CS510 Assignement #3

April 26, 2017

1 Proof

(20p) Prove the following sequents:

(a) $p \to q,\ r \to s \ \vdash\ p \vee r \to q \vee s$

(b) $(p \wedge q) \vee (p \wedge r) \ \vdash\ p \wedge (q \vee r)$

(c) $\vdash\ \neg p \to (p \to (p \to q))$

(d) $q \ \vdash\ (p \wedge q) \vee (\neg p \wedge q)$ using LEM

(e) $p \wedge q \ \vdash\ \neg(\neg p \vee \neg q)$

Answer:

(a)

1	$p \to q$	premise
2	$r \to s$	premise
3	$p \vee r$	assumption
4	p	assumption
5	q	$\to$ e 1,4
6	$q \vee s$	$\vee i_1$ 5
7	r	assumption
8	s	$\to$ e 2,7
9	$q \vee s$	$\vee i_2$ 8
10	$q \vee s$	$\vee e$ 3, 4-6, 7-9
11	$p \vee r \to q \vee s$	

(b)

1	$(p \wedge q) \vee (p \wedge r)$	premise
2	$p \wedge q$	assumption
3	p	$\wedge e_1$ 2
4	q	$\wedge e_2$ 2
5	$q \vee r$	$\vee i_1$ 4
6	$p \wedge (q \vee r)$	$\wedge i$ 3, 5
7	$p \wedge r$	assumption
8	p	$\wedge e_1$ 7
9	r	$\wedge e_2$ 7
10	$q \vee r$	$\vee i_1$ 9
11	$p \wedge (q \vee r)$	$\wedge i$ 8, 10
12	$p \wedge (q \vee r)$	$\vee e$ 1, 2-6, 7-11

(c)

1	$\neg p$	assumption
2	p	assumption
3	$\bot$	$\neg e$ 1,2
4	$p \to q$	$\bot e$ 3
5	$p \to (p \to q)$	$\to I$ 2,4
6	$\neg p \to (p \to (p \to q))$	$\to I$ 1,5

(d)

1	q	premise
2	$p \vee \neg p$	LEM
3	p	assumption
4	$p \wedge q$	$\wedge i$ 1,3
5	$(p \wedge q) \vee (\neg p \wedge q)$	$\vee i_1$ 4
6	$\neg p$	assumption
7	$\neg p \wedge q$	$\wedge i$ 1,6
8	$(p \wedge q) \vee (\neg p \wedge q)$	$\vee i_1$ 7
9	$(p \wedge q) \vee (\neg p \wedge q)$	$\vee e$ 2, 3-5, 6-8

(e)

1	$p \wedge q$	premise
2	p	$\wedge e_1$ 1
3	q	$\wedge e_2$ 1
4	$(\neg p \vee \neg q)$	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	$\neg(\neg p \vee \neg q)$	RAA 4,9

2 Validity

(15p) Determine the validity of the following semantic entailment by constructing the truth table.

$$\neg r \to (p \vee q),\ r \wedge \neg q \ \models\ r \to q$$

Answer:

r	p	q	$r \wedge \neg q$	$\neg r \to (p \vee q)$	$r \to q$
T	T	T	F	T	T
T	T	F	T	T	F
T	F	T	F	T	T
T	F	F	T	T	F
F	T	T	F	T	T
F	T	F	F	T	T
F	F	T	F	T	T
F	F	F	F	F	T

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

$$\vdash (s \to \neg f) \to ((\neg p \to s) \to (f \to p))$$

It equals to

$$\begin{aligned} & \neg(\neg s \vee \neg f) \vee (\neg(\neg\neg p \vee s) \vee (\neg f \vee p)) \\ = \ & (s \wedge f) \vee (\neg p \wedge \neg s) \vee \neg f \vee p \\ = \ & ((s \wedge f) \vee \neg p) \wedge ((s \wedge f) \vee \neg s) \vee \neg f \vee p \\ = \ & ((s \vee \neg p) \wedge (f \vee \neg p) \wedge (s \vee \neg s) \wedge (f \vee s)) \vee \neg f \vee p \\ = \ & ((s \vee \neg p \vee \neg f \vee p) \wedge (f \vee \neg p \vee \neg f \vee p) \wedge \\ & (s \vee \neg s \vee \neg f \vee p) \wedge (f \vee s \vee \neg f \vee p) \end{aligned}$$

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.

$$\neg((s \to \neg f) \to ((\neg p \to s) \to (f \to p)))$$

It equals to

$$\begin{array}{rl} & (s \to \neg f) \wedge \neg((\neg p \to s) \to (f \to p)) \\ = & \neg(s \wedge f) \wedge \neg(\neg(\neg p \wedge s) \to \neg(f \wedge \neg p)) \\ = & \neg(s \wedge f) \wedge \neg((\neg p \wedge s) \vee \neg(f \wedge \neg p)) \\ = & \neg(s \wedge f) \wedge \neg(\neg p \wedge s) \wedge (f \wedge \neg p)) \end{array}$$

