UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales

TESIS DE LICENCIATURA

# La equivalencia entre FO(IFP) y la clase B

**Cynthia Roxana Disenfeld**

Director de tesis: Rafael Grimson

Co-director: Guillermo Martínez

Buenos Aires, 2007

# Agradecimientos

A Rafael Grimson y Guillermo Martínez por toda su ayuda en desarrollar esta tesis.

A los profesores y docentes de la facultad por todos los temas que aprendí.

A Federico, José, Silvana, Alexis, Leandro, Jonathan, Pablo y otros amigos de la facultad por todos los trabajos prácticos y materias compartidas.

A Gisela, Romina, Sandra, Malena, Patricia, Pedro, Laura, Julieta, Diego por su amistad, y todas las conversaciones y momentos compartidos.

Por último y principalmente quiero agradecer a mi hermana Jesi, y a mis padres Héctor y Susana, por su gran ayuda y compañía incondicional.

# Resumen

Desde sus inicios, en la Teoría de Complejidad se definen clases de problemas diferenciadas, o bien a través del tiempo o del espacio utilizado por una máquina de Turing para resolverlos.

Un lenguaje $\mathcal{L}$ pertenece a la clase de complejidad temporal **PTIME** si y sólo si existe una máquina de Turing determinística que decide la pertenencia de una cadena $w$ al lenguaje en tiempo $t(n)$, donde $t$ es una función polinomial y $n$ es la longitud de $w$.

Un lenguaje $\mathcal{L}$ pertenece a la clase de complejidad espacial **LOGSPACE** si y sólo si existe una máquina de Turing determinística que decide la pertenencia de una cadena $w$ al lenguaje en espacio $s(n)$, donde $s$ es una función logarítmica y $n$ es la longitud de $w$.

En este trabajo nos concentraremos en la clase de complejidad temporal **PTIME**.

Con el pasar de los años se encontraron otros modelos alternativos para capturar las clases de complejidad, por ejemplo mediante álgebras de funciones o por la expresividad de diferentes lógicas sobre modelos finitos.

Para **PTIME**, entre otras, existen la caracterización algebraica de Bellantoni-Cook llamada clase **B** y la caracterización de R. Fagin a través de la lógica de *Primer orden con Punto Fijo Inflacionario* (**FO(IFP)**) sobre modelos finitos.

Para mostrar la equivalencia de **PTIME** con **FO(IFP)**, R. Fagin presenta en primer lugar una codificación que a cada estructructura (finita) del vocabulario fijado $\tau$ con $k$ símbolos de relación y $l$ símbolos de constante, le asigna una $(k + l + 1)$-upla de números binarios, es decir, un número binario para codificar cada relación, cada constante y el universo. La función de codificación está dada por $bin(\mathcal{A}) : \tau\text{-estructuras} \rightarrow \{0,1\}^{*^{k+l+1}}$. Denotaremos $\mathcal{L}_\tau^\varphi$ al conjunto de las codificaciones de $\tau$-estructuras que satisfacen $\varphi$ (Notar que $\mathcal{L}_\tau^\varphi$ es un conjunto de $(k + l + 1)$-uplas de números binarios).

Para ver **FO(IFP)** $\subseteq$ **PTIME**, dada la clase de $\tau$-estructuras determinada por $\varphi$ (una fórmula bien formada en **FO(IFP)**), R. Fagin construye la máquina de Turing que decide si la codificación de una $\tau$-estructura $\mathcal{A}$ pertenece a $\mathcal{L}_\tau^\varphi$ en tiempo $t(|bin(\mathcal{A})|)$, donde $t$ es una función polinomial.

Para ver **PTIME** $\subseteq$ **FO(IFP)**, se construye una fórmula $\varphi$ en **FO(IFP)** tal que una máquina de Turing $\mathcal{M} \in$ **PTIME** acepta la codificación de una $\tau-$estructura $\mathcal{A}$ si y solo si $\mathcal{A} \models \varphi$.

Por otro lado, en 1964, A. Cobham dio una caracterización algebraica de la clase **PTIME**. Esta caracterización tiene como desventaja que a la definición habitual del operador de recursión se le debe añadir una condición adicional sobre el tamaño de la salida de la función. En 1992, S. Bellantoni y S. Cook presentan un álgebra de funciones que también

caracteriza **PTIME** y no requiere probar acotaciones para el tamaño de la salida de las funciones. A esta clase de funciones se le llama *clase **B***. S. Bellantoni y S. Cook demuestran la equivalencia entre **B** y **PTIME** de manera indirecta al probar la equivalencia entre las funciones de la clase **B** y el álgebra de funciones definida por A. Cobham.

Esto permite entonces establecer el siguiente cuadro de equivalencias:

$$\mathbf{B} \overset{\text{equivalencia de álgebras de funciones}}{\equiv} \text{clase de Cobham} \overset{\text{máquinas de Turing}}{\equiv} \mathbf{PTIME} \overset{\text{máquinas de Turing}}{\equiv} \mathbf{FO(IFP)}$$

En esta tesis se mostrará directamente la equivalencia entre **B** y **FO(IFP)** sin utilizar máquinas de Turing ni el álgebra de funciones de Cobham, por medio de una asignación directa que permite representar cada función en **B** a través de una fórmula en **FO(IFP)** y viceversa.

# Índice

# Introducción

En la década de 1930 se introdujeron varios modelos para representar las funciones computables, como las máquinas de A. Turing, el cálculo lambda de A. Church o las funciones recursivas de K. Gödel.

Dentro del conjunto de las funciones computables, se puede observar que algunas funciones se pueden calcular en menos tiempo que otras, y a medida que se les ingresan argumentos de mayor tamaño, esta diferencia en tiempo se hace más notoria. En la década de 1960 con el trabajo de J. Hartmanis y R. Stearns ("Algebraic Structure Theory of Sequential Machines") surgen las bases de la complejidad computacional.

Se consideran funciones simples de calcular aquellas que requieren, para obtener un resultado, un tiempo máximo $t(n)$, donde $n$ es el tamaño de la entrada de la función y $t$ una función polinomial.

En particular, en 1964 A. Cobham [COB/64] caracterizó el conjunto de funciones computables en tiempo polinomial mediante un *álgebra de funciones* con una operación de recursión acotada (Bounded Recursion on Notation), que requiere mostrar una cota polinomial del tamaño de la salida de la función. Posteriormente, S. Bellantoni y S. Cook [BEC/92] presentaron una clase de funciones –la clase **B**– que caracteriza la clase de problemas computables por una máquina de Turing determinística en tiempo polinomial respecto al tamaño de la entrada sin el requerimiento de mostrar la cota de la recursión. Las funciones de la clase **B** tienen como dominio $\mathbb{N}_0^*$, las sucesiones finitas de números naturales incluyendo el cero, y como imagen $\mathbb{N}_0$, los números naturales incluyendo el cero.

Dado cualquier vocabulario $\tau$, las funciones de la clase **B** determinan clases de $\tau$-estructuras de la siguiente manera: dada una $\tau$-estructura $\mathcal{A}$ y una función $f$ en **B**, $\mathcal{A}$ *pertenece a la clase determinada por* $f$ si y sólo si $f(bin(\mathcal{A})) > 0$. Esto es, se evalúa la función tomando como argumento la codificación de la estructura. $\mathcal{A}$ pertenece a la clase si y sólo si el resultado de aplicar esta función es mayor que cero. Denotaremos $\mathcal{L}_\tau^f$ el conjunto de las codificaciones de las $\tau$-estructuras que pertenecen a la clase determinada por $f$.

Por otro lado, a partir de 1974, surge la *Complejidad Descriptiva* ([EBF/95, IMM/98]). R. Fagin muestra la equivalencia entre diferentes lógicas y clases de complejidad, entre ellas la equivalencia de la lógica de primer orden con el operador de punto fijo inflacionario –**FO(IFP)**– y **PTIME**.

Las fórmulas en **FO(IFP)** también determinan clases de estructuras de la siguiente manera: dada una $\tau$-estructura $\mathcal{A}$ y una fórmula $\varphi$ en **FO(IFP)**, $\mathcal{A}$ pertenece a la clase determinada por $\varphi$ si y solo si $\mathcal{A} \models \varphi$. Notaremos $\mathcal{L}_\tau^\varphi$ el conjunto de las codificaciones de las $\tau$-estructuras que pertenecen a la clase determinada por $\varphi$.

Entonces, ya sea mediante una fórmula o mediante una función se expresa la propiedad o restricción que deben cumplir las estructuras para pertenecer a la clase. Se puede, por ejemplo, definir la clase de los grafos conexos, como la clase cuyas estructuras son los grafos finitos y la propiedad es "ser conexo" (que es expresable por medio de una fórmula con el vocabulario de grafos en **FO(IFP)**).

Esto permite determinar la clase por la descripción de la propiedad que deben tener las estructuras, más allá de cómo se hace para ver si las estructuras cumplen o no la propiedad (de aquí proviene el nombre *Complejidad Descriptiva*).

Se denominan *problemas de decisión* aquellos problemas en los que se decide la pertenecia de una estructura a una determinada clase.

Denotaremos $\mathcal{T}_{\mathbf{B}}$ el conjunto de las clases de estructuras que se pueden determinar por alguna función en la clase **B**, y $\mathcal{T}_{\mathbf{FO(IFP)}}$ el conjunto de las clases de estructuras que se pueden determinar por alguna fórmula en **FO(IFP)**. Mostraremos en esta tesis la igualdad entre $\mathcal{T}_{\mathbf{B}}$ y $\mathcal{T}_{\mathbf{FO(IFP)}}$.

Una de las primeras dificultades que surgieron, fue la de representar una estructura como argumento de las funciones en **B**. Para resolver esto, presentaremos una función de codificación tal que, dado un vocabulario $\tau$ fijado, a cada $\tau$-estructura le corresponde una $(k + l + 1)$-upla de números $\in \mathbb{N}_0$ representados en binario. Esta función de codificación está basada en la que se presenta en "Finite Model Theory" (Ver [EBF/95]).

Las funciones en **B** se aplican a argumentos en $\mathbb{N}_0^*$ y el conjunto de valores que pueden tomar las variables libres de las fórmulas en **FO(IFP)** es acotado dado que se trabaja sobre modelos finitos. Entonces fue necesario definir una forma de representar los elementos en $\mathbb{N}_0^*$ como elementos pertenecientes al modelo finito. Para esto se representan los números en **B** como una secuencia de elementos en el dominio de la estructura.

Ya teniendo estos dos problemos resueltos, definiremos por inducción en la construcción de una determinada función $f$ en **B**, la fórmula $\varphi$ en **FO(IFP)** tal que $\mathcal{L}_\tau^f = \mathcal{L}_\tau^\varphi$. Presentaremos como herramienta para la demostración la definición de *Programa en **B***. Dado que $f$ es una función cualquiera en **B** podremos deducir que $\mathcal{T}_{\mathbf{B}} \subseteq \mathcal{T}_{\mathbf{FO(IFP)}}$.

Por medio de inducción en la construcción de una fórmula $\varphi$ en **FO(IFP)** definiremos una función $f$ en **B** tal que $\mathcal{L}_\tau^\varphi = \mathcal{L}_\tau^f$. Dado que $\varphi$ es una fórmula (cerrada) cualquiera en **FO(IFP)**, deduciremos que también $\mathcal{T}_{\mathbf{FO(IFP)}} \subseteq \mathcal{T}_{\mathbf{B}}$.

Con las dos demostraciones por inducción presentadas se concluye que $\mathcal{T}_{\mathbf{FO(IFP)}} = \mathcal{T}_{\mathbf{B}}$, es decir, ver si una estructura $\mathcal{A}$ pertenece a una clase $\mathcal{K}$ determinada por una fórmula en **FO(IFP)** es equivalente a ver si $\mathcal{A}$ pertenece a una clase $\mathcal{K}$ determinada por una función $f$ en **FO(IFP)**.

De esta manera se puede ver que para problemas de decisión es equivalente determinarlos

por una fórmula en **FO(IFP)** o por una función en **B**.

Luego estableceremos la equivalencia para la clase **FP** -el conjunto de funciones computables en tiempo polinomial por una máquina de Turing determinística- estableciendo una forma de caracterizar los problemas funcionales en la lógica **FO(IFP)** y en la clase **B**.

Llamaremos a partir de ahora **B** al conjunto $\mathcal{T}_{\mathbf{B}}$ y **FO(IFP)** al conjunto $\mathcal{T}_{\mathbf{FO(IFP)}}$.

En la sección 1 se explicarán tres conceptos preliminares: la definición de **PTIME** mediante el uso de Máquinas de Turing, Complejidad Descriptiva, y la clase de funciones de Bellantoni-Cook, para luego poder relacionar la lógica **FO(IFP)** con la clase **B** de funciones.

En la sección 2 definiremos las fórmulas que intervienen en la demostración de la inclusión **B** $\subseteq$ **FO(IFP)**, incluyendo la noción de *Programa en **B***.

En la sección 3 definiremos las funciones que intervienen en la demostración de **FO(IFP)** $\subseteq$ **B**, incluyendo todos los pasos necesarios de acuerdo al árbol de formación de la fórmula.

En la sección 4 presentaremos la definición clase **FP** (funciones computables en tiempo polinomial) en **FO(IFP)**, con su respectiva equivalencia con la clase **B**.

En las secciones 5 y 6 presentaremos el trabajo a desarrollar en el futuro relacionado con estos temas y las conclusiones obtenidas.

# 1. Conceptos Preliminares

## 1.1. Máquinas de Turing

**Notación 1.1.** Si $\Sigma$ es un alfabeto finito, $\Sigma^*$ es el conjunto de todas las cadenas finitas (incluyendo la cadena vacía) de símbolos de $\Sigma$.

**Definición 1.2.** Un *lenguaje* en $\Sigma$ es un subconjunto $\mathcal{L} \subseteq \Sigma^*$.

**Notación 1.3.** Para una cadena $x \in \Sigma^*$, $|x|$ es la longitud de $x$.

Una máquina de Turing consiste de un una cinta infinita a derecha (con celdas numeradas 0,1,...) y un conjunto finito de estados. En cada celda de la cinta hay un elemento de un alfabeto $\Sigma$ o un espacio en blanco $B$.

La máquina tiene un cabezal de lectura/escritura que según el símbolo que lee en la cinta y el estado actual, realiza las siguientes acciones:

- Escribe un símbolo en el lugar en el que se encuentra el cabezal.
- Se queda en el lugar o se mueve un espacio hacia la derecha o hacia la izquierda.
- Toma un nuevo estado.

La cinta comienza inicialmente en blanco, excepto por la cadena de entrada que pertenece a $\Sigma^*$. La cadena de entrada comienza escrita a partir de la posición 0, y se comienza a partir del estado inicial.

**Definición 1.4.** Se define *máquina de Turing determinística* a la tupla:
$\langle Q, \Sigma, \Gamma, \delta, q_0 \rangle$ donde

- $Q$ es un conjunto finito de estados que contiene los estados de aceptación y rechazo $q_A$, $q_R$, así también como el estado inicial $q_0$.
- $\Sigma$ corresponde al alfabeto de la cadena de entrada.
- $\Gamma$ corresponde al alfabeto utilizado en la cinta. Se cumple que:
  - $\Sigma \subseteq \Gamma$
  - $B \in \Gamma - \Sigma$

- $\delta$ es la función de transición y está dada por:

$$\delta : (Q - \{q_A, q_R\}) \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$$

  Si $q$ es el estado actual, $s$ el símbolo que se lee y $\delta(q, s) = (q', s', h)$ entonces $q'$ el nuevo estado, $s'$ el símbolo que se escribe y $h$ el movimiento que debe realizar el cabezal.

**Definición 1.5.** Una máquina de Turing $\mathcal{M}$ *acepta* una entrada $x \in \Sigma^*$ si termina en $q_A$.

Notaremos $\mathcal{L}(\mathcal{M}) = \{x \in \Sigma^* | \mathcal{M} \text{ acepta } x\}$, $\mathcal{L}(\mathcal{M})$ es el lenguaje aceptado por $\mathcal{M}$.

### 1.1.1. Complejidad

Para poder estudiar el espacio o tiempo que se requiere para resolver un problema en función del tamaño de su entrada se utilizan las siguientes definiciones:

**Definición 1.6.** Se define el *orden* de una función, como una cota superior asintótica en su crecimiento.

$O(f) = \{g : (\exists c > 0)(\exists n_0)[\forall n \geq n_0\, g(n) \leq c \cdot f(n)]\}$. (Ver [CLO/94])

En este caso todas las funciones $g$ que tengan a partir de un $n_0$ a $c \cdot f$ como cota superior, pertenecen a $O(f)$.

Diremos que una función $g$ tiene orden $O(f)$ si $g \in O(f)$.

Llamaremos $t_{\mathcal{M}}(x)$ a la función que devuelve la cantidad de transiciones de estados que se realizan en la máquina de Turing $\mathcal{M}$ para la entrada $x$, con $t_{\mathcal{M}}(n) = \infty$ si $\mathcal{M}$ no termina con $n$ como input. Entonces podemos definir "complejidad temporal en el peor caso" para $\mathcal{M}$ de la siguiente manera:

**Definición 1.7.** La *complejidad temporal en el peor caso* para una máquina de Turing $\mathcal{M}$ es la función $T_{\mathcal{M}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ definida de la siguiente manera:

$$T_{\mathcal{M}}(n) = \text{máx}\{t_{\mathcal{M}}(x) \,|\, x \in \Sigma^*, |x| = n\}.$$

**Definición 1.8.** Una máquina de Turing tiene *complejidad temporal de orden polinomial* si para algún $d \in \mathbb{N}$, $T_{\mathcal{M}}(n) \in O(n^d)$.

La definición de **PTIME** está dada por:

**Definición 1.9. PTIME** = $\{L \mid L = \mathcal{L}(\mathcal{M})$ para alguna máquina de Turing $\mathcal{M}$ de complejidad temporal de orden polinomial$\}$

De igual manera se pueden extender estas definiciones para complejidad espacial:

Llamaremos $s_{\mathcal{M}}(x)$ a la función que devuelve la cantidad de celdas distintas –recordar que estaban numeradas– leídas en la cinta de la máquina de Turing $\mathcal{M}$ para la entrada $x$, con $s_{\mathcal{M}}(n) = \infty$ si $\mathcal{M}$ no termina para el input $n$. Entonces podemos definir "complejidad espacial en el peor caso" para $\mathcal{M}$ de la siguiente manera:

**Definición 1.10.** La *complejidad espacial en el peor caso* para una máquina de Turing $\mathcal{M}$ es la función $S_{\mathcal{M}} : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ definida de la siguiente manera:

$$S_{\mathcal{M}}(n) = \text{máx}\{s_{\mathcal{M}}(x) \mid x \in \Sigma^*, |x| = n\}.$$

**Definición 1.11.** Una máquina de Turing tiene *complejidad espacial de orden logarítmico* si $S_{\mathcal{M}}(n) \in O(\log n)$.

La definición de **LOGSPACE** está dada por:

**Definición 1.12. LOGSPACE** = $\{L \mid L = \mathcal{L}(\mathcal{M})$ para alguna máquina de Turing de complejidad espacial de orden logarítmico $\mathcal{M}\}$

En algunos casos lo que se busca es que la salida del algoritmo sea una cadena, entonces se toma por convención ver el resultado que queda en la cinta cuando la máquina de Turing termina, ya sea por llegar al estado $q_A$ o $q_R$.

Diremos que una máquina de Turing $\mathcal{M}$ *computa* $f : \Sigma^* \to \Sigma^*$ si para cada $x \in \Sigma^*$, $\mathcal{M}$ termina y el resultado en la cinta es $f(x)$.

La definición formal del conjunto **FP** - las funciones computables en tiempo polinomial es la siguiente:

**Definición 1.13. FP** = $\{f : \Sigma^* \to \Sigma^* \mid \mathcal{M}$ computa $f$ para una máquina de Turing de complejidad temporal de orden polinomial $\}$

En el teorema de R. Fagin se utilizan máquinas de Turing con múltiples cintas, pero se puede demostrar que para toda máquina de Turing $\mathcal{M}$ con múltiples cintas existe una máquina de Turing $\mathcal{M}'$ con una única cinta tal que $T_{\mathcal{M}'}(n) \in O((T_{\mathcal{M}}(n))^2)$ (Ver [COO/05]).

