1 Proof

(20p) Prove the following sequents:

(a) $p \rightarrow q, r \rightarrow s \vdash p \lor r \rightarrow q \lor s$
(b) $(p \land q) \lor (p \land r) \vdash p \land (q \lor r)$
(c) $\vdash \neg p \rightarrow (p \rightarrow (p \rightarrow q))$
(d) $q \vdash (p \land q) \lor (\neg p \land q)$ using LEM
(e) $p \land q \vdash (\neg p \lor \neg q)$

Answer:

(a)

1 $p \rightarrow q$ premise
2 $r \rightarrow s$ premise
3 $p \lor r$ assumption
4 $p$ assumption
5 $q$ \rightarrow e 1,4
6 $q \lor s$ \forall 5
7 $r$ assumption
8 $s$ \rightarrow e 2,7
9 $q \lor s$ \forall 8
10 $q \lor s$ \forall e 3, 4-6, 7-9
11 $p \lor r \rightarrow q \lor s$

(b)
(c)
1 \( \neg p \) assumption
2 \( p \) assumption
3 \( \bot \) \( \neg e 1,2 \)
4 \( p \rightarrow q \) \( \bot e 3 \)
5 \( p \rightarrow (p \rightarrow q) \) \( \rightarrow I \ 2,4 \)
6 \( \neg p \rightarrow (p \rightarrow (p \rightarrow q)) \) \( \rightarrow I \ 1,5 \)

(d)
1 \( q \) premise
2 \( p \lor \neg p \) LEM
3 \( p \) assumption
4 \( p \land q \) \( \land i 1,3 \)
5 \( (p \land q) \lor (\neg p \land q) \) \( \lor i 1 \ 4 \)
6 \( \neg p \) assumption
7 \( \neg p \land q \) \( \land i 1,6 \)
8 \( (p \land q) \lor (\neg p \land q) \) \( \lor i 1 \ 7 \)
9 \( (p \land q) \lor (\neg p \land q) \) \( \lor e 2, 3-5, 6-8 \)

(e)
1 \( p \land q \) premise
2 \( p \) \( \land e_1 1 \)
3 \( q \) \( \land e_2 1 \)
4 \( (\neg p \lor \neg q) \) assumption
5 \( \neg p \) assumption
6 \( \bot \) \( \neg e 2, 5 \)
7 \( \neg q \) assumption
8 \( \bot \) \( \neg e 3, 7 \)
9 \( \bot \) \( Ve 4, 5-6, 7-8 \)
10 \( \neg (\neg p \lor \neg q) \) RAA 4,9
2 Validity
(15p) Determine the validity of the following semantic entailment by constructing the truth table.

\[ \neg r \rightarrow (p \lor q), \; r \land \neg q \models r \rightarrow q \]

<table>
<thead>
<tr>
<th>r</th>
<th>p</th>
<th>q</th>
<th>r \land \neg q</th>
<th>\neg r \rightarrow (p \lor q)</th>
<th>r \rightarrow q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Answer: The valuation at rows 2 and 4 makes the premises to be true and the corresponding conclusion to be false. Invalid.

3 CNF and SAT solving
(25p)
If I study, then I will not fail basket weaving 101. If I do not play cards to often, then I will study. I failed basket weaving 101. Therefore, I played cards too often.

(a) Model the above statement to a formula. Turn the formula to its CNF form to decide validity. (15p)
(b) Decide its validity by formulating it as a satisfiability problem and solving it. Please show the parse tree and the value assignment process. (10p)

Answer:
(a) Let \( s, f \) and \( p \) stand for Study, Fail, and PlayCards. It is equivalent to prove

\[ \vdash (s \rightarrow \neg f) \rightarrow ((\neg p \rightarrow s) \rightarrow (f \rightarrow p)) \]

It equals to

\[
\neg (\neg s \lor \neg f) \lor (\neg (\neg p \lor s) \lor (\neg f \lor p)) \\
= (s \land f) \lor (\neg p \land \neg s) \lor \neg f \lor p \\
= ((s \land f) \lor \neg p) \land (s \land f) \lor \neg s \lor \neg f \lor p \\
= ((s \lor \neg p) \land (f \lor \neg p) \land (s \lor \neg s) \land (f \lor s)) \lor \neg f \lor p \\
= ((s \lor \neg p \lor \neg f \lor p) \land (f \lor \neg p \lor \neg f \lor p)) \\
\]

We can see at each clause, there exist a literal and its negation, leading to a valid argument.
(b) We determine the satisfiability of the negation of the original formula.