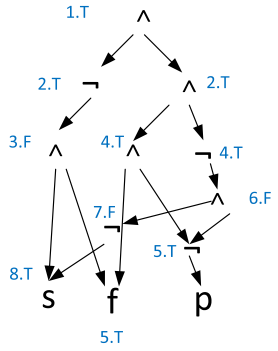

Figure 1:

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

4 SAT solving

(20p)

(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 text book. (10p)

(b) Devise a try-and-backtrack SAT solving algorithm that operates on formulas with $\neg$ and $\wedge$. 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)

(a)

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

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)

```
1 resolve (ast) {
2    linear_resolution();
3    if (contradition was found) return;
4    if (all nodes are consistently assigned) {
6       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  return
15}
```

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 procceed 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: $1 \wedge (2 \vee \neg 3) \wedge (\neg 4 \vee \neg 1) \wedge (\neg 1 \vee \neg 2 \vee 3 \vee 4) \wedge (\neg 2 \vee 4)$.

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.)

$$\omega_1 = x_1 \vee x_3 \vee x_4 \qquad \omega_3 = \neg x_1 \vee x_2 \vee x_3$$
$$\omega_2 = x_1 \vee \neg x_2 \vee x_3 \qquad \omega_4 = x_1 \vee \neg x_2 \vee x_4$$
$$\omega_5 = x_2 \vee x_4 \qquad \omega_6 = x_3 \vee x_4$$

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) $(P \vee \neg Q \vee \neg R) \wedge (Q \vee \neg P \vee R) \wedge (R \vee \neg Q)$

(b) $(P \vee Q \vee R) \wedge (\neg P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee R) \wedge (\neg Q \vee R) \wedge (Q \vee \neg R)$

(a)

Step 1: P=T, we have C1 (the first clause) = T
Step 2: P=T, Q=T, we have C1=T, C2=T
Step 3: P=T, Q=T, R=T, we have C1=C2=C3=T

(b)
Step 1: P=T, we have C1=T
Step 2: P=T, Q=F, we have C1=T, C2=T, C4=T
Step 3: P=T, Q=F, R=T, we have C1=T, C2=T, C3=T, C4=T, C5=F
roll back to 3 and reassign R
Step 4: P=T, Q=F, R=F, we have C1=T, C2=T, C3=F, C4=T, C5=T
roll back to step 2 and reassign Q
Step 5: P=T, Q=T, we have C1=T, C3=T, C5=T
Step 6: P=T, Q=T, R=F, we have C1=T, C2=T, C3=T, C4=F, C5=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 radom 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);
```

- Apply range analysis to the above program and compute the range of variables using the worklist algorithm. Please do not use the per-path analysis, but rather compute the aggregate results directly (10p).
- Does the worklist algorithm terminate on the program(3p)? Does it guarantee termination in general, why(3p)? Is range analysis distributive? If not, give an example (4p).

Answer:

(1)

Iteration one

trace	range
3. x=random(0.0, 1.0);	R_o[x@3]=[0,1]
4. y=random(-50,50);	R_o[y@3]=[-50,50]
5. if (y<0)	
6. p= -y;	R_o[y@6]=[-50,0), R_o[p@6]=(0,50]
7. else	
8. p=y;	R_o[y@8]=[0,50], R_o[p@8]=[0,50]
9. z=100;	R_o[y@9]=R^i[y@9]=R_o[y@6]$\wedge R_o$[y@8]=[-50,50], R_o[p@9]=R^i[p@9]=[0,50] R_o[z@9]=[100,100]
10. while (p>0) {	R^i[z@10]=R_o[z@10]=R_o[z@9]$\wedge R_o$[z@12]=[100,100], R^i[p@10]=[0,50], R_o[p^{true}@10]=(0,50]
11. z=z*x;	R^i[z@11]=[100,100], R_o[z@11]=[0,100], R^i[p@11]=R_o[p@11]=(0,50]
12. p=p-1;	R_o[p@12]=(-1,49]
13. }	
14. output(z);	R^i[z@14]=R_o[z@10]=[100,100]
15. output(p);	R^i[p@15]=R_o[p^{false}@10]=[0,0]

Iteration 2

trace	range
3. x=random(0.0, 1.0);	same as iteration 1
4. y=random(-50,50);	same
5. if (y<0)	
6. p= -y;	same
7. else	
8. p=y;	same
9. z=100;	same
10. while (p>0) {	R^i[z@10]=R_o[z@10]=[0,100],R^i[p@10]=(-1,50], R_o[p^{true}@10]=(0,50]
11. z=z*x;	R^i[z@11]=[0,100], R_o[z@11]=[0,100], R^i[p@11]=R_o[p@11]=(0,50]
12. p=p-1;	R_o[p@12]=(-1,49]
13. }	
14. output(z);	R^i[z@14]=R_o[z@10]=[0,100]
15. output(p);	R^i[p@15]=R_o[p^{false}@10]=(-1,0]

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 inifite height. No, it is not distributive.

Example:

```
 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 yeilds 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].