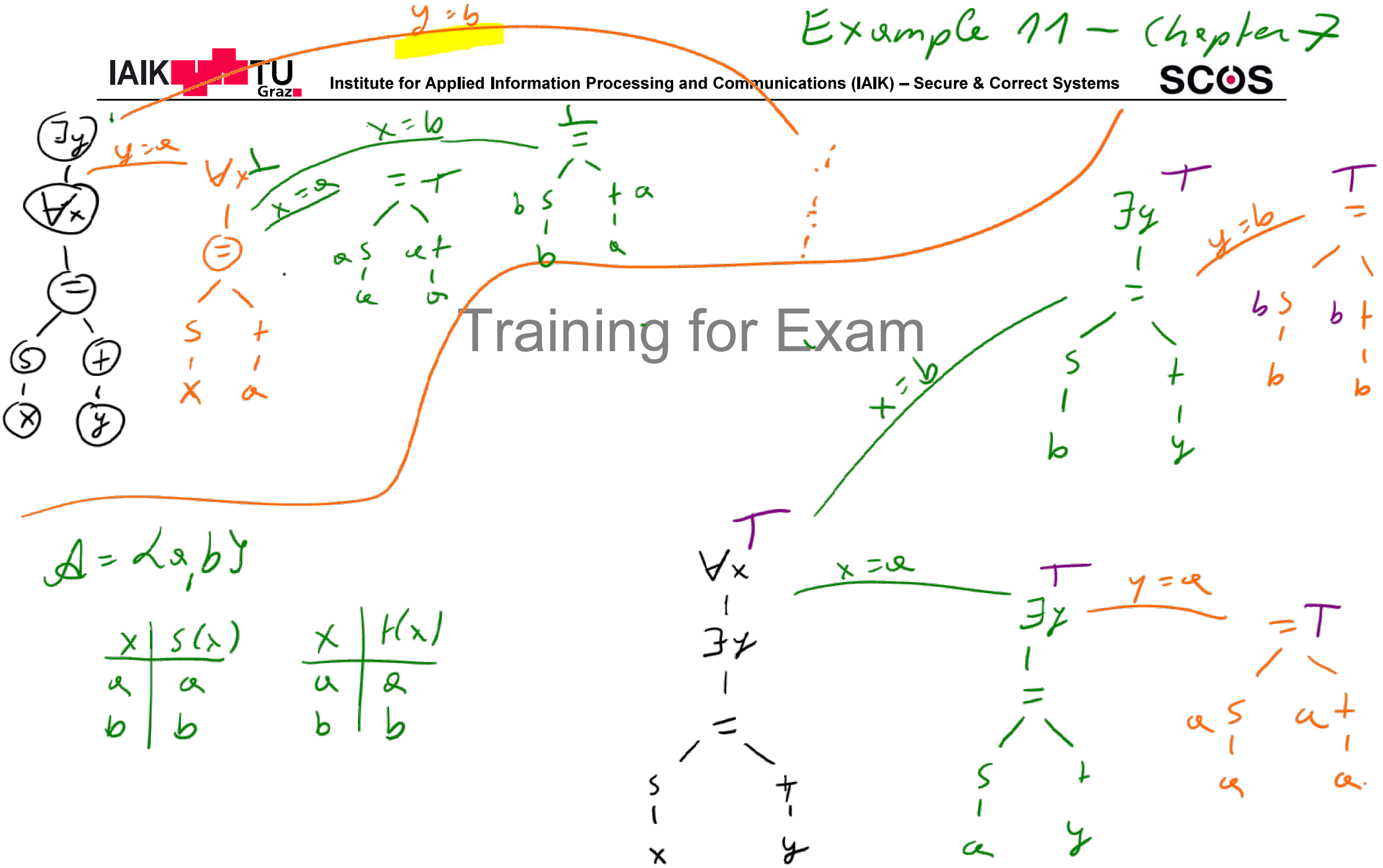
Example 11 – Chapter 7



x	S(x)
a	a
b	b

x	H(x)
a	a
b	b

Consider the propositional formula $\varphi = p \rightarrow (q \rightarrow r)$.

