Satisfiability Checking - WS 2019/2020

Written Exam I

Saturday, February 22, 2020

Sample solution

1.) SAT Checking

14+8 Points

i) Consider the following four variants of the CDCL algorithm. Which of these are both sound and complete (for checking the satisfiability of propositional logic formulas in CNF)? For each variant, please either give a short argument why it is sound and complete, or provide a counterexample formula showing that it is unsound or incomplete. In your arguments you can rely on soundness and completeness of the CDCL algorithm presented in the lecture.

```
if !BCP() then
    return UNSAT
while true do
    if !decide() then
        return SAT
    while !BCP() do
        if !resolve_conflict() then
            return UNSAT
```

(a) Variant A

```
if !BCP() then
    return UNSAT
while true do
    if !decide() then
        return SAT
    if !BCP() then
        if !resolve_conflict() then
            return UNSAT
```

每次resolve conflict后, 又重新回到decide阶段, 而不是对学习的结果BCP

(b) Variant B

```
while true do
    while !BCP() do
        if !resolve_conflict() then
            return UNSAT
    if !decide() then
        return SAT
```

(c) Variant C

```
while true do
    if !decide() then
        return SAT
    while !BCP() do
        if !resolve_conflict() then
            return UNSAT
```

(d) Variant D

Solution:

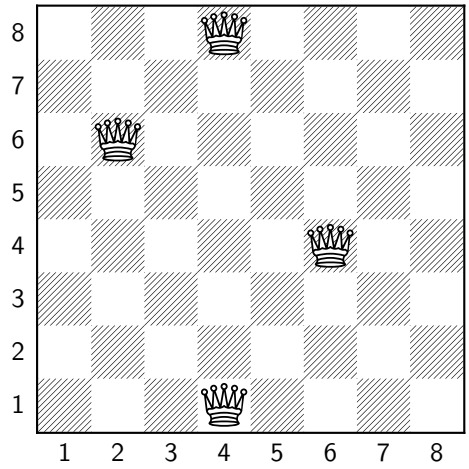
A) This variant is the algorithm from the lecture, therefore it is sound and complete.

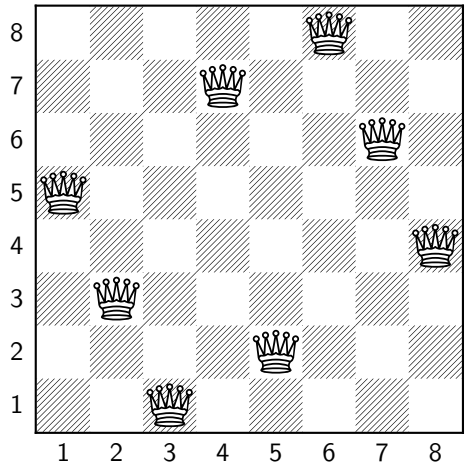
B) Not complete. Let $\varphi = (a \vee b) \wedge (a \vee \neg b)$. BCP does nothing and we decide $\neg a$. Now BCP finds b because of the first clause. The second clause is conflicting and we enter conflict resolution. The conflict clause $(a \vee \neg b)$ is not asserting, we apply resolution to obtain (a). We backtrack the first decision level and continue with the next loop iteration (without propagating again!). We decide $\neg a$ again, so that BCP finds a conflict with (a). As this clause is already asserting, we only backtrack the first decision level and continue with the next loop iteration, entering an infinite loop.

C) This variant is sound and complete. It is gained from the one presented in the lecture by moving the initial call to BCP inside the loop. If the initial call would have returned *false*, resolve_conflict also returns *false* now, as we are still in decision level zero. From this point on, the loop behaves identical (as it has no termination condition).

D) Not correct. Let $\varphi = \mathit{false}$. We immediately return SAT, as no variable is unassigned.

ii) A position $(c, r) \in \{1, \ldots, n\}^2$ on an $n \times n$ chess board is identified by a column index c and a row index r. We say that a queen on a chess board threatens another one (and vice versa) if they are in the same row, column or diagonal. The *n queens problem* poses the question how to place n (chess) queens on an $n \times n$ chess board in such a way that no two queens threaten each other. In the left example below, the queens at positions `(4,8)` and `(2,6)` as well as at `(4,8)` and `(4,1)` threaten each other. On the right, you can see a satisfying solution for the *eight queens problem*.





