Algoritmos y Estructuras de Datos I

Gianfranco Zamboni

31 de enero de 2019

Índice

ESTE RESUMEN NO ESTÁ TERMINADO. LA MATERIA CAMBIO TODA DESDE QUE LA CURSE, HAY COSAS EN LAS DIAPOS QUE NO TENGO IDEA QUE SON Y, LA VERDAD, PAJA LEERLOS

Parte I

Especificación de programas utilizando contratos

1. Lógica propocisional

La lógica propocisional es una **herramienta usada para escribir expresiones**/fórmulas que pueden representar dos posibles valores: Verdadero (V) ó Falso (F), a los que llamaremos valores de verdad.

Sintaxis

Las **fórmulas son secuencias de símbolos ordenadas** de manera tal **que respeten ciertas reglas**. El conjunto de símbolos y de variables que se pueden usar en las fórmulas junto con estas reglas forman la sintaxis del lenguaje (en este caso el lenguaje de la Lógica Proposicional).

- Símbolos: True, False, $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, (,)
- Variables proposicionales (son infinitas): Serán secuencias de una o más letras: p, q, r, ss,...
- Reglas:
  - True y False son fórmulas.
  - Cualquier variable proposicional es una fórmula.
  - Si A es una fórmula entonces $\neg A$ es una fórmula.
  - Si $A_1, A_2, \ldots, A_n$ son fórmulas entonces $(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ es una fórmula.
  - Si $A_1, A_2, \ldots, A_n$ son fórmulas entonces $(A_1 \vee A_2 \vee \cdots \vee A_n)$ es una fórmula.
  - Si A y B son fórmulas entonces $(A \rightarrow B)$ es una fórmula.
  - Si A y B son fórmulas entonces $(A \leftrightarrow B)$ es una fórmula.

Si una secuencia de símbolos cumple con alguna de las reglas mencionadas, entonces diremos que es una **fórmula bien formada**.

Semántica clásica

La semántica de un lenguaje, nos **permite asignarle un significado (valor) a las fórmulas bien formadas**. Como se mencionó al comienzo de esta sección, en la lógica proposicional, una fórmula podrá representar los valores "verdadero"(V) y "falso"(F).

Dada una fórmula, se podrá calcular su valor de verdad usando las siguientes reglas:

- True es verdadero
- False es falso
- Las variables proposicionales podrán ser verdaderas o falsas dependiendo del contexto en el que se las use y que cosa representen, por ejemplo, si p es el valor de verdad de la propocisión "está lloviendo" y, efectivamente, está lloviendo entonces p es verdadero. Pero si p significa esto y no está lloviendo, entonces p es falso.

Sean A y B dos fórmulas con un valor de verdad conocido, entonces:

- $\neg A$ será la negación de A, es decir, si A es verdadero, entonces $\neg A$ y viceversa.
- $A \wedge B$ será la conjunción entre A y B y será verdadero si y solo si ambas fórmulas (A y B) son verdaderas.
- $A \vee B$ será la disjunción entre A y B y será verdadero si alguna de las fórmulas lo es.
- $A \rightarrow B$ define una relación entre ambas fórmulas. Se lee "A implica B" y será falso si y solo si A es verdadero y B es falso.
- $A \leftrightarrow B$ define una relación entre ambas fórmulas. Se lee "A si y solo si B" y será verdadero si y solo si A y B tienen el mismo valor de verdad.

p	q	$\neg p$	$(p \wedge q)$	$(p \vee q)$	$(p \rightarrow q)$	$(p \leftrightarrow q)$
V	V	F	V	V	V	V
V	F	V	F	V	F	F
F	V	-	F	V	V	F
F	F	-	F	F	V	V

Cuadro 1: Tabla de verdad para cada uno de los símbolos descriptos

1.1. Tautologias, contradiciones y contigencias

Diremos que una fórmula es una **tautologíasi siempre toma el valor** V para valores distintos de sus variables propocisionales. Por ejemplo, $((p \wedge q) \rightarrow p)$ es una tautología, esto se puede mostrar usando una tabla de verdad:

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

Cuadro 2: Tabla de verdad de $((p \wedge q) \rightarrow p)$

Una **contradicción será una fórmula que siempre toma el valor** F para valores definidos de sus variables proposicionales.

p	$\neg p$	$(p \wedge \neg q)$
V	F	F
F	V	F

Cuadro 3: $(p \wedge \neg p)$ es una contradicción

Al resto de las fórmulas (las que nos son ni tautologías ni contradicciones) las llamaremos **contigencias**.

