TD3: Algoritmos y Estructuras de Datos

Prof. Agustín Garassino, Gervasio Pérez

Primer Semestre de 2024

Clase Teórica 2
Especificación Formal de Problemas (1)
Predicados y Funciones Auxiliares

Resumen de la clase

- Especificaciones
- Lógica Proposicional y de Primer Orden
- Repaso de cuantificadores
- Tipos de datos
- Lógica trivaluada

Especificación: explicar el *qué*

Recordemos que la especificación (o contrato) de una función...

- ...define su nombre, parámetros de entrada y tipo de retorno (**Encabezado**)
- ...establece los requisitos sobre los parámetros de entrada (**Requiere**)
- ...formaliza las propiedades que cumple el resultado de ejecutarse (**Asegura**)
  - sobre el valor de retorno
  - sobre los parámetros que hayan sido modificados por *referencia*

Ejemplo de TD1:

Calcular el volumen de un cilindro de radio r y altura h.

- Encabezado: `volumen_cilindro(r:float, h:float)` $\rightarrow$ `float`
- Requiere: $r > 0$; $h > 0$
- Devuelve: aproximadamente $\pi \cdot r^2 \cdot h$, donde $\pi \approx 3.1415927$

Contratos en TD3

En esta materia, nuestro formato de especificación será

- la declaración de la función en C++ como **encabezado**
- un apartado **Pre** que especifica la **Pre**condición (**requiere**) de la función
- un apartado **Post** que especifica la **Post**condición (**asegura**) de la función

El mismo ejemplo de TD1

Calcular el volumen de un cilindro de radio `r` *y altura* `h`.

```
float volumen_cilindro(float r, float h)
```

Pre: $r > 0 \land h > 0$

Post: $res = \pi * r^2 * h$

- *la variable especial* ***res*** *representa al valor de retorno de la función*

Pre y **Post** estarán escritos en un *lenguaje formal*:

Lógica de Primer Orden

Repaso: Lógica Proposicional

Elementos sintácticos:

- **valores de verdad constantes** (V, F);
- **variables proposicionales** (p, q, r);
- **conectivos lógicos** proposicionales $(\wedge, \vee, \neg, \implies)$;

Reglas de formación de fórmulas. Si G_1 y G_2 son fórmulas:

- una *variable proposicional* es una **fórmula**;
- $\neg G_1$ es una **fórmula**;
- $(G_1 \wedge G_2)$ es una **fórmula**;
- $(G_1 \vee G_2)$ es una **fórmula**;
- $(G_1 \implies G_2)$ es una **fórmula**;

Semántica: *Tablas de verdad*

Ejemplos.

- $p \vee F$
- $(p \wedge q) \implies r$
- $\neg(p \wedge q) \implies (r \vee \neg s)$

Ejercicios de lógica proposicional

Escribir fórmulas de lógica proposicional que representen las siguientes afirmaciones, usando variables proposicionales como sea necesario:

- "Leo y escribo".
- "Si llueve, se moja la calle y la vereda".
- "Si llueve y llevo paraguas, no me mojo".

- $p \wedge q$
  $p =$ "leo", q = "escribo"
- $p \implies (q \wedge r)$
  $p =$ "llueve", $q =$ "calle mojada", $r =$ "vereda mojada".
- $p \wedge q \implies \neg r$
  $p =$ "llueve", $q =$ "llevo paraguas", $r =$ "me mojo".

Repaso: Lógica de Primer Orden (con igualdad)

Elementos sintácticos:

- **valores de verdad constantes** (V, F);
- **conectivos lógicos** proposicionales $(\wedge, \vee, \neg, \implies)$;
- **predicados** (P, Q, R)
- **cuantificadores** de individuos $((\forall x), (\exists x))$;
  - **variables de individuo** (x, y, z);
- **funciones** (f, g, h);
  - las funciones serán las operaciones provistas por cada tipo de datos.
- el **símbolo de igualdad** $=$.

Reglas de formación de fórmulas y términos. Si P es un predicado, t_i es un término, f es una función, y G_1 y G_2 son fórmulas:

- una *variable* es un **término**;
- la expresión $f(t_1, \ldots, t_n)$ es un **término**;
- la expresión $P(t_1, \ldots, t_n)$ es una **fórmula**;
- la expresión $t_1 = t_2$ es una **fórmula**;
- $\neg G_1$ es una **fórmula**;
- si $*$ es algún conectivo lógico, $(G_1 * G_2)$ es una **fórmula**;
- Si x es una variable, $(\forall x) G_1$ y $(\exists x) G_1$ son **fórmulas**.

