Homework 1
CS 474

Alexandra Levinshteyn

September 20th, 2024

Problem 1

We will prove that for every formula α,

$$len(\alpha) \leq 4 \cdot occ(\alpha) + 1,$$

that is, we are setting $m = 4$ and $n = 1$.

According to the inductive definition of a formula, a formula ϕ is either a proposition p_i, the result of $(\alpha \vee \beta)$, or the result of $(\neg\alpha)$, where α and β are formulas. So, we need to prove the above statement true for the base case formula, a proposition. We also need to prove that if we assume the statement is true for formulas α and β, then the statement is true for formulas $(\alpha \vee \beta)$ and $(\neg\alpha)$.

We proceed by structural induction.

First, we consider the base case of a proposition p_i. We have that

$$\begin{aligned} len(p_i) &= 1 \\ &\leq 1 = 4 \cdot 0 + 1 = 4 \cdot occ(p_i) + 1, \end{aligned}$$

so the statement is true for the base case.

Assume that the statement is true for formulas α and β, so that

$$len(\alpha) \leq 4 \cdot occ(\alpha) + 1$$

and

$$len(\beta) \leq 4 \cdot occ(\beta) + 1.$$

Consider the formula $\phi = (\alpha \vee \beta)$. We have that

$$\begin{aligned} len((\alpha \vee \beta)) &= len(\alpha) + len(\beta) + 3 \\ &\leq (4 \cdot occ(\alpha) + 1) + (4 \cdot occ(\beta) + 1) + 3 \\ &= 4 \cdot occ(\alpha) + 4 \cdot occ(\beta) + 5 \\ &\leq 4 \cdot occ(\alpha) + 4 \cdot occ(\beta) + 6 = 4 \cdot (occ(\alpha) + occ(\beta) + 1) + 1 = 4 \cdot occ((\alpha \vee \beta)) + 1, \end{aligned}$$

where the second line comes from the inductive hypothesis.

Consider the formula $\phi = (\neg\alpha)$. We have that

$$\begin{aligned} len((\neg\alpha)) &= len(\alpha) + 3 \\ &\leq (4 \cdot occ(\alpha) + 1) + 3 \\ &= 4 \cdot occ(\alpha) + 4 \\ &\leq 4 \cdot occ(\alpha) + 5 = 4 \cdot (occ(\alpha) + 1) + 1 = 4 \cdot occ((\neg\alpha)) + 1, \end{aligned}$$

where the second line comes from the inductive hypothesis.

Thus, assuming that the statement is true for formulas α and β, the statement is true for formulas $(\alpha \vee \beta)$ and $(\neg\alpha)$.

As the statement holds for the base case and holds for every possible step in the construction of a formula assuming it held for the previous steps, the statement holds for all formulas. Therefore, for every formula α,

$$len(\alpha) \leq 4 \cdot occ(\alpha) + 1,$$

meaning that the length of a formula is linear in the number of occurrences of Boolean operators.

Problem 2

Part (a)

For each person $p \in P$ and dog $d \in D$, take the proposition $q_{p,d}$ with the meaning that this proposition is true if and only if person p is assigned dog d in an adoption plan.

Define Γ to be the following set of formulas.

1. For each $p \in P$, letting $D_p = \{v_1, \dots, v_k\}$, the formula

$$\begin{aligned}&(q_{p,v_2} \vee q_{p,v_3} \vee \cdots \vee q_{p,v_k})\\ &\wedge (q_{p,v_1} \vee q_{p,v_3} \vee \cdots \vee q_{p,v_k})\\ &\wedge (q_{p,v_1} \vee q_{p,v_2} \vee q_{p,v_4} \cdots \vee q_{p,v_k})\\ &\wedge \dots\\ &\wedge \left(q_{p,v_1} \vee q_{p,v_2} \vee \cdots \vee q_{p,v_{k-1}}\right).\end{aligned}$$

   As D_p is finite for each $p \in P$, these formulas are finite. Each row here removes exactly one dog and checks if any of the others are assigned to p. If p only has one dog v_i assigned, the row without v_i would evaluate to false, making the whole formula false. If p has no dogs, every row would evaluate to false, making the whole formula false. As long as p has at least two dogs (in D_p) assigned, every row must evaluate to true (since in each row, at least one of the dogs will have their corresponding proposition appear), so the whole statement will be true. These formulas capture the constraint that each person is assigned at least two dogs from D_p (this is equivalent to constraint (1) given that constraint (3) is also true; the modification is needed so that the formulas are finite).