Equivalencias entre fórmulas

Dadas dos preposiciones p y q, las siguientes fórmulas son tautologías:

- Idempotencia
  $(p \wedge p) \leftrightarrow p$
  $(p \vee p) \leftrightarrow p$
- Asociatividad
  $(p \wedge q) \wedge r \leftrightarrow p \wedge (q \wedge r)$
  $(p \vee q) \vee r \leftrightarrow p \vee (q \vee r)$
- Conmutatividad
  $(p \wedge q) \leftrightarrow (q \wedge p)$
  $(p \vee q) \leftrightarrow (q \vee p)$
- Distributividad
  $p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$
  $p \vee (q \wedge r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$
- Reglas de Morgan
  $\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$
  $\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$

Que estas fórmulas sean tautologias significa que si encontramos alguno de sus lados en una fórmula, entonces podremos remplazarlo por el otro sin cambiar el valor de la misma.

Relaciones de fuerza

Decimos que A **es más fuerte que**B **cuando** $(A \to B)$ **es una tautología** . Tambien decimos que A **fuerza a** B o que B **es más debil que** A.

Por ejemplo:

- $(p \wedge q)$ es más fuerte que p. Podemos ver en la tabla 1.1 que, efectivamente, $((p \wedge q) \to p)$ es una tautología.
- False es la fórmula más fuerte ya que, si A es una fórmula, entonces $(\mathsf{False} \to A)$ siempre es verdadera, sin importar que fórmula es A.
- True es la más debil, es decir que $(A \to \mathsf{True})$ es una tatutología para toda fórmula A.

1.2. Semántica trivaluada secuencial

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 describir este estado.

Para esto, introducimos el valor de verdad *Indefinido* ($\bot$) y una nueva de forma de evaluar las expresiones: La evaluación **secuencial**.

En esta nueva lógica, los términos se evalúan de izquierda a derecha y la evaluación termina cuando se puede deducir el valor de verdad, aunque el resto esté indefinido.

p	q	$(p \wedge_L q)$	$(p \vee_L q)$	$(p \longrightarrow_L q)$
V	V	V	V	V
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V
T	$\bot$	$\bot$	V	$\bot$
F	$\bot$	F	V	V
$\bot$	V	$\bot$	$\bot$	$\bot$
$\bot$	F	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Cuadro 4: Tablas de verdad para $\wedge_L$, $\vee_L$ y $\longrightarrow_L$

2. Tipos de datos

Un **tipo de datos** es un **conjunto de valores** 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** que pueden ser:

- una variable de tipo T,
- una constante de tipo T,
- o una función (operación) aplicada a otros términos (del tipo T o de otro tipo).

Todos los tipos que definamos van a tener un elemento distinguido que indicará la falta de un valor (indefinición) de una variable y lo notaremos: $\bot$ ó Indef. A continuación se da una pequeña introducción los tipos básicos que vamos a usar durante el curso, para verlos más en detalles ver el apendice A.

2.1. Tipos básicos

- Booleanos (Bool): True, False
- Enteros ($\mathbb{Z}$): $-2, -1, 0, 1, 5$, etc
- Reales ($\mathbb{R}$): $0.0025,\ 1.34,$ etc
- Carácteres (Char): 'a', 'b', 'c', '4', etc.

2.1.1. Tipos enumerados

Los tipos enumerados son tipos con una cantidad finita de elementos denotados por una constante.

$$\mathsf{enum}\ \textbf{\textit{Nombre}}\ \{\ \textit{constantes}\ \}$$

Estos tipos están compuestos por un *nombre* y una lista de *constantes* (en mayúsculas) separadas por comas.

Ejemplo

$$\mathsf{enum}\ \textbf{Día}\ \{\ \text{LUN, MAR, MIER, JUE, VIE, SAB, DOM}\ \}$$

2.1.2. Tuplas

Son uplas de uno o más elementos, cada una de cualquier tipo. El tipo de una upla es $T_0 \times T_1 \times \cdots \times T_k$ donde T_i $(i = 0 \ldots k)$ corresponde al tipo del elemento en la i-ésima posición de la upla. Proporcionan la operación básica enésimo $(\ \bullet\ )_n$ que nos permite acceder a lá enésima componente de la tupla.

$$(1, \mathsf{True}, 1.5)$$

2.2. Cuantificadores

El lenguaje usado en la cátedra, además nos provee formas de predicar sobre los elementos de un tipo de datos.