Términos versus Fórmulas

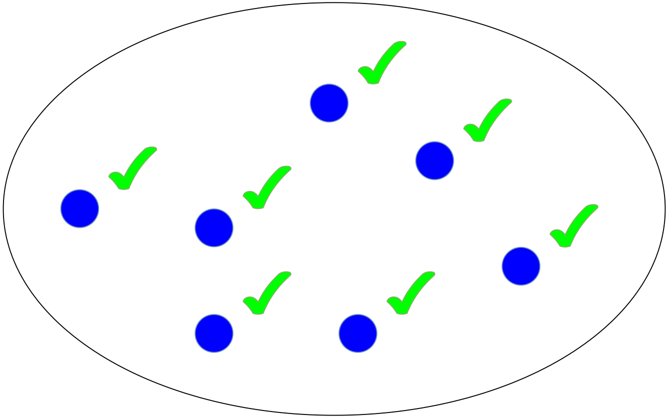
Hay que diferenciar bien entre ambos.

1. Término: expresión de algún tipo T. No es inherentemente verdadera o falsa. **Una expresión `bool` es un término, ¡no confundir!**
   - **float**: $\frac{1}{\sqrt{(5)}}$
   - **int**: $|v|$ v:vector<...>
   - **int**: $v[i]+5$ i:**int**, v:vector<**int**>.
   - **bool**: a || b a,b:**bool**.
2. Fórmula: expresión de la que podemos extraer un valor de verdad *Verdadero/Falso*.
   - $x < \frac{1}{\sqrt{(5)}}$ x: **int**,**float**.
   - $0 \leq i < |v|$ i:**int**, v:vector<...>
   - $v[i+1] = v[i] + 5$ i:**int**, v:vector<**int**|**float**>.
   - a || b = true a,b:**bool**.