(a) Fill out the truth table for φ (and its subformulas). Write T for true and F for false.

Propositional Logic

p	q	r	$(q \rightarrow r)$	$\varphi = p \rightarrow (q \rightarrow r)$
F	F	F	1	1
F	F	T	1	1
F	T	F	0	1
F	T	T	1	1
T	F	F	1	1
T	F	T	1	1
T	T	F	0	0
T	T	T	1	1

(b) Is φ satisfiable? ✓

(c) Is φ valid? ✗

(d) Give a formula ψ that is semantically equivalent to φ, but does not use the "$\rightarrow$" connective. $\neg(p \wedge q \wedge r)$

(e) How can you check whether ψ is semantically equivalent to φ?

$\varphi \oplus \psi$ is UNSAT $\Leftrightarrow \varphi \equiv \psi$

Prop + Pred Logic

Test 2 for the Practicals - Predicate Logic

[Example 1:] Model the following sentences with predicate logic, as detailed as possible. Clearly indicate the intended meaning of all function, predicate, and constant symbols that you use.

Also, model the same sentence in propositional logic, as detailed as possible. Clearly indicate the intended meaning of each propositional variable you use.

a) *"Every even integer greater than 2 is equal to the sum of two prime numbers."*

b) *"Not all birds can fly, but some birds can fly."*

a)

Integer(x) ... x is an Integer Even(x) ... x is even Sum(x,y,z) = "(x = y + z)"

Prime(x) ... x is Prime GT(x,y) ... "x ≥ y"

Pred Logic: $\forall x (\text{Integer}(x) \wedge \text{Even}(x) \wedge GT(x,2) \rightarrow \exists y \exists z. (\text{Prime}(y) \wedge \text{Prime}(z) \wedge \text{Sum}(x,y,z)))$

Propositional Logic: p

b)

Bird(x) ... x is a bird Fly(x) ... x can fly

Pred Logic: $\neg \forall x (\text{Bird}(x) \wedge \text{Fly}(x)) \wedge \exists x (\text{Bird}(x) \wedge \text{Fly}(x))$

propositional Logic: $\neg p \wedge q$

p .... All birds can fly q ... some birds can fly

Construct a (reduced and ordered) BDD for the function

BDD

$$f = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z),$$

using alphabetic variable order. Use complemented edges and a node for "true" as the only constant node. To simplify drawing, you may assume that "dangling" edges point to the constant node. Write down all cofactors that you compute to obtain the final result and clearly mark them in your graph.

$f_x = (y \vee z) \wedge (\neg y \vee \neg z)$

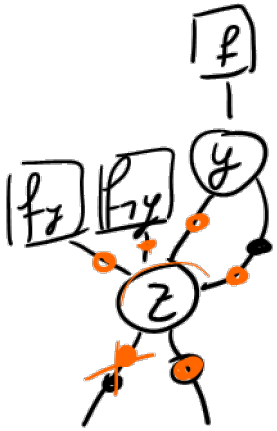
$f_{xz} = \neg z = f_y$

$f_{xyz} = \bot$

$f_{xy\neg z} = \top$

$f_{\neg y} = f_{x\neg y} = z = \neg f_{xz} = \neg f_y$

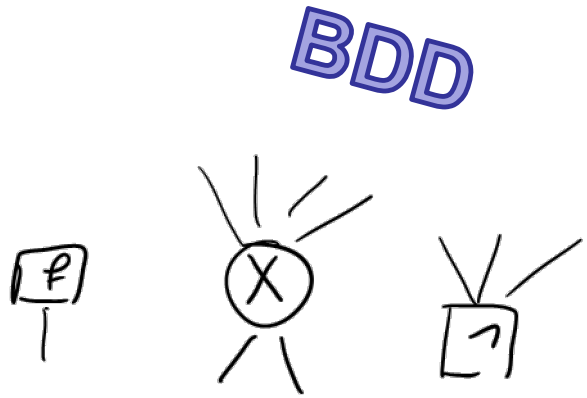
$f_{\neg x} = (y \vee z) \wedge (\neg y \vee \neg z) = f_x$



1. Reduced and Ordered **Binary Decision Diagrams** (BDDs).

   (a) For each type of node that can occur in a BDD, state how many incoming and outgoing edges this type of node has. Write your answers in the following table. If the number of certain edges is always a fixed number, write that number. If the number of certain edges can be arbitrary, write "A".

