Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2017

Departamento de Computación - FCEyN - UBA

Especificación - clase 2

Lógica proposicional - tipos básicos

Lógica proposicional - sintaxis

- Símbolos:

  $$\text{true} \ , \ \text{false} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

  $$p \ , \ q \ , \ r \ , \ \ldots$$

- Fórmulas
  1. true y false son fórmulas
  2. Cualquier variable proposicional es una fórmula
  3. Si A es una fórmula, $\neg A$ es una fórmula
  4. Si $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ es una fórmula
  5. Si $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \vee A_2 \vee \cdots \vee A_n)$ es una fórmula
  6. Si A y B son fórmulas, $(A \rightarrow B)$ es una fórmula
  7. Si A y B son fórmulas, $(A \leftrightarrow B)$ es una fórmula

Semántica clásica

- Dos valores de verdad posibles:
  1. verdadero (o bien "true", "T" o "V")
  2. falso (o bien "false" o "F")
- El valor de verdad de una fórmula se obtiene a partir del valor de verdad de sus subfórmulas:
  - true se interpreta como verdadero.
  - false se interpreta como falso.
  - $\neg A$ se interpreta como "no", se llama negación
  - $\wedge$ se interpreta como "y", se llama conjunción
  - $\vee$ se interpreta como "o" (no exclusivo), se llama disyunción
  - $\rightarrow$ se interpreta como "si... entonces", se llama implicación
  - $\leftrightarrow$ se interpreta como "si y solo si", se llama doble implicación o equivalencia

Semántica clásica

Conociendo el valor de las variables proposicionales de una fórmula, podemos calcular el valor de verdad de la fórmula:

p	$\neg p$
T	F
F	T

p	q	$(p \wedge q)$
T	T	T
T	F	F
F	T	F
F	F	F

p	q	$(p \vee q)$
T	T	T
T	F	T
F	T	T
F	F	F

p	q	$(p \rightarrow q)$
T	T	T
T	F	F
F	T	T
F	F	T

p	q	$(p \leftrightarrow q)$
T	T	T
T	F	F
F	T	F
F	F	T

Ejemplo: tabla de verdad para $((p \wedge q) \rightarrow r)$

p	q	r	$(p \wedge q)$	$((p \wedge q) \rightarrow r)$
T	T	T	T	T
T	T	F	T	F
T	F	T	F	T
T	F	F	F	T
F	T	T	F	T
F	T	F	F	T
F	F	T	F	T
F	F	F	F	T

Dos conectivos bastan

- $\neg$ y $\vee$
  - $(A \wedge B)$ es $\neg(\neg A \vee \neg B)$
  - $(A \rightarrow B)$ es $(\neg A \vee B)$
  - true es $(A \vee \neg A)$
  - false es $\neg$true
- $\neg$ y $\wedge$
  - $(A \vee B)$ es $\neg(\neg A \wedge \neg B)$
  - $(A \rightarrow B)$ es $\neg(A \wedge \neg B)$
  - false es $(A \wedge \neg A)$
  - true es $\neg$false
- $\neg$ y $\rightarrow$
  - $(A \vee B)$ es $(\neg A \rightarrow B)$
  - $(A \wedge B)$ es $\neg(A \rightarrow \neg B)$
  - true es $(A \rightarrow A)$
  - false es $\neg$true

Tautologías, contradicciones y contingencias

- Una fórmula es una tautología si siempre toma el valor T para valores definidos de sus variables proposicionales.
  Por ejemplo, $((p \wedge q) \rightarrow p)$ es tautología:

p	q	$(p \wedge q)$	$((p \wedge q) \rightarrow p)$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	T

- Una fórmula es una contradicción si siempre toma el valor F para valores definidos de sus variables proposicionales.
  Por ejemplo, $(p \wedge \neg p)$ es contradicción:

p	$\neg p$	$(p \wedge \neg p)$
T	F	F
F	T	F

- Una fórmula es una contingencia cuando no es ni tautología ni contradicción.