Cuantificador universal: interpretación

Cuando tenemos un predicado P y una expresión

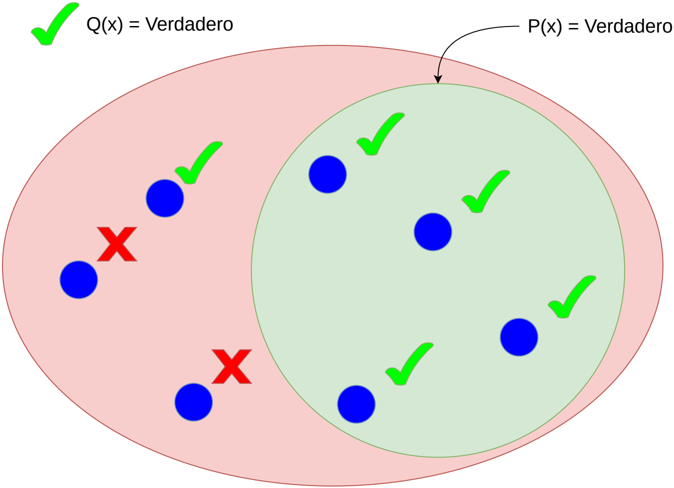
$$(\forall x)P(x)$$



Cuantificador universal + implicación

Cuando tenemos predicados P,Q y una expresión

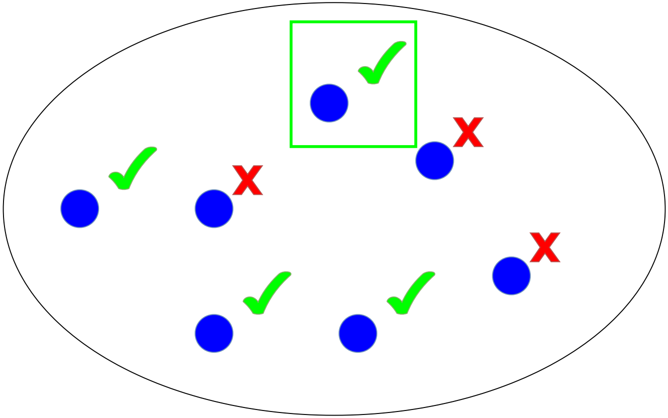
$$(\forall x)P(x) \implies Q(x)$$



Cuantificador existencial

Cuando tenemos un predicado P y una expresión

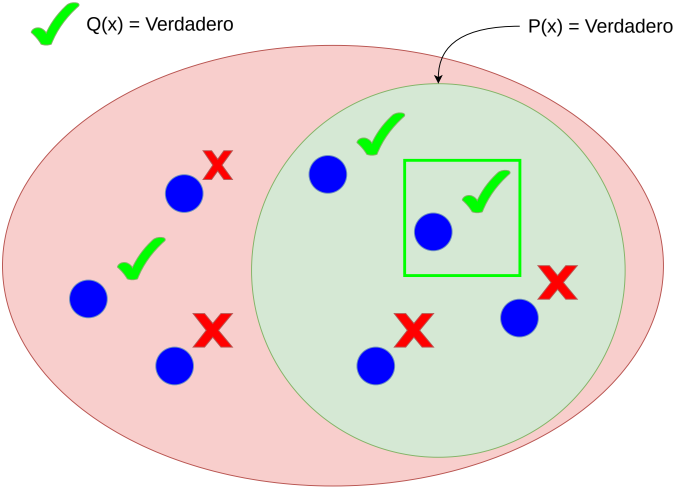
$$(\exists x)P(x)$$



Cuantificador existencial + conjunción lógica

Cuando tenemos predicados P,Q y una expresión

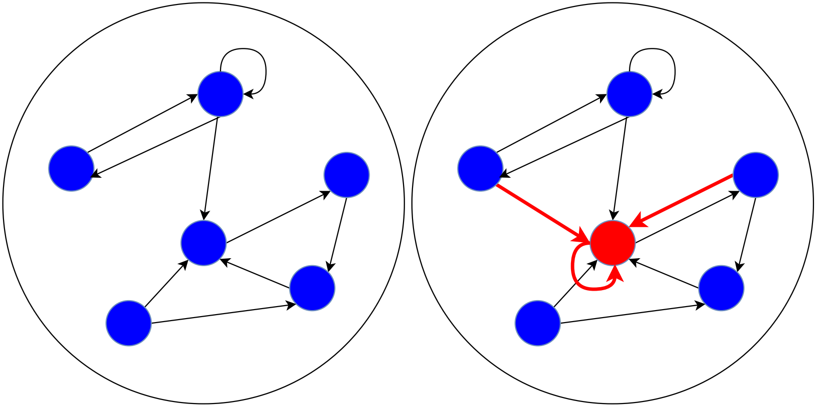
$$(\exists x) P(x) \wedge Q(x)$$



Universal + Existencial versus Existencial + Universal

Si tenemos un predicado $P(x, y)$ que relaciona pares de elementos...

$(\forall x)(\exists y)P(x, y)$ $\qquad\qquad$ $(\exists y)(\forall x)P(x, y)$



Ejercicios de lógica de Primer Orden

Escribir fórmulas de Lógica de Primer Orden que representen las siguientes afirmaciones, usando variables y predicados como sea necesario:

1. "Toda persona tiene derechos"
2. "Toda persona argentina tiene un número de DNI."
3. "Para cada persona hay otra persona que es su madre o su padre"
4. "Para cada persona hay otra persona que es padre de su padre"

Todas las variables x, y, z serán del conjunto de personas

Soluciones

1. $(\forall x)\ \mathit{tieneDerechos}(x)$
2. $(\forall x)\ \mathit{esArgentino}(x) \implies \mathit{tieneDNI}(x)$
3. $(\forall x)\ (\exists y)\ \mathit{esPadre}(y, x)$
   $(\forall x)\ (\exists y)\ y = \mathit{padre}(x)$
4. $(\forall x)\ (\exists y, z)\ \mathit{esPadre}(y, x) \wedge \mathit{esPadre}(z, y)$
   $(\forall p)\ (\exists y, z)\ y = \mathit{padre}(x) \wedge z = \mathit{padre}(y)$

Predicados usados: tieneDerechos, esArgentino, tieneDNI, esPadre

Funciones usadas: padre

Lógica de Primer Orden Tipada

La lógica de Primer Orden básica sólo considera individuos de un único conjunto. Para especificar consas más complejas necesitaremos agregar la posibilidad de manejar múltiples tipos en la misma especificación.

