Program Verification, Review of Propositional Logic

CS 536: Science of Programming, Fall 2021

1. What is program verification? How is it done?
2. How is type-checking a form of program verification?
3. What is propositional logic? What are propositional connectives?
4. Given values for p and q, what are the values of $p \wedge q$, $p \vee q$, etc?
5. Translate each of the following to either $p \rightarrow q$ or $q \rightarrow p$:
   a. if p then q
   b. p is sufficient for q
   c. p only if q
   d. p is necessary for q
   e. p if q
   f. if q then p
6. What are the converse, contrapositive, and inverse of $p \rightarrow q$? How are they related?
7. What is syntactic equality and how is it denoted?
8. What is semantic equality? How do we indicate semantic equality of two arithmetic expressions? Two propositions?
9. How are syntactic and semantic equality related?
10. How do parentheses affect syntactic equality for us?
13. What are the precedence and associativity rules we are using for +, -, *, /, %, ≤, =, (etc.), ∧, ∨, →, ↔, and ¬?
14. For propositions, what are the definitions of tautology, contradiction, and contingency?
15. Consider the six statements "*p* is a *X*" and "¬*p* is a *X*" where *X* ranges over "tautology", "contradiction", and "contingency". How are these statements related?
16. Repeat the previous problem on the six statements "*p* is not a *X*" and "*p* is a *X*".
17. Which of $p \rightarrow q \rightarrow r$, $p \rightarrow (q \rightarrow r)$, and $(p \rightarrow q) \rightarrow r$ are ≡ ?
18. Write out truth tables for the following. Are any of these semantically equivalent? (I.e., do they have the same truth table rows?)
    a. $p \leftrightarrow (q \leftrightarrow r)$
    b. $(p \leftrightarrow q) \wedge (q \leftrightarrow r)$
    c. $(p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$
19. a. Which of $\{p = T, q = T\}$, $\{p \wedge q = T\}$, $\{q = F\}$, $\{ \}$, and $\{r = T, r = F\}$ are well-formed?

b. Of the ones that are well-formed, which ones are proper for $(T \rightarrow F)$, $(p \wedge \neg\neg p)$, and $((p \vee q \vee \neg p) \leftrightarrow q)$?

20. List all the states σ such that $\sigma \vDash p \leftrightarrow (q \leftrightarrow r)$.

21. Are there any σ such that $\sigma \vDash T \rightarrow F$? Does this tell us that $\emptyset \vDash T \rightarrow F$?

22. *Notation*: To write ⇔ in ASCII, let's use <==>. That way, the negation can be written <=/=>, which is a lot easier to read than ⇎. (To me, at least. Or we could use 18 pt and write **⇎**.)

a. Define <=/=> using ⊨. (Hint: Go back to the definition of ⇔ using ⊨.)

b. Give a small example that proves that <=/=> is not transitive. (I.e., fill in the blanks in $\_\_\__1$ <=/=> $\_\_\__2$ and $\_\_\__2$ <=/=> $\_\_\__3$, but $\_\_\__1$ <==> $\_\_\__3$.)

CS 536: Solution to Practice 1 (Program Verification, Review of Propositional Logic)

