# Tema 4

# Especificación basada en modelos

# Lenguaje Z

# Introducción

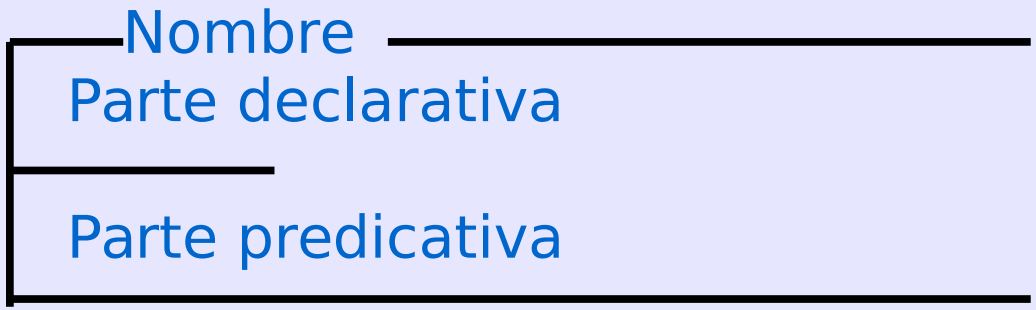
- Características lenguaje Z
  - Lenguaje de especificación formal
    - Propuesto por Jean-Raymond a finales de los 80
    - Desarrollado en la universidad de Oxford
  - Basado en elementos matemáticos
    - Proposiciones y predicados lógicos
    - Conjuntos, relaciones y funciones
  - Se considera un lenguaje declarativo
  - La estructura más importante son los esquemas:
    - Similar a los subprogramas de los lenguajes imperativos
    - Muy útil para la especificación incremental

# Esquemas

- Representación gráfica de las especificaciones
  - Permite descomponer una especificación en piezas más sencillas
  - Se usan para describir
    - Aspectos estáticos del sistema
      - Estado
      - Invariante
    - Aspectos dinámicos del sistema
      - Las operaciones
      - Relaciones entre sus entradas y salidas
      - Cambios que ocurren en el estado

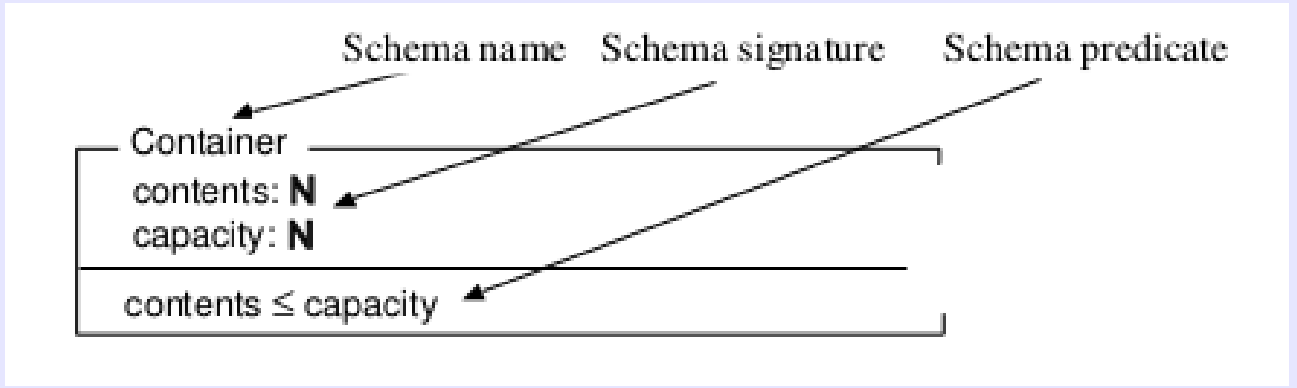
# Esquemas

- Estructura de un esquema
  - Parte declarativa (*schema signature*)
    - Declaraciones locales de variables
    - Tipos, constantes y esquemas
  - Parte predicativa (*schema predicate*)
    - Requisitos que han de satisfacer los valores de las variables