- $(\forall x : T)P(x)$ es un término de tipo Bool que afirma que todos los elementos de tipo T cumplen la propiedad P y se lee ''Para todo x de tipo T se cumple $P(x)$", es decir es equivalente a la siguiente f.

  Este cuantificador se llama **universal** y *generaliza la conjunción*, es decir que nos permite de escribir de manera concisa una concatenacion de conjunciones $(\wedge)$.

- $(\exists x : T)P(x)$ es un término de tipo Bool que afirma que al menos un elemento de tipo T cumplen la propiedad P y se lee ''Existe al menos un x de tipo T que cumple $P(x)$".

  Este cuantificador se llama **existencial** y *generaliza la disyunción* $(\vee)$.

Como el uso de cuantificadores en una expresión devuelve un valor booleano, se les puede aplicar operadores lógicos. En el caso de la negación, el $\forall$ se convierte en un $\exists$ y viceversa.

Variables libres y ligadas: Veamos la siguiente expresión $y \wedge (\forall x : \mathbb{Z})P(x)$. En está expresión, x está **ligada** al cuantificador e y se encuentra **libre**, es decir no hay ningún cuantificador que haga referencia a ella.

2.3. Secuencias

Una secuencia son varios **elementos del mismo tipo** T, posiblemente repetidos, ubicados en un cierto orden y diremos que son de tipo $seq\langle T\rangle$ y T puede ser cualquier tipo, incluso otra secuencia.

$$[1, 2, 3, 4, 5, 7] \text{ es una secuencia de } \mathbb{Z}$$

Además, en todos los tipos de secuencias existe una secuencia especial, la **secuencia vacía**, que notaremos [ ].

Además, hay secuencias de ciertos tipos que tienen nombres especiales y que podremos usar en vez del tipo explicito, por ejemplo, las secuencias de carácteres son **Strings** y las sequencias de sequencias son **matrices**.

$$\mathsf{String} = seq\langle \mathsf{Char}\rangle$$
$$\mathsf{Mat}\langle T\rangle = seq\langle seq\langle T\rangle\rangle$$

2.3.1. Sumatoria

El lenguaje de espicifación provee formas de acumular resultados para los tipos númericos $\mathbb{Z}$ y $\mathbb{R}$. La sumatoria retorna la suma de todas las expresiones númericas $Expr(i)$ entre $from$ y to:

$$\sum_{i=from}^{to} Expr(i) = Expr(from) + Expr(from + 1) + \cdots + Expr(to - 1) + Expr(to)$$

$from$ y to forman un rango finito de valores enteros de tal forma que $from < to$. Si esto no se cumple, entonces la sumatoria devuelve cero. Ademas, si alguno de los valores que se encuentra en este rango hace que la expresión se indefina, entonces toda la sumatoria se indefine.

En el siguiente ejemplo, se suman todos los valores que se encuentran en las posiciones pares de una secuencia:

$$\sum_{i=0}^{|s|-1} (\mathsf{if}\ i\ \mathsf{mod}\ 2 = 0\ \mathsf{then}\ s[i]\ \mathsf{else}\ 0\ \mathsf{fi})$$

Muchas veces, querremos usar esta operación para contar elementos que cumplan alguna propiedad, en estos, podremos hacer el siguiente abuso de notación:

$$\#\{i \in [from, to] : P(i)\} = \sum_{i=from}^{to} (\mathsf{if}\ P(i)\ \mathsf{then}\ 1\ \mathsf{else}\ 0\ \mathsf{fi})$$

2.3.2. Productoria

El término productoria retorna el producto de todas las expresiones $Expr(i)$ entre $from$ y to:

$$\prod_{i=from}^{to} Expr(i) = Expr(from) * Expr(from + 1) * \cdots * Expr(to - 1) * Expr(to)$$

A diferencia de la sumatoria, si $from \leq to$ no se cumple entonces devuelve 1. Si alguna de las expresiones se indefine, entonces tambien lo hace toda la productoria.