## 1.2. Álgebra de funciones de Bellantoni-Cook

**Definición 1.14.** Un *operador* (u *operación*) es una asignación que va de funciones en funciones (Ver [CLO/94]). Si $\chi$ es un conjunto de funciones y OP es un conjunto de

operadores, entonces $[\chi; \text{OP}]$ denota el conjunto más pequeño de funciones que contiene a $\chi$ y es cerrado bajo las operaciones de OP. El conjunto $[\chi; \text{OP}]$ se llama un *álgebra de funciones*.

**Definición 1.15.** Se define la *función característica* $c_P(\overline{x})$ de un predicado $P$:

$$c_P(\overline{x}) = \begin{cases} 1 & \text{si } P(\overline{x}) \\ 0 & \text{sino} \end{cases}$$

Si $\mathcal{F}$ es una clase de funciones, entonces se define $\mathcal{F}_*$ como el conjunto de predicados cuyas funciones características pertenecen a $\mathcal{F}$.

**Notación 1.16.** Notaremos $\overline{x} = x_1,\ldots,x_k$.

Consideremos las funciones de la siguiente forma:

$$f(x_1, x_2, \ldots, x_m; a_1, a_2, \ldots, a_l)$$

donde las variables que aparecen del lado izquierdo $(x_1, x_2, \ldots, x_m)$ se les denomina variables *normales* (en la bibliografía denominadas *normal*) y las que aparecen del lado derecho $(a_1, a_2, \ldots, a_l)$ variables *seguras*(denominadas *safe*). A veces escribiremos $f(\overline{x}; \overline{a})$ para representar $f(x_1, \ldots, x_m, a_1, \ldots, a_l)$.

La idea es que sobre los valores de variables en posiciones safe no se podrán aplicar recursiones.

Los valores que tomarán todas las variables al aplicar una función, se representarán en base 2.

**Notación 1.17.** *yi* en binario con

- $i \in \{0, 1\}$
- $y \in \{0, 1\}^*$

representa el número natural escrito en base dos: $2y + i$. Utilizaremos como cadenas de bits, '0', y aquellas que comiencen con '1'.

**Notación 1.18.** $\overline{r}(\overline{x}; \overline{a}) = r_1(\overline{x}; \overline{a}), r_2(\overline{x}; \overline{a}), \ldots, r_k(\overline{x}; \overline{a})$

Consideremos las funciones:

- $0(;)$: la función que devuelve 0.

- $S_i(;x)$: devuelve $i \in \{0,1\}$ concatenado al final de $x$, es decir $2 \cdot x + i$.
- $p(;x)$: devuelve el predecesor de $x$, $\lfloor x/2 \rfloor$.
- $C(;a,b,c)$: devuelve $b$ en caso de que $a$ sea par, $c$ en caso contrario.
- $\pi_j^{n,m}(x_1,\ldots,x_n;x_{n+1},\ldots,x_{n+m})$: devuelve $x_j$.

Consideremos también los operadores:

- $SRN$:"Safe Recursion on Notation" o "Recursión Predicativa en Notación" dada por:

  $f(0,\overline{x};\overline{a}) = g(\overline{x};\overline{a})$

  $f(yi,\overline{x};\overline{a}) = h_i(y,\overline{x};\overline{a}, f(y,\overline{x};\overline{a}))$ para $yi \neq 0$.

  En el caso base de la recursión se aplica $g$. En caso contrario, dependiendo del último dígito del argumento sobre el que se realiza la recursión, se aplica $h_0$ o $h_1$. Observar que el resultado de la recursión queda del lado safe, restrigiendo así las funciones que se le pueden aplicar (en particular, evitando que se pueda aplicar recursión sobre el resultado de una recursión).

- $SCOMP$: "Safe Composition" o "Composición Segura" dada por:

  $f(\overline{x};\overline{a}) = h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{a}))$.

  La composición así definida garantiza que un argumento ubicado entre las variables safe, no pueda aparecer como normal.

Definimos la clase **B** como el álgebra de funciones dado por [0, $S_0$, $S_1$, $p$, $C$; $SRN$, $SCOMP$], el conjunto más pequeño de funciones que contiene a $0, S_0, S_1, p, C$ y que es cerrado por $SRN$ y $SCOMP$. Su dominio son las sucesiones finitas de los números naturales incluyendo al cero representados en binario.

Las funciones pertenecientes a esta clase tienen la particularidad de que por sus restricciones sintácticas caracterizan las funciones pertenecientes a **PTIME** (Teoremas 3.3 y 4.2 en [BEC/92]).

### 1.2.1. La clase B

Siguiendo a S. Bellantoni, S. Cook [BEC/92] se define entonces **B** como la clase más pequeña que contiene a las funciones 1 - 5 y es cerrada bajo 6, 7 siguientes:

1. (Constante) $0(;) = 0$.
2. (Proyecciones) $\pi_j^{n,m}(x_1 \dots x_n; x_{n+1} \dots x_{n+m}) = x_j$, para $1 \leq j \leq n+m$.
3. (Sucesores) $S_i(; a) = 2a + i$, para $i \in 0, 1$.
4. (Predecesor) $p(; 0) = 0$, $p(; ai) = a$.
5. (Condicional) $C(; a, b, c) = \begin{cases} b & \text{si } a \equiv 0 \bmod 2 \\ c & \text{si } a \equiv 1 \bmod 2 \end{cases}$
6. (Recursión Predicativa en Notación) Es posible definir una nueva función $f$ de la siguiente forma

   $f(0, \overline{x}; \overline{a}) = g(\overline{x}; \overline{a})$

   $f(yi, \overline{x}; \overline{a}) = h_i(y, \overline{x}; \overline{a}, f(y, \overline{x}; \overline{a}))$ para $yi \neq 0$ con $h_i$ y $g$ en **B**.
7. (Composición segura) Es posible definir una nueva funcion $f$ de la siguiente forma $f(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x};); \overline{t}(\overline{x}; \overline{a}))$ donde $h$, $\overline{r}$ y $\overline{t}$ están en **B**.

**Observación 1.19.** De las funciones definidas se pueden obtener los siguientes esquemas:

- Mediante una serie de composiciones y proyecciones se puede definir una función $f$ de la siguiente forma:

$$f(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x_{s_1}}; \overline{x_{s_2}}); \overline{t}(\overline{x_{s_3}}; \overline{x_{s_4}}, \overline{a_{s_5}}))$$

  donde si $\overline{x} = x_1, \dots, x_k$: $\overline{x_{s_i}} = x_{j_1}, \dots, x_{j_m}$ con $j_1, \dots, j_m \in \{1, \dots, k\}$ y si $\overline{a} = a_1, \dots, a_h$: $\overline{a_{s_i}} = a_{j_1}, \dots, a_{j_p}$ con $j_1, \dots, j_p \in \{1, \dots, h\}$.

- Mediante diversas aplicaciones de la composición se puede definir

$$f(\overline{x}; \overline{a}) = h_1(h_2(h_3(\dots(\overline{r}(\overline{x}); \overline{t}(\overline{x}; \overline{a}))\dots)))$$

**Definición 1.20.** Una definición de un conjunto $X$ es *impredicativa* si su definición depende de $X$.

La recursión habitual podría considerarse *impredicativa* ya que se calcula el valor de una función $f$ utilizando $f$ en su definición. Sin embargo, en esta recursión SRN, la aplicación se realiza sobre el último dígito del primer argumento y la recursión utilizará un valor más chico (la parte entera del primer argumento dividido dos, i.e. $\lfloor \frac{\text{primer argumento}}{2} \rfloor$). Por lo tanto no se presenta la impredicatividad. (Ver [BEL/92]).

### 1.2.2. Cota de tamaño de salida en B

**Notación 1.21.** $|\overline{x}| = |x_1| + |x_2| + \cdots + |x_m|$

**Lema 1.22.** *Sea $f$ una función en **B**. Existe un polinomio creciente monótono $q_f$ tal que*

$$|f(\overline{x};\overline{y})| \leq q_f(|\overline{x}|) + \max_i |y_i| \quad \textit{para todo } \overline{x},\overline{y}$$

*Demostración.* La demostración presentada en [BEC/92] es la siguiente:

Se demuestra por inducción en la derivación de la función.

Si $f$ es la función Constante, Proyección, Sucesor, Predecesor o Condicional, $|f(\overline{x};\overline{y})| \leq 1 + \sum_i |x_i| + \max_i |y_i|$.

Si $f$ está definida por Recursión Predicativa en Notación, entonces por hipótesis inductiva tenemos $q_g$, $q_{h_0}$ y $q_{h_1}$ como cotas para $g$, $h_0$ y $h_1$ respectivamente. Si tomamos $q_h = q_{h_0} + q_{h_1}$, obtenemos

$$\begin{aligned} |f(0,\overline{x};\overline{y})| &\leq q_g(|\overline{x}|) + \max_i |y_i| \\ |f(zi,\overline{x};\overline{y})| &\leq q_h(|z| + |\overline{x}|) + \max(\max_i |y_i|, |f(z,\overline{x};\overline{y})|) \end{aligned}$$

Si se define $q_f(|z| + |x|) = |z| \cdot q_h(|z| + |\overline{x}|) + q_g(|\overline{x}|)$, se obtiene trivialmente $|f(0,\overline{x};\overline{y})| \leq q_f(|0| + |x|) + \max_i |y_i|$.

Por ser $q_h$ una función monótona creciente:

Si tomamos que ($|f(z,\overline{x};\overline{y})| \leq q_f(|z| + |x|) + \max_i |y_i|$ y $zi \neq 0$, es decir $|zi| = |z| + 1$, obtenemos:

$$\begin{aligned} |f(zi,\overline{x};\overline{y})| &\leq q_h(|z| + |\overline{x}|) + \max(\max_i |y_i|, |f(z,\overline{x};\overline{y})|) \\ &\leq q_h(|z| + |\overline{x}|) + \max(\max_i |y_i|, q_f(|z| + |x|) + \max_i |y_i|) \\ &\leq q_h(|z| + |\overline{x}|) + q_f(|z| + |x|) + \max_i |y_i| \\ &\leq q_h(|z| + |\overline{x}|) + (|z| \cdot q_h(|z| + |\overline{x}|) + q_g(|\overline{x}|)) + \max_i |y_i| \\ &\leq |zi| \cdot q_h(|z| + |\overline{x}|) + q_g(|\overline{x}|) + \max_i |y_i| \\ &\leq |zi| \cdot q_h(|zi| + |\overline{x}|) + q_g(|\overline{x}|) + \max_i |y_i| \\ &\leq q_f(|zi| + |x|) + \max_i |y_i| \end{aligned}$$

Con esta inducción demostramos que $|f(z,\overline{x};\overline{y})| \leq q_f(|z| + |x|) + \max_i |y_i|$ para todo $z$.

Por último si $f$ está definida por Composición Segura, utilizando hipótesis inductiva en $h$, $\overline{r}$ y $\overline{t}$:

$$\begin{aligned}
|f(\overline{x};\overline{y})| &= |h(\overline{r}(\overline{x};);\overline{t}(\overline{x};\overline{y}))| \\
&\leq q_h(|\overline{r}(\overline{x};)|) + \max_i |t_i(\overline{x};\overline{y})| \\
&\leq q_h(\overline{q_r}(|\overline{x}|;)) + \max_i |t_i(\overline{x};\overline{y})| \\
&\leq q_h(\overline{q_r}(|\overline{x}|;)) + \max_i (q_{t_i}(|\overline{x}|) + \max_j |y_j|) \\
&\leq q_h(\overline{q_r}(|\overline{x}|;)) + \sum_i q_{t_i}(|\overline{x}|) + \max_i |y_i|
\end{aligned}$$

Entonces eligiendo $q_f(|x|) = q_h(\overline{q_r}(|\overline{x}|;)) + \sum_i q_{t_i}(|\overline{x}|)$ obtenemos la cota para la composición.

□

## 1.3. Complejidad Descriptiva

En esta sección unificaremos la notación relacionada con las fórmulas lógicas, para luego presentar los conceptos básicos y principales teoremas de la Complejidad Descriptiva.

### 1.3.1. Conceptos previos

**Definición 1.23.** Un *vocabulario* es un conjunto finito no vacío de símbolos de relación ($P, Q, R, \ldots$) y símbolos de constantes ($c, d, \ldots$). Todo símbolo de relación tiene asociado una aridad. Se denotarán los vocabularios con $\tau, \sigma, \ldots$.

**Definición 1.24.** Una *estructura* $\mathcal{A}$ de vocabulario $\tau$ (o una $\tau$-estructura) consiste en un conjunto no vacío $A$, el universo o dominio de $\mathcal{A}$, de una relación $R^{\mathcal{A}}$ de aridad $k$ para cada símbolo de relación $R$ de aridad $k$ en $\tau$, y un $c^{\mathcal{A}} \in A$ por cada símbolo de constante $c$ en $\tau$.

Se pueden extender los resultados que se presentan a vocabularios que contienen también símbolos de función de una manera simple. Dado un vocabulario $\tau$ con símbolos de función $f_1, \ldots, f_h$, se define el vocabulario $\tau^r$ que se obtiene con los símbolos de relación y de constantes de $\tau$, agregando una nueva relación de aridad $k+1$ por cada función de aridad $k$ en $\tau$ de la manera natural.

Se llamará $Mod(\varphi)$ al conjunto de modelos finitos de $\varphi$.

### 1.3.2. Estructuras ordenadas

Sea $\tau = \{<\}$ un vocabulario con un símbolo de relación binario $<$. Una $\tau$-estructura $\mathcal{A}$=($A$ ,$<^{\mathcal{A}}$) es una estructura *ordenada* si para todo $a, b, c \in A$

1. $\neg(a <^{\mathcal{A}} a)$
2. $(a <^{\mathcal{A}} b) \vee (b <^{\mathcal{A}} a) \vee (a = b)$
3. si $(a <^{\mathcal{A}} b) \wedge (b <^{\mathcal{A}} c)$ entonces $(a <^{\mathcal{A}} c)$

Para trabajar con estructuras ordenadas finitas, se utilizan vocabularios que contienen $\{<, S, \text{mín}, \text{máx}\}$ (i.e. símbolos de relación de menor entre los elementos, sucesor, símbolos de constantes de mínimo y máximo), donde además de cumplirse los puntos anteriores para todo $a, b, c \in A$ debe suceder que:

1. $S^{\mathcal{A}}ab$ si y solo si $((a <^{\mathcal{A}} b)$ y, para todo $c$ si $(a <^{\mathcal{A}} c)$ entonces $(b <^{\mathcal{A}} c \vee b = c))$
2. $\text{mín}^{\mathcal{A}} <^{\mathcal{A}} a$ o $\text{mín}^{\mathcal{A}} = a$
3. $a <^{\mathcal{A}} \text{máx}^{\mathcal{A}}$ o $a = \text{máx}^{\mathcal{A}}$

Si tenemos un vocabulario $\tau_0$ con $\{<\} \subseteq \tau_0 \subseteq \{<, S, \text{mín}, \text{máx}\}$ y $\sigma$ un vocabulario arbitrario tal que $\tau_0 \subseteq \sigma$, $O(\sigma)$ representa las $\sigma$-estructuras ordenadas.

Para codificar las estructuras, tal como se explicará más adelante, se hace fuerte uso del orden entre los elementos del dominio, es por eso que se requiere siempre trabajar con estructuras ordenadas.

En esta tesis utilizaremos estructuras ordenadas con dominios finitos.

### 1.3.3. Teorías de Primer Orden

Para un vocabulario $\tau$, las fórmulas en primer orden van a ser cadenas pertenecientes al alfabeto dado por:

- variables
- conectivos lógicos$(\vee, \neg)$
- cuantificadores

- igualdad
- ( , )
- los símbolos de $\tau$

Un término del vocabulario $\tau$ es una variable o una constante.

Una fórmula de la lógica de primer orden del vocabulario $\tau$ está dada por finitas aplicaciones de las siguientes reglas:

- si $t_0$ y $t_1$ son términos, $t_0 = t_1$ es una fórmula. En particular es una fórmula atómica.
- si $R \in \tau$ es un símbolo de relación de aridad $k$, y $t_1, \ldots, t_k$ son términos, $Rt_1 \ldots t_k$ es una fórmula. En particular es una fórmula atómica.
- si $\psi$ es una fórmula, $\neg\psi$ es una fórmula.
- si $\phi$ y $\psi$ son fórmulas, $(\phi \vee \psi)$ es una fórmula.
- si $\phi$ es una fórmula y $x$ una variable, $(\exists x)\phi$ es una fórmula.

**Variables libres**

Dada una fórmula $\varphi$ se define el conjunto $free(\varphi)$ de las variables libres de $\varphi$:

- si $\varphi$ es atómica, $free(\varphi)$ es el conjunto de variables que aparecen $\varphi$
- $free(\neg\varphi) = free(\varphi)$
- $free(\varphi \vee \psi) = free(\varphi) \cup free(\psi)$
- $free((\exists x)\varphi) = free(\varphi)\backslash\{x\}$

Las variables ligadas son aquellas que se encuentran bajo el alcance de un cuantificador. Observar que una variable puede estar libre y ligada en una misma fórmula.

**La relación de satisfacibilidad**

Sea $\mathcal{A}$ una $\tau-$estructura, una asignación en $A$ es una función $\alpha$ cuyo dominio es el conjunto de variables y su imagen son elementos de $A$, $\alpha : \{v_n \mid n \geq 1\} \to A$. Se puede extender $\alpha$ a constantes, definiendo: $\alpha(c) = c^A$ con $c \in \tau$. Se denota $\alpha\frac{a}{x}$ como la asignación que coincide con $\alpha$, excepto en $x$, $\alpha\frac{a}{x}(x) = a$

Se define la relación

$$\mathcal{A} \models \varphi[\alpha]$$

de la siguiente manera:

$$\begin{array}{rll}
\mathcal{A} \models t_1 = t_2[\alpha] & \text{si y solo si} & \alpha(t_1) = \alpha(t_2) \\
\mathcal{A} \models Rt_1 \dots t_n[\alpha] & \text{si y solo si} & R^{\mathcal{A}}\alpha(t_1) \dots \alpha(t_n) \\
\mathcal{A} \models \neg\varphi[\alpha] & \text{si y solo si} & \text{no } \mathcal{A} \models \varphi[\alpha] \\
\mathcal{A} \models (\varphi \vee \psi)[\alpha] & \text{si y solo si} & \mathcal{A} \models \varphi[\alpha] \text{ o } \mathcal{A} \models \psi[\alpha] \\
\mathcal{A} \models \exists x\varphi[\alpha] & \text{si y solo si} & \text{hay un } a \in A \text{ tal que } \mathcal{A} \models \varphi[\alpha\frac{a}{x}]
\end{array}$$

Además

$$\mathcal{A} \models \varphi$$

si para cualquier $\alpha$

$$\mathcal{A} \models \varphi[\alpha]$$

### 1.3.4. Lógica de Segundo Orden

La lógica de segundo orden es una extensión de la lógica de primer orden que permite cuantificar relaciones. Se agregan entre las fórmulas bien formadas:

- si $X$ es una variable de relación de aridad $k$, y $t_1, \dots, t_k$ son términos, $Xt_1 \dots t_k$ es una fórmula. Es también una fórmula atómica.
- si $\varphi$ es una fórmula y $X$ es una variable de relación, $(\exists X)\varphi$ es una fórmula.

### 1.3.5. Punto Fijo Inflacionario FO(IFP)

Dado que existen enunciados que no se pueden expresar en lógica de primer orden, como la fórmula que caracteriza los grafos conexos, o los dominios de cardinal par, se realizan extensiones a la lógica de primer orden a fin de obtener un mayor poder de expresión. La idea es agregar operaciones que simulen ciclos o iteraciones.

Sea $M$ un conjunto finito no vacío, llamaremos $\mathcal{P}(M)$ a el conjunto de partes de $M$. Dada la función $F : \mathcal{P}(M) \to \mathcal{P}(M)$ obtenemos la siguiente secuencia:

$$\emptyset, F(\emptyset), F(F(\emptyset)), \dots$$

Representables como $F_0, F_1, F_2, \ldots$; con $F_0 = \emptyset$, $F_{n+1} = F(F_n)$. Si existe un $n_0 \geq 0$ tal que $F_{n_0+1} = F_{n_0}$, entonces para todo $m \geq n_0 \quad F_m = F_{n_0}$ y se denota como $F_\infty = F_{n_0}$ el punto fijo de $F$. Si no existe ningún $n_0$ que cumpla lo anterior, entonces el punto fijo no existe y se denota $F_\infty = \emptyset$.

**Definición 1.25.** $F$ es *inflacionaria* si para todo $X \subseteq M$, $X \subseteq F(X)$.

Se puede ver en [EBF/95], que dada una función $F$ inflacionaria, el punto fijo siempre existe y $F_\infty = F_{\|M\|}$.