Equivalencias entre fórmulas

- **Teorema.** Las siguientes son tautologías.

1. Idempotencia
   $(p \wedge p) \leftrightarrow p$
   $(p \vee p) \leftrightarrow p$
2. Asociatividad
   $(p \wedge q) \wedge r \leftrightarrow p \wedge (q \wedge r)$
   $(p \vee q) \vee r \leftrightarrow p \vee (q \vee r)$
3. Conmutatividad
   $(p \wedge q) \leftrightarrow (q \wedge p)$
   $(p \vee q) \leftrightarrow (q \vee p)$
4. Distributividad
   $p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$
   $p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$
5. Reglas de De Morgan
   $\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$
   $\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ es tautología.
- También decimos que A fuerza a B o que B es más débil que A.
- Por ejemplo,
  1. ¿$(p \wedge q)$ es más fuerte que p? sí
  2. ¿$(p \vee q)$ es más fuerte que p? no
  3. ¿p es más fuerte que $(q \to p)$? sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ está indefinido.
  4. ¿p es más fuerte que q? no
  5. ¿p es más fuerte que p? sí
  6. ¿hay una fórmula más fuerte que todas? false!
  7. ¿hay una fórmula más débil que todas? true

Expresión bien definida

- Toda expresión está bien definida en un estado si todas las proposiciones valen T o F.
- Sin embargo, existe la posibilidad de que haya expresiones que no estén bien definidas.
  - Por ejemplo, la expresión x/y no está bien definida si $y = 0$.
- Por esta razón, necesitamos una lógica que nos permita decir que está bien definida la siguiente expresión
  - $y = 0 \vee x/y = 5$
- Para esto, introducimos tres valores de verdad:
  1. verdadero (T)
  2. falso (F)
  3. indefinido ($\perp$)

Semántica trivaluada (secuencial)

Se llama secuencial porque ...

- los términos se evalúan de izquierda a derecha,
- la evaluación termina cuando se puede deducir el valor de verdad, aunque el resto esté indefinido.

Introducimos los operadores lógicos $\wedge_L$ (y-luego, o *conditional and*, o **cand**), $\vee_L$ (o-luego o *conditional or*, o **cor**).

p	q	$(p \wedge_L q)$
T	T	T
T	F	F
F	T	F
F	F	F
T	$\perp$	$\perp$
F	$\perp$	F
$\perp$	T	$\perp$
$\perp$	F	$\perp$
$\perp$	$\perp$	$\perp$

p	q	$(p \vee_L q)$
T	T	T
T	F	T
F	T	T
F	F	F
T	$\perp$	T
F	$\perp$	$\perp$
$\perp$	T	$\perp$
$\perp$	F	$\perp$
$\perp$	$\perp$	$\perp$

Semántica trivaluada (secuencial)

¿Cuál es la tabla de verdad de $\to_L$?

p	q	$(p \to_L q)$
T	T	T
T	F	F
F	T	T
F	F	T
T	$\perp$	$\perp$
F	$\perp$	T
$\perp$	T	$\perp$
$\perp$	F	$\perp$
$\perp$	$\perp$	$\perp$

Tipos de datos

- Un **tipo de datos** es un conjunto de valores (el conjunto base del tipo) provisto de una serie de operaciones que involucran a esos valores.
- Para hablar de un elemento de un tipo T en nuestro lenguaje, escribimos un término o expresión
  - Variable de tipo T (ejemplos: x, y, z, etc)
  - Constante de tipo T (ejemplos: 1, -1, $\frac{1}{5}$, 'a', etc)
  - Función (operación) aplicada a otros términos (del tipo T o de otro tipo)
- Todos los tipos tienen un elemento distinguido: $\perp$ o Indef

Tipos de datos de nuestro lenguaje de especificación

- Básicos
  - Enteros ($\mathbb{Z}$)
  - Reales ($\mathbb{R}$)
  - Booleanos (Bool)
  - Caracteres (Char)
- Enumerados
- Uplas
- Secuencias