3. Especificación

Una **especificación** es una descripción de un problema en un lenguaje formal. Es un contrato que nos proporciona las propiedades que deben cumplir los datos de entrada del problema y las propiedades que debe cumplir su solución. Y, a partir de ella, podremos diseñar **algoritmos** o **programas** que describirán la (o una) solución del problema y como obtenerla.

Si bien **especificación** y **algoritmo** tienen definiciones muy parecidas, una especificación nos dice **qué** queremos resolver sin decirnos como hacerlo mientras que el algoritmo nos da un método que nos indica **como** obtener la solución que buscamos.

Por ejemplo, supongamos que queremos multiplicar dos números. Entonces una especificación podría ser $c = a * b$ que, en lenguaje matemático nos indica que c es el número que va dar como resultado la multiplicación de a y b.

Un algoritmo para realizar esta multiplicación podría ser el que enseñan en las secundarias, o el usado por los mayas, o el de los egipcios. Hay infinitos algoritmos capaces de resolver correctamente este problema.

La especificación, además, nos permite verificar formalmente la correctitud de un algoritmo y pensar mejores casos de test para las aplicaciones que programemos.

Nota: Otra pequeña diferencia a tener en cuenta, un algoritmo y un programa **tampoco son lo mismo**. Un algoritmo es una secuencia de pasos que describe como resolver un problema, esta secuencia puede estar escrita en cualquier lenguaje (ingles, español, chino mandarin, C++, Python, etc), los programas son algoritmos escritos especificamente en un lenguaje entendible por la computadora.

3.1. Partes de la especificación

En algoritmos 1, una especificación tiene la siguiente forma:

```
proc nombre (parámetros) = res : T {
    Pre { P }
    Post { Q }
}
```

donde *nombre* es el nombre que le damos al problema, P y Q son dos predicados denominados la **precondición** y la **postcondición** del procedimiento y *parámetros* es una lista de parámetros separada por comas.

Contrato: El programador escribe un programa P tal que el usuario suministra datos que hacen verdadera la precondición, entonces P termina en una cantidad finita de pasos retornando un valor que hace verdadera la postcondición.

Diremos que P es **correcto** para la especificación dada por esa precondición y esa postcondición cuando se cumpla el contrato.

En otras palabras, la especificación nos da las restricciones que las variables de entrada deben respetar para gantarizar una correcta solución al problema y, la poscondición, es la condición que debe cumplir el resultado de un algoritmo para que sea considerada una solución correcta.

Cuando el usuario no cumpla con la precondición establecida por la espificación el programa fallará. En este caso, es el usuario el que está faltando a su parte del contrato. En cambio, si el usuario respeta la precondición y el programa falla entonces P no es correcto y sus resultados no pueden ser considerados soluciones al problema propuesto.

3.1.1. Parámetros

Cada parámetro del problema tiene la siguiente forma:

$$tipo\_pasaje\ nombre : tipo\_dato$$

Tipos de pasaje

- **Parámetros de entrada (in)**: Se utiliza una copia del parámetro durante la ejecución, quedando el valor original intacto.
- **Parámetros de salida (out)**: Son parámetros que no se inicializan (no contienen ningún valor al principio de la ejecución). Recién al finalizar, se copia un valor al mismo.
- **Parámetros de entrada-salida (inout)**: Son parámetros que funcionan como de entrada (se copia el valor del argumento al inicio) y como de salida (se copia el valor de la viarble al final) simultaneamente. El efecto final es que la ejecución del procedimiento **modifica** el valor del parámetro.

Cuando usamos parámetros de entrada-salida, tendremos **metavariables** que nos permitirán indicar los valores de ese parámetro en distintos puntos de la ejecución. En el siguiente ejemplo, usaremos la metavariable a_0 para referirnos al valor inicial del parámetro a:

```
proc incrementarEnUno (inout a : ℤ) {
    Pre {a_0 = a}
    Post {a = a_0 + 1}
}
```

3.1.2. Errores comunes

Al momento de escribir una especificación podemos cometer dos tipos de errores que afectarán de una manera u otra al contrato:

- **Sobre-especificación:** Ocurre cuando damos una **postcondición más restrictiva** de lo que se necesita, o bien una **precondición mas laxa**. Limita los posibles algoritmos que resuelven el problema porque impone más condiciones para la salida, o amplía los datos de entrada.
- **Sub-especificación:** Ocurre cuando damos una **precondición más restrictiva** de lo que se necesita, o bien una **postcondición mas debil**. Deja afuera datos de entrada o ignora condiciones necesarias para la salida (es decir, permite soluciones no deseadas).