# Esquemas

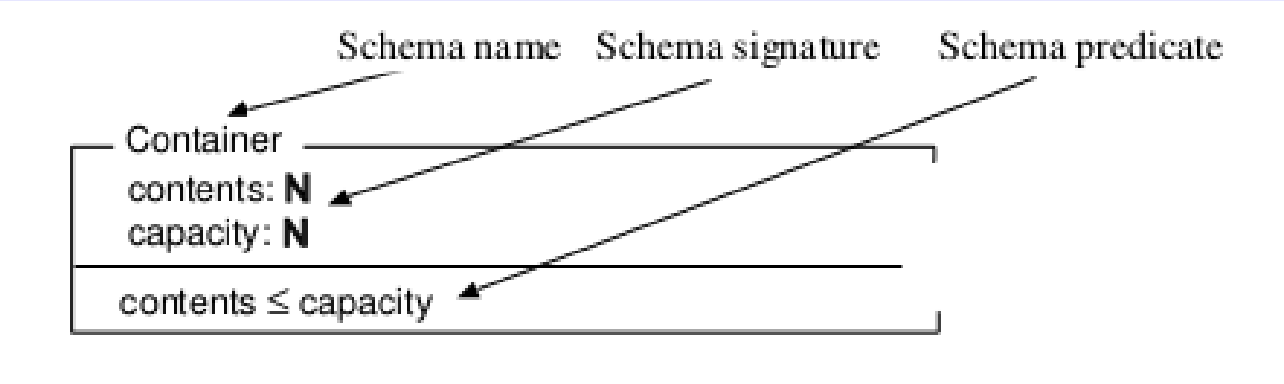
- Ejemplo 1: *Container*
  - Definición parcial de un contenedor que puede almacenar una cantidad discreta de algo




  - Nombre
    - Sirve para referenciarlo en otras partes de la especificación
  - Signature
    - Dos variables de estado
      - `contents` → modelada por un número natural
      - `capacity` → modelada por un número natural

# Esquemas

- Ejemplo 1: *Container*
  - Definición parcial de un contenedor que puede almacenar una cantidad discreta de algo




  - Predicado
    - Relaciones entre las entidades definidas en la signature
      - La cantidad almacenada en el contenedor debe ser menor que su capacidad
      - La especificación no dice nada sobre la capacidad del contenedor, ni sobre lo que contiene

# Esquemas

- Ejemplo 2: *Indicator*
  - Otro esquema básico
  - Se puede asociar con `Container` para proporcionar información sobre su contenido
  - Signature
    - `light` → modelada por los valores {on, off}
    - `reading` → modelada por un número natural
    - `danger_level` → modelada por un número natural

**Indicator**

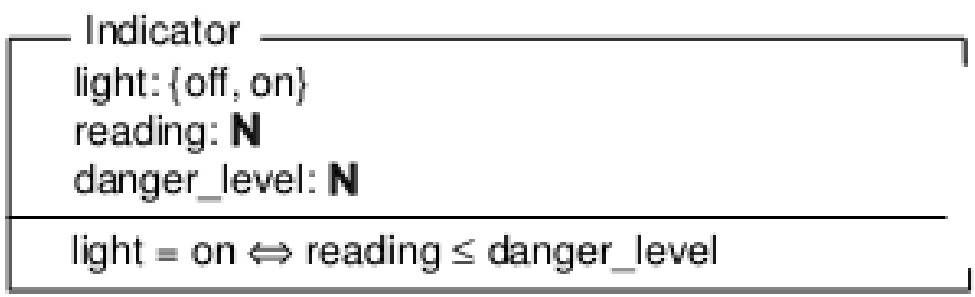
light: {off, on}
reading: $\mathbb{N}$
danger_level: $\mathbb{N}$

---

light = on $\Leftrightarrow$ reading $\leq$ danger_level

# Esquemas

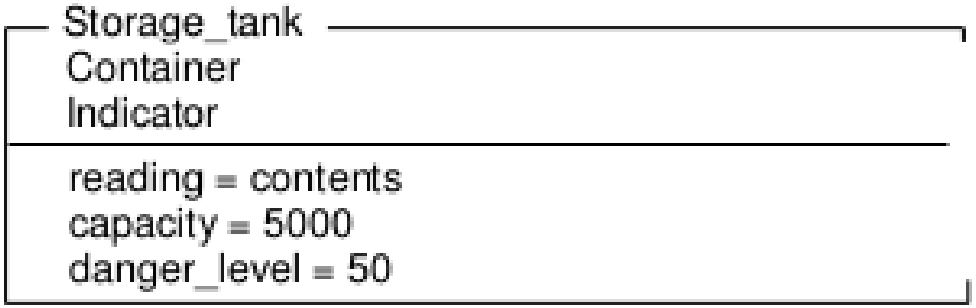
- Ejemplo 2: *Indicator*




  - Predicado
    - ⇔ → sí y sólo si
    - La luz estará encendida, sí y sólo si, *`reading`* es menor o igual que *`danger_level`*
    - *`danger_level`* no está definido