Tipo $\mathbb{Z}$ (números enteros)

- Su conjunto base son los números enteros.
- Constantes: 0 ; 1 ; -1 ; 2 ; -2 ; ...
- Operaciones aritméticas:
  - $a + b$ (suma); $a - b$ (resta); abs(a) (valor absoluto)
  - $a * b$ (multiplicación); a div b (división entera);
  - a mod b (resto de dividir a a por b), a^b o pot(a,b) (potencia)
  - a / b (división, da un valor de $\mathbb{R}$)
- Comparaciones de términos de tipo $\mathbb{Z}$:
  - $a < b$ (menor)
  - $a \leq b$ o a <= b (menor o igual)
  - $a > b$ (mayor)
  - $a \geq b$ o a >= b (mayor o igual)
  - $a = b$ (iguales)
  - $a \neq b$ (distintos)

Tipo $\mathbb{R}$ (números reales)

- Su conjunto base son los números reales.
- Constantes: 0 ; 1 ; -7 ; 81 ; 7,4552 ; π ...
- Operaciones aritméticas:
  - Suma, resta y producto (pero no div y mod)
  - a/b (división)
  - $\log_b(a)$ (logaritmo)
  - Funciones trigonométricas
- Los mismos operadores de comparación que para $\mathbb{Z}$:
  - $a < b$ (menor)
  - $a \leq b$ o a <= b (menor o igual)
  - $a > b$ (mayor)
  - $a \geq b$ o a >= b (mayor o igual)
  - $a = b$ (iguales)
  - $a \neq b$ (distintos)

$\mathbb{Z} \subseteq \mathbb{R}$

- $\mathbb{Z}$ es un subconjunto de $\mathbb{R}$
- Conversión de entero a real:
  - Si $x \in \mathbb{Z}$, entonces $x \in \mathbb{R}$
- Conversión de real a entero (truncamiento):
  - $\lfloor a \rfloor$ o floor(a)
  - Ejemplo $floor(7{,}57) = 7$
- Conversión de real a entero (redondeo):
  - round(a)
  - Ejemplo $round(7{,}57) = 8$

Tipo Bool (valor de verdad)

- Su conjunto base es $\mathbb{B} = \{\textbf{true}, \textbf{false}\}$.
- Conectivos lógicos: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ con la semántica bi-valuada estándar.
- Comparación de términos de tipo Bool:
  - $a = b$
  - $a \neq b$

Tipo Bool (valores de verdad)

- $\neg A$ se puede escribir no(A)
- $(A \wedge B)$ se puede escribir (A `&&` B)
- $(A \vee B)$ se puede escribir (A `||` B)
- $(A \rightarrow B)$ se puede escribir (A `-->` B)
- $(A \leftrightarrow B)$ se puede escribir (A `<-->` B)
- $A \neq B$ se puese escribir $A \neq B$. También se puede escribir $\neg(A = B)$

Ejercicio

¿Cuáles de las siguientes son una especificación adecuada para el problema de decidir si un número entero es positivo?

proc esPositivo (in $x : \mathbb{Z}$, out $r : Bool$) {

$Pre\{True\}$

1. $Post\{x > 0 \wedge r = true\}$
2. $Post\{(x > 0 \wedge r = true) \wedge (x <= 0 \wedge r = false)\}$
3. $Post\{(x > 0 \wedge r = true) \vee (x <= 0 \wedge r = false)\}$
4. $Post\{x \leq 0 \implies r = false\}$
5. $Post\{x > 0 \implies r = true\}$
6. $Post\{(x > 0 \implies r = true) \vee (x <= 0 \implies r = false)\}$
7. $Post\{x > 0 \iff r = true\}$
8. $Post\{x <= 0 \iff r = false\}$

}

Tipo Char (caracteres)

