CSCI 377 Midterm 1

Monday, Sept. 24, 2018

1 True/False

(a) True or **False**? $B \wedge P \models P$ is in the language $\mathcal{L}_{PL}(\Sigma)$, where $\Sigma = \{B, P, F\}$.

(b) **True** or False? Let $RC(S)$ be the resolution closure of a set S of clauses. Then $RC(RC(S)) = RC(S)$.

(c) True or **False**? $\forall x \exists y (x \Rightarrow y)$ is a sentence in first order logic.

(d) **True** or False? $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\gamma \vee \alpha)$ for any sentences $\alpha, \beta, \gamma \in \mathcal{L}_{PL}(\Sigma)$.

(e) **True** or False? If I generate a random 3-CNF sentence with 100 clauses over 100 variables, the probability is greater than 50% that it will be satisfiable.

(f) **True** or False? For sentences $\alpha, \beta \in \mathcal{L}_{PL}(\Sigma)$, $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is satisfiable.

(g) **True** or False? Suppose you have a conjunction of 5 CNF clauses, each of which mentions exactly three different variables (either positive or negative literals). This conjunction MUST be satisfiable.

(h) True or **False**? There are exactly 2^n unique truth tables over n Boolean variables.

(i) **True** or False? $(A \Leftrightarrow (B \wedge \neg C)) \equiv (\neg A \vee B) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee C \vee A)$

(j) **True** or False? If a first-order logical sentence α does not contain the constant (i.e. zero-argument function) A, then $\exists x(Bird(x) \wedge Child(x, Bob)) \models \alpha$ if and only if $(Bird(A) \wedge Child(A, Bob)) \models \alpha$.

2 Resolution Closure

What is the resolution closure of $\{A \vee B \vee \neg C, A \vee \neg B \vee C, \neg A \vee D\}$? Do not include clauses that are logically equivalent to True.

$$\mathbf{A} \lor \mathbf{B} \lor \neg\mathbf{C}$$
$$\mathbf{A} \lor \neg\mathbf{B} \lor \mathbf{C}$$
$$\neg\mathbf{A} \lor \mathbf{D}$$
$$\mathbf{B} \lor \neg\mathbf{C} \lor \mathbf{D}$$
$$\neg\mathbf{B} \lor \mathbf{C} \lor \mathbf{D}$$

3 The Social Club

There's a social club that you want to join. As a condition of entry, they offer you two drinks, one of which is poisoned and one of which is not. You are told you must choose one to drink. For some reason, you still really want to join the club.

Conveniently, each member of the club is either a truthteller (truthtellers always tell the truth) or a liar (liars always lie). To help yourself make the decision, you need to create a first-order logic formulation of the situation.

Use the following signature:

- Drink1 is a zero-argument function that represents the first drink
- Drink2 is a zero-argument function that represents the second drink
- Poisoned(d) is a one-argument predicate that indicates whether d is the poisoned drink
- Member(m) is a one-argument predicate that indicates whether m is a club member
- Liar(m) is a one-argument predicate that indicates whether m is a liar
- SayYes(m, d) is a two-argument predicate that indicates whether m would say "yes, drink d is poisoned" if asked

Express the following in first order logic:

(a) "Exactly one drink is poisoned."

$$\mathsf{Poisoned}(\mathsf{Drink1}) \lor \mathsf{Poisoned}(\mathsf{Drink2})$$
$$\neg\mathsf{Poisoned}(\mathsf{Drink1}) \lor \neg\mathsf{Poisoned}(\mathsf{Drink2})$$

(b) "Exactly one club member is a liar."

$$\exists x(\mathsf{Member}(x) \wedge \mathsf{Liar}(x))$$
$$\forall x \forall y((\mathsf{Member}(x) \wedge \mathsf{Member}(y) \wedge \mathsf{Liar}(x) \wedge \neg(x = y)) \Rightarrow \neg\mathsf{Liar}(y))$$

(c) "When asked about Drink1, a liar always lies about whether Drink1 is poisoned and a truthteller always tells the truth about whether Drink1 is poisoned."

$$\forall m(\mathsf{SayYes}(m, \mathsf{Drink1}) \Leftrightarrow ((\mathsf{Liar}(m) \wedge \neg\mathsf{Poisoned}(\mathsf{Drink1})) \vee (\neg\mathsf{Liar}(m) \wedge \mathsf{Poisoned}(\mathsf{Drink1}))$$

4 Resolution for 2SAT and 3SAT

In class, we used code for a resolution solver that looked something like this:

```
def resolution_solver(C):
    # C is a list of CNF clauses

    class LocalState:
        processed = set()
        unprocessed = C

    def process_next_clause():
        next_to_process = local.unprocessed[0]
        local.unprocessed = local.unprocessed[1:]
        for clause in local.processed:
            for resolvent in resolve(next_to_process, clause):
                if resolvent not in local.processed | set(local.unprocessed):
                    local.unprocessed.append(resolvent)
                if resolvent.is_false():
                    return False
        local.processed.add(next_to_process)
        return True

    local = LocalState()
    while len(local.unprocessed) > 0:
        if not process_next_clause():
            return False
    return True
```

Suppose that the signature contains n symbols, and that a call to $\mathsf{resolve}(\mathsf{c1}, \mathsf{c2})$ returns the list of clauses that can be obtained by resolving clause c1 and clause c2. As usual, assume that a clause cannot contain both the positive and negative literal for the same symbol (e.g. $C \vee \neg C$ is not a clause). Also assume that two clauses that contain the exact same literals are the same clause (e.g. $A \vee \neg B$ should be considered to be the same clause as $\neg B \vee A$).

(a) What is the worst-case number of calls to process_next_clause if every clause in C contains at most 2 literals? Justify your answer.

(b) What is the worst-case number of calls to process_next_clause if every clause in C contains at most 3 literals? Justify your answer.

(a) $4\binom{n}{2} + 2n + 1 = O(n^2)$. **Resolving a 2-literal clause with another 2-literal clause can only generate a clause with at most 2 literals, thus we can only ever generate clauses with at most 2 literals. There are** $4\binom{n}{2}$ **clauses with exactly two literals,** $2n$ **clauses with exactly one literal, and** 1 **clause with no literals (i.e.** $False$**).**

(b) 3^n. **You can obtain a clause with 4 literals by resolving two 3-literal clauses. Once 4-literal clauses are generated, then you can resolve these with 3-literal clauses to obtain 5-literal clauses, etc., until you have potentially generated any clause. There are** 3^n **possible clauses over** n **symbols.**