Agregados sintácticos

- especificación de tipos a los cuantificadores: $(\forall x : \mathtt{tipo})$ y $(\exists x : \mathtt{tipo})$
- los predicados y las funciones incluyen el tipo de cada uno de sus parámetros: $P(t_1 : T_1, \ldots, t_n : T_n)$ y $f(t_1 : T_1, \ldots, t_n : T_n)$
- un término y una fórmula estarán bien formados si los tipos de sus variables son coherentes con los tipos esperados por sus fórmulas y funciones involucradas.

Semántica. Análoga a la de la Lógica de Primer Orden, pero con una partición del *dominio de discurso* diferente para cada tipo.

Tipos de datos

En nuestro lenguaje de especificación usaremos un subconjunto de los tipos disponibles en C++:

- `bool`: valores de verdad
- `char`: caracteres `'a'`, `'b'`, `'c'`
- `int`: números enteros con signo
- `float`: números en punto flotante
- `vector<T>`: secuencias de elementos de tipo T.

En el lenguaje de especificación (lógica de primer orden) no existe el concepto de referencia ni de puntero, pues no existe un **modelo de memoria** en el cual podamos hablar de la **ubicación de un valor**.

Esto no quita que hablemos (más adelante) de parámetros pasados por copia y por referencia al definir un contrato.

Tipos de datos: **bool**

Su conjunto de valores son las constantes {`true`, `false`}.

- Funciones: Operadores lógicos `&&` `||` `!`, devuelven `true`, `false`.
- Predicados: Igualdad ($=$ y $\neq$) (fórmulas; devuelven *Verdadero*/*Falso*)

Notar que `true`/`false` no son *Verdadero* / *Falso* de la lógica y no son intercambiables. Es decir, la siguiente fórmula para b : **bool** es incorrecta:

$$b \implies i > 0$$

La manera correcta de escribirlo es comparar b con una constante **bool** lo cual devolverá un valor lógico *Verdadero*/*Falso*:

$$b = \texttt{true} \implies i > 0$$

Tipos de datos: `int`

Su conjunto de valores son los enteros ($\mathbb{Z}$).
Ejemplos: `1`, `-2`, `333`, `0`.

Funciones: Operadores aritméticos

- suma ($c + d$), resta ($c - d$), valor absoluto ($|c|$)
- Producto (*) y división entera(/)
- módulo ($c \bmod b$), potenciación (c^d)

Predicados: operadores relacionales

- menor, menor o igual ($c < d$, $c \leq d$)
- mayor, mayor o igual ($c > d$, $c \geq d$)
- igual, distinto ($=, \neq$)