Sea $\psi(x_1, \ldots, x_k, \bar{u}, X, \bar{Y})$ una fórmula en el vocabulario $\tau$ ($x_1, \ldots, x_k, \bar{u}$ variables de primer orden de $\psi$, y $X, \bar{Y}$ variables de segundo orden), donde la aridad de $X$ es $k$. Dada una $\tau$-estructura $A$ con $\bar{b}$ una interpretación de $\bar{u}$ y $\bar{S}$ una interpretación de $\bar{Y}$, definimos la función $F^\psi : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ de la siguiente forma:

$$F^\psi(R) = \{(a_1, \ldots, a_k) \mid A \models \psi[a_1, \ldots, a_k, \bar{b}, R, \bar{S}]\}$$

Para un vocabulario $\tau$ la clase **FO(IFP)**$[\tau]$ de formulas está dada por las siguientes reglas:

- si $\psi$ es una fórmula atómica de segundo orden en $\tau$, $\psi \in$ **FO(IFP)**$[\tau]$
- si $\psi$ es una fórmula, $\neg\psi$ es una fórmula
- si $\psi$ y $\phi$ son fórmulas, $(\psi \vee \phi)$ es una fórmula
- si $\psi$ es una fórmula y $x$ es una variable, $(\exists x)\psi$ es una fórmula
- si $\psi$ es una fórmula, $X$ es una variable de aridad $k$, y las longitudes de $\bar{x}$ y $\bar{t}$ es $k$, entonces $[IFP_{\bar{x},X}\psi]\bar{t}$ es una fórmula.

**FO(IFP)** se obtiene al clausurar la lógica de primer orden con el operador de punto fijo.

**Variables libres**

Dada una fórmula $\varphi$ en **FO(IFP)** se agrega la siguiente definición al conjunto de las variables libres:

- $free([IFP_{\bar{x},X}\varphi]\bar{t}) = free(\bar{t}) \cup (free(\varphi) \backslash \{\bar{x}, X\})$

**La relación de satisfacibilidad**

Sea $\mathcal{A}$ una $\tau$-estructura y $\varphi$ una fórmula con el vocabulario de $\tau$. Si $X$ es una variable de relación de aridad $k$, y las variables libres de $[IFP_{\bar{x},X}\varphi]\bar{t}$ están entre $\bar{u}$ y $\bar{Y}$; $\bar{b}$ y $\bar{S}$ son interpretaciones en $\mathcal{A}$ de $\bar{u}$ y $\bar{Y}$ respectivamente, entonces:

$$\mathcal{A} \models [IFP_{\bar{x},X}\varphi]\bar{t}[\bar{b}, \bar{S}] \quad \text{si y solo si} \quad (t_1[\bar{b}], \ldots, t_k[\bar{b}]) \in F_\infty^{(X\bar{x}\vee\varphi)}$$

**Ejemplo 1.26.** Sea $\tau = \langle E \rangle$ el vocabulario de los grafos, donde $E$ es el símbolo de relación que representa los ejes. Sea $\varphi(x, y, X) \equiv Exy \vee \exists z(Xxz \wedge Ezy)$ entonces al evaluar $F_n^\varphi$ sobre una $\tau$-estructura $\mathcal{A}$ se obtiene:

$F_0^\varphi$ es el conjunto vacío (por definición).

$F_1^\varphi$ es el conjunto de los $(x, y)$ tal que $A \models F(x, y, \emptyset)$, es decir se obtienen los $(x, y)$ tales que existe un eje entre $x$ e $y$ en $\mathcal{A}$.

$F_2^\varphi$ es el conjunto de los $(x, y)$ tal que $A \models F(x, y, F_1^\varphi)$, es decir obtenemos los $(x, y)$ entre los que hay un camino de longitud a lo sumo igual a dos.

Y así sucesivamente se puede definir $F_\infty^\varphi$ como el conjunto de los $(x, y)$ tales que existe un camino entre ellos en $\mathcal{A}$.

Usando esto se puede definir una propiedad que caracteriza los grafos conexos de la siguiente manera:

$$\phi_{conexo} \equiv \forall x, y[IFP_{xy,X}\varphi(x, y, X)]xy$$

### 1.3.6. Clausura Transitiva Determinística FO(DTC)

**Definición 1.27.** Sea $R$ una relación binaria en un conjunto $M$, $R \subseteq M^2$, se define la *clausura transitiva determinística* como:

$$\begin{aligned} \mathrm{DTC}(R) \quad = \quad & \{(a, b) \in M^2 \mid \text{existe } n > 0, \text{ y } e_0, \ldots, e_n \in M, a = e_0, b = e_n, \\ & \forall i < n, e_{i+1} \text{ es el único } e \text{ tal que } (e_i, e) \in R\} \end{aligned}$$

Entonces, $[\mathrm{DTC}_{\bar{x},\bar{y}}\varphi(\bar{x}, \bar{y}, \bar{u})]\bar{s}\bar{t}$ si y solo si $(\bar{s}, \bar{t}) \in \mathrm{DTC}(\{(\bar{x}, \bar{y}) | \varphi(\bar{x}, \bar{y}, \bar{u})\})$

Ver Flum [EBF/95].

**Definición 1.28.** $C \leq C'$

Una clase de complejidad descriptiva $C$ es *menor o igual* que una clase de complejidad descriptiva $C'$ si para cada vocabulario $\tau$, para cada clase de $\tau$-estructuras $\mathcal{K}$ determinable por una fórmula $\varphi$ en $C$ existe una fórmula en $C'$ que también determina la clase.

**Proposición 1.29.** ***FO(DTC)*** $\leq$ ***FO(IFP)***

*Se puede ver que para toda fórmula $\varphi$ que pertenece a* ***FO(DTC)***$[\tau]$*, existe una fórmula en $\psi$ perteneciente a* ***FO(IFP)***$[\tau]$ *también con vocabulario $\tau$ tal que $Mod(\varphi) = Mod(\psi)$*

*Ver [EBF/95], Proposición 6.1.5.*

### 1.3.7. Clases de Complejidad en estructuras

**Definición 1.30.** Una lógica $\mathcal{L}$ *captura* una clase de complejidad $C$ si dado cualquier vocabulario $\tau$ que contiene los símbolos de relación y constantes que se utilizan en estructuras ordenadas, y $K \subseteq O(\tau)$ se tiene:

$$K \in C \Leftrightarrow K \text{ es axiomatizable en } \mathcal{L}$$

En particular por el Teorema de Fagin (Ver [EBF/95]), **FO(IFP)** captura **PTIME** y **FO(DTC)** captura **LOGSPACE**.

Entonces, si $C$ es una clase de complejidad en teoría de complejidad clásica (en máquinas de Turing), denotamos como $C'$ su correspondiente clase de complejidad en estructuras.

**Ejemplo 1.31.** Para **PTIME**:

> $C$ consiste en todos los lenguajes $L$, $L \subseteq A^+$, tal que existe una máquina de Turing determinística que acepta $L$ en tiempo polinomial.
>
> $C'$ consiste en todas las clases de estructuras ordenadas $K$, $K \subseteq O(\tau)$ para algún $\tau$, tal que existe una máquina de Turing determinística que acepta la codificación binaria de $K$ en tiempo polinomial.

La Teoría de Complejidad Descriptiva presenta una serie de equivalencias entre clases de complejidad en máquinas de Turing y poder expresivo en distintas lógicas.

Con las definiciones dadas se puede ver que probar la diferencia o equivalencia de poder expresivo entre dos lógicas es equivalente a mostrar la diferencia o equivalencia entre las clases de complejidad que capturan.

Entonces, el problema de hallar la existencia de una máquina de Turing con ciertos requerimientos de tiempo o espacio, se traduce en el problema de hallar una fórmula en una determinada lógica que exprese lo mismo.

# 2. $\mathbf{B} \subseteq$ FO(IFP)

En esta sección presentaremos la demostración de la inclusión $\mathbf{B} \subseteq$ **FO(IFP)**. En primer lugar –en la sección 2.1– presentaremos la función que nos permitirá codificar cada estructura como una tupla de números binarios. Luego, en la sección 2.2 presentaremos la definición de un *Programa en* ***B***, que nos permitirá formalizar la relación entre las funciones y el orden de construcción de la fórmula equivalente. En la sección 2.3 demostraremos una cota polinomial en el tamaño de la salida de los programas en **B**. En la sección 2.4 presentaremos la representación de los argumentos en **B**. En la sección 2.5 presentaremos un mecanismo de sustitución en fórmulas, el cual será utilizado para los operadores del álgebra de Bellantoni-Cook. Las secciones anteriores sirven como herramientas para las secciones 2.6 y 2.7 donde demostraremos por inducción en la función en **B**, la fórmula que determina la misma clase de $\tau$-estructuras.

## 2.1. Codificación de estructuras

Sea $\tau = \langle R_1, \ldots, R_k, c_1, \ldots, c_l \rangle$.

Las funciones en la clase **B** trabajan con números, entonces para que una función pueda decirnos si una estructura tiene o no una propiedad se necesita una forma de codificar $\tau-$estructuras con números. Definiremos la función **bin** de la siguiente manera:

$$\begin{aligned} \textbf{bin} : \tau\text{-estructura} &\to \{0,1\}^{*^{k+l+1}} \\ \mathcal{A} &\mapsto (n, r_1, \ldots, r_k, c_1, \ldots, c_l) \end{aligned}$$

- $n$ es una secuencia de *unos* de longitud $n = \|A\|$
- Sea $R = R_i$ una relación en $\tau$ de aridad $r$ tal que $R^{\mathcal{A}} \subseteq \{0, \ldots, n-1\}^r$. Dadas las tuplas $(0, \ldots, 0), (0, \ldots, 1), \ldots, (n-1, \ldots, n-1)$ se define $|j|_r$ la tupla número $j$ en el orden lexicográfico de $\{0, \ldots, n-1\}^r$.

  **Ejemplo 2.1.** Algunos ejemplos:

  - Para $j = 0$
    $$|j|_r = (0, \ldots, 0)$$
  - Para $j = 1$
    $$|j|_r = (0, \ldots, 1)$$
  - Para $j = n^r - 1$
    $$|j|_r = (n-1, \ldots, n-1)$$

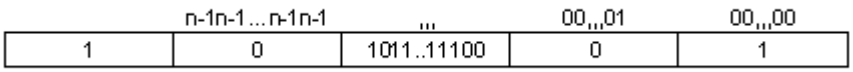
$$j = j_1 \cdot n^{r-1} + j_2 \cdot n^{r-2} + \cdots + j_{r-1} \cdot n + j_r$$

Entonces para cada relación hay un input dado por: $r_i = 1a_{n^r-1}a_{n^r-2}\ldots a_1a_0$, donde $a_i = 1$ si y solo si $R^A|j|_i$

Esta codificación está basada en la que se presenta en [EBF/95], con las siguientes modificaciones:

- Comienza con un uno para no perder la longitud de la relación.
- El orden de las tuplas comienza por aquella menor en el orden lexicográfico ubicada en el bit menos significativo, para lograr una correspondencia directa el recorrer las relaciones utilizando el operador de recursión de la clase **B**.

**Ejemplo 2.2.** Ejemplo de una codificación




En el gráfico, si representamos la codificación de una relación $R$, se ve que por ejemplo $(0, 0, \ldots, 0, 0) \in R$, $(0, 0, \ldots, 0, 1) \notin R$ y $(n-1, n-1, \ldots, n-1, n-1) \notin R$.

- Para cada símbolo de constante $c_i$, se codifica con la representación binaria de $c_i$.

## 2.2. Programa en B

En primer lugar se presentará la definición de la relación *Dep* sobre funciones en **B**. $(f, g) \in Dep$ si $f$ *depende* de $g$ para su definición. Esto servirá luego para establecer la relación entre las distintas funciones del programa.

**Definición 2.3.** *Dep* es una relación binaria entre las funciones en **B**. Se define de la siguiente manera:

- Si $f$ está definida por composición

$$f(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x}); \overline{t}(\overline{x}; \overline{a}))$$

Entonces valen:

- $Dep(f, h)$
- $Dep(f, r_i) \forall r_i \in \overline{r}$

- $Dep(f, t_i) \forall t_i \in \overline{t}$
- Y para toda función $g$ tal que $(g \neq h) \wedge (g \neq r_i) \wedge (g \neq t_i)$: $\neg Dep(f, g)$

- Si $f$ está definida por recursión

$$f(0, \overline{x}; \overline{a}) = g(\overline{x}; \overline{a})$$
$$f(yi, \overline{x}; \overline{a}) = h_i(y, \overline{x}; \overline{a}, f(y, \overline{x}; \overline{a}))$$

  Entonces valen:

  - $Dep(f, g)$
  - $Dep(f, h_0)$
  - $Dep(f, h_1)$
  - Y para toda función $r$ tal que $(r \neq g) \wedge (r \neq h_0) \wedge (r \neq h_1)$: $\neg Dep(f, r)$

**Definición 2.4.** Se define $Dep^+$ como la clausura transitiva de $Dep$.

**Definición 2.5.** Un *programa* $\mathcal{P}$ en **B** va a estar dado por la tupla $\langle f, C \rangle$ donde $f$ es una función en **B**, sin variables safe, que toma como argumentos las variables necesarias para la codificación de una $\tau$-estructura. $C$ es un conjunto de funciones en **B**. La tupla debe cumplir lo siguiente:

- Si $f$ está definida por composición:

$$f(\overline{x};) = h(\overline{r}(\overline{x};); \overline{t}(\overline{x};))$$

  entonces $h$, $\overline{r}$ y $\overline{t}$ pertenecen a $C$.

- Si $f$ está definida por recursión

$$f(0, \overline{x};) = g(\overline{x};)$$
$$f(yi, \overline{x};) = h_i(y, \overline{x}; f(y, \overline{x};))$$

  entonces g, $h_0$, $h_1$ pertenecen a $C$.

- Para toda función $k \in C, \forall h \in B, Dep^+(k, h) \rightarrow h \in C$ (toda función necesaria para construir $k$ pertenece a $C$).

**Notación 2.6.** Se utilizará la notación $pr_f$ para representar el programa que computa la función $f$.

**Definición 2.7.** Los argumentos de un programa $pr_f$ en un programa en **B**, $\mathcal{P} = \langle f, C\rangle$ son los *argumentos de* $\mathcal{P}$.

**Definición 2.8.** Una clase de $\tau$-estructuras $\mathcal{K}$ es *determinada* por un programa $\mathcal{P} = \langle f, C\rangle$, si $\mathcal{K}$ es determinada por $f$.

## 2.3. Cota del tamaño de la salida de un programa $\mathcal{P}$

Dados un vocabulario $\tau$, una $\tau$-estructura $\mathcal{A}$, un programa $\mathcal{P} = \langle f, C\rangle$ y una clase de $\tau$-estructuras $\mathcal{K}$ que determina $\mathcal{P}$, definiremos a continuación una serie de lemas que nos permitirán acotar el tamaño de la salida de $\mathcal{P}$.

El objetivo de esta sección es obtener una cota polinomial en la longitud de la codificación de la estructura, para esto utilizaremos la notación de $f, g, h$ como los programas que computan las funciones $f, g, h$ respectivamente.

**Lema 2.9.** *Si $\tau$ es un vocabulario, $\mathcal{A} = \langle A, R_1^A, \ldots, R_k^A, c_1^A, \ldots, c_l^A\rangle$ es una $\tau$-estructura donde la aridad de cada $R_i$ es $a_i$, y $f(n, r_1, \ldots, r_k, c_1, \ldots, c_l;)$ es una función en **B**, entonces existe un d tal que $|f(bin(\mathcal{A});)| \leq ||A||^d$.*

*Demostración.* Por el lema 1.22 existe un polinomio monótono creciente $p_f$ tal que $|f(n, r_1, \ldots, r_k, c_1, \ldots, c_l;)| \leq p_f(|n| + \sum_i |r_i| + \sum_j |c_j|)$. Entonces,

- Tomando $||A|| = n$.
- Sabiendo que la longitud de la codificación de una relación de aridad $a$ es $n^{a_i} + 1$.
- Sabiendo que la longitud de la codificación de una constante es $\log_2(n)$, o sea es menor o igual a $n$.
- $n \geq 2$, ya que siempre tomamos estructuras cuyo dominio tenga por lo menos dos elementos

$$\begin{aligned} |f(bin(\mathcal{A});)| &\leq p_f(n + \sum_i (n^{a_i} + 1) + \sum_j n) \\ &\leq n^d \end{aligned}$$

□

Cuando en un programa $\mathcal{P} = \langle f, C\rangle$, donde $f$ define una clase de $\tau$-estructuras $\mathcal{K}$, se aplica $f$ a la codificación de una estructura $\mathcal{A}$, para evaluar el resultado de la función -si no es una función inicial- se tiene que primero obtener el resultado de las funciones de las que depende. Por ejemplo, si

$$f(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ) = h(\overline{r}(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ); \overline{t}(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ))$$

entonces para obtener el resultado de $f$, primero habrá que obtener los resultados de

$$r_i(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ) \text{ para todo } i$$

$$t_j(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ) \text{ para todo } j$$

Dentro del programa $\mathcal{P}$, los valores que toman las variables de las funciones no son arbitrarios, sino que dependen de la codificación de la estructura.

Por ejemplo $r_i(n, r_1, \ldots, r_k, c_1, \ldots, c_l; )$ se aplicará a la codificación de la estructura cuya pertenencia a la clase se quiere decidir. Esta idea la notaremos de la siguiente manera:

**Notación 2.10.** Dado un programa $\mathcal{P} = \langle f, C\rangle$, donde $f$ define una clase de $\tau$-estructuras $\mathcal{K}$, si $g \in C$, notaremos $|g_{\mathcal{A}}(\overline{x}; \overline{y})|$ al tamaño de la salida de la función $g$ dentro del contexto de $\mathcal{A}$, es decir cuando se la aplica a valores que influyen en decidir si $\mathcal{A}$ pertenece o no a $\mathcal{K}$.

**Lema 2.11.** *Si $\tau$ es un vocabulario, $\mathcal{A} = \langle A, R_1^A, \ldots, R_k^A, c_1^A, \ldots, c_l^A\rangle$ es una $\tau$-estructura donde la aridad de cada $R_i$ es $a_i$, $\mathcal{P} = \langle f, C\rangle$ es un programa en **B** donde $f$ define una clase de $\tau$-estructuras $\mathcal{K}$, entonces con $|A| = n$, para todo $f' \in C$ tal que $Dep(f, f')$,*

$$|f'_{\mathcal{A}}(\overline{x}; \overline{y})| \leq n^d$$

*Demostración.* Si $f'$ es tal que $Dep(f, f')$ entonces hay dos casos posibles:

1. $f$ definida por Composición Segura:

   $$f(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ) = h(\overline{r}(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ); \overline{t}(n, r_1, \ldots, r_k, c_1, \ldots, c_l; ))$$

   - Si $f' = r_i$ para algún $i$:

   $$\begin{aligned} |r_{i_{\mathcal{A}}}(\overline{x}; \overline{a})| &= |r_{i_{\mathcal{A}}}(bin(\mathcal{A}); )| \\ &\leq p_{r_i}(|bin(\mathcal{A})|) \\ &\leq p_f(|bin(\mathcal{A})|) \quad (1) \\ &\leq n^d \quad (2) \end{aligned}$$

   La línea 1 sale de la definición de $p_f$ según la demostración de 1.22.
   La línea 2 sale de la desigualdad presentada en el lema 2.9.

- Si $f' = t_i$ para algún $i$:

$$\begin{aligned} |t_{i_{\mathcal{A}}}(\overline{x};\overline{a})| &= |t_{i_{\mathcal{A}}}(bin(\mathcal{A});)| \\ &\leq p_{t_i}(|bin(\mathcal{A})|) \\ &\leq p_f(|bin(\mathcal{A})|) \quad (3) \\ &\leq n^d \quad (4) \end{aligned}$$

  La línea 3 sale de la definición de $p_f$ según la demostración de 1.22.
  La línea 4 sale de la desigualdad presentada en el lema 2.9.

- Si $f' = h$:

$$\begin{aligned} |h_{\mathcal{A}}(\overline{r}(n, r_1, \ldots, r_k, c_1, \ldots, c_l;);& \\ \overline{t}(n, r_1, \ldots, r_k, c_1, \ldots, c_l;))| &= |f(n, r_1, \ldots, r_k, c_1, \ldots, c_l;)| \\ &\leq p_f(|bin(\mathcal{A})|) \quad (5) \\ &\leq n^d \quad (6) \end{aligned}$$

  La línea 5 sale de la definición de $p_f$ según la demostración de 1.22.
  La línea 6 sale de la desigualdad presentada en el lema 2.9.