Encode for the n queens problem the following statements as formulae, using the Boolean variables x_{ij} to encode whether there is a queen at position (i, j):

(a) At least one queen is in column i.
$\varphi_{col}(i) =$ 不是所有行, 所以式子前面不需要加上^

(b) At least one queen is in row j.
$\varphi_{row}(j) =$

(c) A queen at position (i, j) is threatened.
$\varphi_{threat}(i, j) =$

Using the above formulae, please build a formula whose models encode the solutions to the n queens problem.

Solution:

(a) $\varphi_{col}(i) = \bigvee_{j=1}^{n} x_{ij}$

(b) $\varphi_{row}(j) = \bigvee_{i=1}^{n} x_{ij}$

(c) $\varphi_{threat}(i, j) = \bigvee_{i'=1}^{n} \bigvee_{j'=1}^{n} \underbrace{(i \neq i' \vee j \neq j')}_{\text{not the same}} \wedge x_{i'j'} \wedge$

$$\left(\underbrace{i = i'}_{\text{same column}} \vee \underbrace{j = j'}_{\text{same row}} \vee \underbrace{i - i' = j - j' \vee i - i' = j' - j}_{\text{same diagonal}} \right)$$

$$\varphi = \bigwedge_{i=1}^{n} \varphi_{col}(i) \wedge \bigwedge_{j=1}^{n} \varphi_{row}(j) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (x_{ij} \rightarrow \neg \varphi_{threat}(i, j))$$

2.) SMT Solving

23 Points

Consider the equality logic formula

$$\begin{array}{rcccccccc} \varphi^{EQ} & := & (& x_1 = x_2 & \vee & \neg(x_3 = x_4) & & &) \\ & \wedge & (& x_1 = x_2 & \vee & \neg(x_1 = x_4) & \vee & x_1 = x_3 &) \\ & \wedge & (& x_2 = x_3 & \vee & x_1 = x_4 & & &) \\ & \wedge & (& \neg(x_2 = x_3) & \vee & x_1 = x_3 & & &) \end{array}$$

with the Boolean abstraction

$$\begin{array}{rcccccccc} \varphi^{skel} & := & c_1 : (& a & \vee & \neg b & & &) \\ & \wedge & c_2 : (& a & \vee & \neg d & \vee & e &) \\ & \wedge & c_3 : (& c & \vee & d & & &) \\ & \wedge & c_4 : (& \neg c & \vee & e & & &) \end{array}$$

Show how a *less-lazy* SMT solver solves φ^{EQ} for satisfiability as presented in the lecture. Stop the computations after the second conflict has been resolved. Describe

- the initial watch lists of the SAT solver,
- all propagations and decisions of the SAT solver,
- the watch lists of the SAT solver after every decision level,
- the theory call(s) and
- the conflict resolutions in the SAT solver.

Assume that

- in each original clause initially the first two literals are watched,
- the SAT solver uses the lexicographically smallest unassigned variable for a decision and assigns *false* to it,
- all assigned constraints (the true constraints and the negations of false constraints) are passed to the theory solver (i.e. no constraints are dropped because only one polarity is present).

Name: Student number:

Solution:

Watch lists: ☐ as before

a: c_1, c_2	b:	c: c_3	d: c_3	e: c_4
$\neg a$:	$\neg b$: c_1	$\neg c$: c_4	$\neg d$: c_2	$\neg e$:

Assignments:

	1.	2.	3.	4.	5.
DL0					
DL1	$\neg a$	$\neg b$ c_1			

Watch lists: ☐ as before

a: c_1	b:	c: c_3	d: c_3	e: c_4, c_2
$\neg a$:	$\neg b$: c_1	$\neg c$: c_4	$\neg d$: c_2	$\neg e$:

Theory call:

Equalities

x_1

x_4 x_2

x_3

Disequalities

$x_1 \neq x_2$

$x_3 \neq x_4$

Conflict

☐ yes ☒ no

Conflict clause:

Conflict resolution:

Newly learnt clause:

Watch lists: ☒ as before

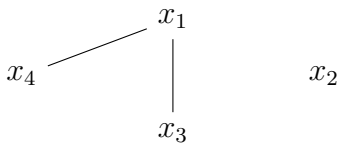
Assignments:

	1.	2.	3.	4.	5.
DL0					
DL1	$\neg a$	$\neg b$ c_1			
DL2	$\neg c$	d c_3	e c_2		

Watch lists: ☒ as before

Theory call:

Equalities



Disequalities

$x_1 \neq x_2$

$x_3 \neq x_4$

$x_2 \neq x_3$

Conflict

☒ yes ☐ no

Conflict clause:

$(x_3 = x_4 \vee x_1 \neq x_4 \vee x_1 \neq x_3)$

Conflict resolution:

$$\frac{(b \vee \neg d \vee \neg e) \quad c_2 : (a \vee e \vee \neg d)}{(a \vee b \vee \neg d)}$$

Newly learnt clause:

$c_5 : (a \vee b \vee \neg d)$

Watch lists: ☐ as before

a: c_1, c_5	b: c_5	c: c_3	d: c_3	e: c_4, c_2
$\neg a$:	$\neg b$: c_1	$\neg c$: c_4	$\neg d$: c_2	$\neg e$:

Name: Student number:

Attention: c_5 can also be added to a and $\neg d$ or b and $\neg d$ instead, see other comment below.

Assignments:

	1.	2.	3.	4.	5.
DL0					
DL1	$\neg a$	$\neg b \quad c_1$	$\neg d \quad c_5$	$c \quad c_3$	$e \quad c_4$

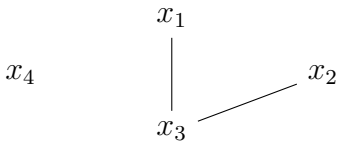
Watch lists: ☐ as before

a: c_1	b: c_5	c: c_3	d: c_3	e: c_4, c_2
$\neg a$:	$\neg b$: c_1	$\neg c$: c_4	$\neg d$: c_2, c_5	$\neg e$:

Attention: If we use the watches b and $\neg d$ when inserting the clause $(a \vee b \vee \neg d)$ (instead of a and b) they do *not* change here. We have not really specified which watches to use here, so both possibilities are okay.

Theory call:

Equalities



Disequalities

$x_1 \neq x_2$
$x_3 \neq x_4$
$x_1 \neq x_4$

Conflict

☒ yes ☐ no

Conflict clause:
$(x_1 = x_2 \vee x_1 \neq x_3 \vee x_2 \neq x_3)$

Conflict resolution:

$$\dfrac{\dfrac{\dfrac{\dfrac{(a \vee \neg c \vee \neg e) \quad c_4 : (\neg c \vee e)}{(a \vee \neg c) \quad c_3 : (c \vee d)}}{(a \vee d) \quad c_5 : (a \vee b \vee \neg d)}}{(a \vee b) \quad c_1 : (a \vee \neg b)}}{(a)}$$

Newly learnt clause:
$c_6 : (a)$

3.) Fourier-Motzkin Variable Elimination 4+6+2+2 Points

Consider the following set of linear real-arithmetic constraints:

$$\begin{array}{lrcrcrcr} c_1: & -x & & & + & 3z & \leq & 6 \\ c_2: & & & & - & z & \leq & 5 \\ c_3: & x & - & 2y & + & 2z & \geq & 0 \\ c_4: & x & - & y & & & = & -1 \\ c_5: & -2x & + & y & + & z & \leq & 2 \\ c_6: & x & - & 2y & & & \geq & -4 \end{array}$$

i) *Eliminate* x by applying Gauß variable elimination. Bring the resulting constraint set into *matrix form* $A \cdot x \leq b$.