3.2. Funciones y predicados auxiliares

Las funciones y los predicados auxiliares son expresiones del lenguaje a las que se les asigna un nombre. Permiten **modularizar** la escritura de las especificaciones y facilita su lectura. Tienen la siguiente forma:

$$\mathsf{fun}\ f(argumentos) : tipo = expresion$$
$$\texttt{pred p}\ (\text{argumentos})\ \{formula\}$$

Donde f y p son los nombres asignados que podrán usarse en el resto de le especificación en lugar de la *expresion/formula*. Los argumentos son opcionales y se reemplazan en la *expresion/formula* cada vez que se usa f/p.

Se debe tener en cuenta que las funciones auxiliares son solo un **remplazo sintáctico** dentro de una especificación, por lo que no se permiten definiciones recursivas. Además, no se permite usar una especificación dentro de otra, ya que las mismas no pertenecen la lógica de primer orden.

Para que quede más claro, veamos dos ejemplos, el primero es una función auxiliar que nos permite sumar dos números reales. El segundo un predicado que devuelve true si y solo el entero pasado por parámetro es primo:

$$\mathsf{fun}\ suma(x, y : \mathbb{R}) : \mathbb{R} = x + y$$
$$\texttt{pred esPrimo}\ (\text{x} : \mathbb{Z})\ \{(\forall\ y : \mathbb{Z})\ (1 < y < x)\ \wedge\ (x\ \mathsf{mod}\ y \neq 0)\}$$

3.2.1. Expresiones condicionales

Las expresiones condicionales, son expresiones cuyo valor depende del valor de verdad de una formula lógica (*guarda*). Un ejemplo, es la función auxiliar IfThenELse que toma como argumentos un elemento e de tipo Bool y dos valores s y r de tipo T. Si e es True, entonces, esta función, retorna s, si no, retorna r. Se podrá usar esta función de la siguientes formas:

$$a = \mathsf{IfThenElse} < T > (exp, s, r)$$

$$a = \mathsf{if}\ exp\ \mathsf{then}\ s\ \mathsf{else}\ r\ \mathsf{fi}$$

3.3. Funciones auxiliares imprescindibles

Cantidad de apariciones: Cuenta la cantidad de aparciiones de un elemento e en la secuencia s.
$\texttt{aux \#apariciones}\ (\text{s: } seq\langle T\rangle, \text{e: } T) : \mathbb{Z} = \sum_{i=0}^{|s|-1}(\mathsf{if}\ s[i] = 0\ \mathsf{then}\ 1\ \mathsf{else}\ 0\ \mathsf{fi})$;

Es permutación: Devuelve verdadero si y solo si una secuencia es permutación de otra.
$\texttt{pred es\_permutacion}\ (\text{s1, s2} : seq\langle T\rangle)\ \{(\forall x\ :\ T)\}$

3.4. Notas sobre la especificación de problemas

Nombres de predicados y funciones: Deben ser lo más declarativos posibles, es decir, deben describir el significado de lo que devuelven, ya que la idea es que nos permitan realizar especificaciones más concisas.

Comentarios: Muchas veces, puede resultar dificil separar los predicados de las postcondicion o precondiciones en funciones o predicados auxiliares. En estos casos, se pueden agregar comentarios en castellano que aclaren lo que indica cada parte del predicado.

```
proc menorElemDistinos (in s : seq⟨ℤ⟩, out result : ℤ) {
    Pre {noNegativos(s) ∧ noHayRepetidos(s)}
    Post {
          /* result es un indice válido de s */
          0 ≤ result < |s| ∧_L
          /* s[result] es el menor elemento de s */
          (∀i : ℤ((0 ≤ i < |s|) ⟶_L s[result] ≤ s[i]))
          }
}
```

4. Temario

- Conceptos básicos de los programas imperativos
- Estructuras de control
  - Variables
- Ciclos
  - Construcción
  - Terminación
  - Corrección
- Corrección de programas
  - Tipos
  - Tratamientos de secuencias
  - Fundamentos de ordenamiento
  - Fundamentos de testing

Apéndices

A. Tipos de Datos y sus funciones

A.1. Enteros

Conjunto base: Números enteros

Constantes: 0; 1; -1; 2; -2; ...