2. $f$ definida por Recursión Predicativa en Notación:

$$\begin{aligned} f(0, r_1, \ldots, r_k, c_1, \ldots, c_l;) &= g(r_1, \ldots, r_k, c_1, \ldots, c_l;) \\ f(ni, r_1, \ldots, r_k, c_1, \ldots, c_l;) &= h_i(n, r_1, \ldots, r_k, c_1, \ldots, c_l; \\ &\quad f(n, r_1, \ldots, r_k, c_1, \ldots, c_l;)) \text{ para } yi \neq 0 \end{aligned}$$

- Si $f' = g$:

$$\begin{aligned} |g_{\mathcal{A}}(\overline{x};\overline{a})| &= |g_{\mathcal{A}}(r_1, \ldots, r_k, c_1, \ldots, c_l;)| \\ &\leq p_g(|bin(\mathcal{A})|) \\ &\leq p_f(|bin(\mathcal{A})|) \\ &\leq n^d \end{aligned}$$

- si $f' = h_i$, tomando $p_h = p_{h_0} + p_{h_1}$, y $|n'| < |n|$:

$$\begin{aligned} |h_{i_{\mathcal{A}}}| &= |h_{i_{\mathcal{A}}}(n', r_1, \ldots, r_k, c_1, \ldots, c_l;)| \\ &\leq p_h(|bin(\mathcal{A})|) + |f(n', r_1, \ldots, r_k, c_1, \ldots, c_l;)| \\ &\leq p_h(|bin(\mathcal{A})|) + |n'| \cdot p_h(|bin(\mathcal{A})|) + p_g(|bin(\mathcal{A})|) \\ &\leq |ni| \cdot p_h(|bin(\mathcal{A})|) + p_g(|bin(\mathcal{A})|) \\ &\leq p_f(|bin(\mathcal{A})|) \\ &\leq n^d \end{aligned}$$

□

**Lema 2.12.** *Si $\tau$ es un vocabulario, $\mathcal{A} = \langle A, R_1^A, \ldots, R_k^A, c_1^A, \ldots, c_l^A \rangle$ es una $\tau$-estructura con $\|A\| = n$ donde la aridad de cada $R_i$ es $a_i$, y $\mathcal{P} = \langle f, C \rangle$ es un programa en **B** donde $f$ define una clase de $\tau$-estructuras $\mathcal{K}$, para todo $f' \in C$ tal que $Dep^+(f, f')$,*

$$|f(bin(\mathcal{A}); )| \leq p_f(|bin(\mathcal{A})|) \leq n^d \rightarrow |f'_{\mathcal{A}}(\overline{x}; \overline{y})| \leq p_{f'}(|\overline{x}|) + \max_i |y_i| \leq n^d$$

*Demostración.* Por inducción en la clausura de $Dep$.

- Si $f'$ es tal que $Dep(f, f')$ entonces por la demostración de la cota presentada por [BEC/92] y por el lema 2.11 se cumple.
- Si $f'$ no es tal que $Dep(f, f')$ entonces existe $h'$ tal que $Dep^+(f, h')$ y $Dep(h', f')$ y $|h'_{\mathcal{A}}(\overline{x}; \overline{a})| \leq p_{h'}(|\overline{x}|) + \max_i |a_i| \leq n_d$.

  1. $h'$ definida por Composición Segura:
     $h'(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x}; ); \overline{t}(\overline{x}; \overline{a}))$
     - Si $f' = r_i$ para algún $i$:

$$\begin{aligned} |r_{i_{\mathcal{A}}}(\overline{x}; )| &\leq p_{r_i}(|\overline{x}|) \\ &\leq p_{h'}(|\overline{x}|) \\ &\leq n^d \end{aligned}$$

     - Si $f' = t_i$ para algún $i$:

$$\begin{aligned} |t_{i_{\mathcal{A}}}(\overline{x}; \overline{a})| &\leq p_{t_i}(|\overline{x}|) + \max_i |a_i| \\ &\leq p_{h'}(|\overline{x}|) + \max_i |a_i| \\ &\leq n^d \end{aligned}$$

     - Si $f' = h$:

$$\begin{aligned} |h_{\mathcal{A}}(\overline{r}(\overline{x}; ); \overline{t}(\overline{x}; \overline{a}))| &\leq p_h(|\overline{r}(\overline{x}; )|) + \max_i t_i(\overline{x}; \overline{a}) \\ &\leq p_h(\overline{p_r}(|\overline{x}|)) + \max_i(p_{t_i}(|\overline{x}|) + \max_j |a_j|) \\ &\leq p_h(\overline{pr}(|\overline{x}|)) + \sum_i p_{t_i}(|\overline{x}|) + \max_j |a_j| \\ &\leq p_{h'}(|\overline{x}|) + \max_i |a_i| \\ &\leq n^d \end{aligned}$$

2. $g'$ definida por Recursión Predicativa en Notación:
   $h'(0, \overline{x}; \overline{a}) = g(\overline{x}; \overline{a})$
   $h'(yi, \overline{x}; \overline{a}) = h_i(y, \overline{x}; \overline{a}, h'(y, \overline{x}; \overline{a}))$ para $yi \neq 0$
   - Si $f' = g$:

$$\begin{aligned} |g_{\mathcal{A}}(\overline{x}; \overline{a})| &\leq p_g(|\overline{x}|) + \max_i |a_i| \\ &\leq p_{h'}(|\overline{x}|) + \max_i |a_i| \\ &\leq n^d \end{aligned}$$

   - Si $f' = h_i$, tomando $p_h = p_{h_0} + p_{h_1}$, con $|z'| < |z|$:

$$\begin{aligned} |h_{i_{\mathcal{A}}}(z', \overline{x}; \overline{a}, f(z', \overline{x}; \overline{a}))| &\leq p_h(|z'| + |\overline{x}|) + \max(\max_i |a_i|, |h'(z', \overline{x}; \overline{a})|) \\ &\leq p_h(|z'| + |\overline{x}|) + |f(z', \overline{x}; \overline{a})| + \max_i |a_i| \\ &\leq p_h(|z'| + |\overline{x}|) + |z'| \cdot p_h(|z'| + |\overline{x}|) + \\ &\quad p_g(|\overline{x}|) + \max_i |a_i| \\ &\leq |z' + 1| \cdot p_h(|z'| + |\overline{x}|) + p_g(|\overline{x}|) + \max_i |a_i| \\ &\leq |z| \cdot p_h(|z| + |\overline{x}|) + p_g(|\overline{x}|) + \max_i |a_i| \\ &\leq p_{h'}(|\overline{x}|) + \max_i |a_i| \\ &\leq n^d \end{aligned}$$

□

Con los resultados de los lemas 2.9 y 2.12 obtenemos que al evaluar la función principal de un programa $\mathcal{P} = \langle f, C \rangle$ en la codificación de una $\tau$-estructura $\mathcal{A}$, existe un $d$ tal que el resultado de todas las funciones va a estar acotado por $n^d$.

## 2.4. Representación en FO(IFP) de argumentos del programa en B

Las variables de las funciones en **B** pueden tomar valores más grandes que el cardinal del dominio de la estructura a decidir, con lo que se debe definir una codificación de los argumentos de tal manera que puedan ser utilizados en las fórmulas pertenecientes a **FO(IFP)**.

La primera idea que surgió fue la de representar los números de la codificación de la estructura en base $n = |A|$ (cardinal del dominio). Esta idea hubo que desecharla ya que no existe una cota polinomial para estos valores. En efecto, para una $\tau$-estructura $\mathcal{A}$, cuyo

dominio tiene cardinal $n$, una de las componentes de la codificación es $2^n - 1$ y no existe un $d$ tal que $2^n - 1 \leq n^d$ para todo $n$.

La solución hallada para representar los números fue la de utilizar fórmulas con $d$ variables libres.

Antes de pasar a nuestra representación recordemos que los números en su expansión binaria pueden representarse a través de un conjunto definido de la siguiente manera:

**Notación 2.13.**

$$\widetilde{n} = \left\{ i \mid \left\lfloor \frac{n}{2^i} \right\rfloor \text{ es impar} \right\}$$

En otras palabras, es el conjunto de los índices $i$ tales que el dígito en la posición $i$ de $n$ es 1.

**Ejemplo 2.14.** Estos son algunos ejemplos:

- $\widetilde{0} = \emptyset$
- $\widetilde{1} = \{0\}$
- $\widetilde{2} = \{1\}$
- $\widetilde{3} = \{0, 1\}$
- $\widetilde{4} = \{2\}$
- $\widetilde{5} = \{0, 2\}$
- $\widetilde{6} = \{1, 2\}$
- $\widetilde{7} = \{0, 1, 2\}$
- $\widetilde{8} = \{3\}$

Observar que con esta representación se pueden hacer las operaciones usuales:

**Ejemplo 2.15.** Sumar uno.

Si un número está dado por un conjunto $X$, $\widetilde{R} = \widetilde{X} + 1$ va a estar dado por:

$$
\begin{aligned}
&\exists x(\ (\forall y\ 0 \le y \le x \to y \in X) \land \\
&\ \ x+1 \notin X \land \\
&\ \ (\forall y\ 0 \le y \le x \to y \notin R) \land \\
&\ \ (x+1 \in R) \land \\
&\ \ (\forall y\ y > x+1 \to\ \ (y \in X \leftrightarrow y \in R)) \\
&) \lor \\
&(0 \notin X \land 0 \in R \land (\forall y > 0\ y \in X \leftrightarrow y \in R))
\end{aligned}
$$

Se busca el primer cero desde el dígito menos significativo. Todos los dígitos a la derecha quedan en cero, el cero pasa a valer uno (por sumar uno), y los dígitos a la izquierda se mantienen igual.

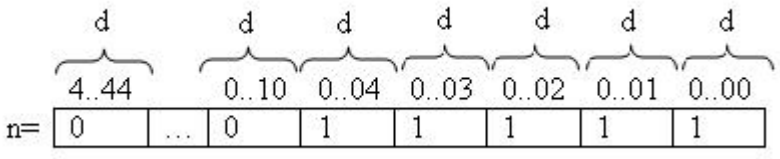
Concatenar dos números $X, Y$.

Dados dos números $X, Y$, se obtiene la concatenación: $\widetilde{R} = \widetilde{X} \oplus \widetilde{Y}$:

$$
\begin{aligned}
&(Y \neq \emptyset \land \\
&\ \ (\forall x \in X\ (x + \text{máx}\, Y + 1) \in R) \land \\
&\ \ (\forall x \in Y\ x \in R) \land \\
&\ \ (\forall x \in R\ ((x > \text{máx}\, Y \land (x - (\text{máx}\, Y + 1)) \in X) \lor (x \le \text{máx}\, Y \land x \in Y)))) \lor \\
&(Y = \emptyset \land R = X)
\end{aligned}
$$

En este trabajo utilizaremos un caso particular de esta representación. Los números, en vez de estar representados por conjuntos, estarán representados mediante relaciones. Dado un número $n$, la relación que representa a $n$ cumplirá lo siguiente: $(x_1, \ldots, x_d)$ pertenece a la relación si y solo si el bit en la posición $x = x_1 \cdot n^{d-1} + \cdots + x_d$ es un uno en la representación binaria de $n$, es decir, si $x \in \left\{ i \mid \left\lfloor \frac{n}{2^i} \right\rfloor \text{ es impar} \right\}$.

**Ejemplo 2.16.** Si $n = |A| = 5$, es decir, $A = \{0, 1, 2, 3, 4\}$:




La codificación de $n$ consiste en el número 11111, esto lo interpretaremos como que la fórmula $N(x_1, \ldots, x_d)$ es verdadera únicamente para:

- (0,0,...,0,0)
- (0,0,...,0,1)
- (0,0,...,0,2)
- (0,0,...,0,3)
- (0,0,...,0,4)

Definiremos a continuación las fórmulas que dependen de $d$ variables y representan la codificación de una $\tau$-estructura $\mathcal{A}$:

1. Para representar $n = |A|$:

$$N(x_1, \ldots, x_d) \equiv x_1 = 0 \wedge x_2 = 0 \wedge \cdots \wedge x_{d-1} = 0$$

2. Para representar $R_i$ de aridad $a_i$:

$$\begin{aligned} R_i(x_1, \ldots, x_d) \quad \equiv \quad & (x_1 = 0 \wedge \cdots \wedge x_{d-a_i} = 0 \wedge R_i(x_{d-a_i+1}, \ldots, x_d)) \vee \\ & (x_1 = 0 \wedge \cdots \wedge x_{d-a_i-1} = 0 \wedge x_{d-a_i} = 1 \wedge \\ & x_{d-a_i+1} = 0 \wedge \cdots \wedge x_d = 0) \end{aligned}$$

   La segunda parte de la disyunción es para representar el 1 que se agrega en la codificación de la relación $R_i$ para no perder su longitud.

3. Para representar $c_i$, pertenecen los $(x_1, \ldots, x_d)$ tal que $x_1 \cdot n^{d-1} + x_2 \cdot n^{d-2} + \cdots + x_d = c_i$:

$$\begin{aligned} C_i(x_1, \ldots, x_d) \quad \equiv \quad & x_1 = c_i \bmod n^{d-1} \wedge \\ & x_2 = c_i \bmod n^{d-2} - c_i \operatorname{div} n^{d-1} * 10 \wedge \\ & \cdots \wedge \\ & x_d = c_i \bmod n^0 - c_i \operatorname{div} n * 10 \end{aligned}$$

## 2.5. Sustitución en Fórmulas

Para poder representar los operadores $SCOMP$ y $SRN$ mediante fórmulas en **FO(IFP)** introduciremos a continuación la idea de sustitución en fórmulas. Esto resultó necesario debido a que, si bien las fórmulas en **FO(IFP)** pueden tener variables de segundo orden libres, no pueden tener subfórmulas libres. Con la sustitución de fórmulas se elimina la ambigüedad que puede surgir con las fórmulas que representan el operador de composición o la recursión.

**Definición 2.17.** Presentaremos por inducción en $G$ el significado de:

$$\begin{array}{l} G[\overline{X}](\overline{x}) \\ \{ \\ \quad X_i \rightsquigarrow H[\overline{Y}](\overline{y}) \\ \} \end{array}$$

con $G, H$ fórmulas en **FO(IFP)**, $\overline{X}, \overline{Y}$ conjuntos de variables de segundo orden, y $\overline{x} = x_1, \ldots, x_{a_x}$, $\overline{y} = y_1, \ldots, y_{a_y}$.

- Si $G$ es una fórmula atómica: $R(z_1, \ldots, z_{a_z})$
  - Si $R \neq X_i$:

    $$\begin{array}{lll} G[\overline{X}](\overline{x}) & & \equiv G[\overline{X}](\overline{x}) \\ \{ & & \\ \quad X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & & \\ \} & & \end{array}$$

    La sustitución no modifica nada.
  - Si $R = X_i$:

    $$\begin{array}{lll} G[\overline{X}](\overline{x}) & & \equiv H[\overline{Y}](\overline{y}, z_{a_y+1}, \ldots, z_{a_z}) \\ \{ & & \\ \quad X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & & \\ \} & & \end{array}$$

    Se sustituye $R$ por $H[\overline{Y}]$ y las primeras $a_y$ variables de $H$ por $\overline{y}$
- Si $G$ es $t_1 = t_2$, donde $t_1$ y $t_2$ son términos

  $$\begin{array}{lll} G[\overline{X}](\overline{x}) & & \equiv G[\overline{X}](\overline{x}) \\ \{ & & \\ \quad X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & & \\ \} & & \end{array}$$

- Si $G \equiv \neg G'$:

  $$\begin{array}{lll} \neg G'[\overline{X}](\overline{x}) & & \equiv \neg(G'[\overline{X}](\overline{x}) \\ \{ & & \quad \{ \\ \quad X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & & \quad\quad X_i \rightsquigarrow H[\overline{Y}](\overline{y}) \\ \} & & \quad \}) \end{array}$$

- Si $G \equiv (G_1 \vee G_2)$:

$$\begin{array}{lll} (G_1 \vee G_2)[\overline{X}](\overline{x}) & \equiv (G_1[\overline{X}](\overline{x}) & \vee \quad (G_2[\overline{X}](\overline{x}) \\ \{ & \{ & \{ \\ X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & X_i \rightsquigarrow H[\overline{Y}](\overline{y}) \\ \} & \}) & \}) \end{array}$$

- Si $G \equiv \exists x\, G'$:

$$\begin{array}{lll} \exists x\ G'[\overline{X}](\overline{x}) & \equiv \exists x\ (G'[\overline{X}](\overline{x}) \\ \{ & \{ \\ X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & X_i \rightsquigarrow H[\overline{Y}](\overline{y}) \\ \} & \}) \end{array}$$

- Si $G \equiv [IFP_{\overline{y},Y}G']$:
  - Si $X_i = Y$:

$$\begin{array}{lll} [IFP_{\overline{y},Y}G'][\overline{X}](\overline{x}) & \equiv G[\overline{X}](\overline{x}) \\ \{ \\ X_i \rightsquigarrow H[\overline{Y}](\overline{y}) \\ \} \end{array}$$

  - Si $X_i \neq Y$:

$$\begin{array}{lll} [IFP_{\overline{y},Y}G'][\overline{X}](\overline{x}) & \equiv & [IFP_{\overline{y},Y}(G' \\ \{ & & \{ \\ X_i \rightsquigarrow H[\overline{Y}](\overline{y}) & & X_i \rightsquigarrow H[\overline{Y}](\overline{y}) \\ \} & & \})][\overline{X}](\overline{x}) \end{array}$$

## 2.6. Construcción de la fórmula en FO(IFP)

A continuación definiremos la construcción de la fórmula para un programa $\mathcal{P} = \langle f, C \rangle$. Definiremos la fórmula que representa $g$ para toda función $g \in \{f\} \cup C$.

### 2.6.1. Cero

Para la función 0(;), definimos la fórmula:

$$0_*(x_1, \ldots, x_d) \equiv x_1 \neq x_1$$

Como $0_*(x_1, \ldots, x_d)$ es falsa para todo $x_1, \ldots, x_d$, esto representa que no hay ningún uno en la codificación binaria del resultado, con lo que obtenemos la representación del 0 en base dos.

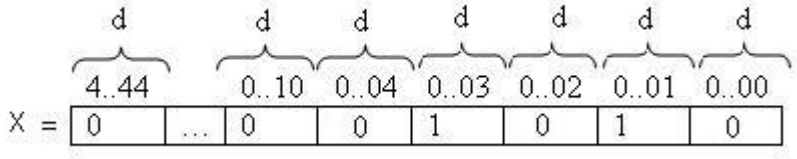
### 2.6.2. $S_0$

Para la función $S_0(;x)$ definimos la fórmula:

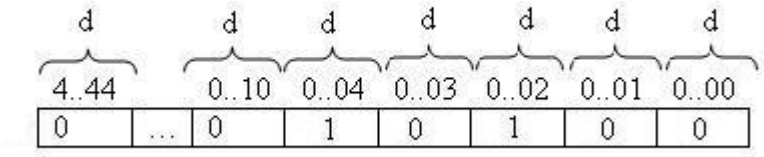
$$\begin{aligned} S_{0_*}[X](x_1, \ldots, x_d) \equiv\ & \exists y_1, \ldots, y_d\, X y_1, \ldots y_d \wedge \\ & \bigvee_{0 \le i \le d} (y_{i+1} = \ldots = y_d = \text{máx} \wedge \neg y_i = \text{máx} \\ & \wedge S(y_i, x_i) \wedge \bigwedge_{i<j\le d} x_j = \text{mín} \wedge \bigwedge_{0 \le j < i} y_j = x_j) \end{aligned}$$

$X$ es la fórmula que representa el valor en el que se aplica $S_0(;x)$.