# Esquemas

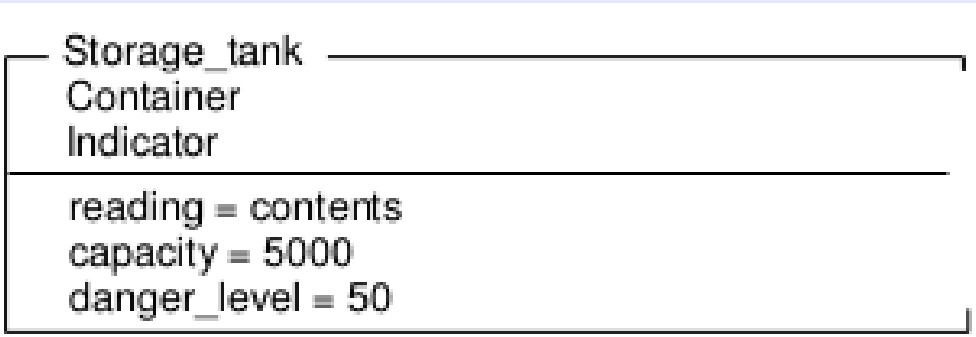
- Ejemplo 3: *Storage tank*
  - Combinando esquemas
    - Se combinan *`Indicator`* y *`Container`* para definir un tanque de almacenamiento con cierta capacidad y un indicador luminoso




    - La nueva especificación incluye todas las variables y predicados definidos en las especificaciones incluidas
      - Se combinan con cualquier signature o predicado nuevo
      - Los predicados se combinan mediante *and* → todos verdaderos

# Esquemas

- Ejemplo 3: *Storage tank*
  - Predicados nuevos
    - Restricciones sobre las variables de estado introducidas en los esquemas Container e Indicator
    - Los predicados se escriben en líneas separadas
      - Se entienden que están combinados mediante and




    - *reading* igual a *contents* y *capacity* igual a 5000 y *danger_level* igual a 50

# Esquemas

- Ejemplo 3: *Storage tank*
  - Incluir esquemas en otro esquema es equivalente a mezclar los esquemas

**Storage_tank**

contents: $\mathbb{N}$
capacity: $\mathbb{N}$
reading: $\mathbb{N}$
danger_level: $\mathbb{N}$
light: {off, on}

---

contents $\leq$ capacity
light = on $\Leftrightarrow$ reading $\leq$ danger_level
reading = contents
capacity = 5000
danger_level = 50

# Definición de operaciones

- Operaciones
  - Se pueden especificar usando esquemas
  - Se define el efecto que tienen sobre el estado del sistema
    - Se indica el estado después de la operación en términos del estado antes de la operación y de los parámetros de la operación
- Convenciones
  - Si una variable N va seguida de ', N' representa el estado de la variable N después de una operación
    - Decoración con dash
  - Si el nombre de un esquema es decorado con un ', el ' se aplica a todas las variables definidas, junto con el invariante

# Definición de operaciones

- Convenciones
  - Si una variable N es decorada con signo !, significa que N es una variable de salida
    - message!
  - Si una variable N es decorada con signo ?, significa que N es una variable de entrada
    - amount?
  - Si el nombre de un esquema va precedido por la letra Ξ , significa que las versiones ' de las variables son las mismas que sin '
    - El estado de las variables no cambia con la operación
    - ΞStorage_tank

# Definición de operaciones

- Convenciones
  - Si el nombre de un esquema va precedido por la letra Δ , implica que el valor de una o más variables del esquema van a cambiar al realizarse la operación

**Fill-OK**

$\Delta$ Storage_tank
amount?: $\mathbb{N}$

---

contents + amount? ≤ capacity
contents' = contents + amount?

**OverFill**

Ξ Storage-tank
amount?: $\mathbb{N}$
r!: seq CHAR

---

capacity < contents + amount?
r! = "Insufficient tank capacity - Fill cancelled"

# Definición de operaciones

- Ejemplo 4: operación Fill-OK
  - Operación que añade una cantidad a un tanque
  - El Δ indica que la operación cambia el estado de Storage_tank
  - La cantidad a añadir (amount?) es una entrada
  - El estado cambia si hay suficiente capacidad en el tanque

**Fill-OK**

Δ Storage_tank
amount?: **N**

---

contents + amount? ≤ capacity
contents' = contents + amount?