- Sus elementos son las letras, dígitos y símbolos.
- Constantes: ‘a‘, ‘b‘, ‘c‘, . . . , ‘z‘, . . . , ‘A‘, ‘B‘, ‘C‘, . . . , ‘Z‘, . . . , ‘0‘, ‘1‘, ‘2‘, . . . , ‘9‘ (en el orden dado por el estándar ASCII).
- Función ord, que numera los caracteres, con las siguientes propiedades:
  - ord(‘a‘) $+ 1 =$ ord(‘b‘)
  - ord(‘A‘) $+ 1 =$ ord(‘B‘)
  - ord(‘1‘) $+ 1 =$ ord(‘2‘)
- Función char, de modo tal que char(ord(c)) $= c$.
- Las comparaciones entre caracteres son comparaciones entre sus órdenes, de modo tal que $a < b$ es equivalente a ord(a) $<$ ord(b).

Tipos enumerados

- Cantidad finita de elementos.
  Cada uno, denotado por una constante.

  enum *Nombre* { *constantes* }

- *Nombre* (del tipo): tiene que ser nuevo.
- *constantes*: nombres nuevos separados por comas.
- Convención: todos con mayúsculas.
- `ord`(a) da la posición del elemento en la definición (empezando de 0).
- Inversa: El nombre del tipo funciona como inversa de `ord`.

Ejemplo de tipo enumerado

Definimos el tipo `Día` así:

```
enum Día {
  LUN, MAR, MIER, JUE, VIE, SAB, DOM
}
```

- Valen:
  - ord(LUN) $= 0$
  - Día(2) $=$ MIE
  - JUE $<$ VIE

Tipo upla (o tupla)

- Uplas, de dos o más elementos, cada uno de cualquier tipo.
- $T_0 \times T_1 \times \cdots \times T_k$: Tipo de las k-uplas de elementos de tipos T_0, T_1, ... T_k, respectivamente, donde k es fijo.
- Ejemplos:
  - $\mathbb{Z} \times \mathbb{Z}$ son los pares ordenados de enteros.
  - $\mathbb{Z} \times$ Char $\times$ Bool son las triplas ordenadas con un entero, luego un carácter y luego un valor booleano.
- $n$ésimo: $(a_0, \ldots, a_k)_m$ es el valor a_m en caso de que $0 \leq m \leq k$. Si no, está indefinido.
- Ejemplos:
  - $(7, 5)_0 = 7$
  - (‘a’, *Domingo*, $78)_2 = 78$

Términos

- Simples (variables y constantes del tipo).
- Complejos (combinaciones de funciones aplicadas a funciones, constantes y variables).

Ejemplos de términos de tipo $\mathbb{Z}$

- $0 + 1$
- $((3 + 4) * 7)^2 - 1$
- $2 * \text{if } (1 + 1 = 2) \text{ then } 1 \text{ else } 0 \text{ fi}$
- $1 + \text{ord}(\text{`}A\text{`})$
- con x variable de tipo $\mathbb{Z}$; y de tipo $\mathbb{R}$; z de tipo Bool
  - $2 * x + 1$
  - $(\text{if } (y^2 > \pi) \text{ then } 1 \text{ else } 0 \text{ fi }) + x$
  - $(x \bmod 3) * \text{if } (z) \text{ then } 1 \text{ else } 0 \text{ fi}$

Semántica de los términos

- Vimos que los términos representan elementos de los tipos
- Los términos tienen valor indefinido cuando no se puede hacer alguna operación
  - 1 div 0
  - $(-1)^{1/2}$
- Las operaciones son estrictas
  - Si uno de sus argumentos es indefinido, el resultado también está indefinido
  - Ejemplos:
    - $0 * (-1)^{1/2}$ es indefinido ($*$ es estricto)
    - $0^{1/0}$ es indefinido (pot es estricto)
    - $((1 + 1 = 2) \vee_L (0 > 1/0))$ es verdadero ($\vee_L$ no es estricto)
  - Las comparaciones con $\perp$ son indefinidas
    - En particular, si x está indefinido, $x = x$ es indefinido (no es verdadero)