2. For each $d \in D$ and $p, r \in P$ such that $p \neq r$, the formula

$$\neg q_{p,d} \vee \neg q_{r,d}.$$

   These formulas capture the constraint that no dog is paired with multiple people. If there is a dog d assigned to two people p and r, the corresponding formula $\neg q_{p,d} \vee \neg q_{p,r}$ will be false. If each dog is assigned to one or no people, there is no way for any of these formulas to be false.

3. For each $d \in D$ and person $p \in P$ such that $d \notin D_p$, the formula

$$\neg q_{p,d}.$$

   These formulas capture the constraint that dogs are paired with only people who want them. If there is a dog d assigned to some $p \in P$ with $d \notin D_p$, the corresponding formula will evaluate to false. If there is no such dog, all of these formulas will evaluate to true.

4. For each $p \in P$, $d \in D$, and $e \in E_d$, the formula

$$\neg q_{p,d} \vee \neg q_{p,e}.$$

   These formulas capture the constraint that the dogs paired with a person must get along. If there are two dogs d and e such that $e \in E_d$ assigned to the same person p, the corresponding formula will evaluate to false. If there is no such pair of dogs, all of these formulas must be true.

Thus, Γ is an infinite set of propositional logic formulas, where each of these formulas is finite. Furthermore, Γ is satisfiable if and only if there is an adoption plan.

Part (b)

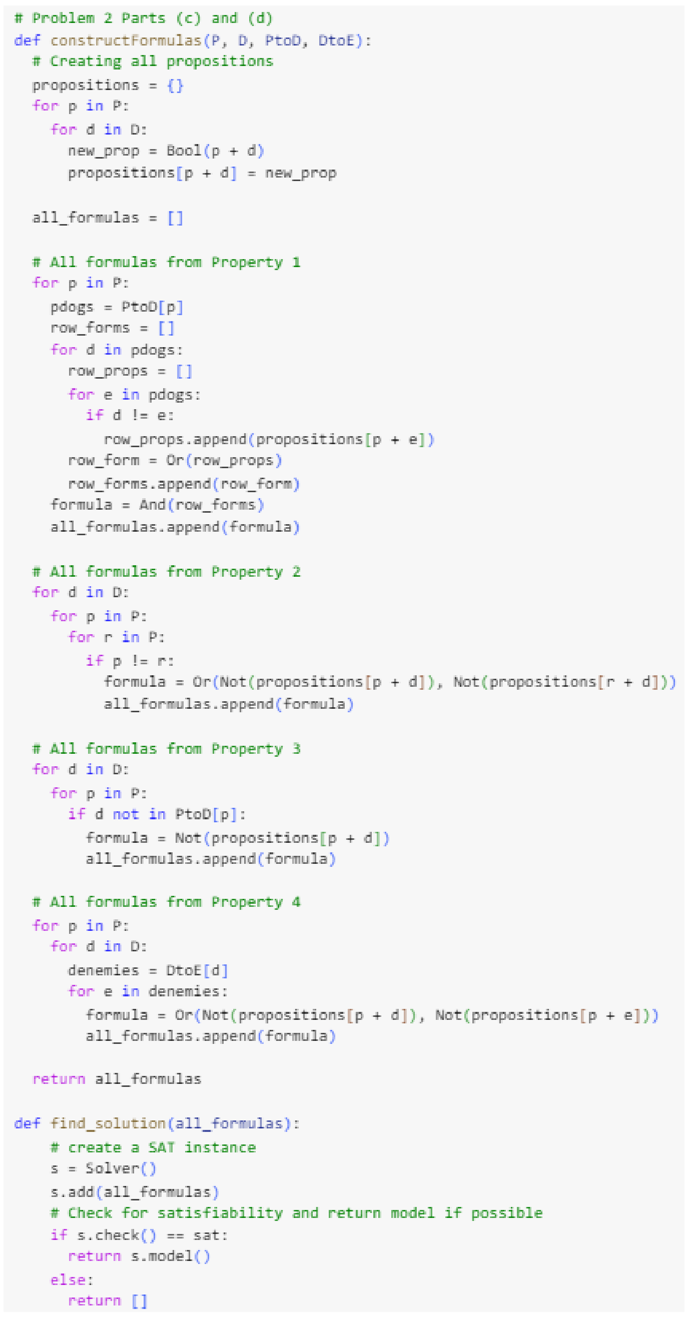
We are given that for any finite subset of people, there is an adoption plan for that set of people.