Solution: We use c_4 to eliminate x by $x = y - 1$. Furthermore, we invert c'_3 and c'_6 to obtain $\leq$ everywhere. The resulting constraint set is:

$$\begin{array}{lrcrcrcr} c'_1: & -y & + & 3z & \leq & 5 \\ c_2: & & - & z & \leq & 5 \\ c'_3: & -y & + & 2z & \geq & 1 \\ c'_5: & -y & + & z & \leq & 0 \\ c'_6: & -y & & & \geq & -3 \end{array} \quad \rightarrow \quad \begin{array}{lrcrcrcr} c'_1: & -y & + & 3z & \leq & 5 \\ c_2: & & - & z & \leq & 5 \\ c'_3: & y & - & 2z & \leq & -1 \\ c'_5: & -y & + & z & \leq & 0 \\ c'_6: & y & & & \leq & 3 \end{array}$$

ii) *Eliminate* y from the resulting constraint set by applying the Fourier-Motzkin algorithm.

Solution: We first identify lower and upper bounds for y and obtain:

$$\begin{array}{lrcrcl} c'_1: & 3z - 5 & \leq & y & & \\ c'_3: & & & y & \leq & 2z - 1 \\ c'_5: & z & \leq & y & & \\ c'_6: & & & y & \leq & 3 \end{array}$$

We now combine all lower-upper-bound pairs, i.e. (c'_1, c'_3), (c'_1, c'_6), (c'_5, c'_3) and (c'_5, c'_6).

$$\begin{array}{lrcl} (c'_1, c'_3): & 3z - 5 & \leq & 2z - 1 \\ \Rightarrow & z & \leq & 4 \\ \hline (c'_1, c'_6): & 3z - 5 & \leq & 3 \\ \Rightarrow & z & \leq & 8/3 \\ \hline (c'_5, c'_3): & z & \leq & 2z - 1 \\ \Rightarrow & 1 & \leq & z \\ \hline (c'_5, c'_6): & z & \leq & 3 \end{array}$$

The result of the elimination are the above constraints extended with $c_2 : -z \leq 5$, which does not contain y and was thus not involved in the elimination of y.

iii) Sketch the *solution set* of the resulting constraints graphically.

Solution: The variable z can take any values between $max\{-5, 1\} = 1$ and $min\{4, 8/3, 3\} = 8/3$:

iv) Give a *satisfying assignment* for all variables.

Solution: We choose some value for z in $[1, 8/3]$. Then we substitute this value in the upper and lower bounds for y and choose a value for y within the resulting bounds. The assignment for x is then $y-1$ according to c_4. So for example α with $\alpha(x) = 0,\ \alpha(y) = 1$ and $\alpha(z) = 1$ is a satisfying assignment. Also β with $\beta(x) = 1,\ \beta(y) = 2$ and $\beta(z) = 2$ and γ with $\gamma(x) = 2,\ \gamma(y) = 3$ and $\gamma(z) = 2$

4.) Simplex

12+3 Points

i) Apply one *pivoting step* and update the current assignment, where the variable order is $x_1 < x_2 < s_1 < s_2 < s_3 < s_4$ and where the current tableau, the bounds and the current assignment are given by:

	s_1	s_3
s_2	1	2
x_2	-2	0
s_4	3	-1
x_1	2	-1

$$\begin{array}{lcl} s_1 & \geq & 3 \\ s_2 & \leq & -1 \\ s_3 & \leq & -1 \\ s_4 & \leq & 3 \end{array}$$

$$\begin{array}{ll} \alpha(s_1) & = 3 \\ \alpha(s_2) & = 1 \\ \alpha(s_3) & = -1 \\ \alpha(s_4) & = 10 \\ \alpha(x_1) & = 7 \\ \alpha(x_2) & = -6 \end{array}$$

Solution: Two basic variables violate their respective bounds: s_2 and s_4. According to the variable order, we try to pivot s_2 and only s_3 is suitable for pivoting. We decrease the value of s_2 by 2.

We pivot the first row and second column with

$$s_2 = s_1 + 2 \cdot s_3 \Leftrightarrow s_3 = 1/2 \cdot s_2 - 1/2 \cdot s_1$$

and obtain

	s_1	s_2
s_3	$-1/2$	$1/2$
x_2	-2	0
s_4	$7/2$	$-1/2$
x_1	$5/2$	$-1/2$

$$\begin{array}{lcl} s_1 & \geq & 3 \\ s_2 & \leq & -1 \\ s_3 & \leq & -1 \\ s_4 & \leq & 3 \end{array}$$

$$\begin{array}{ll} \alpha(s_1) & = 3 \\ \alpha(s_2) & = -1 \\ \alpha(s_3) & = -2 \\ \alpha(s_4) & = 11 \\ \alpha(x_1) & = 8 \\ \alpha(x_2) & = -6 \end{array}$$

ii) The simplex method terminates on the resulting tableau. Is the tableau satisfied or conflicting? Why? Give the satisfying assignment for the original variables or the infeasible subset.