Node Type	Incoming Edges	Outgoing Edges
Function Node	0	1
Internal Node	A	2
Terminal Node	A	0



BDD

(b) Use the following BDD to check if the function it represents evaluates to *true* or *false* with the following assignments for the variables. Tic the correct answers.

- ☐ The BDD evaluates to *true* with the assignment $a = \top, b = \bot, c = \top$.
- ☒ The BDD evaluates to *false* with the assignment $a = \top, b = \bot, c = \top$.
- ☒ The BDD evaluates to *true* with the assignment $a = \bot, b = \top, c = \bot$
- ☐ The BDD evaluates to *false* with the assignment $a = \bot, b = \top, c = \bot$

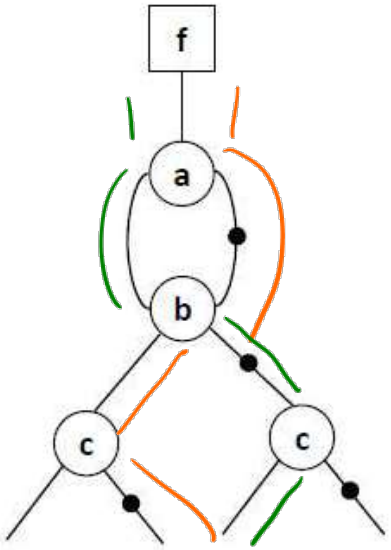
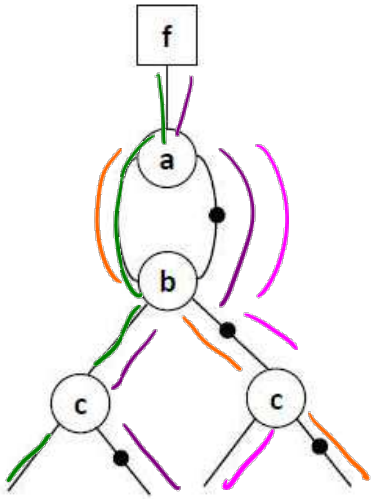


Figure 1: BDD



BDD

Figure 1: BDD

(c) Tic the correct propositional formula for the function f that is represented by the BDD in Figure 1.

- ☐ $(a \wedge b \wedge c) \vee (\neg a \wedge \neg b \wedge \neg c)$
- ☒ $(a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$
- ☐ $(a \vee b \vee c) \wedge (\neg a \vee \neg b \vee \neg c)$
- ☐ $(a \vee b \vee c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$

(d) Is the BDD in Figure 1 reduced? Tic the correct answer.

- ☐ Yes.
- ☐ No.

Consider the domain $A = \{Spain, France, Italy, Germany\}$ and the two different symbolic encodings for A given below. Which one gives a shorter characteristic function for the set $B = \{France, Germany\}$? Illustrate your answer by giving the characteristic function for B in both encodings.

