Lógica

Profesor. Carlos Barrón Romero

Guía para el segundo examen.

Instrucciones. Esta guía tiene dos partes, ejercicios y repaso de teoría necesaria para entender, relacionar y aplicar la Teoría de la Inferencia, Teoría de la Demostración o Teoría de la Deducción Lógica a la programación.

El marco de sus respuestas y comprensión de los temas de Lógica son los objetivos de la UEA de Lógica (clave 111222) que transcribo a continuación:

1. Comprender los principios básicos de la lógica matemática.
2. Demostrar la validez de argumentos mediante reglas formales.
3. Aplicar principios de lógica matemática en la elaboración de programas de cómputo.

Contenido

El esquema o diagrama consta de dos partes, separadas por una raya $\dfrac{\text{Suposiciones: proposición dada}}{\text{Dedución: proposición resultante}}$

Los esquemas clásicos de inferencia.

Modus Ponendo Ponens $\dfrac{\begin{array}{l}p \Rightarrow q \\ p\end{array}}{q}$. Doble negación $\dfrac{p}{\neg\neg p}$.

Modus Tollendo Tollens $\dfrac{\begin{array}{l}p \Rightarrow q \\ \neg q\end{array}}{\neg p}$. Regla de Adjunción $\dfrac{\begin{array}{l}p \\ q\end{array}}{p \wedge q}$.

Modus Tollendo Ponens $\dfrac{\begin{array}{l}p \vee q \\ \neg p\end{array}}{q}$ $\dfrac{\begin{array}{l}p \vee q \\ \neg q\end{array}}{p}$.

Ejemplos de inferencia:

Dado: $p \Rightarrow q \vee r$, $r \Rightarrow t$, $\neg q$

$\dfrac{\begin{array}{l}p \Rightarrow q \vee r \\ p\end{array}}{t}$. Se tiene ya que $\dfrac{\begin{array}{l}p \Rightarrow q \vee r \\ p\end{array}}{q \vee r}$, $\dfrac{\begin{array}{ll}q \vee r & \\ \neg q & \end{array}}{r}$, $\dfrac{\begin{array}{ll}r \Rightarrow t & \\ r & \end{array}}{t}$.

Dado: $p \Rightarrow \neg q$, $\neg q$

$\dfrac{\begin{array}{l}p \Rightarrow \neg q \\ \neg q\end{array}}{\neg p}$. Equivocado. Lo correcto es $\dfrac{\begin{array}{l}p \Rightarrow \neg q \\ q\end{array}}{\neg p}$ en efecto $p \Rightarrow \neg q \equiv \neg\neg q \Rightarrow \neg p \equiv q \Rightarrow \neg p$

finalmente $\dfrac{\begin{array}{l}q \Rightarrow \neg p \\ q\end{array}}{\neg p}$. Note que $\neg\neg q \equiv q$. Y que el símbolo $\equiv$ significa equivalente y se refiere a la igualdad de las columnas de una tabla de verdad, por ejemplo: $\neg\neg q \equiv q$ ya que las columnas de q y de $\neg\neg q$ en

q	$\neg q$	$\neg\neg q$
0	1	0
1	0	1

son iguales.

Se puede usar el álgebra de proposiciones.

Demostrar (es decir dar los esquemas o las equivalencias para obtener la proposición que se pide o explicar que no se puede).

1) Demostrar p, dado $p \vee q, \neg t, q \Rightarrow t$.

2) Demostrar $a \wedge b$, dado $b, b \Rightarrow \neg d, a \vee d$.

3) Demostrar p, dado $t \Rightarrow p \vee q, \neg\neg t, \neg q$.

3) Demostrar q, dado $t \Rightarrow p \vee q, \neg\neg t, \neg q., s, \neg p$.

4) Explique si son equivalentes $t \Rightarrow (p \vee q)$ y $(t \Rightarrow p) \vee q$, por tanto los paréntesis son importantes y cual debe ser la forma de interpretar $t \Rightarrow p \vee q$, es decir la prioridad de $\Rightarrow$ y $\vee$.

5) Simplifique: $\neg (q \vee r) \wedge (q \vee 0) \equiv$

6) Simplifique: $\neg (q \vee (r \wedge \neg q)) \wedge (q \vee 1) \equiv$

Demostración o verificación de programas.

Se le da una explicación de que es la demostración de correctez de programas con un esquema similar al de inferencia ({} algo {}) donde la segunda {} se llena con lo que "algo" significa o realiza o define.