**Ejemplo 2.18.** Si $X$ representa el número 1010:




La fórmula que obtenemos al aplicar $S_0$ va a ser verdadera para los $x_1, \ldots, x_d$ tales que verifiquen $S_{0_*}[X](x_1, \ldots, x_d)$:




### 2.6.3. $S_1$

Para la función $S_1(;x)$ definimos la fórmula:

$$S_{1_*}[X](x_1, \ldots, x_d) \equiv S_{0_*}[X](x_1, \ldots, x_d) \vee (x_1 = \cdots = x_d = 0)$$

### 2.6.4. $\pi_j^{n,m}$

Para la función $\pi_j^{n,m}(x_1, \ldots, x_n; x_{n+1}, \ldots, x_{n+m})$ definimos la fórmula:

$$\pi_{j_*}^{n,m}[X_1, \ldots, X_{n+m}](x_1, \ldots, x_d) \equiv X_j(x_1, \ldots, x_d)$$

### 2.6.5. $C$

Para la función $C(; a, b, c)$ definimos la fórmula:

$$C_*[A, B, C](x_1, \ldots, x_d) \equiv (\neg A(0, \ldots, 0) \wedge B(x_1, \ldots, x_d)) \vee (A(0, \ldots, 0) \wedge C(x_1, \ldots, x_d))$$

Si $A$ representa un número par, entonces el resultado va a estar dado por $B(x_1, \ldots, x_d)$, en caso contrario va a estar dado por $C(x_1, \ldots, x_d)$.

### 2.6.6. $p$

Para la función $p(; x)$ definimos la fórmula:

$$\begin{aligned} p_*[X](x_1, \ldots, x_d) \quad \equiv \quad & \exists y_1 \ldots, y_d \, X y_1, \ldots, y_d \wedge \\ & \bigvee_{0 \leq i \leq d} (x_{i+1} = \ldots = x_d = \text{máx} \wedge \neg x_i = \text{máx} \\ & \wedge S(x_i, y_i) \wedge \bigwedge_{i < j \leq d} y_j = \text{mín} \wedge \bigwedge_{0 \leq j < i} y_j = x_j) \end{aligned}$$

**Observación 2.19.** Si $X$ representa el cero, no existen $(y_1, \ldots, y_d) \in X$, por lo tanto, el resultado también va a representar el cero.

### 2.6.7. Composición Segura

Si $f$ está dada por composición segura:

$f(\overline{x}; \overline{a}) = h(\overline{r}(\overline{x};); \overline{t}(\overline{x}; \overline{a}))$

$$\begin{aligned} f(x_1, \ldots, x_{n_1}; a_1, \ldots, a_{n_2}) \quad = \quad & h(\overline{r}(x_1, \ldots, x_{n_1};); \\ & \overline{t}(x_1, \ldots, x_{n_1}; a_1, \ldots, a_{n_2})) \end{aligned}$$

Con $\overline{r} = r_1, \ldots, r_{k_1}$ y $\overline{t} = t_1, \ldots, t_{k_2}$.

Por hipotesis inductiva existen:

- $R_{i_*}[Y_1, \ldots, Y_{n_1}](y_1, \ldots, y_d)$
- $T_{j_*}[Y_1, \ldots, Y_{n_1}, Z_1, \ldots, Z_{n_2}](y_1, \ldots, y_d)$
- $H_*[R_1, \ldots, R_{k_1}, T_1, \ldots, T_{k_2}](x_1, \ldots, x_d)$

Entonces definiremos:

$$\begin{array}{rcl} F_*[X_1 \ldots, X_{n_1}, A_1, \ldots, A_{n_2}](x_1, \ldots, x_d) & \equiv & H_*[R_1, \ldots, R_{k_1}, T_1, \ldots, T_{k_2}](x_1, \ldots, x_d) \\ & \{ & R_i \rightsquigarrow R_{i_*}[X_1, \ldots, X_{n_1}] \\ & & T_i \rightsquigarrow T_{i_*}[X_1, \ldots, X_{n_1}, A_1, \ldots, A_{n_2}] \\ & \} & \end{array}$$

Esto significa que $F_*$ va a ser equivalente al resultado de reemplazar cada aparición de $R_i$ por $R_{i_*}$, y cada aparición de $T_i$ por $T_{i_*}$ en $H_*$.

**Ejemplo 2.20.**

$$f(x_1, x_2; x_3, x_4) = \pi^{2,2}_2(\pi^{2,0}_2(x_1, x_2;), \pi^{2,0}_1(x_1, x_2;); \pi^{2,2}_4(x_1, x_2; x_3, x_4), \pi^{2,2}_3(x_1, x_2; x_3, x_4))$$

- $\pi^{2,0}_{2_*}[X_1, X_2](x_1, \ldots, x_d) \equiv X_2(x_1, \ldots, x_d)$
- $\pi^{2,0}_{1_*}[X_1, X_2](x_1, \ldots, x_d) \equiv X_1(x_1, \ldots, x_d)$
- $\pi^{2,2}_{4_*}[X_1, X_2, X_3, X_4](x_1, \ldots, x_d) \equiv X_4(x_1, \ldots, x_d)$
- $\pi^{2,2}_{3_*}[X_1, X_2, X_3, X_4](x_1, \ldots, x_d) \equiv X_3(x_1, \ldots, x_d)$
- $\pi^{2,2}_{2_*}[R_1, R_2, T_1, T_2](x_1, \ldots, x_d) \equiv R_2(x_1, \ldots, x_d)$

Defino:

$$\begin{array}{rcl} F_*[X_1, X_2, X_3, X_4](x_1, \ldots, x_d) & \equiv & \pi^{2,2}_{2_*}[R_1, R_2, T_1, T_2](x_1, \ldots, x_d) \\ & \{ & R_1 \rightsquigarrow \pi^{2,0}_{2_*}[X_1, X_2] \\ & & R_2 \rightsquigarrow \pi^{2,0}_{1_*}[X_1, X_2] \\ & & T_1 \rightsquigarrow \pi^{2,2}_{4_*}[X_1, X_2, X_3, X_4] \\ & & T_2 \rightsquigarrow \pi^{2,2}_{3_*}[X_1, X_2, X_3, X_4] \\ & \} & \end{array}$$

Entonces:

$$F_*[X_1, X_2, X_3, X_4](x_1, \ldots, x_d) \equiv \pi^{2,0}_{1_*}[X_1, X_2](x_1, \ldots, x_d)$$

### 2.6.8. Recursión Predicativa en Notación

A continuación presentaremos la construcción de la fórmula que representa una función definida mediante el operador de Recursión Predicativa en Notación, el paso de mayor dificultad en la inclusión **B** $\subseteq$ **FO(IFP)**. Esta construcción requiere la definición de algunas fórmulas auxiliares para llegar finalmente a una definición clara de la fórmula que buscamos construir.

Para representar la recursión predicativa en notación utilizaremos el operador de punto fijo inflacionario de **FO(IFP)**. La idea es la siguiente:

Dada $f(z, \overline{y}; \overline{a})$

$$\begin{aligned} f(0, \overline{y}; \overline{a}) &= g(\overline{y}; \overline{a}) \\ f(zi, \overline{y}; \overline{a}) &= h_i(z, \overline{y}; \overline{a}, f(z, \overline{y}; \overline{a})) \end{aligned}$$

donde $f(z, \overline{y}; \overline{a}) = w$ con la representación binaria de $w = w_{|w|-1} \dots w_0$, se utilizará una relación $X$ que contendrá las tuplas $(c_1, \dots, c_d, x_1, \dots, x_d)$ tales que $c_1, \dots, c_d$ en base $n$ indican el bit de $z$ que se está evaluando y el bit $i = x_1 \cdot n^{d-1} + \dots + x_d$ del resultado en este paso sea uno ($w_i = 1$).

Si $z$ es un número $b_1 \dots b_{|z|}$ entonces en el paso 0 se obtiene $g(\overline{y}; \overline{a})$, luego en el paso 1 $h_{b_1}(1, \overline{y}; \overline{a}, g(\overline{y}; \overline{a}))$ y así hasta obtener el valor en $z$.

Observar que si en algún paso el resultado de la función es 0, entonces la relación en ese paso será vacía. Agregamos una variable ($y_0$) que nos indica si la relación en un determinado paso es vacía o no.

Utilizaremos las siguientes fórmulas auxiliares:

Para representar $(x_0 \cdot n^{d-1} + x_1 \cdot n^{d-2} + \dots + x_d \cdot n^0) + 1 = y_0 \cdot n^{d-1} + y_1 \cdot n^{d-2} + \dots + y_d \cdot n^0$, definiremos la relación $S^d(x_0, \dots, x_d, y_0, \dots, y_d)$ como:

$$S^d(x_0, \dots, x_d, y_0, \dots, y_d) \equiv \bigvee_{0 \le i \le d} (x_{i+1} = \dots = x_d = \text{máx} \wedge \neg x_i = \text{máx}$$

$$\wedge S(x_i, y_i) \wedge \bigwedge_{i < j \le d} y_j = \text{mín} \wedge \bigwedge_{0 \le j < i} y_j = x_j)$$

Luego, para representar la relación $x + y = z$, se define

$$ADD^d(x_0, \dots, x_d, y_0, \dots, y_d, z_0, \dots, z_d)$$

de la siguiente manera:

$$[IFP_{\overline{xyz}, X}((\overline{y} = \overline{\text{mín}} \wedge \overline{z} = \overline{x}) \vee \exists \overline{v} \exists \overline{w}(X(\overline{x}, \overline{v}, \overline{w}) \wedge S^d(\overline{v}, \overline{y}) \wedge S^d(\overline{w}, \overline{z})))]\overline{xyz}$$

La fórmula $MaximoUno_d[N](p_1, \ldots, p_d)$ que vale para los $p_1, \ldots, p_d$ tales que en el bit $p_1 \cdot n^{d-1} + \cdots + p_d$ de $N$ hay un uno y es el primero en el número evaluando los bits de izquierda a derecha:

$$\begin{aligned} MaximoUno_d[N](p_1, \ldots, p_d) \quad \equiv \quad & N(p_1, \ldots, p_d) \wedge \\ & (\forall q_1 \ldots q_d \, N(q_1, \ldots, q_d) \rightarrow \geq (p_1, \ldots, p_d, q_1, \ldots, q_d) \end{aligned}$$

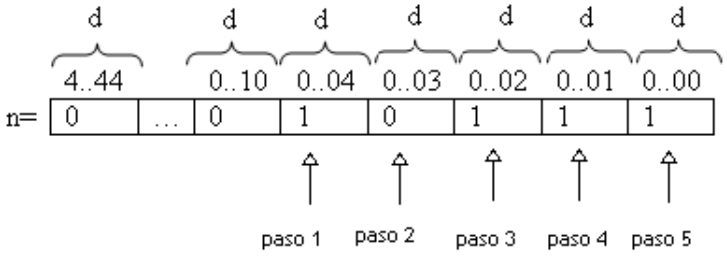
$$\geq (p_1, \ldots, p_d, q_1, \ldots, q_d) \equiv \exists r_1 \ldots r_d \, Add_d(q_1, \ldots, q_d, r_1, \ldots, r_d, p_1, \ldots, p_d)$$

La siguiente fórmula nos servirá para evaluar si en el bit $t = c_1 \cdot n^{d-1} + \cdots + c_d$ desde el lugar en el que se encuentra el uno más significativo (utilizando *MaximoUno*) hay un uno.

$$\begin{aligned} bit[N](c_1, \ldots, c_d) \quad \equiv \quad & \exists p_1, \ldots, p_d \, MaximoUno[N](p_1, \ldots, p_d) \wedge \\ & \exists w_1, \ldots, w_d \\ & [IFP_{q_1 \ldots q_d m_1 \ldots m_d, X} \\ & (q_1 = \cdots = q_d = 0 \wedge m_1 = p_1 \wedge \cdots \wedge m_d = p_d) \vee \\ & (\exists r_1, \ldots, r_d, n_1, \ldots, n_d \, X r_1 \ldots r_d n_1 \ldots n_d \wedge \\ & S_d(r_1, \ldots, r_d, q_1, \ldots, q_d) \wedge \\ & S_d(m_1, \ldots, m_d, n_1, \ldots, n_d))] c_1 \ldots c_d w_1 \ldots w_d \wedge \\ & N(w_1, \ldots, w_d) \end{aligned}$$

En cada paso de la recursión se evalua $h_0$ o $h_1$ con el valor actual de *z*, para esto hay que tomar una subcadena de *z*.

**Ejemplo 2.21.** Si la recursión se hace sobre un número $z = 10111$, entonces el paso cero es el caso base de la recursión, y en cada paso *i* evaluaremos la subcadena a partir del bit más significativo de *z* de *i* dígitos:




- $Subcadena[Z](0, x_1, \ldots, x_d) \equiv \emptyset$ (0)

- $Subcadena[Z](1, x_1, \ldots, x_d) \equiv (x_1 = \cdots = x_d = 0)$ (1)
- $Subcadena[Z](2, x_1, \ldots, x_d) \equiv (x_1 = \cdots = x_{d-1} = 0 \wedge x_d = 1)$ (10)
- $Subcadena[Z](3, x_1, \ldots, x_d) \equiv (x_1 = \cdots = x_d = 0) \vee (x_1 = \ldots x_{d-1} = 0 \wedge x_d = 2)$ (101)

$$
\begin{aligned}
&Subcadena[Z](f_1, \ldots, f_d, f'_1, \ldots, f'_d) \equiv \\
&\quad [IFP_{e_1,\ldots,e_d,x_1,\ldots,x_d,X}(e_0 = \cdots = e_d = 0 \wedge Cero(x_1, \ldots, x_d)) \vee \\
&\quad (\exists c_1, \ldots, c_d, y_1, \ldots, y_d X(c_1, \ldots, c_d, y_1, \ldots, y_d) \wedge S_d(c_1, \ldots, c_d, e_1, \ldots, e_d) \wedge \\
&\quad (bit[Z](e_1, \ldots, e_d) \rightarrow S_1[X'](x_1, \ldots, x_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\ X' \rightsquigarrow X(c_1, \ldots, c_d) \\
&\qquad\qquad \} \quad ) \wedge \\
&\quad (\neg bit[Z](e_1, \ldots, e_d) \rightarrow S_0[X'](x_1, \ldots, x_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\ X' \rightsquigarrow X(c_1, \ldots, c_d) \\
&\qquad\qquad \} \quad ) \\
&\quad ) f_1, \ldots, f_d, f'_1, \ldots, f'_d
\end{aligned}
$$

Para definir la fórmula correspondiente a la recursión definimos las siguientes subfórmulas:

- En el caso base, es decir el paso 0, si $g(\overline{x}; \overline{a}) = 0$ definimos la siguiente fórmula:

$$
\begin{aligned}
Paso_0^{=0}[\overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \quad \equiv \quad & (c_1 = \cdots = c_d = 0) \wedge \\
& (\neg \exists w_1 \ldots w_d\, G[\overline{Y}, \overline{A}](w_1, \ldots, w_d)) \wedge \\
& (y_0 = 1)
\end{aligned}
$$

  Esta fórmula representa el caso base de la recursión con $c_1 = \cdots = c_d = 0$, y que el resultado es 0, es decir, no hay ninguna tupla $(w_1 \ldots w_d)$ que haga verdadera $G$. Como se necesita que haya alguna tupla en la relación para el paso cero, para luego hallar el punto fijo inflacionario, cualquier tupla con $c_1 = \cdots = c_d = 0$ y $y_0 = 1$ hace verdadera la fórmula.

- Cuando el paso es 0 y $g(\overline{x};\overline{a}) \neq 0$ definimos la siguiente fórmula:

$$\begin{aligned} Paso_0^{\neq 0}[\overline{Y},\overline{A}](c_1,\ldots,c_d,y_0,\ldots,y_d) \quad \equiv \quad & (c_1 = \cdots = c_d = 0) \wedge \\ & (\exists w_1 \ldots w_d\, G[\overline{Y},\overline{A}](w_1,\ldots,w_d)) \wedge \\ & (y_0 = 0) \wedge \\ & (G[\overline{Y},\overline{A}](y_1,\ldots,y_d)) \end{aligned}$$

  En este caso los $y_1,\ldots,y_d$ que van a hacer verdadera la fórmula en el paso cero van a ser aquellos que hacen verdadero $G$ con los valores que representan las relaciones dadas por $\overline{Y},\overline{A}$.

- Cuando el paso es $p = k+1$, el bit en la posición $p$ es un cero y el resultado en este paso de la recursión es cero definimos la siguiente fórmula:

$$\begin{aligned} & Paso_{p,0}^{=0}[X,Z,\overline{Y},\overline{A}](c_1,\ldots,c_d,y_0,\ldots,y_d) \equiv \\ & \quad \exists e_1,\ldots,e_d,x_1,\ldots,x_d \quad X(e_1,\ldots,e_d,x_1,\ldots,x_d) \wedge \\ & \quad S_d(e_1,\ldots,e_d,c_1,\ldots,c_d) \wedge \neg bit[Z](c_1,\ldots,c_d) \wedge \\ & \quad (\neg\exists z_1,\ldots,z_d\, H_0[Z',\overline{Y},\overline{A},F](z_1,\ldots,z_d) \\ & \qquad\qquad \{ \\ & \qquad\qquad\ Z' \rightsquigarrow Subcadena[Z](c_1,\ldots,c_d) \\ & \qquad\qquad\ F \rightsquigarrow X(e_1,\ldots,e_d,0) \\ & \qquad\qquad \} \\ & \quad ) \wedge y_0 = 1 \end{aligned}$$

- Cuando el paso es $p = k+1$, el bit en la posición $p$ es un cero y el resultado en este paso es distinto de cero definimos la siguiente fórmula:

$$
\begin{aligned}
&Paso^{\neq 0}_{p,0}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \equiv \\
&\quad \exists e_1, \ldots, e_d, x_1, \ldots, x_d \quad X(e_1, \ldots, e_d, x_1, \ldots, x_d) \wedge \\
&\quad S_d(e_1, \ldots, e_d, c_1, \ldots, c_d) \wedge \neg bit[Z](c_1, \ldots, c_d) \wedge \\
&\quad (\exists z_1, \ldots, z_d \, H_0[Z', \overline{Y}, \overline{A}, F](z_1, \ldots, z_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\quad Z' \rightsquigarrow Subcadena[Z](c_1, \ldots, c_d) \\
&\qquad\qquad\quad F \rightsquigarrow X(e_1, \ldots, e_d, 0) \\
&\qquad\qquad \} \\
&\quad ) \wedge y_0 = 0 \wedge H_0[Z', \overline{Y}, \overline{A}, F](y_1, \ldots, y_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\quad Z' \rightsquigarrow Subcadena[Z](c_1, \ldots, c_d) \\
&\qquad\qquad\quad F \rightsquigarrow X(e_1, \ldots, e_d, 0) \\
&\qquad\qquad \}
\end{aligned}
$$

- Cuando el paso es $p = k + 1$, el bit en la posición $p$ es uno y el resultado en este paso de la recursión es cero definimos:

$$
\begin{aligned}
&Paso^{=0}_{p,1}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \equiv \\
&\quad \exists e_1, \ldots, e_d, x_1, \ldots, x_d \quad X(e_1, \ldots, e_d, x_1, \ldots, x_d) \wedge \\
&\quad S_d(e_1, \ldots, e_d, c_1, \ldots, c_d) \wedge bit[Z](c_1, \ldots, c_d) \wedge \\
&\quad (\neg \exists z_1, \ldots, z_d \, H_1[Z', \overline{Y}, \overline{A}, F](z_1, \ldots, z_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\quad Z' \rightsquigarrow Subcadena[Z](c_1, \ldots, c_d) \\
&\qquad\qquad\quad F \rightsquigarrow X(e_1, \ldots, e_d, 0) \\
&\qquad\qquad \} \\
&\quad ) \wedge y_0 = 1
\end{aligned}
$$

- Cuando el paso es $p = k + 1$, el bit en la posición $p$ es uno y el resultado en este

paso es distinto de cero definimos:

$$
\begin{aligned}
&Paso^{\neq 0}_{p,1}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \equiv \\
&\quad \exists e_1, \ldots, e_d, x_1, \ldots, x_d \quad X(e_1, \ldots, e_d, x_1, \ldots, x_d) \wedge \\
&\quad S_d(e_1, \ldots, e_d, c_1, \ldots, c_d) \wedge bit[Z](c_1, \ldots, c_d) \wedge \\
&\quad (\exists z_1, \ldots, z_d \, H_1[Z', \overline{Y}, \overline{A}, F](z_1, \ldots, z_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\quad Z' \rightsquigarrow Subcadena[Z](c_1, \ldots, c_d) \\
&\qquad\qquad\quad F \rightsquigarrow X(e_1, \ldots, e_d, 0) \\
&\qquad\qquad \} \\
&\quad ) \wedge y_0 = 0 \wedge H_1[Z', \overline{Y}, \overline{A}, F](y_1, \ldots, y_d) \\
&\qquad\qquad \{ \\
&\qquad\qquad\quad Z' \rightsquigarrow Subcadena[Z](c_1, \ldots, c_d) \\
&\qquad\qquad\quad F \rightsquigarrow X(e_1, \ldots, e_d, 0) \\
&\qquad\qquad \}
\end{aligned}
$$

Utilizando las fórmulas anteriores, podemos ahora definir la fórmula para representar el resultado de la recusión por medio de una disyunción en la cual la primer parte representa el caso en que el valor sobre el que se hace la recursión es cero, y la segunda parte el caso en que es mayor a cero:

$$
\begin{aligned}
F[Z, \overline{X}, \overline{A}](x_1, \ldots, x_d) \quad \equiv \quad & (\neg \exists w_1, \ldots, w_d \quad MaximoUno[Z](w_1, \ldots, w_d) \wedge \\
& \qquad\qquad G[\overline{X}, \overline{A}](x_1, \ldots, x_d)) \vee \\
& (\exists w_1, \ldots, w_d \quad MaximoUno[Z](w_1, \ldots, w_d) \wedge \\
& [IFP_{c_1, \ldots, c_d, y_0, \ldots, y_d, X} \\
& \quad Paso^{=0}_0[\overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \vee \\
& \quad Paso^{\neq 0}_0[\overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \vee \\
& \quad Paso^{=0}_{p,0}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \vee \\
& \quad Paso^{\neq 0}_{p,0}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \vee \\
& \quad Paso^{=0}_{p,1}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d) \vee \\
& \quad Paso^{\neq 0}_{p,1}[X, Z, \overline{Y}, \overline{A}](c_1, \ldots, c_d, y_0, \ldots, y_d)] \\
& \quad w_1, \ldots, w_d, x_1, \ldots, x_d)
\end{aligned}
$$

## 2.7. Fórmula Principal

Para poder obtener la fórmula $\varphi$ en **FO(IFP)** que determina la misma clase de estructuras que la función principal $f$ de un programa $\mathcal{P} = \langle f, C \rangle$, el primer paso consiste en obtener las fórmulas correspondientes a cada función $g \in \{f\} \cup C$, utilizando las definiciones presentadas para cada función básica y operador $(0, S_0, S_1, p, C, \pi, SRN, SCOMP)$.