Symbolic Encoding

Encoding 1		
Element	x	y
Spain	0	0
France	1	0
Italy	0	1
Germany	1	1

Encoding 2		
Element	x	y
Spain	0	0
France	1	0
Italy	1	1
Germany	0	1

$(x \wedge \neg y) \vee (x \wedge y)$

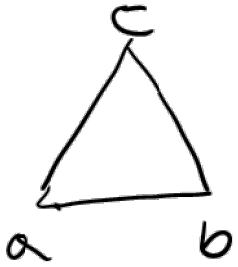
$\equiv x$

$(x \wedge \neg y) \vee (\neg x \wedge y)$

$\equiv x \oplus y$

shorter encoding

The Graph Based Reduction is used to reduce a formula φ_{in} in $\mathcal{T}_E$ to a formula in prop. logic that is equisatisfiable. Two formulas are equisatisfiable if both sat. or both unsat. In the first step of the algorithm, we create a Non-Polar Equality Graph and in the next step we make it chordal. The graph is chordal, if no cycles with size > 3. We introduce fresh propositional variables for each equation to ensure symmetry. In order to ensure transitivity, the algorithm adds constraints of the form $e_{a=b} \wedge e_{b=c} \rightarrow e_{a=c} \wedge \ldots$ for all Δ in the graph. The resulting equisatisfiable formula consists of two parts and is of the form: $\varphi_{out} := \varphi_{TC} \wedge \hat{\varphi}_{in}$. The right part of the formula $\hat{\varphi}_{in}$ describes the flattening original formula in which we replace equalities with the fresh variables



SMT

Ergor Enc.

$\mathcal{T}_{EUF}$

↓ Ackermans

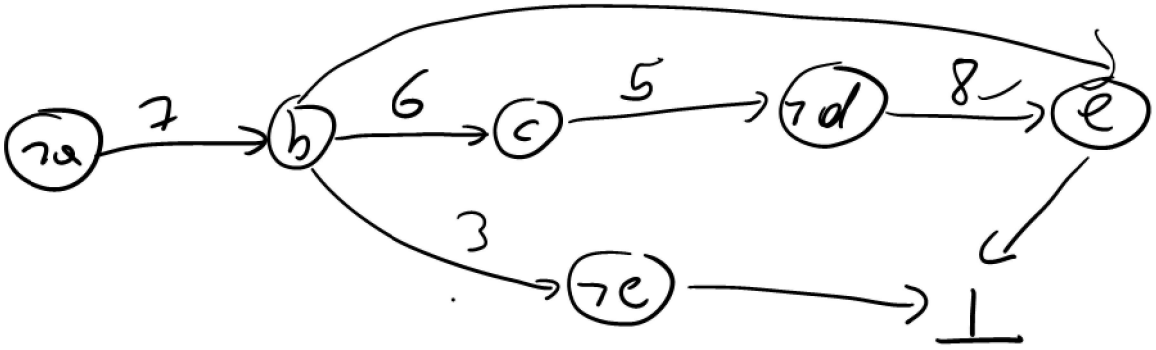
$\mathcal{T}_E$

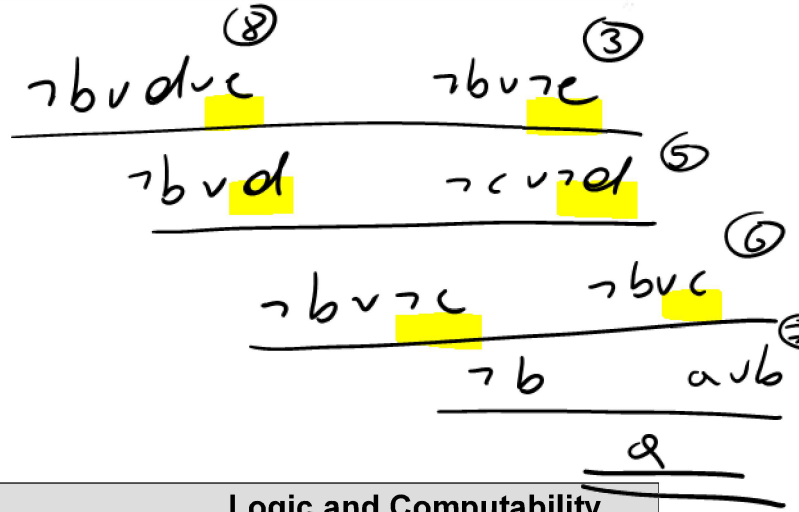
↓ Graph Based R.

prop. logic

DPLL

Step	1	2	3	4	5	6				
Level	0	1	1	1	1	1				
Ass.	✓	¬a	¬a b	¬a b c	¬a b c d	¬a b c ¬d e				
1. b, d	1	1	✓	✓	✓	✓				
2. b, c	2	2	✓	✓	✓	✓				
3. ¬b, ¬e	3	3	¬e	¬e	¬e	✓				
4. ¬a, ¬c	4	✓	✓	✓	✓	✓				
5. ¬c, ¬d	5	5	¬c ¬d	¬d	✓	✓				
6. ¬b, c	6	6	c	✓	✓	✓				
7. a, b	7	b	✓	✓	✓	✓				
8. ¬b, d, e	8	8	d e	d e	e	¬e				
9. a (LC)						a				
BCP		b	c	¬d	¬c					
PL										
Dec	¬a									