1. Program verification aims to get reliable programs by mechanically reasoning about them. We name sets of values or states using predicates and simulate program execution on them, using rules of logic plus rules describing program execution.
2. Type-checking uses types to denote sets of values (e.g., values of type int); it uses mechanical type-checking rules to show how values manipulated (e.g., plus takes two ints and yields an int).
3. Propositional logic is logic over proposition variables (i.e., Boolean variables). The connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ are standard functions on boolean values.
4. See the table in the lecture notes. Be especially knowledgeable about $p \rightarrow q$.
5. (a), (b), and (c) mean $p \rightarrow q$; (d), (e), and (f) mean $q \rightarrow p$.
6. $p \rightarrow q$ is equivalent to its contrapositive $\neg q \rightarrow \neg p$. The converse $q \rightarrow p$ and inverse $\neg p \rightarrow \neg q$ are equivalent to each other. In general, an implication is not equivalent to its converse.
7. Syntactic equality is equality as structured text. It is denoted by $\equiv$
8. Semantic equality is equality of meaning. To denote it, we write $p \Leftrightarrow q$ for logical propositions and $e_1 = e_2$ for expressions with non-logical values.
9. Syntactic equality implies semantic equality (but not vice versa).
10. Redundant parentheses are ignored. E.g., $1 + 2 * 3 \equiv (1 + (2 * 3))$.
13. From low to high: $\leftrightarrow$, $\rightarrow$, $\vee$, $\wedge$, comparison operators ($\leq$, $\geq$, etc), (+ and –), (* and / and %), (unary – and $\neg$). [Operators written in parentheses have equal precedence.]
14. A tautology is true for all possible combinations of truth values for its variables. A contradiction is false for all possible combinations. A contingency is true for at least one combination and is false for at least one combination.
15. (p is a tautology) and ($\neg p$ is a contradiction) are equivalent; so are (p is a contingency) and ($\neg p$ is a contingency).
16. Every proposition falls into exactly one of the three categories, so if it's not some particular one of the three, it must be one of the other two.
    (p is not a tautology) is equivalent to (p is a contradiction or p is a contingency).
    (p is not a contradiction(is equivalent to (p is a tautology or p is a contingency).
    (p is not a contingency) is equivalent to (p is a tautology or p is a contradiction).
17. $p \rightarrow q \rightarrow r \equiv p \rightarrow (q \rightarrow r)$.
18. The table is below. The only pair of semantically equivalent propositions are $(p \leftrightarrow q) \wedge (q \leftrightarrow r)$ and $(p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$. Note $((p \leftrightarrow q) \wedge (q \leftrightarrow r)) \leftrightarrow ((p \wedge q \wedge r) \vee (\neg p \wedge$

¬ q ∧ ¬ r)) is a tautology and ¬((p ↔ q) ∧ (q ↔ r)) ↔ ((p ∧ q ∧ r) ∨ (¬p ∧ ¬q ∧ ¬r)) is a contradiction.

p	*q*	*r*	*p ↔ (q ↔ r)*	*(p ↔ q) ∧ (q ↔ r)*	*p ∧ q ∧ r ∨ ¬p ∧ ¬q ∧ ¬r*
F	F	F	F	T	T
F	F	T	T	F	F
F	T	F	T	F	F
F	T	T	F	F	F
T	F	F	T	F	F
T	F	T	F	F	F
T	T	F	F	F	F
T	T	T	T	T	T

19. a. All except {p ∧ q = T} and {r = T, r = F} are well-formed.
    b. All three of {p = T, q = T}, {q = F}, and { } are proper for (T → F). Only {p = T, q = T} is proper for (p ∧ ¬¬p) because {q = F} and { } don't have a value for p. Only {p = T, q = T} is proper for ((p ∨ q ∨ ¬p) ↔ q); {q = F} because { } don't have values for p and q.
20. The σ such that σ ⊨ p ↔ (q ↔ r) are {p = T, q = T, r = T}, {p = T, q = F, r = F}, {p = F, q = T, r = F}, and {p = F, q = F, r = T}.
21. There are no σ such that σ ⊨ T → F. In this context, ∅ is also a state (the empty one), so ∅ ⊨ T → F does not hold either. (We can also write σ ⊭ T → F for all σ, including ∅.)
22. a. Since p <==> q means ⊨ p ↔ q, p <=/=> q means ⊭ p ↔ q. (I.e., for some σ, we have σ ⊭ p ↔ q. One way to expand out this last statement is that σ ⊨ (p ∧ ¬q) ∨ (¬p ∧ q).
    b. One example that proves <=/=> is not transitive is (T <=/=> F and F <=/=> T but T <==> T).