Solution: The tableau is conflicting. s_4 is not within its bound and no suitable non-basic variable exists for pivoting s_4.

The infeasible subset thus corresponds to the third row, i.e. the constraints corresponding to s_4, s_1 and s_2.

5.) Integer Arithmetic

6+3 Points

Branch&Bound was presented as an incomplete method for linear integer arithmetic.

i) Give the core algorithm of Branch&Bound as pseudocode.

Solution:

```
function Branch&Bound(P) {
        res = LRA(Relaxed(P));
        if (res == UNSAT) return UNSAT;
        if (isInteger(res)) return SAT;
        v = variable s.t. res(v) ∉ ℤ;
        if (Branch&Bound(P ∪ {v ≤ ⌊res(v)⌋}) == SAT) return SAT;
        if (Branch&Bound(P ∪ {v ≥ ⌈res(v)⌉}) == SAT) return SAT;
        return UNSAT;
}
```

ii) Can we apply Branch&Bound to nonlinear integer arithmetic as well?
Sketch how the pseudocode can be adapted, or argue why this can not be done.

Solution: This is possible. We only need to use a method for nonlinear real arithmetic (like CAD) instead of Simplex.

6.) Interval Constraint Propagation **6+1+2 Points**

i) Apply *interval arithmetic* as presented in the lecture to compute the following:

Solution:

a) $[1;2] - [-3;4] + [1;3] = [-3;5] + [1;3] = [-2;8]$

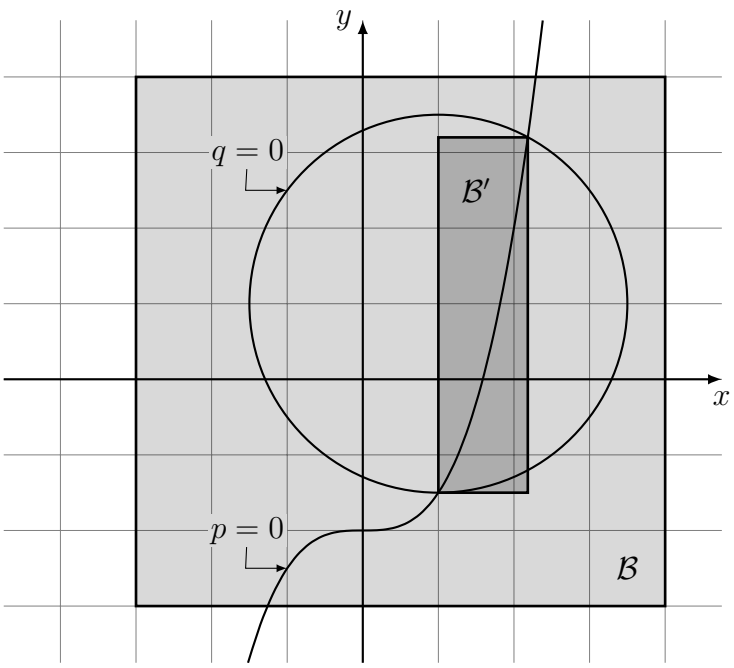
b) $[3;5] \cdot [-2;2] = [-10;10]$

c) $[4;5]/(-\infty;-2] = [-\frac{5}{2};0]$

d) $[1;2] + [-3;4] \cdot [5;6] = [1;2] + [-18;24] = [-17;26]$

ii) Assume two constraints $p(x,y) = 0$ and $q(x,y) = 0$ in the variables x and y and an initial box $\mathcal{B} \subseteq \mathbb{R}^2$ restricting the values of the variables x and y, as shown below. Sketch *the smallest box* that contains all common solutions to both constraints inside $\mathcal{B}$.