A partir de la fórmula $F$ que representa el valor obtenido por la función principal del programa, definiremos la siguiente fórmula como la fórmula que define la misma clase de estructuras que el programa en **B**, $\varphi$ vale cuando el resultado de la función es distinto de cero:

$$\varphi \equiv \exists x_1, \ldots, x_d F(x_1, \ldots, x_d)$$

Donde en $F$ pueden aparecer $N$, $R_i$, $C_j$, que se definen como en 1, 2, 3, en la sección 2.4, página 28. Se aplica la sustitución en fórmulas sustituyendo cada apariciòn de estas variables por sus fórmulas correspondientes.

**Teorema 2.22.** *Para cualquier vocabulario $\tau$ y función $f \in$ **B** existe una fórmula $\varphi \in$ **FO(IFP)** tal que para toda $\tau$-estructura $\mathcal{A}$*

$$f(bin(\mathcal{A});) > 0 \text{ si y solo si } \mathcal{A} \models \varphi$$

*Demostración.* Según la definición de $f$:

- Si $f$ es una función inicial:
  - Si $f$ es la función $0(;)$, se define $\varphi$ tal como en la sección 2.6.1.
  - Si $f$ es la función $S_0(; x)$, se define $\varphi$ tal como en la sección 2.6.2.
  - Si $f$ es la función $S_1(; x)$, se define $\varphi$ tal como en la sección 2.6.3.
  - Si $f$ es la función $\pi_j^{n,m}(\overline{x}; \overline{a})$, se define $\varphi$ tal como en la sección 2.6.4.
  - Si $f$ es la función $C(; a, b, c)$, se define $\varphi$ tal como en la sección 2.6.5.
  - Si $f$ es la función $p(; x)$, se define $\varphi$ tal como en la sección 2.6.6.
- Si $f$ está definida por un operador:
  - Si $f$ está definida por composición segura, se define $\varphi$ tal como en la sección 2.6.7.
  - Si $f$ está definida por recursión predicativa en notación, se define $\varphi$ tal como en la sección 2.6.8.

□

**Corolario 2.23.** *Toda clase de estructuras que se puede determinar por un programa en* ***B****, puede determinarse por una fórmula en* ***FO(IFP)****, es decir* $\mathcal{T}_{\mathbf{B}} \subseteq \mathcal{T}_{\mathbf{FO(IFP)}}$.

*Demostración.* Un programa $\mathcal{P} = \langle f, C \rangle$ determina una clase de estructuras $\mathcal{K}$, cuando $f$ determina $\mathcal{K}$, y, como ya vimos, para toda función $f$ en **B** existe una fórmula en **FO(IFP)** que caracteriza la misma clase, $\mathcal{T}_{\mathbf{B}} \subseteq \mathcal{T}_{\mathbf{FO(IFP)}}$. □

# 3. FO(IFP) $\subseteq$ B

Para obtener el programa en **B** que, con input $bin(\mathcal{A})$, devuelve 1 si $\mathcal{A} \models \varphi$ y devuelve 0 en caso contrario, se define un conjunto de funciones que se utilizarán según la construcción de la fórmula. Por ejemplo para la fórmula:

$$\varphi_{CONN} \equiv \forall x, y\,[IFP_{xy,X}(Exy \vee \exists z(Exz \wedge Xzy))]xy$$

Su construcción está dada por:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{Exz}\ (1) \qquad \overline{Xzy}\ (2)}{Exz \wedge Xzy}\ (3)}{\exists z(Exz \wedge Xzy)}\ (5)}{Exy \vee \exists z(Exz \wedge Xzy)}\ (6)}{[\mathrm{IFP}_{xy,X}Exy \vee \exists z(Exz \wedge Xzy)]xy}\ (7)}{\forall y\,[\mathrm{IFP}_{xy,X}Exy \vee \exists z(Exz \wedge Xzy)]xy}\ (8)}{\forall x, y\,[\mathrm{IFP}_{xy,X}Exy \vee \exists z(Exz \wedge Xzy)]xy}\ (9)$$

Se realizan funciones para: (1),(2),...,(9). Donde la función principal de (9) será la función principal del programa.

En la sección 3.1 presentaremos una biblioteca de funciones que utilizaremos luego para trabajar con la codificación de las relaciones. En la sección 3.2 definiremos las funciones que representarán las transformaciones de las relaciones (como la negación, la disyunción). En las secciones 3.3, 3.4, 3.5, 3.6, 3.7, 3.8 y 3.9 definiremos la construcción de las funciones que formarán parte del programa en **B** que determine la misma clase de $\tau$-estructuras.

## 3.1. Funciones básicas

Para explicar la idea de las funciones se utilizará la siguiente definición de longitud:

**Definición 3.1.** La función $|\cdot|$ se define de la siguiente manera:

- $|0| = 0$
- $|wi| = |w| + 1$ para $wi \neq 0$

1. $P(x;y)$ representa $\lfloor y/2^{|x|} \rfloor$

$$\begin{aligned} P(0;y) &= y \\ P(xi;y) &= p(;P(x;y)) \end{aligned}$$

2. $ones(x;)$ devuelve $|x|$ 1's

$$\begin{aligned} ones(0;) &= 0 \\ ones(xi;) &= S_1(;ones(x;)) \end{aligned}$$

3. $pad(x;y)$ devuelve $y \times 2^{|x|}$

$$\begin{aligned} pad(0;y) &= y \\ pad(xi;y) &= S_0(;pad(x;y)) \end{aligned}$$

4. $\oplus(x;y)$ devuelve $x$ concatenado al final de $y$, para $x$ distinto de 0

$$\begin{aligned} \oplus(0;y) &= y \\ \oplus(xi;y) &= S_i(;\oplus(x;y)) \end{aligned}$$

5. $sg(x;)$ devuelve 0 cuando $x$ es cero, 1 cuando $x$ es distinto de cero

$$sg(x;) = C(;ones(x;),0,1)$$

6. $\overline{sg}(x;)$ devuelve 1 cuando $x$ es cero, y 0 cuando $x$ es distinto a cero (al revés de $sg(x;)$)

$$\overline{sg}(x;) = C(;sg(x;),1,0)$$

7. Se definen algunos operadores lógicos:

$$\begin{aligned} not(x;) &= \overline{sg}(x;) \\ and(x,y;) &= C(;sg(x;),0,C(;sg(y;),0,1)) \\ or(x,y;) &= not(and(not(x;),not(y;););) \\ xor(x,y;) &= or(and(x,y;),and(not(x;),not(y;););) \end{aligned}$$

8. Para luego poder definir la igualdad, se define ahora $Eq_b(c;x,y)$ (igual acotado o bounded equality). $Eq_b(c;x,y)$ devuelve 1 cuando los últimos $|c|$ bits de $x$ e $y$ son iguales, 0 en caso contrario

$$\begin{aligned} &Eq_b(0;x,y) = 1 \\ &Eq_b(wi;x,y) = \\ &\quad C(;Eq_b(w;x,y),0,C(;P(w;x),C(P(w;y),1,0),C(P(w;y),0,1))) \end{aligned}$$

9. $Eq(x, y;)$ devuelve 1 si $x$ e $y$ son iguales, 0 en caso contrario

$$Eq(x, y;) = Eq_b(\oplus(x; y); x, y)$$

**Notación 3.2.** Para simplificar notación cuando sea suficientemente claro, escribiremos $x == y$ para representar $Eq(x, y;)$.

10. $last(; x)$ devuelve el último bit de $x$

$$last(; x) = C(; x, 0, 1)$$

11. Algunas funciones relacionadas con la suma.

- $carry(c; x, y)$ devuelve 1 si la suma de los dígitos de $x$ e $y$ en el lugar $|c|$ mirando desde el bit menos significativo recibe acarreo.

$$\begin{aligned}
carry(0; x, y) &= 0 \\
carry(wi; x, y) &= C(; P(w; x), \\
&\quad C(; P(w; y), \\
&\quad\ 0, \\
&\quad\ C(; carry(w; x, y), 0, 1) \\
&\quad ), \\
&\quad C(; P(wi; y), \\
&\quad\ C(; carry(w; x, y), 0, 1), \\
&\quad\ 1) \\
&\quad )
\end{aligned}$$

- $digit\_sum(a, b, c;)$ se va a utilizar para, dado el acarreo y los dos dígitos de la suma, saber qué devuelve

$$digit\_sum(a, b, c;) = xor(xor(a, b;), c;)$$

- $+_{b_{dupl\_cota}}(r, c; x, y)$ realiza la suma entre $x$ e $y$, utilizando $r$ para la recursión y $c$ como la cota.

$$\begin{aligned}
+_{b_{dupl_cota}}(0, c; x, y) &= C(; digit_sum(last(; P(c; x)), \\
&\quad last(; P(c; y)), \\
&\quad carry(p(; c); x, y);), 0, 1) \\
+_{b_{dupl_cota}}(wi, c; x, y) &= C(; digit_sum(last(; P(P(wi; c); x)), \\
&\quad last(; P(P(wi; c); y)), \\
&\quad carry(P(wi; c); x, y);), \\
&\quad S_0(; +_{b_{dupl_cota}}(w, c; x, y)), \\
&\quad S_1(; +_{b_{dupl_cota}}(w, c; x, y)))
\end{aligned}$$

- $+_b(c; x, y)$ realiza la suma de los últimos $|c|$ dígitos de $x$ e $y$.

$$+_b(c; x, y) = +_{b_{dupl\_cota}}(c, c; x, y)$$

- $+(x, y; )$ devuelve la suma de $x$ e $y$

$$+(x, y; ) = +_b(S_1(; \oplus(x; y)); x, y)$$

**Notación 3.3.** Si es suficientemente claro se utilizará $x + y$ para notar $+(x, y; )$

12. $long(x; )$ devuelve la longitud de $x$ para números distintos de cero.

$$long(0; ) = 1;$$
$$long(xi; ) = +_b(S_1(; x); 1, long(x; ))$$

13. $<_b (b; x, y)$ devuelve 1 si el número dado por los $|b|$ bits menos significativos bits de $x$ es menor que el dado por los $|b|$ bits menos significativos de $y$, 0 en caso contrario.

$$<_b (0; x, y) = 0$$
$$<_b (wi; x, y) =$$
$$C(; P(w; x), C(P(w; y), <_b (w; x, y), 1), C(P(w; y), 0, <_b (w, x, )))$$

14. $< (x, y; )$ devuelve 1 si $x$ es menor que $y$, 0 en caso contrario.

$$< (x, y; ) =<_b (\oplus(x; y); x, y)$$

15. $resta_1(x; y)$ resta unaria : $|y| - |x|$.

$$resta_1(x; y) = P(x; y)$$

16. $div_1(x, y; )$ division unaria : $\lfloor |y|/|x| \rfloor$), el resultado va a ser a lo sumo $x$.

$$div_1(x, y; ) = div_{1_b}(x, x, y; )$$

**Notación 3.4.** $x \text{ div } y = div_1(y; x)$

17. $\leq (x, y; )$ devuelve 1 si $x \leq y$, 0 en caso contrario.

$$\leq (x, y; ) = or(< (x, y; ), Eq(x, y; ); )$$

18. $div_{1_b}(w, x, y; )$ division unaria acotada por $|w|$

$$\begin{aligned} div_{1_b}(0, x, y; ) &= 0 \\ div_{1_b}(wi, x, y; ) &= C(; \leq (\times_1(wi, x; ), y; ), div_{1_b}(w, x; y), wi) \end{aligned}$$

19. $mod_1(x, y;)$ resto unario, devuelve $|y| \mod |x|$

$$mod_1(x, y;) = resta_1(\times_1(div_1(y, x;), x;); y)$$

**Notación 3.5.** $x \bmod y = mod_1(y; x)$

20. $\times_1(x, y;)$ producto unario

$$\times_1(0, y;) = 0$$
$$\times_1(xi, y;) = \oplus(y; \times_1(x, y;))$$

21. $exp^i(n;)$: $|n|^i$

$$exp^0(n;) = 1$$
$$exp^1(n;) = n$$
$$exp^k(n;) = \times_1(n, exp^{k-1}(n;))$$

**Notación 3.6.** $n^k = exp^k(n;)$

**Observación 3.7.** No estamos definiendo la función exponencial, ya que no pertenece a la clase **B**, si en cambio para cada $k$ fijado, la función que devuelve $|n|^k$.

22. $+_1(x; y)$ suma unaria (concatenacion)

$$+_1(x; y) = \oplus(x; y)$$

## 3.2. Funciones para trabajar con relaciones

Dado que las relaciones están codificadas de una manera particular (ver 2.1) se define a continuación una serie de funciones cuyo fin será el de evaluar y modificar las relaciones que intervienen en las fórmulas, para que luego instanciando en una estructura en particular se obtenga el resultado esperado.

Sea $\varphi$ la fórmula perteneciente a **FO(IFP)** para la cual se quiere obtener el programa que devuelve un número mayor a 0 si y solo si $\mathcal{A} \models \varphi$. Sea $\psi$ una subfórmula de $\varphi$. Si $\{x_1, \ldots, x_k, X_1, \ldots, X_h\}$ son las variables libres (tanto de primer como segundo orden) de $\psi$, se definirá una función $f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_k, X_1, \ldots, X_h)$, es decir una función que dependerá de la codificación de la estructura y de las variables libres que recibe. La función $f_\varphi$ no debe tener variables libres más allá de las utilizadas para la codificación de la estructura.

Se definen a continuación algunas funciones que serán útiles para ir modificando las variables que representan relaciones:

1. $arg_i^k$

   En primer lugar, se describirán las funciones que obtienen los distintos argumentos de una relación.

   **Ejemplo 3.8.** Sea $R$ es una relación de aridad $k$ que está codificada por un número $r$, donde el último dígito se corresponde al valor de $R(0,\ldots,0,0)$, el anteúltimo es $R(0,\ldots,0,1)$, $\ldots$, el tercer dígito es $R(n-1,\ldots,n-1,n-2)$, el segundo es $R(n-1,\ldots,n-1,n-1)$ y el primero es un 1 para no perder la longitud de la relación: dado uno de los dígitos (en unario) se obiene cada uno de sus argumentos

   Para el último dígito la función $arg_i^k(n,w;)$ se instancia con $w = 0$ y $arg_i^k(n,0;)$ es cero para todo $i$. En cambio para el penúltimo dígito la función se instancia con $w = 1$ y $arg_i^k(n,1;)$ es igual a cero para todo $i$ distinto de $k$, y uno para $i = k$. Como un último ejemplo, para el tercer dígito $w = n^k - 2$ (recordando que el primer dígito no es un valor de $R$) $arg_i^k(n,n^k-2;)$ es $n-1$ para todo $i$ distinto de $k$, y $n-2$ para $i = k$

$$\begin{aligned} arg_k^k(n,w;) &= long(w \bmod n) \\ arg_{k-i}^k(n,w;) &= long((w \operatorname{div} n^i) \bmod n) \end{aligned}$$

2. $EvalConst_i^k$

   Se presenta una función que dada una relación de aridad $k$ codificada en una variable $r$, devuelve un número que va a representar una relación con el argumento $i$ instanciado en una variable $c$.

   **Ejemplo 3.9.** Si $R$ es una relación ternaria, y $r$ es su codificación, entonces podría ser de interés obtener $R(x,4,y)$. En este caso $k = 3$, $r$ es la codificación de $R$, $i = 2$ y $c = 4$.

$$\begin{aligned} EvalConst_i^k(r,n,c;) &= EvalConst_{i_b}^k(p(;r),r,n,ones(c;);r) \\ EvalConst_{i_b}^k(0,tamr,n,c;r) &= 1 \\ EvalConst_{i_b}^k(wi,tamr,n,c;r) &= C(;arg_i^k(n,p(;P(wi;tamr));) == c, \qquad (7) \\ &\quad EvalConst_{i_b}^k(w,tamr,n,c;r), \\ &\quad C(;P(p(;P(wi;tamr));r), \qquad (8) \\ &\quad S_0(;EvalConst_{i_b}^k(w,tamr,n,c;r)), \\ &\quad S_1(;EvalConst_{i_b}^k(w,tamr,n,c;r)))) \end{aligned}$$

En la línea 7 el objetivo de obtener el argumento $i$ de $p(;P(wi;r))$ es ir recorriendo del último al primer dígito. Entonces se le sacan $wi$ bits a $r$ y uno más por el 1 inicial en la codificación de la relación.

Y en la línea 8 mediante $P(p(;P(wi;r));r)$ se obtien el dígito que corresponde de acuerdo al recorrido de $r$ comenzando a partir del último dígito.

3. $EvalVar^k_{i=j}$

A continuación se define la función cuya utilidad es la de instanciar una variable con el contenido de otra.

**Ejemplo 3.10.** Se puede obtener la relación $R'(x_1, x_2, x_3, x_4, x_6)$ dada por

$$R'(x_1, x_2, x_3, x_4, x_6) = R(x_1, x_2, x_3, x_4, x_2, x_6)$$

llamando a la función $EvalVar^6_{2=5}(r, n;)$ que devuelve un número que es la codificación de $R'$.

$EvalVar^k_{i=j}(r, n;)$ se define para $i < j$ y la variable $x_j$ es la que deja de existir.

$$\begin{aligned}
EvalVar^k_{i=j}(r, n;) &= EvalVar^k_{i=j_b}(p(;r), r, n; r) \\
EvalVar^k_{i=j_b}(0, tamr, n; r) &= 1 \\
EvalVar^k_{i=j_b}(wi, tamr, n; r) &= C(; arg^k_i(n, p(;P(wi;tamr));) == \\
&\quad arg^k_j(n, p(;P(wi;tamr));), \\
&\quad EvalVar^k_{i=j_b}(w, tamr, n; r), \\
&\quad C(;P(p(;P(wi;tamr));r), \\
&\quad S_0(;EvalVar^k_{i=j_b}(w, tamr, n; r)), \\
&\quad S_1(;EvalVar^k_{i=j_b}(w, tamr, n; r))))
\end{aligned}$$

4. $SwapVar^k_{i,j}$

Se define la función para intercambiar el lugar de dos variables en una relación.