Semántica de los términos

Con x e y variables de tipo $\mathbb{Z}$, decidir el valor de verdad de los siguientes términos cuando $x = 0$ e $y = 5$:

- $y \neq 5$
- $x/(y - 5) \geq 0$
- $y = 5 \vee_L x/(y - 5) \geq 0$
- $x/(y - 5) \geq 0 \vee_L y = 5$
- $y \neq 5 \wedge_L x/(y - 5) \geq 0$
- $x/(y - 5) \geq 0 \wedge_L y \neq 5$
- $(y \neq 5 \wedge_L x/(y - 5) \geq 0) \vee_L x = 0$
- $y + \text{abs}(x/(y - 5)) \geq 0$

Funciones y predicados auxiliares

- Asignan un nombre a una expresión.
- Facilitan la lectura y la escritura de especificaciones.
- Modularizan la especificación.

  fun f(argumentos) : *tipo* = *expresion*;

  pred p(argumentos){*formula*}

  - f es el nombre de la función, que puede usarse en el resto de la especificación en lugar de la expresión e.
  - Los argumentos son opcionales y se reemplazan en e cada vez que se usa f.
  - *tipo* es el tipo del resultado de la función (el tipo de e).
- Ejemplo:

  fun $suc(x : \mathbb{Z}) : \mathbb{Z} = x + 1$;

Ejemplos de funciones auxiliares

- fun $e() : \mathbb{R} = 2{,}7182$;
- fun *inverso*$(x : \mathbb{R}) : \mathbb{R} = 1/x$;
- pred *par*$(n : \mathbb{Z})$ { (n mod 2) $= 0$ }

  pred *impar*$(n : \mathbb{Z})$ { $\neg$ (par(n)) }
- pred *esFinde*$(d :$ Día$)\{d =$ SAB $\vee$ $d =$ DOM$\}$

  Otra forma:

  pred *esFinde2*$(d :$ Día$)\{d >$ VIE$\}$

Expresiones condicionales

Función que elige entre dos elementos del mismo tipo, según una condición booleana (guarda)

- si la guarda es verdadera, elige el primero
- si no, elige el segundo

Por ejemplo

- expresión que devuelve el máximo entre dos elementos:

  fun máx$(a, b : \mathbb{Z}) : \mathbb{Z} =$ IfThenElseFi$\langle\mathbb{Z}\rangle(a > b, a, b)$;

  cuando los argumentos se deducen del contexto, se puede escribir directamente

  fun máx$(a, b : \mathbb{Z}) : \mathbb{Z} =$ IfThenElseFi$(a > b, a, b)$; o bien

  fun máx$(a, b : \mathbb{Z}) : \mathbb{Z} =$ if $a > b$ then a else b fi;
- expresión que dado x devuelve $1/x$ si $x \neq 0$ y 0 sino

  fun unoSobre$(x : \mathbb{R}) : \mathbb{R} =$ $\underbrace{\text{if } x \neq 0 \text{ then } 1/x \text{ else } 0 \text{ fi;}}_{\text{no se indefine cuando } x = 0}$

Definir funciones auxiliares versus especificar problemas

Definimos funciones auxiliares

- Expresiones del lenguaje, que se usan dentro de las especificaciones como reemplazos sintácticos. Son de cualquier tipo.
- Dado que es un reemplazo sintáctico, no se permiten definiciones recursivas!

Especificamos problemas

- Condiciones (el contrato) que debería cumplir un algoritmo para ser solución del problema.
- En una especificación dando la precondición y la postcondición con predicados de primer orden.
- No podemos usar otros problemas en la especificación (dado que no pertenecen a la lógica de primer orden). Sí podemos usar predicados y funciones auxiliares ya definidos.