\[ \neg((s \rightarrow \neg f) \rightarrow ((\neg p \rightarrow s) \rightarrow (f \rightarrow p))) \]

It equals to

\[ (s \rightarrow \neg f) \land \neg((\neg p \rightarrow s) \rightarrow (f \rightarrow p)) \]
\[ = \neg(s \land f) \land \neg((\neg p \land s) \rightarrow \neg(f \land \neg p)) \]
\[ = \neg(s \land f) \land \neg((\neg p \land s) \lor \neg(f \land \neg p)) \]
\[ = \neg(s \land f) \land \neg(\neg(\neg p \land s) \land (f \land \neg p)) \]

\[ \neg(x \land y) \]

Figure 1:

Observe that there is a contradiction between steps 3, 5, and 8.

4 SAT solving

(a) Find a formula such that the cubic SAT solving algorithm cannot decide its satisfiability, that is, it fails to converge on a deterministic value for any unmarked node. Please do not use examples from the textbook. (10p)

(b) Devise a try-and-backtrack SAT solving algorithm that operates on formulas with \( \neg \) and \( \land \). You can use the linear algorithm as a primitive to construct your algorithm, e.g. linear_solve() means performing the linear inference as much as possible. (10p)
When the linear resolution gets stuck at the conjunction, the algorithm tries to proceed with both T/F with x, then the corresponding y has F/T value. However, neither has a deterministic value. Therefore the algorithm gets stuck.

(b)

```python
1 resolve(ast) {
2    linear_resolution();
3    if (contradition was found) return;
4    if (all nodes are consistently assigned) {
5        output the assigned ast;
6        exit;
7    }
8    for (each unmarked node n in ast) {
9        assign(n)=T;
10       resolve(ast);
11        assign(n)=F;
12       resolve(ast);
13       unassign(n);
14    }
15    return
16}
```

At each call of resolve(), the algorithm tries its best to perform linear inference of assignments. If there are still unresolved nodes, it traverses each unmarked node, trying to proceed with assigning T to the node and then assigning F. Note that it remove the mark at the end of each iteration of the loop.
5 The DIMACS Format
(10p) Most fast SAT solvers require the input formula in CNF. The input CNF formula is specified in the DIMACS format. Consider the following file sample.cnf in DIMACS format.

```
p cnf 4 5
1 0
2 -3 0
-4 -1 0
-1 -2 3 4 0
-2 4 0
```

The first line (p cnf x y) says that the input is a CNF formula containing x variables and y clauses. Our example has 4 variables (1, 2, 3, 4) and five clauses. The negation of a variable is denoted by putting a minus sign in front of the variable number. Each clause is described in a line terminated by a zero. Note that 0 cannot be used as a variable number. So sample.cnf denotes the following CNF formula: $1 \land (2 \lor \neg 3) \land (\neg 4 \lor \neg 1) \land (\neg 1 \lor \neg 2 \lor 3 \lor 4) \land (\neg 2 \lor 4)$.

Express the following clause set in the DIMACS format. Use the variable name $i$ in the DIMACS format for the variable named $x_i$ in the the above clauses. (For example, use 4 for $x_4$.)

\[
\begin{align*}
\omega_1 &= x_1 \lor x_3 \lor x_4 \\
\omega_2 &= x_1 \lor \neg x_2 \lor x_3 \\
\omega_5 &= x_2 \lor x_4 \\
\omega_3 &= \neg x_1 \lor x_2 \lor x_3 \\
\omega_4 &= x_1 \lor \neg x_2 \lor x_4 \\
\omega_6 &= x_3 \lor x_4
\end{align*}
\]

Answer:
```
p cnf 4 5
1 3 4 0
1 -2 3 0
-1 2 3 0
-1 -2 4 0
2 4 0
3 4 0
```

6 Using a SAT solver
(10p)

Some publicly available fast SAT solvers are MiniSat, zChaff, siege. For this assignment you will install and use the MiniSat SAT solver which was the fastest SAT solver in the SAT-competitions of 2005 and 2006. You can run MiniSat SAT solver simply by the following command:

```
/path/MiniSat_v1.14_linux sample.cnf sample.result
```

The file sample.cnf is a description of a CNF formula in DIMACS format. MiniSat reports whether the given formula is (un)satisfiable in the file sample.result. If the formula is satisfiable, then a satisfying assignment is also written to sample.result.
Run MiniSat on the DIMACS file you made from the previous problem. What is the output stored in results file?

Answer: Satisfiable, -1 -2 -3 4 0
7. Describe the execution of DPLL on the following formulae (20p).