Observe that Γ is finitely satisfiable. Let Γ_0 be any finite subset of Γ. Let P_0 be the set of people p such that $q_{p,d}$ appears in Γ for some $d \in D$. Then, P_0 is a finite set of people, so we know that there is an adoption plan A for that set of people. Let Γ_1 be the subset of Γ with every single statement in Γ that contains any of the people in P_0. We are given that A will satisfy Γ_1. This adoption plan A will satisfy whichever finite set of formulas appear in Γ_0 since it will satisfy every formula in Γ_1 and $\Gamma_0 \subseteq \Gamma_1$.

Since Γ is finitely satisfiable, Γ is satisfiable. Therefore, there is an adoption plan for all people P.

Part (c)

The function code used for both Part (c) and Part (d) to encode the constraints as a set of formulas and attempt to find a solution to this set is shown below.

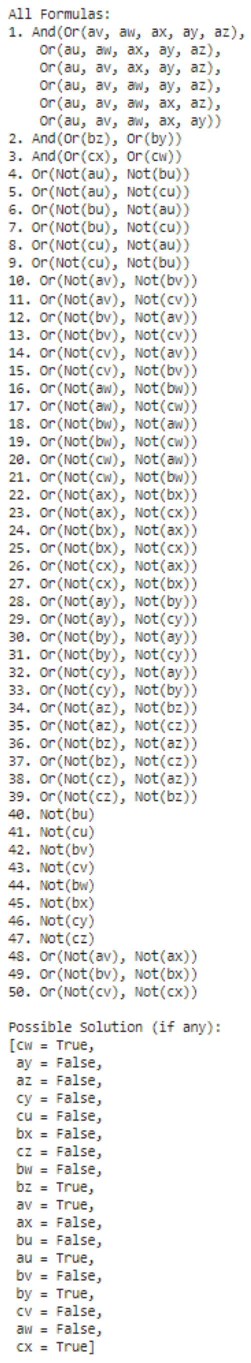


```
# Problem 2 Parts (c) and (d)
def constructFormulas(P, D, PtoD, DtoE):
  # Creating all propositions
  propositions = {}
  for p in P:
    for d in D:
      new_prop = Bool(p + d)
      propositions[p + d] = new_prop

  all_formulas = []

  # All formulas from Property 1
  for p in P:
    pdogs = PtoD[p]
    row_forms = []
    for d in pdogs:
      row_props = []
      for e in pdogs:
        if d != e:
          row_props.append(propositions[p + e])
      row_form = Or(row_props)
      row_forms.append(row_form)
    formula = And(row_forms)
    all_formulas.append(formula)

  # All formulas from Property 2
  for d in D:
    for p in P:
      for r in P:
        if p != r:
          formula = Or(Not(propositions[p + d]), Not(propositions[r + d]))
          all_formulas.append(formula)

  # All formulas from Property 3
  for d in D:
    for p in P:
      if d not in PtoD[p]:
        formula = Not(propositions[p + d])
        all_formulas.append(formula)

  # All formulas from Property 4
  for p in P:
    for d in D:
      denemies = DtoE[d]
      for e in denemies:
        formula = Or(Not(propositions[p + d]), Not(propositions[p + e]))
        all_formulas.append(formula)

  return all_formulas

def find_solution(all_formulas):
    # create a SAT instance
    s = Solver()
    s.add(all_formulas)
    # Check for satisfiability and return model if possible
    if s.check() == sat:
      return s.model()
    else:
      return []
```

The code specifically for Part (c) is shown below.

```python
# Problem 2 Part (c)
P = ['a', 'b', 'c']
D = ['u', 'v', 'w', 'x', 'y', 'z']
Da = ['u', 'v', 'w', 'x', 'y', 'z']
Db = ['y', 'z']
Dc = ['w', 'x']
PtoD = {'a': Da, 'b': Db, 'c': Dc}
Ev = ['x']
DtoE = {'u': [], 'v': Ev, 'w': [], 'x': [], 'y': [], 'z': []}

print("All Formulas:")
count = 1
all_formulas = constructFormulas(P, D, PtoD, DtoE)
for formula in all_formulas:
  print(str(count) + ". " + str(formula))
  count = count + 1

print()

solution = find_solution(all_formulas)
print("Possible Solution (if any):")
print(solution)
```

The code outputs the following:



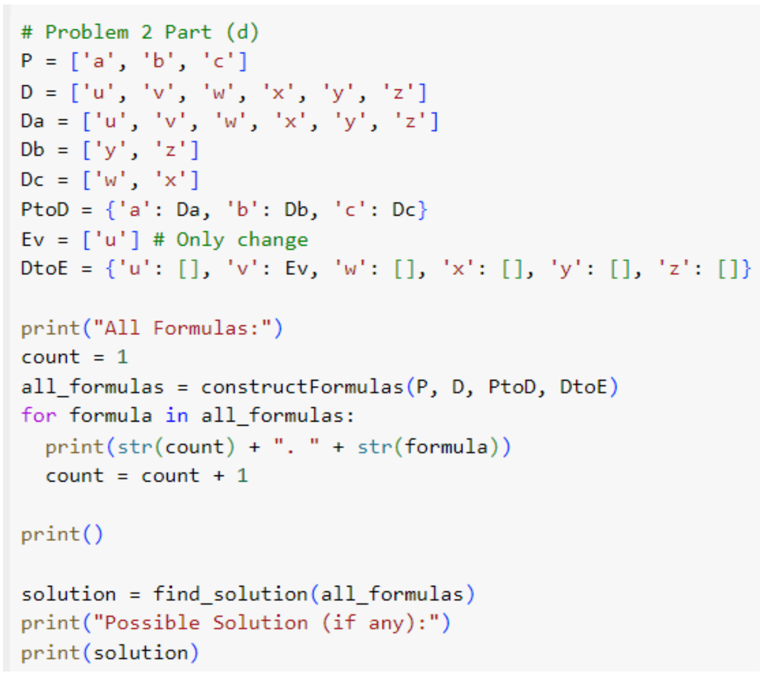
```
All Formulas:
1. And(Or(av, aw, ax, ay, az),
    Or(au, aw, ax, ay, az),
    Or(au, av, ax, ay, az),
    Or(au, av, aw, ay, az),
    Or(au, av, aw, ax, az),
    Or(au, av, aw, ax, ay))
2. And(Or(bz), Or(by))
3. And(Or(cx), Or(cw))
4. Or(Not(au), Not(bu))
5. Or(Not(au), Not(cu))
6. Or(Not(bu), Not(au))
7. Or(Not(bu), Not(cu))
8. Or(Not(cu), Not(au))
9. Or(Not(cu), Not(bu))
10. Or(Not(av), Not(bv))
11. Or(Not(av), Not(cv))
12. Or(Not(bv), Not(av))
13. Or(Not(bv), Not(cv))
14. Or(Not(cv), Not(av))
15. Or(Not(cv), Not(bv))
16. Or(Not(aw), Not(bw))
17. Or(Not(aw), Not(cw))
18. Or(Not(bw), Not(aw))
19. Or(Not(bw), Not(cw))
20. Or(Not(cw), Not(aw))
21. Or(Not(cw), Not(bw))
22. Or(Not(ax), Not(bx))
23. Or(Not(ax), Not(cx))
24. Or(Not(bx), Not(ax))
25. Or(Not(bx), Not(cx))
26. Or(Not(cx), Not(ax))
27. Or(Not(cx), Not(bx))
28. Or(Not(ay), Not(by))
29. Or(Not(ay), Not(cy))
30. Or(Not(by), Not(ay))
31. Or(Not(by), Not(cy))
32. Or(Not(cy), Not(ay))
33. Or(Not(cy), Not(by))
34. Or(Not(az), Not(bz))
35. Or(Not(az), Not(cz))
36. Or(Not(bz), Not(az))
37. Or(Not(bz), Not(cz))
38. Or(Not(cz), Not(az))
39. Or(Not(cz), Not(bz))
40. Not(bu)
41. Not(cu)
42. Not(bv)
43. Not(cv)
44. Not(bw)
45. Not(bx)
46. Not(cy)
47. Not(cz)
48. Or(Not(av), Not(ax))
49. Or(Not(bv), Not(bx))
50. Or(Not(cv), Not(cx))

Possible Solution (if any):
[cw = True,
 ay = False,
 az = False,
 cy = False,
 cu = False,
 bx = False,
 cz = False,
 bw = False,
 bz = True,
 av = True,
 ax = False,
 bu = False,
 au = True,
 bv = False,
 by = True,
 cv = False,
 aw = False,
 cx = True]
```

According to the SAT solver, the formula is satisfiable, so there is an adoption plan. Specifically, one potential adoption plan (which in this case happens to be the only one) is