Especificar problemas

- **Ejemplo:** Especificar el problema de retornar el i-ésimo dígito de la representación decimal del número π.
- proc *piesimo*(in $i : \mathbb{Z}$, out *result* $: \mathbb{Z}$) {
  Pre $\{i > 0\}$
  Post $\{result = \lfloor \pi * 10^i \rfloor \bmod 10\}$
  }

Cuantificadores

El lenguaje de especificación provee formas de predicar sobre los elementos de un tipo de datos

- $(\forall x : T)\ P(x)$: Término de tipo Bool. Afirma que todos los elementos de tipo T cumplen la propiedad P.
  - Se lee "Para todo x de tipo T se cumple $P(x)$"
- $(\exists x : T)\ P(x)$: Término de tipo Bool. Afirma que al menos un elemento de tipo T cumple la propiedad P.
  - Se lee "Existe al menos un x de tipo T que cumple $P(x)$"

En la expresión $(\forall x : T)\ P(x)$, la variable x está ligada al cuantificador. Una variable es libre cuando no está ligada a ningún cuantificador.

Ejemplo

- **Ejemplo:** Crear un predicado `esPrimo` que sea **Verdadero** si y sólo si el número n es un número primo.
- pred esPrimo($n : \mathbb{Z}$) {
  $n > 1 \ \wedge\ (\forall n' : \mathbb{Z})(1 < n' < n \rightarrow n \bmod n' \neq 0)$
  }
- **Ejemplo:** Especificar el problema de, dado un número mayor a 1, indicar si el número es un número primo.
- proc *primo*(in $n : \mathbb{Z}$, out *result* : Bool) {
  Pre $\{n > 1\}$
  Post $\{result = \text{esPrimo}(n)\}$
  }

Operando con cuantificadores

- **Ejemplo:** Todos los enteros entre 1 y 10 son pares:
  $(\forall n : \mathbb{Z})(1 \leq n \leq 10 \rightarrow n \bmod 2 = 0)$.
- **Ejemplo:** Existe un entero entre 1 y 10 que es par:
  $(\exists n : \mathbb{Z})(1 \leq n \leq 10 \wedge n \bmod 2 = 0)$.
- En general, si queremos decir que todos los enteros x que cumplen $P(x)$ también cumplen $Q(x)$, decimos:
  $(\forall x : \mathbb{Z})(P(x) \rightarrow Q(x))$.
- Para decir que existe un entero que cumple $P(x)$ y que también cumple $Q(x)$, decimos:
  $(\exists x : \mathbb{Z})(P(x) \wedge Q(x))$.

Operando con cuantificadores

- La negación de un cuantificador universal es un cuantificador existencial, y viceversa:
  $\neg(\forall n : \mathbb{Z})P(n) \ \leftrightarrow\ (\exists n : \mathbb{Z})\neg P(n)$.
  $\neg(\exists n : \mathbb{Z})P(n) \ \leftrightarrow\ (\forall n : \mathbb{Z})\neg P(n)$.
- Un cuantificador universal generaliza la conjunción:
  $(\forall n : \mathbb{Z})\big(a \leq n \leq b \rightarrow P(n)\big) \ \wedge\ P(b+1)$
  $\leftrightarrow\ (\forall n : \mathbb{Z})\big(a \leq n \leq b+1 \rightarrow P(n)\big)$.
- Un cuantificador existencial generaliza la disyunción:
  $(\exists n : \mathbb{Z})\big(a \leq n \leq b \wedge P(n)\big) \ \vee\ P(b+1)$
  $\leftrightarrow\ (\exists n : \mathbb{Z})\big(a \leq n \leq b+1 \wedge P(n)\big)$.