(a) \((P \lor \neg Q \lor \neg R) \land (Q \lor \neg P \lor R) \land (R \lor \neg Q)\)

(b) \((P \lor Q \lor R) \land (\neg P \lor \neg Q \lor \neg R) \land (\neg P \lor Q \lor R) \land (\neg Q \lor R) \land (Q \lor \neg R)\)

(a)

Step 1: \(P=T\), we have \(C_1\) (the first clause) = T
Step 2: \(P=T, Q=T\), we have \(C_1=T, C_2=T\)
Step 3: \(P=T, Q=T, R=T\), we have \(C_1=C_2=C_3=T\)

(b)

Step 1: \(P=T\), we have \(C_1=T\)
Step 2: \(P=T, Q=F\), we have \(C_1=T, C_2=T, C_4=T\)
Step 3: \(P=T, Q=F, R=T\), we have \(C_1=T, C_2=T, C_3=T, C_4=T, C_5=F\)
    roll back to 3 and reassign \(R\)
Step 4: \(P=T, Q=F, R=F\), we have \(C_1=T, C_2=T, C_3=F, C_4=T, C_5=T\)
    roll back to 2 and reassign \(Q\)
Step 5: \(P=T, Q=T\), we have \(C_1=T, C_3=T, C_5=T\)
Step 6: \(P=T, Q=T, R=F\), we have \(C_1=T, C_2=T, C_3=T, C_4=F, C_5=T\)
    roll back to 6 and reassign \(R\)
...
    roll back to 1 and reassign \(P\)

Step 10: \(P=F\)
Step 11: \(P=F, Q=T\)
Step 12: \(P=F, Q=T, R=F, SAT\)
Static Analysis from Homework 2 (20p)

1. int y, t, p;
2. float x, z;
3. x=random(0.0, 1.0); /*x is a random sample from [0.0, 1.0];*/
4. y=random(-50, 50); /*y is a random sample in [-50, 50];*/
5. if (y<0)
6. p= -y;
7. else
8. p=y;
9. z=100;
10. while (p>0) {
11. z=z*x;
12. p=p-1;
13. }
14. output(z);
15. output(p);

Answer:

(1)

Iteration one

<table>
<thead>
<tr>
<th>trace condition</th>
<th>range</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=random(0.0, 1.0)</td>
<td>([0,1])</td>
<td>([0,1])</td>
</tr>
<tr>
<td>y=random(-50, 50)</td>
<td>([-50,50])</td>
<td>([-50,50])</td>
</tr>
<tr>
<td>if (y&lt;0)</td>
<td>([-\infty,0)) (&amp;) ([-50,50])</td>
<td>([-50,50])</td>
</tr>
<tr>
<td>p= -y</td>
<td>([0,50])</td>
<td>([0,50])</td>
</tr>
<tr>
<td>p=y</td>
<td>([0,50])</td>
<td>([0,50])</td>
</tr>
<tr>
<td>z=100</td>
<td>([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>while (p&gt;0)</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>z=z*x</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>p=p-1</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
</tbody>
</table>

Iteration 2

<table>
<thead>
<tr>
<th>trace condition</th>
<th>range</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=random(0.0, 1.0)</td>
<td>same as iteration 1</td>
<td>same</td>
</tr>
<tr>
<td>y=random(-50, 50)</td>
<td>same</td>
<td>same</td>
</tr>
<tr>
<td>if (y&lt;0)</td>
<td>same</td>
<td>same</td>
</tr>
<tr>
<td>p= -y</td>
<td>same</td>
<td>same</td>
</tr>
<tr>
<td>p=y</td>
<td>same</td>
<td>same</td>
</tr>
<tr>
<td>z=100</td>
<td>same</td>
<td>same</td>
</tr>
<tr>
<td>while (p&gt;0)</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>z=z*x</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>p=p-1</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>output(z)</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
<tr>
<td>output(p)</td>
<td>([0,100]) &amp; ([0,100])</td>
<td>([0,100])</td>
</tr>
</tbody>
</table>
Iteration 3 is the same as iteration 2.

(2) It terminates on the program. However, it does not guarantee termination. For example, “while (...) x++”. The lattice has infinite height. No, it is not distributive.

Example:

```c
if (...) {
    a=[0,2];
    b=[0,2];
} else {
    a= [3,5];
    b=[3,5];
}
```

c=a-b;

For per-path analysis, the true branch yields c=[-2,2] and the false branch yields c=[-2,2], so c is in the range of [-2,2].

However, if we merge first, we get a=[0,5], b=[0,5], c=a-b=[-5,5].