# Definición de operaciones

- Ejemplo 4: operación Fill-OK

**Fill-OK**

$\Delta$ Storage_tank
amount?: $\mathbb{N}$

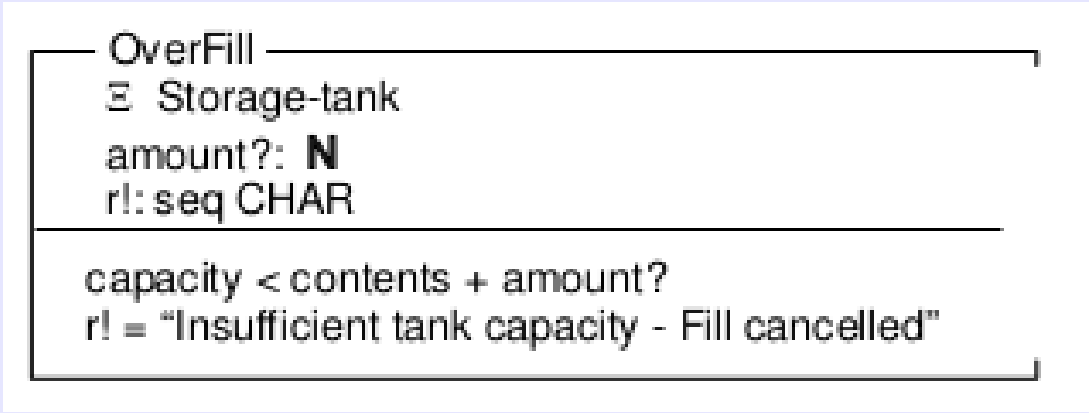
---

contents + amount? $\leq$ capacity
contents' = contents + amount?

  - El predicado especifica que después de la operación
    - `contents'` es igual al contenido anterior a la operación más la cantidad añadida
    - Esto se cumple, si la cantidad no excede la capacidad
    - Si la cantidad supera la capacidad del tanque → no definido

# Definición de operaciones

- Ejemplo 5: operación OverFill
  - En Z las operaciones se especifican por parte
    - Un primer esquema define la operación “correcta”
    - Los siguientes esquemas definen lo que ocurriría en situaciones excepcionales
  - Los esquemas se combinan mediante una disyunción para especificar la operación completa



# Definición de operaciones

- Ejemplo 5: operación OverFill
  - Especifica que pasaría si añadiéramos una cantidad que excediera la capacidad del tanque
    - No se añade nada
    - Se muestra un mensaje de aviso
  - El uso del esquema indica que las variables de estado no cambiarán su valor
  - El predicado asociado será verdad cuando la capacidad sea menor que el contenido actual + la cantidad a añadir

**OverFill**

Ξ Storage-tank

amount?: **N**
r!: seq CHAR

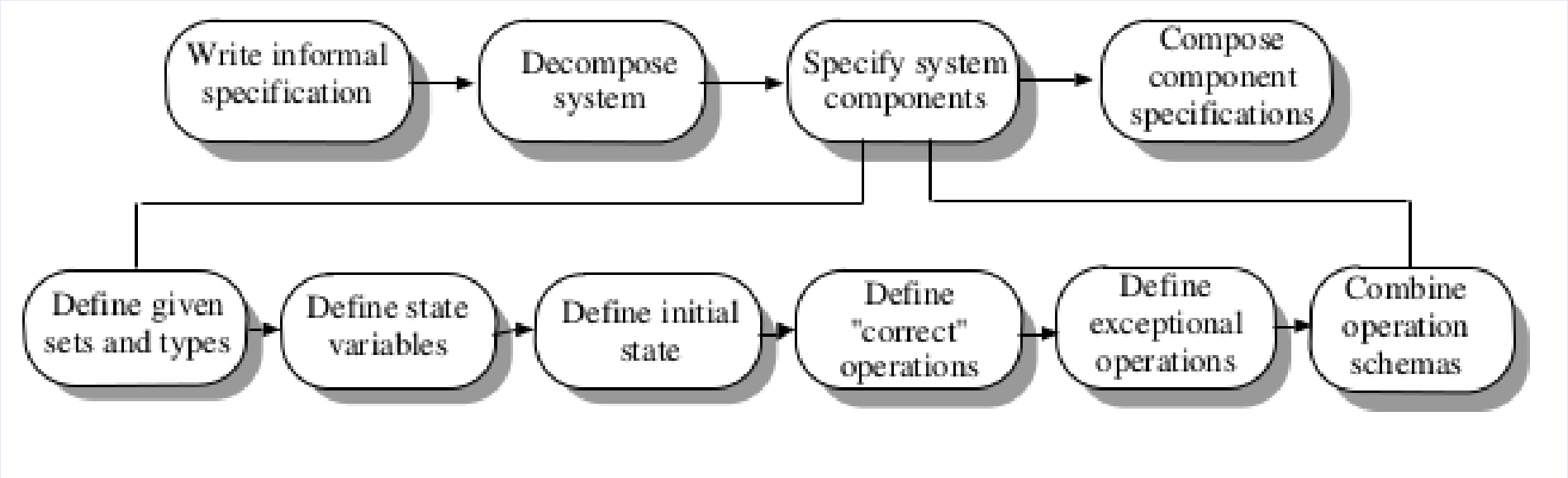
capacity < contents + amount?
r! = "Insufficient tank capacity - Fill cancelled"