Tipos de datos: **char**

Su conjunto de valores son las constantes de caracteres definidas por el codigo ASCII:

```
 !"#$%&\'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

Se escriben entre comillas simples; por ejemplo `'a'`, `'b'`, `'c'`.

Funciones

- $\mathit{ord}(c)$ devuelve un **int** con el número de orden de c.
- $\mathit{char}(n)$ devuelve el **char** correspondiente al **int** n.
  Se cumple que $\mathit{char}(\mathit{ord}(c)) = c$ para todo c **char**.

Predicados: operadores relacionales

- El tipo **char** soporta los mismos tipos de comparación que **int**:
  $< \ \leq \ > \ \geq \ = \ \neq$
- El orden relativo entre elementos de tipo **char** es el mostrado arriba; en particular se cumple que `'0'` $<$ `'9'`, `'a'` $<$ `'z'`, y `'A'` $<$ `'Z'`.

Tipos de datos: **float**

Su conjunto de valores son los reales ($\mathbb{R}$).
Se admite escribir cualquier constante real (por ejemplo $7.28, e, \pi$).

Funciones: Operadores aritméticos (*)

- Suma ($c + d$), resta ($c - d$), valor absoluto ($|c|$)
- Producto (*) y división (/)
- Potenciación y logaritmo ($c^d, \log_c d$)
- Parte entera superior e inferior ($\lceil . \rceil$ $\lfloor . \rfloor$); devuelve **int**

Predicados: Fórmulas de comparación (*)

- menor, menor o igual ($c < d$, $c \leq d$)
- mayor, mayor o igual ($c > d$, $c \geq d$)
- igual, distinto ($=, \neq$)

** si alguno de los parámetros es* **int***, se convierte al* **float** *equivalente*.

Tipos de datos: vector<T>

Representa a una secuencia de elementos de tipo T.

Las constantes de tipo `vector<T>` se denotan $\{v_1, v_2, \ldots, v_n\}$ donde v_i es algún valor de tipo T; por ejemplo el `vector<int> {1, -5, 3, 0}`.

> Para `vector<char>` usaremos la sintaxis simplificada
> `"hola mundo"`
> para referirnos al vector
> `{'h','o','l','a',' ','m','u','n','d','o'}`

Funciones

- tamaño de un vector v: $|v|$, equivale a `v.size()` (devuelve **int**)
- acceso a la posición i de un vector v: `v[i]`; sólo definido si $0 \leq i < |v|$, devuelve un elemento de tipo T.

Predicados: comparación

- igual, distinto $(=, \neq)$, que comparan longitud e igualdad elemento a elemento usando el operador $=_T$ (igualdad definida para T).

Expresiones indefinidas

- La semántica estándar de la Lógica de Primer Orden asume que los predicados sólo tomarán un valor de verdad *Verdadero* o *Falso*.
- Al introducir otros tipos de datos al lenguaje, nos encontraremos con que ciertas expresiones van a estar *indefinidas*:

$$[1]\ n > 1 \land (\forall k : \mathbf{int})\ (n \bmod k\ = 0 \implies (k = 1 \lor k = n))$$

- En este caso, para $p = 0$ el predicado se indefine.
- Supongamos que quisiéramos arreglarlo así:

$$[2]\ n > 1 \land (\forall k : \mathbf{int})\ ((k \geq 1\ \land n \bmod k = 0) \implies (k = 1 \lor k = n))$$

- El problema es que, para $k = 0$, el predicado se sigue indefiniendo, con lo que no podemos determinar el valor de verdad total. ¡Pero queremos poder escribir algo así y determinar su valor de verdad!

Entra en escena la lógica trivaluada.

Lógica trivaluada

Para escribir propiedades como [2] de la slide anterior, necesitamos

- introducir en la semántica de la lógica el valor de verdad *indeterminado* (lo notamos $\perp$, *bottom*) y
- extender la semántica de los conectivos lógicos para poder razonar sobre ellos.

x	y	$x \wedge y$
V	V	V
V	F	F
F	V	F
F	F	F
V	$\perp$	$\perp$
F	$\perp$	F
$\perp$	...	$\perp$

Y lógico

x	y	$x \vee y$
V	V	V
V	F	V
F	V	V
F	F	F
V	$\perp$	V
F	$\perp$	$\perp$
$\perp$	...	$\perp$

O lógico

x	y	$x \implies y$
V	V	V
V	F	F
F	V	V
F	F	V
V	$\perp$	$\perp$
F	$\perp$	V
$\perp$	...	$\perp$

Implicación lógica ($\equiv \neg x \vee y$)

Esta nueva semántica permite escribir, con recaudos, expresiones lógicas (como [2]) que no se indefinen si están correctamente escritas.

