Assessment 1

Logic, Regular Languages, and Finite Automata

Norbert Logiewa
nl253

November 2017

Propositional Logic

1. Model as a propositional formal the poroperty that the inital state cannot be reached from c or b

Answer:
$model \rightarrow \neg((c \rightarrow a') \vee (b \rightarrow a'))$

2. Prove the following sequent:

$$a \rightarrow b'.b \rightarrow c'.c \rightarrow b' \vdash (a \vee b) \rightarrow (b' \vee c')$$

1. $a \rightarrow b'.b \rightarrow c'.c \rightarrow b'$ [premise]

---

2. $(a \rightarrow b') \wedge (b \rightarrow c') \wedge (c \rightarrow b')$ $[\wedge_i]$ 1
3. $(a \rightarrow b') \wedge (c \rightarrow b') \wedge (b \rightarrow c')$ [commutativity of $\wedge$] 2
4. $(a \wedge c \rightarrow b') \wedge (b \rightarrow c')$ [distributivity of $\rightarrow$] 3
5. $a \wedge c \rightarrow b'$ $[\wedge_{e2}]$ 4
6. $a \rightarrow b'$ $[\wedge_{e2}]$ 5
7. $a \vee b \rightarrow b'$ $[\vee_{i2}]$ 6

---

8. $(a \vee b) \rightarrow (b' \vee c')$ $[\vee_i]$ 7 [conclusion]

3. Model the property that the automate can only be in one stat at a time via a defintion of a propositional function

Answer:
$A \oplus B \equiv (A \vee B) \wedge (\neg A \vee \neg B)$ [definiton of the XOR operator]
$goodState(a, b, c) \equiv (a \rightarrow b') \oplus (b \rightarrow c') \oplus (c \rightarrow b')$

4. Negate the formula and convert it to CNF

$$((a \rightarrow b') \wedge (a \wedge \neg b) \wedge (b' \wedge \neg a')) \rightarrow (b \rightarrow \neg a')$$

Answer:

1. Negate the formula

$$\neg(((a \rightarrow b') \wedge (a \wedge \neg b) \wedge (b' \wedge \neg a')) \rightarrow (b \rightarrow \neg a'))$$

2. Remove $\rightarrow$ using $A \rightarrow B \equiv \neg A \vee B$ property

$$\neg(((\neg a \vee b') \wedge (\neg a \vee \neg b) \wedge (b' \wedge \neg a')) \rightarrow (\neg b \vee \neg a'))$$

$$\equiv$$

$$\neg(\neg((\neg a \vee b') \wedge (\neg a \vee \neg b) \wedge (b' \wedge \neg a')) \vee (\neg b \vee \neg a'))$$

3. Push $\neg$ inwards using De Morgan's Laws

$$((\neg a \vee b') \wedge (\neg a \vee \neg b) \wedge (b' \wedge \neg a')) \wedge \neg(\neg b \vee \neg a')$$

$$\equiv$$

$$((\neg a \vee b') \wedge (\neg a \vee \neg b) \wedge (b' \wedge \neg a')) \wedge (b \wedge a')$$

4. Distribute

$$b \wedge ((\neg a \vee b') \wedge (\neg a \vee \neg b) \wedge (b' \wedge \neg a')) \wedge a' \wedge ((\neg a \vee b') \wedge (\neg a \vee \neg b) \wedge (b' \wedge \neg a'))$$

$$\equiv$$

$$b \wedge (\neg a \vee b') \wedge b \wedge (\neg a \vee \neg b) \wedge b \wedge (b' \wedge \neg a')$$
$$\wedge\ a' \wedge (\neg a \vee b') \wedge a' \wedge (\neg a \vee \neg b) \wedge a' \wedge (b' \wedge \neg a')$$

5. Simplify (remove duplicate elements because $A \wedge A \equiv A$)

$$(\neg a \vee b') \wedge b \wedge (b' \wedge \neg a') \wedge (\neg a \vee \neg b) \wedge a'$$

---

5. Explain the principle of unit propagation from DPLL, and use it to show that CNF formula of the previous question is unsatisfiable.

Answer:
a) Unit propagation refers to a technique that is used to simplify logical formulas. When we have a clause that *only* contains a single literal such as x, in all other clauses that contain x, we can replace it with $True$.

Answer:
b) We can use it to show that the previous formula is unsatisfiable by demonstrating that it is a contradiction. We simplify it and continue to substitute $True$ for single literal clauses. When we rearrange the formula, it becomes clear that we have $b \wedge \neg b$, this is a contradiction.

$$(True \vee b') \wedge True \wedge \neg b \wedge b' \wedge \neg a' \wedge b$$
$$(True \vee True) \wedge True \wedge \neg b \wedge True \wedge \neg a' \wedge b$$
$$\neg b \wedge True \wedge \neg a' \wedge b$$
$$\neg b \wedge \neg a' \wedge b$$
$$\neg b \wedge b \wedge \neg a'$$

6. Comment on the validity of the original formula.

Answer:
The formula is not satisfiable and therefore not valid as satisfiability is a prerequisite for validity. In other words, because that logical statement was a contradiction (*assumed something can be False and True at the same time*), there was no way in which it could have been true and because of that it is not valid.