Especificar problemas

- La conjetura de Goldbach dice que todo entero par mayor que 2 puede ser expresado como la suma de dos números primos.
- **Ejemplo:** Especificar un procedimiento que indique si esta conjetura es verdadera.
- proc *goldbach*(out *result* : Bool) {
  Pre $\{True\}$
  Post $\{result = true \leftrightarrow (\forall n : \mathbb{Z})(n > 2 \wedge n \bmod 2 = 0 \rightarrow (\exists p : \mathbb{Z})(\exists q : \mathbb{Z})(esPrimo(p) \wedge esPrimo(q) \wedge n = p + q))\}$
  }
- Observar que estamos especificando el problema. Es posible que en este punto no sepamos cómo vamos a resolver este problema (si es que se puede resolver!), y es bueno no pensar en eso al momento de especificar.

Especificar problemas

- **Ejemplo:** Especificar un procedimiento que calcule el máximo común divisor (mcd) entre dos números positivos.
- proc *mcd*(in $n : \mathbb{Z}$, in $m : \mathbb{Z}$, out *result* : $\mathbb{Z}$) {
  Pre $\{n \geq 1 \wedge m \geq 1\}$
  Post $\{n \bmod result = 0 \wedge m \bmod result = 0 \wedge \neg(\exists p : \mathbb{Z})(p > result \wedge n \bmod p = 0 \wedge m \bmod p = 0)\}$
  }
- Observar que no damos una fórmula que especifica el valor de retorno, sino que solamente damos las propiedades que debe cumplir!

Repaso: Pasaje de parámetros

in, out, inout

- Parámetros de entrada (**in**): Parámetros con valores que se copian hacia el procedimiento.
- Parámetros de salida (**out**): Parámetros para que el procedimiento informe resultados al código llamador.
- Parámetros de entrada-salida (**inout**): Parámetros de entrada y salida.

Especificar problemas

- **Ejemplo:** Especificar un procedimiento que incremente en una unidad una variable de tipo $\mathbb{Z}$ que representa un contador.
- proc *inc*(inout $n : \mathbb{Z}$) {
  Pre $\{n = n_0\}$
  Post $\{n = n_0 + 1\}$
  }
- La variable n_0 es una metavariable, que representa el valor inicial de la variable n, y que usamos en la postcondición para relacionar el valor de salida de n con su valor inicial.

Especificar problemas

- **Ejemplo:** Especificar un procedimiento que decremente en una unidad una variable de tipo $\mathbb{Z}$ que representa un contador. Para poder aplicar esta operación el contador tiene que ser mayor que 0.

- proc *dec*(inout $n : \mathbb{Z}$) {
  Pre $\{n = n_0 \wedge n \geq 1\}$
  Post $\{n = n_0 - 1\}$
  }

Especificar problemas

- **Ejemplo:** Dados cuatro enteros a, b, c, d tales que
  - a y b representan un número fraccionario $\frac{a}{b}$ (con $b \neq 0$) y son coprimos entre sí, y
  - c y d representan el número fraccionario $\frac{c}{d}$ (con $d \neq 0$)y son coprimos entre sí,

  escribir la especificación de un procedimiento que compute $\frac{a}{b} + \frac{c}{d}$ y coloque el resultado en dos números enteros.

- proc *suma*(in $a, b, c, d : \mathbb{Z}$, out $e, f : \mathbb{Z}$) {
  Pre $\{\text{coprimos}(a, b) \wedge \text{coprimos}(c, d) \wedge b \neq 0 \wedge d \neq 0\}$
  Post $\{\frac{e}{f} = \frac{a}{b} + \frac{c}{d}\}$
  }

- pred *coprimos*($n, m : \mathbb{Z}$) {
  $\neg(\exists i : \mathbb{Z})(i \geq 2 \wedge n \bmod i = 0 \wedge m \bmod i = 0)$
  }

Especificar problemas

- **Continuación:** Modificar la solución anterior para que el resultado sean dos números coprimos entre sí.