**Ejemplo 3.11.** Dada una relación $R(x1, x2, x3, x4, x5)$ se puede obtener

$$R'(y1, y2, y3, y4, y5) = R(y1, y4, y3, y2, y5)$$

mediante la función $SwapVar^5_{2,4}(r, n;)$.

$$
\begin{aligned}
SwapVar^{k}_{i,j}(r,n;) &= SwapVar^{k}_{i,j_b}(p(;r),r,n;r)\\
SwapVar^{k}_{i,j_b}(0,tamr,n;r) &= 1\\
SwapVar^{k}_{i,j_b}(wi,tamr,n;r) &= C(;EvalConst^{1}_{1}(\ldots(EvalConst^{i}_{i}(\ldots(\\
&\quad EvalConst^{j}_{j}(\ldots(\\
&\quad EvalConst^{k}_{k}(tamr,\\
&\quad n,arg^{k}_{k}(n,p(;P(wi;tamr)););r)\ldots),\\
&\quad n,arg^{k}_{i}(n,p(;P(wi;tamr)););)\ldots),\\
&\quad n,arg^{k}_{j}(n,p(;P(wi;tamr)););)\ldots),\\
&\quad n,arg^{k}_{1}(n,p(;P(wi;tamr)););),\\
&\quad S_0(;SwapVar^{k}_{i,j_b}(w,tamr,n;r)),\\
&\quad S_1(;SwapVar^{k}_{i,j_b}(w,tamr,n;r)))
\end{aligned}
$$

5. *AddVar*

   A continuación se presenta la función que incrementa en uno la aridad de la función.

   **Ejemplo 3.12.** Dada una relación $R(y1,y2)$ llamando a $AddVar(r;)$ se obtiene la codificación de $R'(x1,x2,x3) = R(x2,x3)$

$$
\begin{aligned}
AddVar(r;) &= AddVar_b(p(;r),r;r)\\
AddVar_b(0,tamr;r) &= r\\
AddVar_b(wi,tamr;r) &= C(;P(p(;P(wi;tamr));r),\\
&\quad S_0(;AddVar_b(w,tamr;r)),\\
&\quad S_1(;AddVar_b(w,tamr;r)))
\end{aligned}
$$

6. $rel_{=}$

   Con la función $rel_{=}(n;)$ se genera la codificación de la relación de igualdad.

$$
\begin{aligned}
rel_{=}(n;) &= rel_{=_b}(n,n;)\\
rel_{=_b}(0,n;) &= 1\\
rel_{=_b}(wi,n;) &= rel_{=_{baux}}(n,wi,n;rel_{=_b}(w,n;))\\
rel_{=_{baux}}(0,m,n;z) &= z\\
rel_{=_{baux}}(wi,m,n;z) &= C(;ones(P(w;n);) == ones(P(p(;m);n);),\\
&\quad S_0(;rel_{=_{baux}}(w,m,n;z)),S_1(;rel_{=_{baux}}(w,m,n;z)))
\end{aligned}
$$

7. $or_{rel}$

   Se define la función $or_{rel}$ que devuelve la aplicación del conectivo binario entre la codificación de dos fórmulas ($q$,$r$) que tienen las mismas variables y en el mismo orden.

$$
\begin{aligned}
or_{rel}(q, r; ) &= or_{rel_b}(p(; q); q, r) \\
or_{rel_b}(0; q, r) &= 1 \\
or_{rel_b}(wi; q, r) &= C(; P(p(; P(wi; q)); q), \\
&\quad C(; P(p(; P(wi; r)); r), \\
&\quad S_0(; or_{rel_b}(w; q, r)), \\
&\quad S_1(; or_{rel_b}(w; q, r))), \\
&\quad S_1(; or_{rel_b}(w; q, r)))
\end{aligned}
$$

8. $not_{rel}$

   Se define la función $not_{rel}$ que devuelve la codificación de la negación de una relación.

$$
\begin{aligned}
not_{rel}(r; ) &= not_{rel_b}(p(; r); r) \\
not_{rel_b}(0; r) &= 1 \\
not_{rel_b}(wi; r) &= C(; P(p(; P(wi; r)); r), S_1(; not_{rel_b}(w; r)), S_0(; not_{rel_b}(w; r)))
\end{aligned}
$$

9. *ObtenerRel*

   Se define la función *ObtenerRel*, tal que dada una relación de aridad $k$, y un $w$ que representa los argumentos $1, \ldots, k-1$ devuelve la relación que define el argumento $k$.

$$
\begin{aligned}
&ObtenerRel(0, w, tamr; r) = 1\\
&ObtenerRel(xi, w, tamr; r) =\\
&\quad C(; arg_1^k(n, p(; P(xi; tamr)); ) == arg_1^{k-1}(n, w),\\
&\quad ObtenerRel(x, w, tamr; r),\\
&\quad C(; arg_2^k(n, p(; P(xi; tamr)); ) == arg_2^{k-1}(n, w),\\
&\quad ObtenerRel(x, w, tamr; r), \ldots\\
&\quad C(; arg_{k-1}^k(n, p(; P(xi; tamr)); ) == arg_{k-1}^{k-1}(n, w),\\
&\quad ObtenerRel(x, w, tamr; r),\\
&\quad C(; P(p(; P(wi; tamr)); r),\\
&\quad\quad S_0(; ObtenerRel(x, w, tamr; r)),\\
&\quad\quad S_1(; ObtenerRel(x, w, tamr; r)))) \ldots ))
\end{aligned}
$$

10. *HayUnUno*

    Defino la función *HayUnUno* que dada una relación $r$ que codifica $R$ devuelve 1 si y solo si $R \neq \emptyset$, es decir si hay algún uno en el número que codifica la relación.

$$
\begin{aligned}
HayUnUno(0; r) &= 0\\
HayUnUno(wi; r) &= C(; P(p(; P(wi; r)); r), HayUnUno(w; r), 1)
\end{aligned}
$$

## 3.3. Idea de Construcción del Programa $\mathcal{P}$

Ahora, para construir el programa en B se construyen las funciones necesarias para cada subfórmula, de acuerdo a las reglas aplicadas. Es decir, realizando inducción sobre la última regla aplicada.

Así como está definida una codificación para las relaciones, se utilizará esta misma forma de codificación para las subfórmulas. Entonces si una fórmula $\phi$ depende de $x_1, \ldots, x_k$, $X_1, \ldots X_j$, la función en **B** que caracteriza $\phi$ devuelve un número que en cada lugar vale 1 si y solo si evaluando en los distintos $x_1, \ldots, x_k, X_1, \ldots X_j$ la fórmula se satisface, y 0 en caso contrario. En el caso de que la fórmula no tenga variables de primer orden libres, devolverá 10 cuando no se satisface, y 11 en caso contrario.

Las reglas son las siguientes:

- $\dfrac{}{R(t_1, \ldots, t_k)}$ con $R$ una relación de aridad $k$ y $t_1, \ldots, t_k$ términos.

En 3.4.1 se construye la función correspondiente.

- $\overline{X(t_1, \ldots, t_k)}$ con $X$ una variable de segundo orden.

  En 3.4.2 se construye la función correspondiente.

- $\overline{t_1 = t_2}$

  En 3.4.3 se construye la función correspondiente.

- $\dfrac{\psi}{\neg\psi}$

  En 3.5 se construye la función correspondiente.

- $\dfrac{\psi \qquad \phi}{\psi \vee \phi}$

  En 3.6 se construye la función correspondiente.

- $\dfrac{\psi}{\exists x \psi}$

  En 3.7 se construye la función correspondiente.

- $\dfrac{\psi}{[IFP_{\overline{x},X}\psi]\overline{y}}$

  En 3.8 se construye la función correspondiente.

Recordar que para el conectivo ($\wedge$) y para el cuantificador ($\forall$):

$$(\psi \wedge \phi) \equiv \neg(\neg\psi \vee \neg\phi)$$

$$(\forall x\, \psi) \equiv \neg(\exists x\, \neg\psi)$$

## 3.4. Fórmulas atómicas

### 3.4.1. Fórmula definida mediante un símbolo de relación del vocabulario

Para la fórmula $\varphi \equiv R(t_1, \ldots, t_k)$ se realizan dos pasos:

1. **Constantes**

   El objetivo es obtener una nueva codificación de la relación en la que ya esté realizada la instanciación en los términos que son constantes.

   Sean $i_1, \ldots, i_h$ tal que para todo $s \in \{1, \ldots, h\}, t_{i_s}$ es un símbolo de constante, y $m < n \Rightarrow i_m < i_n$.

   Sea $w$ tal que $r_w$ es la codificación de la relación $R$

   - Si $h > 0$
     Se define con $k' = k - h$

$$\begin{aligned} f^{k'}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) &= EvalConst^{k-h+1}_{i_1}(\ldots( \\ & \quad EvalConst^{k-1}_{i_{h-1}}( \\ & \quad EvalConst^{k}_{i_h}(r_w, n, t_{i_h}), \\ & \quad n, t_{i_{h-1}})\ldots), n, t_{i_1}) \end{aligned}$$

   - Si $h = 0$
     Se define con $k' = k$

$$f^{k}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) = r_w$$

2. **Variables repetidas**

   Mediante la función $f_{\varphi,Ctes}$ se obtiene la codificación de una relación $R'$ de aridad $k'$ que no se va a instanciar en constantes pero puede suceder que dos de las variables deban tener el mismo valor. En ese caso sea $D = \{(i_1, j_1), \ldots, (i_h, j_h)\}$ el conjunto de los pares que deben tener iguar valor (con $(i_t < j_t \forall t)$ y $(u < v \Rightarrow j_u > j_v)$). Por ejemplo para $R(x, y, z, y, x)$, $D = \{(1, 5), (2, 4)\}$.

   - Si $h > 0$:
     Para cada par se definen las funciones

$$\begin{aligned} f^{1,k'}_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; & \\ x_1, \ldots, x_s) &= \\ & EvalVar^{k'}_{i_1=j_{1_b}}(n^{k'}, S_0(; n^{k'}), n; \\ & \quad f^{k'}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s)) \end{aligned}$$

$$\begin{aligned} f^{2,k'-1}_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; & \\ x_1, \ldots, x_s) &= \\ & EvalVar^{k'-1}_{i_2=j_{2_b}}(n^{k'-1}, S_0(; n^{k'-1}), n; \\ & \quad f^{1,k'}_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s)) \end{aligned}$$

Y así sucesivamente hasta llegar a:

$$f^{h,k'-h+1}_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) = \\ EvalVar^{k'-h+1}_{i_h=j_{h_b}}(n^{k'-h+1}, S_0(; n^{k'-h+1}), n; \\ f^{h-1,k'-h+2}_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s))$$

Por último se define

$$f_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) = \\ f^{h,k-h+1}_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s)$$

- Si $h = 0$:
  Se define

$$f_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) = \\ f^{k'}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s)$$

Por último, se define

$$f_{\varphi}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) = \\ f_{\varphi,Rep}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s)$$

### 3.4.2. Fórmula definida a partir de una variable de segundo orden

Para la fórmula $\varphi \equiv X(t_1, \ldots, t_k)$ la idea es igual a 3.4.1, pero en lugar de definir $f_{\varphi,Ctes}$ usando $r_w$, se utiliza la variable que representa a la codificación de la variable de segundo orden $X$

Es decir,

- Si $h > 0$
  Se define con $k' = k - h$

$$f^{k'}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s, X) = EvalConst^{k-h+1}_{i_1}(\ldots( \\ EvalConst^{k-1}_{i_{h-1}}( \\ EvalConst^{k}_{i_h}(X, n, t_{i_h}), \\ n, t_{i_{h-1}})\ldots), n, t_{i_1})$$

- Si $h = 0$

  Se define con $k' = k$

$$f^{k}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s, X) = X$$

Más allá de esta particularidad, la eliminación de las constantes y variables repetidas es exactamente la misma idea (ver 3.4.1).

### 3.4.3. Relación de igualdad

En este caso nuevamente se sigue la misma idea pero en lugar de utilizar $r_w$ como en 3.4.1 o $X$ como en 3.4.2 se utiliza $rel_{=}(n;)$

Es decir,

- Si $h > 0$

  Se define con $k' = k - h$

$$\begin{aligned} f^{k'}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) \quad = \quad & EvalConst^{k-h+1}_{i_1}(\ldots( \\ & EvalConst^{k-1}_{i_{h-1}}( \\ & EvalConst^{k}_{i_h}(rel_{=}(n;), n, t_{i_h}), \\ & n, t_{i_{h-1}}) \ldots), n, t_{i_1}) \end{aligned}$$

- Si $h = 0$

  Se define con $k' = k$

$$f^{k}_{\varphi,Ctes}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s) = rel_{=}(n;)$$

## 3.5. Negación

Para $\varphi \equiv \neg\psi$, siendo $f_{\psi}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s, X_1, \ldots, X_t)$ la función que devuelve la codificación de la fórmula definida por $\psi$ (que depende de $k$ variables de primer orden), se define:

$$\begin{aligned} f_{\neg\psi}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s, X_1, \ldots, X_t) \quad = \quad & \\ & not_{rel_b}(n^k; \\ & \quad f_{\psi}(n, r_1, \ldots, r_m, c_1, c_l; \\ & \qquad x_1, \ldots, x_s, X_1, \ldots, X_t)) \end{aligned}$$

## 3.6. Disyunción

Para la fórmula $\varphi \equiv \psi \vee \phi$ ya están definidas las funciones:

- $f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'})$ codifica la fórmula $\psi$ dependiendo de $s$ variables de primer orden.
- $f_\phi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; y_1, \ldots, y_t, Y_1, \ldots, Y_{t'})$ codifica la fórmula $\phi$ dependiendo de $t$ variables de primer orden

Para definir la función se realizan dos pasos:

1. El primer paso consiste en agregar a la relación que se obtiene de $f_\psi$ las variables de $f_\phi$ que aún no están, y lo mismo para $f_\phi$.

   Sean $i_1, \ldots, i_v \in \{1, \ldots, t\}$ tal que $y_{i_j}$ es una variable de primer orden $\notin \{x_1, \ldots, x_s\} \forall j$

   Entonces se construye la extensión de $\psi$ a las nuevas variables:

   - Si $v > 0$:

     Se definen las siguientes funciones:

     Para $j = 1$:

$$
\begin{aligned}
&f^1_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&\quad x_1, \ldots, x_s, y_{i_1}, X_1, \ldots, X_{s'}) \quad = \\
&\qquad AddVar_b(n^s, S_0(; n^s);\\
&\qquad\quad f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&\qquad\qquad x_1, \ldots, x_s,\\
&\qquad\qquad X_1, \ldots, X_{s'}))
\end{aligned}
$$

     Para $j = 2$:

$$
\begin{aligned}
&f^2_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&x_1, \ldots, x_s, y_{i_1}, y_{i_2}, X_1, \ldots, X_{s'}) \quad = \\
&\qquad AddVar_b(n^{s+1}, S_0(; n^{s+1});\\
&\qquad\quad f^1_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&\qquad\quad x_1, \ldots, x_s, y_{i_1}, X_1, \ldots, X_{s'}))
\end{aligned}
$$

y así sucesivamente, para $j = v$:

$$f^{v}_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, y_{i_1}, \ldots, y_{i_v}, X_1, \ldots, X_{s'}) = AddVar_b(n^{s+v-1}, S_0(; n^{s+v-1}); f^{v-1}_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, y_{i_1}, \ldots, y_{i_{v-1}}, X_1, \ldots, X_{s'}))$$

- Si $v = 0$:

$$f^{v}_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s) = f_{\psi}(n, r_1, \ldots, r_m, c_1, c_l; x_1, \ldots, x_s)$$

2. Ahora se agregan las variables de segundo orden.

   Sean $\{i_1, \ldots, i_{v'}\}$ tal que para todo $t \in \{1, \ldots, v'\}$, $Y_{i_t} \notin \{X_1, \ldots, X_{s'}\}$

   Se define

$$f_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, y_{i_1}, \ldots, y_{i_v}, X_1, \ldots, X_{s'}, Y_{i_1}, \ldots, Y_{i_{v'}}) = f^{v}_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, y_{i_1}, \ldots, y_{i_v}, X_1, \ldots, X_{s'})$$

3. De igual manera se define la extensión de $\phi$, obteniendo

$$f_{\psi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; y_1, \ldots, y_t, x_{i_1}, \ldots, x_{i_w}, Y_1, \ldots, Y_{t'}, X_{i_1}, \ldots, X_{i_{w'}})$$

4. Luego se reordenan las variables de primer orden de $f_{\psi,ext}$ componiendo la función $SwapVar$ todas las veces que sea necesario para que queden en igual orden que $f_{\phi,ext}$.

   Con este paso se obtiene:

$$f_{\psi,orden_1}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, y_{i_1}, \ldots, y_{i_v}, Y_1, \ldots, Y_{t'}, X_{i_1}, \ldots, X_{i_{w'}})$$

5. Se reordenan las variables de segundo orden, tomando en cuenta que algún $Y_i$ puede ser igual a algún $X_j$

$$
\begin{aligned}
f_{\psi,orden_2}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
x_1, \ldots, x_s,\\
y_{i_1}, \ldots, y_{i_v},\\
X_1, \ldots, X_{s'},\\
Y_{i_1}, \ldots, Y_{i_v}) \quad = \quad & \\
f_{\psi,orden_1}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
x_1, \ldots, x_s, y_{i_1}, \ldots, y_{i_v},\\
Y_1, \ldots, Y_{t'}, X_{i_1}, \ldots, X_{i_{w'}})
\end{aligned}
$$

De esta manera las funciones correspondientes a las subfórmulas que se comportan como operandos de la disyunción reciben los mismos argumentos en el mismo orden.

6. Por último

$$
\begin{aligned}
f_{\psi\vee\phi}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
x_1, \ldots, x_s,\\
y_{i_1}, \ldots, y_{i_v},\\
X_1, \ldots, X_{s'},\\
Y_{i_1}, \ldots, Y_{i_v}) \quad = \quad & \\
or_{rel_b}(n^{s+v};\ f_{\psi,orden_2}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
x_1, \ldots, x_s,\\
y_{i_1}, \ldots, y_{i_v},\\
X_1, \ldots, X_{s'},\\
Y_{i_1}, \ldots, Y_{i_v}),\\
f_{\phi,ext}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
x_1, \ldots, x_s,\\
y_{i_1}, \ldots, y_{i_v},\\
X_1, \ldots, X_{s'},\\
Y_{i_1}, \ldots, Y_{i_v}))
\end{aligned}
$$

## 3.7. Cuantificador existencial

Para la fórmula $\varphi \equiv \exists x_w \psi$, con la fórmula

$$f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'})$$

correspondiente a $\psi$ que tiene $s$ variables libres, se definen las siguientes funciones:

- Si $x_w \in free(\psi)$ entonces reordenamos para que $x_w$ sea la última variable libre:

$$\begin{aligned}
&f_{\psi,ord}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&x_1, \ldots, x_{w-1}, x_s, x_{w+1}, \ldots, x_{s-1}, x_w,\\
&X_1, \ldots, X_{s'}) =\\
&\quad SwapVar^s_{w,s_b}(n^s, S_0(; n^s), n;\\
&\quad\quad f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'}))
\end{aligned}$$

  Utilizamos la siguiente función auxiliar que genera una nueva relación evaluando para cada elemento del dominio.

$$\begin{aligned}
f_{\varphi_b}(0, tamr; r) &= 1\\
f_{\varphi_b}(wi, tamr; r) &= C(; HayUnUno(p(; ObtenerRel(tamr, w; r));\\
&\qquad ObtenerRel(tamr, w; r)),\\
&\qquad S_0(; f_{\varphi_b}(w, tamr; r)), S_1(; f_{\varphi_b}(w, tamr; r)))
\end{aligned}$$

  Tomando $y_1, \ldots, y_s = x_1, \ldots, x_{w-1}, x_s, x_{w+1}, \ldots, x_{s-1}, x_w$ definimos la siguiente función para representar el cuantificador existencial:

$$\begin{aligned}
&f_\varphi(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&y_1, \ldots, y_{s-1}, X_1, \ldots, X_{s'}) =\\
&\quad f_{\varphi_b}(n^{s-1}, n^s; f_{\psi,ord}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; y_1, \ldots, y_{s-1}, 0, X_1, \ldots, X_{s'})
\end{aligned}$$

- Si $x_w \notin free(\psi)$

$$\begin{aligned}
&f_\varphi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'}) =\\
&\quad f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'})
\end{aligned}$$

## 3.8. IFP

Si la fórmula está dada por: $\varphi \equiv [IFP_{\overline{x},X}\psi]\overline{t}$

Entonces existe una función $f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'})$.

- Si $X$ no es una variable libre de $\psi$ entonces se define:

$$f_\varphi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'}) = f_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_1, \ldots, X_{s'})$$

- Si $X$ es una variable libre de $\psi$, entonces existe $w$ tal que $X_w$ (argumentos de $f_\psi$) representa $X$ y $i_1, \ldots, i_k$ tal que $x_{i_1}, \ldots, x_{i_k}$ aparecen libres en $\psi$ y son argumentos de $f_\psi$.