Operaciones aritméticas:

Operación	Descripción	Entrada	Salida
$a + b$	Suma	$a, b \in \mathbb{Z}$	$c \in \mathbb{Z}b$
$a - b$	Resta	$a, b \in \mathbb{Z}$	$c \in \mathbb{Z}$
$a * b$	Multiplicación	$a, b \in \mathbb{Z}$	$c \in \mathbb{Z}$
$\mathsf{abs}(a)$	Valor Absoluto	$a \in \mathbb{Z}$	$c \in \mathbb{Z}$
a div b	División entera	$a, b \in \mathbb{Z}$	$c \in \mathbb{Z} \ : \ a = c * b + r$ con $r \in \mathbb{Z} \wedge 0 < r < b$
a mod b	Resto de dividir a por b	$a, b \in \mathbb{Z}$	$r \in \mathbb{Z} \ : \ 0 < r < b \ \wedge \ a = c * b + r$ con $c \in \mathbb{Z}_{>0}$
a/b	División	$a, b \in \mathbb{Z}$	$c \in \mathbb{R} \ : \ a = c * b$

Formulas de comparación:

Formula	Descripción
$a < b$	Menor
$a \leq b$	Menor o igual
$a > b$	Mayor
$a \geq b$	Mayor o igual
$a = b$	Igual
$a \neq b$	Distinto

A.2. Reales

Conjunto base: Números reales

Constantes: 0; 1; -7; 81.455; π; ...

Operaciones aritméticas:

Operación	Descripción	Entrada	Salida
$a + b$	Suma	$a, b \in \mathbb{R}$	$c \in \mathbb{R}$
$a - b$	Resta	$a, b \in \mathbb{R}$	$c \in \mathbb{R}$
$a * b$	Multiplicación	$a, b \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{abs}(a)$	Valor Absoluto	$a \in \mathbb{R}$	$c \in \mathbb{R}$
a/b	División	$a, b \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{log}_b(a)$	Logaritmo de a en base b	$a, b \in \mathbb{R}$	$c \in \mathbb{R} \ : \ a = b^c$
$\mathsf{sin}(b)$	Seno	$a \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{cos}(b)$	Coseno	$a \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{tan}(b)$	Tangente	$a \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{cot}(b)$	Cotangente	$a \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{sec}(b)$	Secante	$a \in \mathbb{R}$	$c \in \mathbb{R}$
$\mathsf{csec}(b)$	Cosecante	$a \in \mathbb{R}$	$c \in \mathbb{R}$

Formulas de comparación:

Formula	Descripción
$a < b$	Menor
$a \leq b$	Menor o igual
$a > b$	Mayor
$a \geq b$	Mayor o igual
$a = b$	Igual
$a \neq b$	Distinto

A.3. Booleanos

Conjunto base: {**true**, **false**}

Constantes: `true`, `false`

Operaciones (Conectivos lógicos)

Operación	Descripción	Entrada	Salida
$!a$	Negación	$a \in \mathsf{Bool}$	$c \in \mathsf{Bool} \ : \ c = \neg a$
$a \ \&\& \ b$	Conjunción	$a, b \in \mathsf{Bool}$	$c \in \mathsf{Bool} \ : \ c = a \wedge b$
$a \ \|\| \ b$	Disjunción	$a, b \in \mathsf{Bool}$	$c \in \mathsf{Bool} \ : \ c = a \vee b$

Formulas de comparación:

Formula	Descripción
$a = b$	Igual
$a \neq b$	Distinto

A.4. Carácteres

Conjunto base: Letras, dígitos y símbolos.

Constantes: 'a', 'b', 'c',...,'z','A','B','C',..., 'Z',...,'0','1',...,'9' (en el orden dado por el estándar ASCII)

Operaciones:

Operación	Descripción	Entrada	Salida
$\mathsf{ord}(a)$	Numero los carácteres según el orden dado por el estándar ASCII	$a \in \mathsf{Char}$	$c \in \mathbb{Z}_{\geq 0}$
$\mathsf{char}(a)$	Inversa de $\mathsf{ord}()$	$a \in \mathbb{Z}_{\geq 0}$	$c \in \mathsf{Char} \ : \ a = \mathsf{ord}(c)$

Formulas de comparación:

La comparación entre caracteres son comparaciones entre sus órdenes, de modo que $a \bullet b$ es equivalente a $\mathsf{ord}(a) \bullet \mathsf{ord}(b)$.