El nuevo comportamiento se llama "lógica secuencial" y coincide con el de los operadores `&&`, `||`, `and`, `or` del tipo **bool** de C++.

Ejercicios: Lógica de Primer Orden tipada

1. "Hay un número positivo que es divisible solamente por 1 y por sí mismo."
2. "Para cada vector de enteros no vacío, hay un elemento que es el mínimo".
3. "Toda posición impar de todo vector de enteros tiene valores positivos".
4. "Hay un vector de números reales en donde cada posición es el doble de la anterior".

Soluciones:

1. $(\exists p : \texttt{int})\ p > 0 \land (\forall q : int)\ q \neq 0 \land p \bmod q = 0 \implies q = 1 \lor q = p$
2. $(\forall v : \text{vector}<\texttt{int}>)(\exists i : int)\ 0 \leq i < |v| \land (\forall j : int)\ 0 \leq j < |v| \implies v[i] \leq v[j]$
3. $(\forall v : \text{vector}<\texttt{int}>)(\forall i : \texttt{int})\ 0 \leq i < |v| \land i \bmod 2 = 1 \implies v[i] > 0$
4. $(\exists v : \text{vector}<\texttt{int}>)\ (\forall i : \texttt{int})\ 1 \leq i < |v| \implies v[i] = 2 * v[i-1]$

Entendiendo cuantificadores

- $(\forall x : \texttt{int})\ P(x)$

  $$-\infty \ldots \;\; y \;\; P(-2) \;\; y \;\; P(-1) \;\; y \;\; P(0) \;\; y \;\; P(1) \;\; y \;\; P(2) \;\; y \;\; \ldots \infty$$

- Si algún $P(x)$ se indefine entonces toda la fórmula se indefine.
- Si la fórmula no se indefine, alcanza con un único **`int`** n que cumpla $P(n) \equiv \textit{Falso}$ para que toda la fórmula sea *Falsa*.

- $(\exists x : \texttt{int})\ P(x)$

  $$-\infty \ldots \;\; ó \;\; P(-2) \;\; ó \;\; P(-1) \;\; ó \;\; P(0) \;\; ó \;\; P(1) \;\; ó \;\; P(2) \;\; ó \;\; \ldots \infty$$

- Si algún $P(x)$ se indefine entonces toda la fórmula se indefine.
- Si la fórmula no se indefine, alcanza con un único **`int`** n que cumpla $P(n) \equiv \textit{Verdadero}$ para que toda la fórmula sea *Verdadera*.

Entendiendo cuantificadores: $\forall$

- $(\forall i : \texttt{int})\ (0 \leq i < |v| \implies v[i] = 2)$
  - Significa que **todas** las siguientes fórmulas tienen que ser Verdaderas:

$$-\infty$$

$$\vdots$$

$$(0 \leq -2 < |v| \mathit{Falso} \implies v[-2] = 2) \mathit{Verdadero}$$
$$(0 \leq -1 < |v| \mathit{Falso} \implies v[-1] = 2) \mathit{Verdadero}$$
$$(0 \leq 0 < |v| \mathit{Verdadero} \implies v[0] = 2) v[0] = 2$$
$$(0 \leq 1 < |v| \mathit{Verdadero} \implies v[1] = 2) v[1] = 2$$

$$\vdots$$

$$(0 \leq |v| - 2 < |v| \mathit{Verdadero} \implies v[|v| - 2] = 2) v[|v| - 2] = 2$$
$$(0 \leq |v| - 1 < |v| \mathit{Verdadero} \implies v[|v| - 1] = 2) v[|v| - 1] = 2$$
$$(0 \leq |v| < |v| \mathit{Falso} \implies v[|v|] = 2) \mathit{Verdadero}$$
$$(0 \leq |v| + 1 < |v| \mathit{Falso} \implies v[|v| + 1] = 2) \mathit{Verdadero}$$

$$\vdots$$

$$\infty$$

Entendiendo cuantificadores: $\exists$

- $(\exists i : \textbf{int})\ (0 \leq i < |v| \ \land \ v[i] = 2)$
  - Significa que **alguna** de las sigs. fórmulas tiene que ser Verdadera:

$-\infty$

$\vdots$

$(0 \leq -2 < |v| \mathit{Falso} \ \land \ v[-2] = 2) \mathit{Falso}$

$(0 \leq -1 < |v| \mathit{Falso} \ \land \ v[-1] = 2) \mathit{Falso}$

$(0 \leq 0 < |v| \mathit{Verdadero} \ \land \ v[0] = 2) v[0] = 2$

$(0 \leq 1 < |v| \mathit{Verdadero} \ \land \ v[1] = 2) v[1] = 2$

$\vdots$

$(0 \leq |v| - 2 < |v| \mathit{Verdadero} \ \land \ v[|v| - 2] = 2) v[|v| - 2] = 2$

$(0 \leq |v| - 1 < |v| \mathit{Verdadero} \ \land \ v[|v| - 1] = 2) v[|v| - 1] = 2$

$(0 \leq |v| < |v| \mathit{Falso} \ \land \ v[|v|] = 2) \mathit{Falso}$

$(0 \leq |v| + 1 < |v| \mathit{Falso} \ \land \ v[|v| + 1] = 2) \mathit{Falso}$

$\vdots$

∞

Sintaxis: predicados y funciones auxiliares

Para modularizar especificaciones es útil descomponer un predicado complejo en múltiples **predicados auxiliares** y reemplazar algunas expresiones usadas frecuentemente por **funciones auxiliares**.

$$\mathit{nombre\_predicado}(x_1 : T_1, \ldots, x_n : T_n) \equiv$$