1. Person a gets dogs u and v
2. Person b gets dogs y and z
3. Person c gets dogs w and x

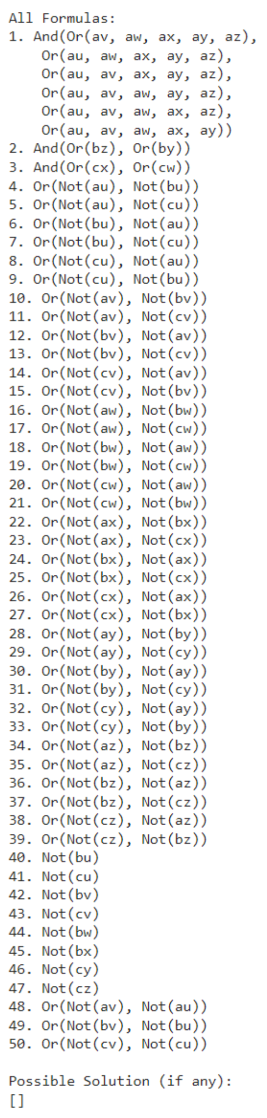
Part (d)

The code specifically for Part (d) is shown below.



```python
# Problem 2 Part (d)
P = ['a', 'b', 'c']
D = ['u', 'v', 'w', 'x', 'y', 'z']
Da = ['u', 'v', 'w', 'x', 'y', 'z']
Db = ['y', 'z']
Dc = ['w', 'x']
PtoD = {'a': Da, 'b': Db, 'c': Dc}
Ev = ['u'] # Only change
DtoE = {'u': [], 'v': Ev, 'w': [], 'x': [], 'y': [], 'z': []}

print("All Formulas:")
count = 1
all_formulas = constructFormulas(P, D, PtoD, DtoE)
for formula in all_formulas:
  print(str(count) + ". " + str(formula))
  count = count + 1

print()

solution = find_solution(all_formulas)
print("Possible Solution (if any):")
print(solution)
```

The code outputs the following:



```
All Formulas:
1. And(Or(av, aw, ax, ay, az),
    Or(au, aw, ax, ay, az),
    Or(au, av, ax, ay, az),
    Or(au, av, aw, ay, az),
    Or(au, av, aw, ax, az),
    Or(au, av, aw, ax, ay))
2. And(Or(bz), Or(by))
3. And(Or(cx), Or(cw))
4. Or(Not(au), Not(bu))
5. Or(Not(au), Not(cu))
6. Or(Not(bu), Not(au))
7. Or(Not(bu), Not(cu))
8. Or(Not(cu), Not(au))
9. Or(Not(cu), Not(bu))
10. Or(Not(av), Not(bv))
11. Or(Not(av), Not(cv))
12. Or(Not(bv), Not(av))
13. Or(Not(bv), Not(cv))
14. Or(Not(cv), Not(av))
15. Or(Not(cv), Not(bv))
16. Or(Not(aw), Not(bw))
17. Or(Not(aw), Not(cw))
18. Or(Not(bw), Not(aw))
19. Or(Not(bw), Not(cw))
20. Or(Not(cw), Not(aw))
21. Or(Not(cw), Not(bw))
22. Or(Not(ax), Not(bx))
23. Or(Not(ax), Not(cx))
24. Or(Not(bx), Not(ax))
25. Or(Not(bx), Not(cx))
26. Or(Not(cx), Not(ax))
27. Or(Not(cx), Not(bx))
28. Or(Not(ay), Not(by))
29. Or(Not(ay), Not(cy))
30. Or(Not(by), Not(ay))
31. Or(Not(by), Not(cy))
32. Or(Not(cy), Not(ay))
33. Or(Not(cy), Not(by))
34. Or(Not(az), Not(bz))
35. Or(Not(az), Not(cz))
36. Or(Not(bz), Not(az))
37. Or(Not(bz), Not(cz))
38. Or(Not(cz), Not(az))
39. Or(Not(cz), Not(bz))
40. Not(bu)
41. Not(cu)
42. Not(bv)
43. Not(cv)
44. Not(bw)
45. Not(bx)
46. Not(cy)
47. Not(cz)
48. Or(Not(av), Not(au))
49. Or(Not(bv), Not(bu))
50. Or(Not(cv), Not(cu))

Possible Solution (if any):
[]
```

According to the SAT solver, the formula is not satisfiable, so there is no possible adoption plan.