En este caso el esquema es

{premisas}

Instrucción o declaración

{consecuentes}

que se interpreta con la inferencia y el significado de la instrucción para construir el consecuente. Ejemplos:

{instrucciones y declaraciones nativas de C}

#include <iostream>

{funciones de biblioteca, en particular printf(), instrucciones y declaraciones nativas de C}

int main() {

{funciones de biblioteca, en particular printf(), instrucciones y declaraciones nativas de C, modulo main sin parámetros que debe regresar un valor entero}

float epsi = 1.0;

{funciones de biblioteca, en particular printf(), instrucciones y declaraciones nativas de C, modulo main sin parámetros que debe regresar un valor entero, variable epsi tipo número real con valor 1.0 representada internamente como "seven digits of precision and a range of about 1.E-36 to 1.E+36. A float takes four bytes to store" según el help de Dev-Cpp.

}

Nota de la notación de punto flotante y aritmética: significa que solo se pueden representar valores entre $[-1.\text{E}{+}36, 1.\text{E}{+}36]$ y el numero más pequeño es $1.\text{E}{-}36$ y la mantisa en esta notación de punto flotante es de siete dígitos.

Además si float $x = 0.123456789$; la asignación significa que el valor almacenado en x de acuerdo a lo anterior es 0.1234567

En este esquema

{etc. }

float $x = 0.123456789$;

{etc. variable real x de tipo número real con valor 0.1234567}

El resto de las operaciones aritméticas se realizan como se definen o sea como usted las opera, pero es en la asignación o sea al momento de guardar los valores que estos se cortan a la notación de punto flotante de siete dígitos con exponente de -36 a + 36.

Y de esta forma se sigue hasta terminar de analizar el programa, si tiene ciclos se repite el proceso hasta que se demuestre que se cumple alguna condición de terminación. Si tiene if se analizan las dos ramas o consecuencias que tenga por separado siguiendo las ruta o los caminos que genera el if. El programa será correcto si se logra determinar que termina o se detiene (halt) después de analizar todos los caminos y ciclos del algoritmo de programa.

Ahora debe aplicar lo anterior, dado el siguiente programa C.

```
#include <iostream>
int main() {
float epsi = 1.0;
float uno = 1.0;
float suma =.0;
int cont = 0;
while (1){
cont = cont + 1;
epsi = epsi / 2.0;
suma = 1.0 + epsi;
if (suma == uno)
break;
}
printf("\nIteraciones => %d Epsilon => %16.12e\n",cont,epsi);
system("PAUSE");
return EXIT_SUCCESS;
}
```

1) para que lógicamente se demuestre que es correcto.

2) Se compilo con Dev-Cpp y se ejecuto en una laptop VAIO y aparece en la pantalla:

Iteraciones => 24 Epsilon => 5.960464477539e-008

Presione una tecla para continuar . . .

Además justifique si el resultado anterior es correcto con lo que se espera del lenguaje C++ y su definición de float.

3) Explique que pasa si se cambian las variables a double, puede demostrar que esta nueva versión del programa es correcto o no. ¿Que resultados se obtienen y porqué?

4) Usted conoce que matemáticamente $a + b = b + a$, sin embargo esto no ocurre en los lenguajes de programación. No lo explique con palabras, sino con un programa correcto desde el punto de vista de funcionamiento (es decir que termine) pero que claramente dados valores a, b y c se tenga que $a + b + c \neq c + b + a$.

5) Explique porque es correcto en los lenguajes de programación que representan los números reales por un rango y con una mantisa finita ($\mathbb{RC}$ es el conjunto de los números con representación por computadora) la afirmación: $\exists \varepsilon \in \mathbb{RC}$, $\varepsilon > 0$ tal que si $1+\varepsilon \in \mathbb{RC}$, $s := 1+\varepsilon$ entonces $s = 1$. ¿No contradice esto la proposición matemática: $\forall \varepsilon \in \mathbb{R}$, $\varepsilon > 0$ se tiene que $1 + \varepsilon \neq 1$? Donde $\mathbb{R}$ son los números reales,:= es la asignación, $=$ es la igualdad y $\neq$ la desigualdad. En forma resumida que consecuencias tienen estas dos afirmaciones que son ciertas, una en el universo numérico de las computadoras y la otra en el universo de la Lógica Matemática.

Suerte en su examen y por favor estudien y practiquen.

Los ejemplos de inferencia fueron tomados del libro Introducción a la Lógica Matemática, P. Suppes y S. Hill, Editorial Reverte, S.A., 2009. En este libro o en uno similar encontrarán muchos más.

Todo lo que esta escrito en está guía es susceptible de verificación lógica y de experimentación con su computadora. Puede utilizar Prolog para inferencia, o sea, para deducir de ejemplos de proposiciones. Y tiene otra herramienta de experimentación para la verificación de conceptos de Lógica.