  Entonces se modifica el orden de las variables de segundo orden, dejando la que queda ligada en $\varphi$ al principio.

$$f'_\psi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_1, \ldots, x_s, X_w, X_1, \ldots, X_{w-1}, X_{w+1}, \ldots, X_{s'})$$

  Sea $y_1, \ldots, y_t$ las variables incluidas en $\{x_1, \ldots, x_s\}$ tal que $x_w \notin \{y_1, \ldots, y_t\}$ y $x_{i_j} \notin \{y_1, \ldots, y_t\}$ para todo $j$.

  Luego se modifica el orden de las variables de primer orden para que queden al principio aquellas que quedan ligadas en $\varphi$ usando la composición de $SwapVar$. Se obtiene de esta manera:

$$f_{\psi,ord}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; x_{i_1}, \ldots, x_{i_k}, y_1, \ldots, y_t, X_w, X_1, \ldots, X_{w-1}, X_{w+1}, \ldots, X_{s'})$$

  Entonces se define:

$$f_\varphi(n, r_1, \ldots, r_m, c_1, \ldots, c_l; y_1, \ldots, y_t) = rec_\varphi(S_1(; exp_k(n;)), pad(exp_k(n;); 1); y_1, \ldots, y_t)$$

  Observar que con $exp_k(n;)$ se obtiene la cota de la cantidad de iteraciones con las que se llega al punto fijo, y con $pad(exp_k(n;); 1)$ se obtiene la relación inicial: $X = \emptyset$.

  Se describe a continuación la función $rec_\varphi$ que va a simular las aplicaciones de la función $F^\varphi$

$$\begin{aligned} rec_\varphi(0, rel; y_1, \ldots, y_t) &= Modificar Rel(p(; rel), 0, rel; y_1, \ldots, y_t, rel) \\ rec_\varphi(wi, rel; y_1, \ldots, y_t) &= Modificar Rel(p(; rel), wi, rel; \\ &\quad y_1, \ldots, y_t, rec_\varphi(w, rel; y_1, \ldots, y_t)) \end{aligned}$$

$$
\begin{aligned}
&Modificar Rel_{\varphi}(0, w, tamr; y_1, \ldots, y_t, rel) = 1\\
&Modificar Rel_{\varphi}(xi, w, tamr; y_1, \ldots, y_t, rel) =\\
&\quad C(; P(p(; P(xi; tamr)); rel),\\
&\quad C(; f_{\psi,ord}(n, r_1, \ldots, r_m, c_1, \ldots, c_l;\\
&\quad arg_1^k(n, p(; P(xi; tamr)); ),\\
&\quad arg_2^k(n, p(; P(xi; tamr)); ),\\
&\quad \vdots\\
&\quad arg_k^k(n, p(; P(xi; tamr)); ),\\
&\quad y_1, \ldots, y_t),\\
&\quad S_0(; Modificar Rel_{\varphi}(x, w, tamr;\\
&\qquad y_1, \ldots, y_t, rel)),\\
&\quad S_1(; Modificar Rel_{\varphi}(x, w, tamr;\\
&\qquad y_1, \ldots, y_t, rel)))
\end{aligned}
$$

Luego con $\bar{t}$ se realiza la instanciación en las constantes y variables como en 3.4.1 para obtener la codificación de una fórmula que depende de variables (y no constantes) todas distintas.

## 3.9. Función principal

A partir de la última regla aplicada en el árbol de formación de la fórmula se tiene:

$$f_{\varphi}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; )$$

Esta función –como las definidas anteriormente– devuelve un número que codifica la fórmula, en este caso con cero variables libres. Los dos resultados posibles son 10, 11 (recordemos que el uno inicial forma parte de la codificación de las relaciones para no perder su longitud). Evaluando el último dígito de este resultado, obtenemos si la codificación de la estructura pertenece o no a la clase $\mathcal{K}$ en cuestión.

Entonces, se define como función principal del programa:

$$f_{main}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; ) = last(; f_{\varphi}(n, r_1, \ldots, r_m, c_1, \ldots, c_l; ))$$

**Teorema 3.13.** *Para cualquier vocabulario $\tau$ y fórmula $\varphi \in$ **FO(IFP)** existe una función $f \in$ **B** tal que para toda $\tau$-estructura $\mathcal{A}$*

$$\mathcal{A} \models \varphi \text{ si y solo si } f(bin(\mathcal{A});) > 0$$

*Demostración.* Por inducción en la construcción de la fórmula, siguiendo las secciones 3.4.1, 3.4.2, 3.4.3, 3.5, 3.6, 3.7, 3.8 y mediante la definición de $f_{main}$, se obtiene la función tal que

$$\mathcal{A} \models \varphi \text{ si y solo si } f_{main}(bin(\mathcal{A});) > 0$$

□

**Corolario 3.14.** *Toda clase de estructuras que se puede determinar por una fórmula en **FO(IFP)**, puede determinarse por una función en **B**, es decir $\mathcal{T}_{\mathbf{FO(IFP)}} \subseteq \mathcal{T}_{\mathbf{B}}$.*

**Teorema 3.15.** *Para toda clase de estructuras, se puede determinar por una fórmula en **FO(IFP)** si y solo si puede determinarse por una función en **B**.*

*Demostración.* Ya demostramos ambas inclusiones:

- $\mathcal{T}_{\mathbf{B}} \subseteq \mathcal{T}_{\mathbf{FO(IFP)}}$: Ver 2.23.
- $\mathcal{T}_{\mathbf{FO(IFP)}} \subseteq \mathcal{T}_{\mathbf{B}}$: Ver 3.14.

□

# 4. Extensión a FP

Para extender la equivalencia de los problemas de decisión a problemas que representan funciones computables en tiempo polinomial (**FP**), se presenta a continuación la idea de Query en **FO(IFP)** (Ver [IMM/98]).

**Definición 4.1.** Se define *Query* como una función

$$I : \sigma\text{-estructuras} \rightarrow \tau\text{-estructuras}$$

de $\sigma$-estructuras en $\tau$-estructuras donde $I$ es polinomialmente acotada, es decir, existe un polinomio $p$ tal que para toda $\sigma$-estructura $\mathcal{A}$: $\|I(\mathcal{A})\| \leq p(\|\mathcal{A}\|)$.

Entonces se extiende la idea de First Order Queries presentada en [IMM/98] a **FO(IFP)**.

Para $\tau = \langle R_1, \ldots, R_m, c_1, \ldots, c_l \rangle$:

$$I = \langle \varphi_0, \varphi_1, \ldots, \varphi_m, \psi_1, \ldots, \psi_l \rangle$$

Con $\varphi_i, \psi_j \in$ **FO(IFP)**$[\sigma] \quad \forall i, j$

Entonces, si $\mathcal{A}$ es una $\sigma$-estructura, $I(A)$ es una $\tau$-estructura definida de la siguiente manera:

$$I(\mathcal{A}) = \langle |I(\mathcal{A})|, R_i^{I(\mathcal{A})}, \ldots, R_r^{I(\mathcal{A})}, c_1^{I(\mathcal{A})}, \ldots, c_l^{\mathcal{A}} \rangle$$

.

El dominio de $I(\mathcal{A})$ se define a partir de la fórmula $\varphi_0$ que pertenece a **FO(IFP)**:

$$|I(\mathcal{A})| = \{\langle b^1, \ldots, b^k \rangle | \mathcal{A} \models \varphi_0(b^1, \ldots, b^k)\}$$

Luego, para cada relación se tiene con $\varphi_i$ una fórmula que pertenece a **FO(IFP)**$[\sigma]$:

$$R_i^{I(\mathcal{A})} = \{(\langle b_1^1, \ldots, b_1^k \rangle, \ldots, \langle b_{a_i}^1, \ldots, b_{a_i}^k \rangle) \in |I(\mathcal{A})|^{a_i} | \mathcal{A} \models \varphi_i(b_1^1, \ldots, b_{a_i}^k)\}$$

Por último, para cada símbolo de constante se puede definir un elemento en $I(\mathcal{A})$ mediante las fórmulas $\psi_1, \ldots, \psi_l \in$ **FO(IFP)**$[\sigma]$:

$$c_j^{I(\mathcal{A})} = \text{ el único } \langle b^1, \ldots, b^k \rangle \in |I(\mathcal{A})| \text{ tal que } \mathcal{A} \models \psi_j(b^1, \ldots, b^k)$$

Para cada una de estas fórmulas se construye el programa en **B** tal como se presentó en la sección 3 y utilizando las funciones que representa las relaciones y constantes se obtiene la función $I$ (ver B $\subseteq$ **FO(IFP)** (Ver sección 2))

**Ejemplo 4.2.** Sean $\sigma = \langle N^1, S^2, <^2, \text{mín}^0, \text{máx}^0\rangle$, $\tau = \langle R^1, carry^0, S^2, <^2, \text{mín}^0, \text{máx}^0\rangle$ los vocabularios que intervienen en el problema.

Una $\sigma$-estructura $\mathcal{A}$ representa un número en binario($N$) de longitud $|A|$.

Por ejemplo, para representar el 10110:

$$\mathcal{A} = \langle A, N^A, S^A, <^A, \overset{A}{\text{mín}}, \overset{A}{\text{máx}}\rangle$$

con $A = \{0, 1, 2, 3, 4\}$, $N = \{\langle 1\rangle, \langle 2\rangle, \langle 4\rangle\}$ y $S^A, <^A, \text{mín}^A, \text{máx}^A$ se interpretan como se espera con las estructuras ordenadas.

Se obtendrá mediante una *Query* $I$ un número en binario representado por $R = N + 1$ y la constante *carry* valdrá 1 si y solo si el resultado de incrementar en uno $N$ tiene $n + 1$ dígitos en base $n = \|A\|$.

Entonces, con $k = 1$

$$\varphi_0(x) \equiv \textbf{true}$$

$$Carry(x) \equiv x > 0 \wedge \forall y(y < x \rightarrow N(y))$$

Se define $\varphi_1$ tal que vale $\varphi_1(x)$ si la posición $x$ vale uno en el resultado de sumarle 1 a $N$

$$\begin{aligned} \varphi_1(x) \quad \equiv \quad & (N(x) \wedge \neg Carry(x)) \vee \\ & (\neg N(x) \wedge Carry(x)) \end{aligned} \tag{9}$$

Para la constante de carry: si en el bit de la posición $n - 1$ se genera carry, es decir se va a necesitar un bit más, entonces $\psi_1$ vale para $x = 1$, sino vale para $x = 0$

$$\begin{aligned} \psi_1(x) \quad \equiv \quad & (Carry(\text{máx}) \wedge x = 1) \vee \\ & (\neg Carry(\text{máx}) \wedge x = 0) \end{aligned} \tag{10}$$

Observar que $\varphi_0$, $\varphi_1$ y $\psi_1$ pertenecen a **FO(IFP)**[$\sigma$], entonces para cada una de estas fórmulas existe un programa $\mathcal{P}_0$, $\mathcal{P}_1$ y $\mathcal{Q}_1$ que devuelven 1 si y solo si se satisfacen en la estructura para cierto input.

De igual modo, dadas las estructuras presentadas previamente, y teniendo los programas que corresponden a cada una de las fórmulas, utilizando lo presentado en la inclusión **B⊆FO(IFP)** (Ver sección 3) se obtienen las fórmulas que pertenecen a **FO(IFP)**[$\sigma$] que definen la Query.

**Teorema 4.3.** *Los modelos de complejidad **FO(IFP)** y **B** son equivalentes en su caracterización de **FP**. Para toda Query definida a través de un conjunto de s programas en **B**, existe una Query equivalente definida mediante s fórmulas en **FO(IFP)**, y para toda Query definida mediante s fórmulas en **FO(IFP)** existe una Query equivalente definida a través de s programas en **B**.*

# 5. Trabajo futuro

En esta tesis fue presentada la demostración de la equivalencia entre dos caracterizaciones de **PTIME**: **FO(IFP)** y la clase **B**. Esta idea se puede extender para demostrar la equivalencia entre otras clases de complejidad.

Algunos ejemplos de distintas caracterizaciones que tienen las clases de complejidad son las siguientes:

- **LOGSPACE**
  - La caracterización algebraica presentada en [BEL/92], que es similar a **B** pero sin la función inicial $S_0$.
  - **FO(DTC) -** la lógica de primer orden con la clausura transitiva determinística.
  - $BC_\epsilon^-$ (Ver [NEE/04]).
- Otras caracterizaciones de **PTIME** – se basan en entender las demostraciones como programas.
  - Light Linear Logic (Ver [GIR/95]).
  - Soft Lambda Calculus (Ver [BAI/03]).
- **NP**
  - $\Sigma_1^1$ - las fórmulas de segundo orden de la forma $\exists X_1 \ldots X_m \psi$ donde $\psi$ es una fórmula de primer orden.
- **PSPACE**
  - **FO(PFP) -** la lógica de primer orden con punto fijo parcial.

# 6. Conclusiones

Una clase $\mathcal{K}$ de $\tau$-estructuras pertenece a **PTIME** si existe una máquina de Turing determinística $\mathcal{M}$, de complejidad temporal de orden polinomial, tal que $\mathcal{M}$ decide la pertenencia de las $\tau$-estructuras a la clase.

En 1964, A. Cobham presentó un álgebra de funciones que caracteriza **PTIME**. Con este álgebra, no es necesario hallar la función $t_{\mathcal{M}}$ que devuelve la cantidad de transiciones de estados que se realizan en una máquina de Turing. Esto permite que sea más legible el problema de decidir la pertenencia a una clase de $\tau$-estructuras. Sin embargo, en este álgebra de funciones, hay que mostrar cada vez que se utiliza el operador de recursión, que la longitud del resultado se encuentra acotada por una función perteneciente a la clase.

El álgebra de funciones presentada en 1992 por S. Bellantoni y S. Cook agrega restricciones sintácticas para prescindir de mostrar cotas en la recursión.

En 1974, R. Fagin muestra la equivalencia entre **FO(IFP)** y **PTIME**. Con este resultado, comienza a desarrollarse la Teoría de la Complejidad Descriptiva. La existencia de una fórmula $\varphi \in$ **FO(IFP)** que caracteriza una clase de $\tau$-estructuras $\mathcal{K}$ es equivalente a la existencia de una máquina de Turing determinística de complejidad temporal polinomial que decida la pertenencia a la clase $\mathcal{K}$.

La demostración presentada en esta tesis utiliza como modelos de **PTIME**, la clase **B** y la lógica **FO(IFP)**. Al no utilizar máquinas de Turing, obtenemos la equivalencia en complejidad como la equivalencia en las clases de $\tau$-estructuras que pueden ser determinadas por funciones en **B** o fórmulas en **FO(IFP)**.

Una clase $\mathcal{K}$ de $\tau$-estructuras es determinada por una fórmula $\varphi \in$ **FO(IFP)** si para toda $\tau$-estructura $\mathcal{A}$: $\mathcal{A} \in \mathcal{K} \Leftrightarrow \mathcal{A} \models \varphi$. Para obtener una función $f \in$ **B** que determine la clase $\mathcal{K}$ hubo que encontrar una manera de representar las estructuras como elementos del dominio de $f$, es decir, elementos en $\mathbb{N}_0^*$. Para esto, presentamos una función de codificación $bin(\mathcal{A})$ inyectiva, que nos permite representar las estructuras como números. También utilizamos esta idea para representar las subfórmulas de $\varphi$ como números.

Por otro lado, una clase $\mathcal{K}$ de $\tau$-estructuras es determinada por una función $f \in$ **B**, si para toda $\tau$-estructura $\mathcal{A}$: $\mathcal{A} \in \mathcal{K} \Leftrightarrow f(bin(\mathcal{A});) > 0$. Para obtener una fórmula $\varphi \in$ **FO(IFP)** que determine la clase $\mathcal{K}$ hubo que encontrar una forma de representar las funciones sobre los números en **FO(IFP)**. Demostramos en esta tesis una cota polinomial en el cardinal del dominio de la estructura para la salida de las funciones en **B**, y utilizando esta cota presentamos la representación de los números y resultados de las funciones como fórmulas. Para llegar a esta representación, primero se evaluó la posibilidad de utilizar una representación en base $n = ||A||$ de los números, pero no existe una cota polinomial para poder representar la codificación de la estructura en base $n$.

Hemos utilizado, tanto para la representación de las relaciones con números como para la representación de los números con fórmulas, que las estructuras son ordenadas. Todos los resultados presentados no valdrían sin esta hipótesis. Sin embargo, es razonable trabajar con estructuras ordenadas, ya que al trabajar con problemas computacionales de modelos finitos, necesitamos hallar un isomorfismo con una estructura cuyo dominio esté dado por $\{0, \ldots, n-1\}$.

La diferenciación entre variables *safe* y variables *normal* fue utilizada para hallar una cota polinomial de la longitud del resultado de las funciones. En las fórmulas en **FO(IFP)** las relaciones que representan las variables son utilizadas indistintamente.

Para expresar el operador de recursión en **FO(IFP)**, utilizamos el hecho que la cantidad de pasos que se realizan es la longitud del primer argumento (acotada por $n^d$). Gracias a esto pudimos utilizar $d$ variables para representar cada paso de la recursión. Además, representamos el resultado en cada paso con los bits que están en uno. Si el resultado en algún paso $p$ es cero, no habría ninguna tupla para $p$ en el punto fijo inflacionario. Para evitar este problema agregamos una variable adicional que representa el cero como resultado.

Para la demostración de **B**⊆**FO(IFP)** introdujimos la noción de programa en **B** que nos permitió formalizar la dependencia entre las funciones que intervienen en la decisión de una clase de estructuras.

La función principal de un programa $\mathcal{P}$ en **B** que decide una clase de $\tau$-estructuras $\mathcal{K}$, siempre es evaluada en la codificación de alguna estructura $\mathcal{A}$. Esto implica que podemos acotar el tamaño de la salida de la función por $n^d$. Extendimos este resultado a todas las funciones que intervienen en la decisión.

Para mostrar la inclusión de **FO(IFP)** en **B** codificamos las relaciones y todas las subfórmulas como números en $\mathbb{N}_0$. Hemos definido funciones que actúan como transformaciones de relaciones para representar los distintos conectivos y cuantificadores lógicos. Para obtener la disyunción entre dos fórmulas, hubo que generar codificaciones auxiliares de fórmulas que tuvieran los mismos argumentos en el mismo orden.

La equivalencia de **B** y **FO(IFP)** en **FP** se realiza como una extensión a la equivalencia ya definida para problemas de decisión, introduciendo la definición de Query en **FO(IFP)** y tomando cada fórmula o función interviniente.

Con la formalización de la inclusión de la clase **B** en la lógica **FO(IFP)** y la inclusión de **FO(IFP)** en **B**, se concluye que dada una estructura ordenada $\mathcal{A}$ con universo finito, una propiedad se puede expresar en **B** si y solo si se puede expresar en **FO(IFP)**. Si **B** caracteriza a una clase de complejidad temporal $C$ dada por $O(f(n))$ y **FO(IFP)** caracteriza a una clase de complejidad temporal $C'$ dada por $O(g(n))$ con la definición presentada en 1.3.7, entonces $C = C'$.

# Referencias


[BAI/03] P. Baillot y V. Mogbil, “Soft lambda-calculus: a language for polynomial time computation” (2003).

[BEC/92] S. Bellantoni y S. Cook, “A new recursion-theoretic characterization of the polytime functions”, Computational Complexity 2 (1992) 97-110.

[BEL/92] S. Bellantoni, “Predicative recursion and computational complexity”, Technical Report 264/92, University of Toronto, Computer Science Department (1992).

[CLO/94] P. Clote, “Computation Models and Function Algebras”, Logic and Computational Complexity (1994) 98-130.

[COB/64] A. Cobham, “The intrinsic computational difficulty of functions”, Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science (1964)

[COO/05] S. Cook, “Turing Machines and Reductions”, www.cs.toronto.edu/~sacook/csc365h/notes/turing.ps (2005).

[EBF/95] H.D. Ebbinghaus y J. Flum, “Finite Model Theory” (1995).

[GIR/95] J. Y. Girard, “Light Linear Logic” (1995).

[IMM/98] N. Immerman, “Descriptive Complexity Theory” (1998).

[LEI/94] D. Leivant, “A foundational delineation of poly-time”, Information and Computation 110 (1994).

[MEN/64] E. Mendelson, “Introduction to Mathematical Logic” (1964).

[NEE/04] P. Neergard, “$BC_\epsilon^-$: A Recursion-Theoretic Characterization of LOGSPACE” (2004).