Learned Clause a

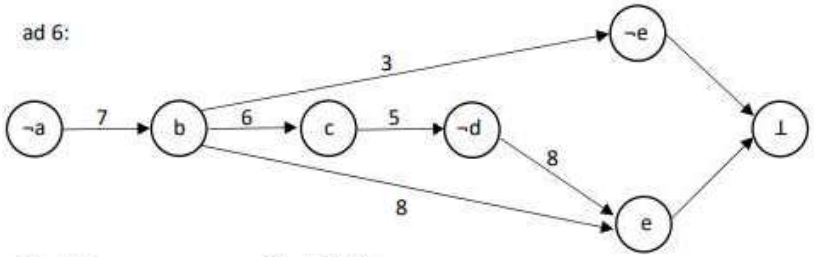
DPLL

C1 = {b, d}
C2 = {b, c}
C3 = {¬b, ¬e}
C4 = {¬a, ¬c}
C5 = {¬c, ¬d}
C6 = {¬b, c}
C7 = {a, b}
C8 = {¬b, d, e}

Solution:

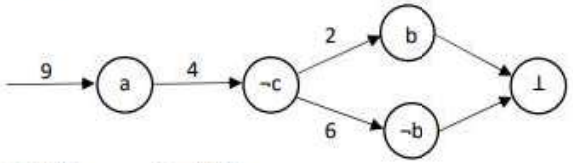
Step	1	2	3	4	5	6	7	8	9	10
Level	0	1	1	1	1	1	0	0	0	0
Ass.	/	¬a	¬a,b	¬a,b,c	¬a,b,c,¬d	¬a,b,c,¬d,¬e	/	a	a,¬c	a,¬b,¬c
1. b, d	b,d	b,d	√	√	√	√	b,d	b,d	b,d	d
2. b, c	b,c	b,c	√	√	√	√	b,c	b,c	b	{ }
3. ¬b, ¬e	¬b,¬e	¬b, ¬e	¬e	¬e	¬e	√	¬b,¬e	¬b,¬e	¬b,¬e	√
4. ¬a, ¬c	¬a,¬c	√	√	√	√	√	¬a,¬c	¬c	√	√
5. ¬c, ¬d	¬c,¬d	¬c,¬d	¬c,¬d	¬d	√	√	¬c,¬d	¬c,¬d	√	√
6. ¬b, c	¬b,c	¬b,c	c	√	√	√	¬b,c	¬b,c	¬b	√
7. a, b	a,b	b	√	√	√	√	a,b	√	√	√
8. ¬b, d, e	¬b,d,e	¬b,d,e	d,e	d,e	e	{ }	¬b,d,e	¬b,d,e	¬b,d,e	√
9. a (LC)							a	√	√	√
BCP		b	c	¬d	¬e		a	¬c	¬b	
PL										
Dec	¬a									

ad 6:



3. ¬b ∨ ¬e 8. ¬b ∨ d ∨ e
¬b ∨ d 5. ¬c ∨ ¬d
¬b ∨ ¬c 6. ¬b ∨ c
¬b 7. a ∨ b
a

ad 10:



2. b ∨ c 6. ¬b ∨ c
c 4. ¬a ∨ ¬c
¬a 9. a
⊥

UNSAT

For the following example, the SMT solver Z3 is not able to return a *model*. In general, what does Z3 understand under a model? Why is there no model for this example?

Bonus-SMT-Solver

```
1 (declare-fun x () Bool)
2 (assert (and x (not x)))
3 (check-sat)
4 (get-model)
```

Z3: Model is assignment that makes the formula true

$(x \wedge \neg x) \rightarrow$ UNSAT $\rightarrow$ there is no model (that makes the formula true)