First Order Logic

1. Write a formula in first-order logic, using the parent relation, that states that two entities x and y are siblings if they share a parent

Answer:
$\forall i.\forall j.siblings(i, j) \equiv (parent(x, i) = parent(x, j))$

2. Assuming there is an equality relation $\equiv$ on P such that $x \equiv y$ means x and y are the same entity, write a formula in first order logic stating that every entity has two distinct parents.

Answer:
$\exists!x.P(x) \equiv \exists x(P(x) \wedge \forall y(P(y) \rightarrow y = x))$ [1]
$\forall i.\exists!y.\exists!x(parent(x, i) \wedge parent(y, i) \wedge \neg(x \equiv y))$

3. Prove that the following sequent is valid:

$$parent(p, q).\forall x.\forall y(parent(x, y) \rightarrow \exists z.parent(z, x)) \vdash \exists.parent(z, p)$$

---

1. $parent(p, q).\forall x.\forall y(parent(x, y) \rightarrow \exists z.parent(z, x))$ [premise]
2. —
3. $\exists.parent(z, p)$ [conclusion] 1 – ?

---

4. Interpret the following formula into simple English statement

$\forall x.\exists y.parent(y, x) \rightarrow \forall a.\exists b.parent(a, b)$

Answer:
If all children have a parent, then all parents have a child.

5. The sequent below is not valid. Show this by proving a counter-example: a concrete definition of the universe P (a set) and the relation parent. Explain why this is a counterexample.

$$\vdash \forall x.\exists y.parent(y, x) \rightarrow \forall a.\exists b.parent(a, b)$$

In English
if all children have a parent then all parents have a child.

1. $\forall x.\exists y.parent(y, x) \rightarrow \forall a.\exists b.parent(a, b)$ [premise]

---

2. $P = \{ \ x \mid x \ \}$

---

3. $False$ [conclusion]

[1] https://math.stackexchange.com/questions/228285/how-can-i-get-the-negation-of-exists-unique-existential-quantification

6. What is the smallest possible counterexample universe P and relation parent. Explain your reasoning.

7. Prove that the following sequent is valid.

$$\forall x.\forall y.parent(x,y) \rightarrow \neg(x \equiv y) \vdash \neg\exists x.parent(x,x)$$

ie. if someone is a parent and has a child, then that child cannot be it's parent.

1. $\forall x.\forall y.parent(x,y) \rightarrow \neg(x \equiv y)$ [premise]

---

2. $parent(x_i, y_j) \rightarrow \neg(x_i \equiv y_j)$ $\forall_e$ 1
3. $\neg parent(x_i, y_j) \vee \neg(x_i \equiv y_j)$ because $P \rightarrow Q \equiv \neg P \vee Q$
4. $\neg(parent(x_i, y_j) \wedge (x_i \equiv y_j))$ De Morgan's Law

---

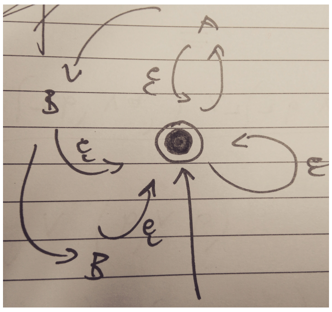
5. $\neg\exists x.parent(x,x)[conclusion]$

Regular Lanugages and Finite Automata

1. For the reular expression $(A|AB|ABB)^*$ do the following steps (and justify them):

a) translate it into an NFA

Answer:

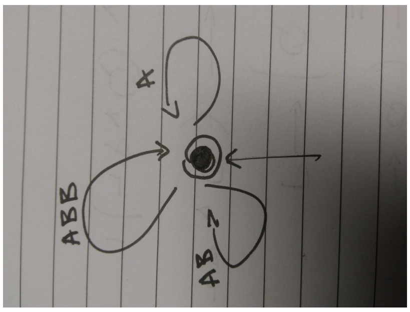


b) remove (semantic preserving) the ϵ transitions

Answer:

c) create the corresponding DFA

Answer:



2. Consider the following language:
A word is in the language if and only if it contains an even number of As an odd number of Bs and precisely one C.

a) Design a DFA for this language

   Answer:
   Not possible.

b) Explain your design (what information do you "store " in your states?)

Additional Sources

- https://en.wikipedia.org/wiki/Negation
- https://en.wikipedia.org/wiki/Distributive_property
- https://en.wikipedia.org/wiki/Conjunctive_normal_form
- https://en.wikipedia.org/wiki/De_Morgan%27s_laws
- https://en.wikipedia.org/wiki/Modus_ponens
- https://en.wikipedia.org/wiki/Exclusive_or
- https://en.wikipedia.org/wiki/Finite-state_machine
- https://en.wikipedia.org/wiki/Uniqueness_quantification