- proc *suma*(in $a, b, c, d : \mathbb{Z}$, out $e, f : \mathbb{Z}$) {
  Pre $\{\text{coprimos}(a, b) \wedge \text{coprimos}(c, d) \wedge b \neq 0 \wedge d \neq 0\}$
  Post $\{\frac{e}{f} = \frac{a}{b} + \frac{c}{d} \wedge \text{coprimos}(e, f)\}$
  }

Especificando un semáforo

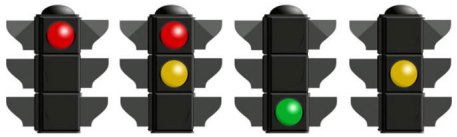
- **Ejemplo:** Representamos con tres valores de tipo *Bool* el estado de la luz verde, amarilla y roja de un semáforo.

- Escribir el procedimiento que inicializa el semáforo con la luz roja y el resto de las luces apagadas.

- proc *iniciar*(out v, a, r : Bool) {
  Pre $\{True\}$
  Post $\{r = \textbf{true} \wedge a = \textbf{false} \wedge v = \textbf{false}\}$
  }

Especificando un semáforo

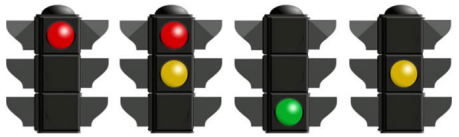
- **Continuación:** Especificar un procedimiento que avance el estado de las luces de un semáforo.



- Podemos especificar un predicado para representar cada estado válido del semáforo:
- pred esRojo(v, a, r: Bool) { $v = \textbf{false} \land a = \textbf{false} \land r = \textbf{true}$ }
- pred esRojoAmarillo(v, a, r: Bool) {
  $v = \textbf{false} \land a = \textbf{true} \land r = \textbf{true}$ }
- pred esVerde(v, a, r: Bool){ $v = \textbf{true} \land a = \textbf{false} \land r = \textbf{false}$ }
- pred esAmarillo(v, a, r: Bool){ $v = \textbf{false} \land a = \textbf{true} \land r = \textbf{false}$ }

Especificando un semáforo

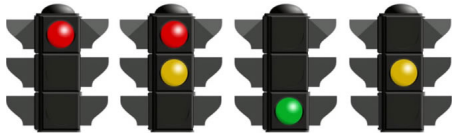
- **Continuación:** Ahora podemos escribir una nueva versión de iniciar usando los predicados definidos anteriormente.



- proc $iniciar$(out v, a, r : Bool) {
  Pre {$True$}
  Post {$esRojo(v, a, r)$}
  }

Especificando un semáforo

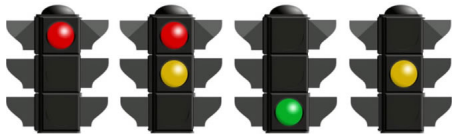
- **Continuación:** Especificar un procedimiento que avance el estado de las luces de un semáforo.



- Podemos especificar un predicado para representar que el semáforo está en un estado válido:
- pred esValido(v, a, r: Bool) {
  $esRojo(v, a, r)$
  $\lor\ esRojoAmarillo(v, a, r)$
  $\lor\ esVerde(v, a, r)$
  $\lor\ esAmarillo(v, a, r)$
  }

Especificando un semáforo

- Ahora especificamos el problema de avanzar:



- proc $avanzar$(inout v, a, r : Bool) {
  Pre {
  $esValido(v, a, r)$
  $\land v = v_0 \land r = r_0 \land a = a_0$
  }
  Post {
  $(esRojo(v_0, a_0, r_0) \rightarrow esRojoAmarillo(v, a, r))$
  $\land\ (esRojoAmarillo(v_0, a_0, r_0) \rightarrow esVerde(v, a, r)$
  $\land\ (esVerde(v_0, a_0, r_0) \rightarrow esAmarillo(v, a, r))$
  $\land\ (esAmarillo(v_0, a_0, r_0) \rightarrow esRojo(v, a, r))\}$
  }

Bibliografía

- David Gries - The Science of Programming
  - Chapter 1 - Propositions (Fórmulas, Tautologías, etc.)
  - Chapter 2 - Reasoning using Equivalence Transformations (Propiedades, De Morgan, etc.)