Formula	Descripción
$a < b$	Menor
$a \leq b$	Menor o igual
$a > b$	Mayor
$a \geq b$	Mayor o igual
$a = b$	Igual
$a \neq b$	Distinto

A.5. Tipos enumerados

Conjunto base: Conjuntos finitos de constantes

Forma:

$$\mathsf{enum}\ \textbf{\textit{Nombre}}\ \{\ \textit{constantes}\ \}$$

- *Nombre*: con el que se va a designar al tipo del conjunto.
- *constantes*: Lista de elementos del conjunto separados por coma.

Operaciones:

Operación	Descripción	Entrada	Salida
$\mathsf{ord}(a)$	Da la posición del elemento en la definición del tipo	$a \in \mathsf{Char}$	$c \in \mathbb{Z}_{\geq 0}$
$\texttt{nombre}(a)$	Inversa de $\mathsf{ord}()$, "`nombre`"debe ser remplazado por el nombre del tipo	$a \in \mathbb{Z}_{\leq 0}$	$c \in \mathsf{Char} \ : \ a = \mathsf{ord}(c)$

Formulas de comparación:

La comparación entre caracteres son comparaciones entre sus órdenes, de modo que $a \bullet b$ es equivalente a $\mathsf{ord}(a) \bullet \mathsf{ord}(b)$.

Formula	Descripción
$a < b$	Menor
$a \leq b$	Menor o igual
$a > b$	Mayor
$a \geq b$	Mayor o igual
$a = b$	Igual
$a \neq b$	Distinto

A.6. Tuplas (Conjunto de tipos)

Conjunto base: Conjuntos finitos ordenados de elementos.

Tipos: $T_1 \times \cdots \times T_n$.

Forma:

$$(e_1, \ldots, e_n)$$

- e_i: Elemento de tipo T_i.

Operaciones:

Operación	Descripción	Entrada	Salida
$(e_1, \ldots, e_n)_n$	nésimo, es el elemento que se encuentra en la posición	$c \in \mathbb{Z}_{\geq 0}$	e_n

A.7. Secuencia

Conjunto base: Elementos de un tipo T, posiblemente repetidos, ordenados de una manera específica.

Tipos: $seq\langle T \rangle$.

Forma:

$$[e_1, \ldots, e_n]$$

- e_i: Elemento de tipo T.

Operaciones:

Operación	Descripción	Entrada	Salida	Precondición
$\|a\|$	Longitud de la secuencia a	$a \in seq\langle T \rangle$	$n \in \mathbb{Z}$	
$a[i]$	Elemento en la i-esima posición de a	$a \in seq\langle T \rangle,\ i \in \mathbb{Z}$	$e_i \in T$	
$\mathsf{head}(a)$	Primer elemento de la sequencia	$a \in seq\langle T \rangle$	$e \in T$	$\|a\| > 0$
$\mathsf{tail}(a)$	Elimina el primer elemento	$a \in seq\langle T \rangle$	$b \in seq\langle T \rangle$	$\|a\| > 0$
$\mathsf{addFirst}(t, a)$	Agrega un elemento al principio	$t \in T,\ a \in seq\langle T \rangle$	$b \in seq\langle T \rangle$	
$a + +b$	Secuencia con los elementos de a seguidos de b	$a, b \in seq\langle T \rangle$	$b \in seq\langle T \rangle$	
$\mathsf{subseq}(a, d, h)$	Elementos de la secuencia a entre las posiciones d y h	$a \in seq\langle T \rangle, d, h \in \mathbb{Z}$	$b \in seq\langle T \rangle$	$0 \leq d \leq h \leq \|a\|$
$\mathsf{setAt}(a, i, val)$	La misma secuencia pero con el valor val en la posición i	$a \in seq\langle T \rangle, d \in \mathbb{Z}, h \in \mathbb{Z}$	$b \in seq\langle T \rangle$	

Formulas de comparación:

La comparación entre secuencias involucra la comparación de sus elementos uno a uno. Si los elementos de dos secuencias son los mismos y se encuentran en el mimos orden, entonces son iguales.

Formula	Descripción
$a = b$	Igual