...fórmula correctamente tipada donde aparecen libres las variables $x_1, \ldots, x_n$ ...

$$\mathit{nombre\_funcion}(x_1 : T_1, \ldots, x_n : T_n) : T \equiv$$

...expresión correctamente tipada de tipo T donde aparecen libres las variables $x_1, \ldots, x_n$ ...

Tanto predicados como funciones auxiliares funcionan como **reemplazos sintácticos** y como tales **no soportan el uso de ningún tipo de recursión directa o indirecta.**

Ejemplo: descomponiendo un problema complejo

Conjetura de Goldbach: Todo número par mayor que 2 se puede escribir como la suma de dos números primos

$$(\forall n : \texttt{int})\ ((n > 2 \land n \bmod 2 = 0) \implies \mathit{esSumaDeDosPrimos}(n))$$

El entero n se puede escribir como la suma de dos números primos

$$\mathit{esSumaDeDosPrimos}(n : \texttt{int}) \equiv$$

$$(\exists p, q : \texttt{int})\ (\mathit{esPrimo}(p) \land \mathit{esPrimo}(q) \land n = p + q)$$

El entero n es primo

$$\mathit{esPrimo}(n : \texttt{int}) \equiv$$

$$n > 1 \land (\forall k : \texttt{int})\ (k \geq 1 \land n \bmod k = 0 \implies k = 1 \lor k = n)$$

Otras expresiones: $\sum, \prod, \beta$

- $\sum_{i=N}^{M} E(i)$ (sumatoria) sobre índices i de tipo **`int`**.
  - E es alguna expresión de i y es de tipo **`int`** o **`float`**
  - su valor será del mismo tipo que E;
  - se indefine si alguno de los términos de la sumatoria es indefinido.
  - Su valor es 0 si el rango de iteración es vacío ($N > M$)
- $\prod_{i=N}^{M} E(i)$ (productoria); mismas reglas que la sumatoria.
  - Su valor es 1 si el rango de iteración es vacío ($N > M$)
- $\beta(G)$: (de tipo **`int`**) vale 1 si la fórmula G es *Verdadera* y 0 si G es *Falsa*; notar que su valor será *Indefinido* si G se indefine.

$$\sum_{i=1}^{N} \beta(i \bmod 2 = 0)$$

Cuenta cuántos números entre 1 y N son pares

$$1 - \prod_{i=N}^{M} \beta(\neg esPrimo(i))$$

Vale 1 si algún número entre N y M es primo

Otras expresiones: $if - then - else - fi$

if cond then E_1 *else* E_2 *fi*: equivale al operador ternario de C++:

- E_1 y E_2 son expresiones **de un mismo tipo** T.
  ¡No se puede obviar el *else* E_2**!**
- la expresión vale E_1 si $cond = Verdadero$, E_2 si $cond = Falso$ o *Indefinido* si la *cond* se indefine;
- si la expresión elegida (E_1 o E_2) se indefine el valor final también será indefinido.
- No confundir su funcionamiento con el **`if`** de C++: este último no es una expresión (no tiene un valor asociado) y tiene **`else`** opcional.

$$\sum_{i=N}^{M} \text{if } esPrimo(i) \text{ then } i^2 \text{ else } 0 \text{ fi}$$

Calcula la sumatoria de los cuadrados de los primos entre N y M

Predicados y funciones auxiliares: Algunas recomendaciones

- Descomponer un predicado complejo en múltiples predicados/funciones más simples que tengan un significado claro y un nombre declarativo. Suele ser útil trabajar "de arriba hacia abajo" descomponiendo progresivamente el problema.
  Ver ejemplo de Goldbach [slide **??**]
- Usar el cuantificador Existencial ($\exists$) para "hacer aparecer" un valor que nos resulte útil. El patrón general es
  $(\exists x : T)\ \mathit{cumplePropiedadesUtiles}(x) \land \ldots \mathit{uso}\ x \ldots$.
  Suele ser útil especificar un x que sea un valor intermedio que nos facilite especificar, por ejemplo, el resultado esperado en una postcondición.

Repaso de la clase

Hoy vimos

- Un repaso del concepto de especificar;
- Lógica de primer orden tipada y trivaluada;
- Uso de predicados y funciones auxiliares;
- Otras expresiones útiles.

Guía de ejercicios

- Ya pueden comenzar la Guía 2.

Bibliografía:

- Para lógica proposicional: **Gries**, capítulo 1.
  - En **Gries**, capítulo 2, van a encontrar reglas de equivalencia para lógica proposicional.
- Para lógica de primer orden y lógica trivaluada: **Gries**, capítulo 4.