Solution:



iii) Why is the ICP algorithm as presented in the lecture designed to be able to return UNSAT but not designed to return SAT? What does the ICP algorithm do instead of returning SAT?

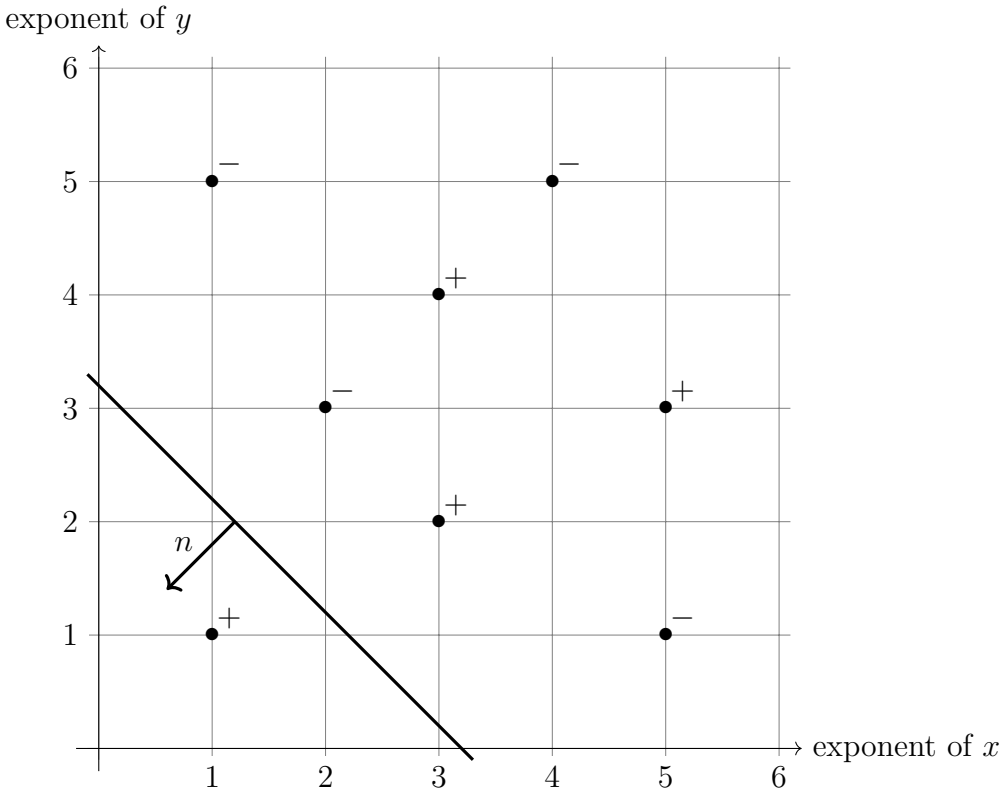
Solution: ICP as presented in the lecture performs contraction until a certain interval width is reached for each dimension. In case no contradiction (no empty interval) has been found, the resulting box is a solution candidate. Nonetheless there is no guarantee that contraction will result in a single solution point eventually, thus ICP can only provide solution candidate boxes potentially containing a solution. However, in case the algorithm contracts every box derived from the initial box to the empty set, the algorithm can return UNSAT.

7.) Subtropical Satisfiability **8 Points**

Consider the polynomial $f(x,y) = 2xy - 2xy^5 - 3x^2y^3 + x^3y^2 + x^3y^4 - 2x^4y^5 + 7x^5y^3 - 6x^5y$. Show the points of both $\mathit{frame}^+(f)$ and $\mathit{frame}^-(f)$ in the graphic below.

Graphically identify a hyperplane that separates some $p \in \mathit{frame}^+(f)$ from the remaining frame and give its normal vector n. Use n to provide a $f^*(a)$ such that $f^*(a) > 0$ for a sufficiently large a.

Solution:



We read the exponents of f and put them into the graphic. We identify the given hyperplane with $n = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$ and obtain

$$f^*(a) = f(a^{-1}, a^{-1}) = 2a^{-2} - 2a^{-5} - 8a^{-6} + a^{-7} + 7a^{-8} - 2a^{-9}$$

8.) Virtual Substitution **6+3 Points**

i) Assume we want to virtually substitute the test candidate $t = 2 + \epsilon$ *for the variable* y in the constraint $2 \cdot x^2 \cdot y - 3 \cdot x^2 > 0$. Please identify the appropriate substitution rule from below, apply virtual substitution and simplify the result as far as possible.

- Substitute $e + \varepsilon$ for x in $b \cdot x + c > 0$:

$$\begin{array}{cccccc} & (& (bx + c > 0)[e//x] & & &) \\ \vee & (& (bx + c = 0)[e//x] & \wedge & (b > 0)[e//x] &) \end{array}$$

- Substitute $e + \varepsilon$ for x in $b \cdot x + c \geq 0$:

$$\begin{array}{cccccc} & (& (bx + c > 0)[e//x] & & &) \\ \vee & (& (bx + c = 0)[e//x] & \wedge & (b > 0)[e//x] &) \\ \vee & (& b = 0 & \wedge & c = 0 &) \end{array}$$

- Substitute $e + \varepsilon$ for x in $a \cdot x^2 + b \cdot x + c > 0$:

$$\begin{array}{cccccccc} & (& (ax^2 + bx + c > 0)[e//x] & & & & &) \\ \vee & (& (ax^2 + bx + c = 0)[e//x] & \wedge & (2ax + b > 0)[e//x] & & &) \\ \vee & (& (ax^2 + bx + c = 0)[e//x] & \wedge & (2ax + b = 0)[e//x] & \wedge & (2a > 0)[e//x] &) \end{array}$$

Solution: Note that we substitute for y and not for x. We need to compute $(b \cdot y + c > 0)[e + \epsilon//y]$ with $e = 2$, $b = 2 \cdot x^2$ and $c = -3 \cdot x^2$. I.e., we need to instantiate the first rule, where we instantiate x, b, c and e, respectively, by y, $2 \cdot x^2$, $-3 \cdot x^2$ and 2:

Substitute $e + \varepsilon$ into $b \cdot y + c > 0$:

$$\begin{array}{cccccc} & (& (by + c > 0)[e//y] & & &) \\ \vee & (& (by + c = 0)[e//y] & \wedge & (b > 0)[e//y] &) \end{array}$$

$$\begin{array}{cccccc} & (& (2x^2y - 3x^2 > 0)[2//y] & & &) \\ \vee & (& (2x^2y - 3x^2 = 0)[2//y] & \wedge & (2x^2 > 0)[2//y] &) \end{array}$$

which evaluates to

$$\begin{array}{cccccc} & (& (x^2 > 0) & & &) \\ \vee & (& (x^2 = 0) & \wedge & (x^2 > 0) &) \end{array}$$

and further simplifies to

$$x^2 > 0 \ .$$

Note: the last rule is also applicable with $a = 0$ (actually it is then the same).

ii) Give a univariate constraint whose test candidates are $-\infty$, $-1 + \varepsilon$ and $1 + \varepsilon$. Show how you derive it.

Solution: From the test candidates $-1 + \varepsilon$ and $1 + \varepsilon$, we know that the polynomial has the roots -1 and 1. Thus, we have $(x + 1) \cdot (x - 1) = x^2 - 1$. As we only have test candidates with $+\varepsilon$ (and $-\infty$) we need to have a strict inequality:

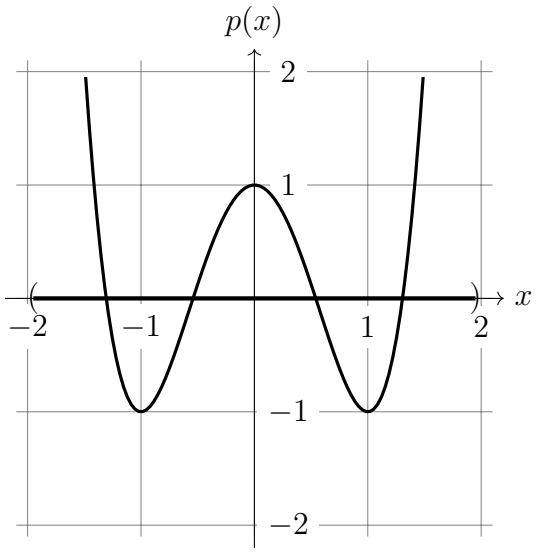
$$x^2 - 1 < 0 \ .$$

9.) Cylindrical Algebraic Decomposition

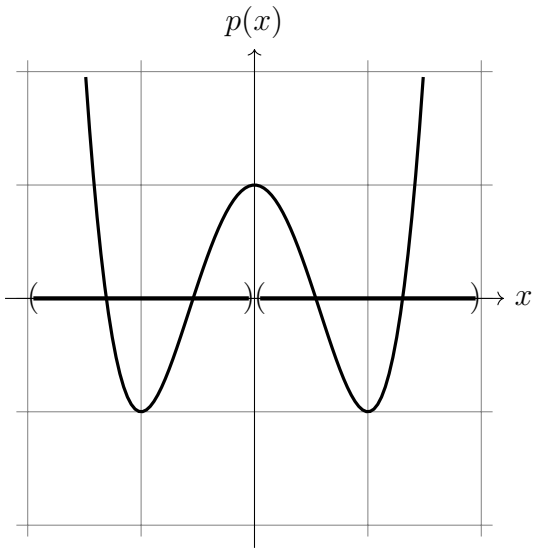
7+4 Points

i) Isolate all real zeros of the univariate polynomial $p = 2x^4 - 4x^2 + 1$ in the interval $(-2, 2)$ with the method presented in the lecture, using interval midpoints for splitting. You do not need to compute Sturm sequences, you can read off all needed information from the plots below. After each step, please depict all pending intervals and indicate when a root has been isolated (there are more plots than necessary).

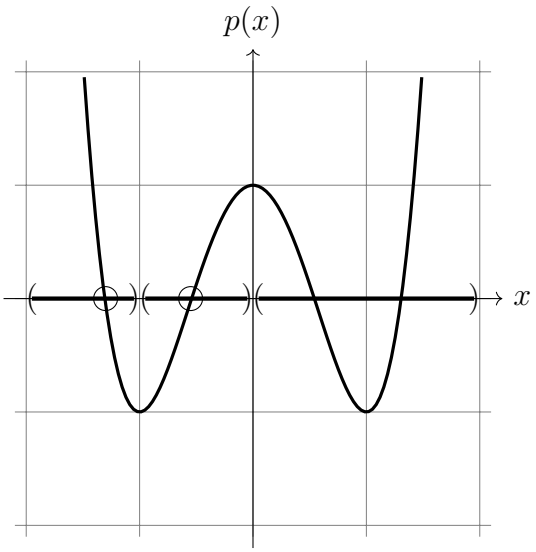
Solution:



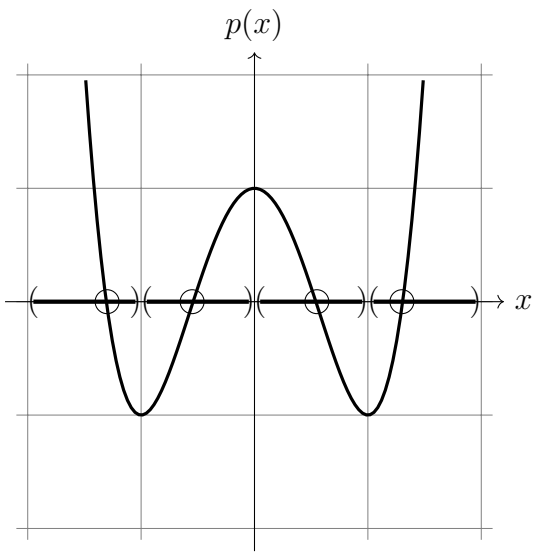
Split at 0



Split at -1



Split at 1



Isolated all roots

ii) One of the utility methods for the cylindrical algebraic decomposition method presented in the lecture are *Sturm sequences*. What is the *input* of this method and what does it compute?

Solution: A method calculating a Sturm sequence accepts a *univariate polynomial* and an *open interval* and returns a *positive number* c where c is the number of real roots of the polynomial within the open interval.