# Definición de operaciones

- Ejemplo 6: operación Fill
  - Los esquemas Fill-OK y OverFill se combinan mediante una disyunción para especificar la operación completa
    - Las signaturas se mezclan
    - Los predicados son independientes y están separados por el operador $\vee$.
      - Uno de los dos será verdad.

# Proceso de especificación Z

- Especificación formal de un sistema
  - Proceso incremental
    - Descripción individual de componente
    - Composición de los componente individuales



# Definición de tipos

- Tipos en Z
  - Definidos como conjuntos
- Clases de tipos
  - *Incorporados*
    - $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$
  - *Tipos dados*
    - Definiciones parciales
      - Sólo se define el nombre
    - Dependen de los conjuntos utilizados en las aplicaciones que se modelen
      - [NAME, DATE]
      - [Libro, Persona, Autor, Tema]

# Definición de tipos

- Clase de tipos
  - *Definidos explícitamente*
    - En forma de signature
      - `x:t`
    - Con una restricción
      - `x: t | x ∈ s`
  - *Tipos libres: enumeraciones*
    - Mensaje ::= Ok | yaMatriculado | sinSitio | noMatriculado
    - sem_model_types = {relation, entity, attribute}

# Definición de tipos

- Un esquema puede definir un nuevo tipo
  - Similar a una estructura en C
  - Para referirnos a un componente del esquema usamos el operador ".",
    - Como si se accediera a un campo de una estructura

Month::=jan | feb | mar | apr | may | jun | jul | aug | sep | oct | nov |dec

**Date**

month:Month
day:1..31

month $\subseteq$ {sep,apr,jun,nov} $\Rightarrow$ day $\leq$ 30
month = feb $\Rightarrow$ day $\leq$ 29

# Relaciones

- En una especificación formal necesitamos establecer relaciones entre objetos
  - Conexiones entre pares de objetos
- Definición
  - Conjunto de pares ordenados.
    - Subconjunto del producto cartesiano
  - $X \leftrightarrow Y$
    - X e Y dos conjuntos cualesquiera
    - $X \leftrightarrow Y == P(X \times Y)$
  - Para indicar uno de los elementos de la relación
    - $(x,y)$ o $x \mapsto y$

# Relaciones

- Ejemplo
  - Las relaciones se definen como los axiomas

Conductores == {elena,indra,jaime,cati}
Coche == {alfa,escarabajo,ford,renault}

_conduce_:Conductores ↔ Coches

conduce = {elena ↦ escarabajo, indra ↦alfa, jaime ↦ escarabajo, cati ↦ ford}

# Relaciones

- Obtener información de las relaciones
  - Dominio de una relación
    - $\text{dom } R = \{x:X, y:Y \mid x \mapsto y \in R \bullet x\}$
      - ***dom*** *conduce = {elena, indra, jaime,cati}*
  - Rango de una relación
    - $\text{ran } R = \{x:X, y:Y \mid x \mapsto y \in R \bullet y\}$
      - ***ran*** *conduce = {alfa, escarabajo, ford}*
  - Restricción de dominio, dado un conjunto $A \subseteq X$
    - $A \lhd R = \{x:X, y:Y \mid x \mapsto y \in R \land x \in A \bullet x \mapsto y\}$
      - *{jaime, cati}* $\lhd$ *conduce = {jaime* $\mapsto$ *escarabajo, cati* $\mapsto$ *ford}*
  - Restricción de rango, dado un conjunto $B \subseteq X$
    - $R \rhd B = \{x:X, y:Y \mid x \mapsto y \in R \land y \in B \bullet x \mapsto y\}$
      - *conduce* $\rhd$ *{alfa,renault} = {indra* $\mapsto$*alfa}*

# Relaciones

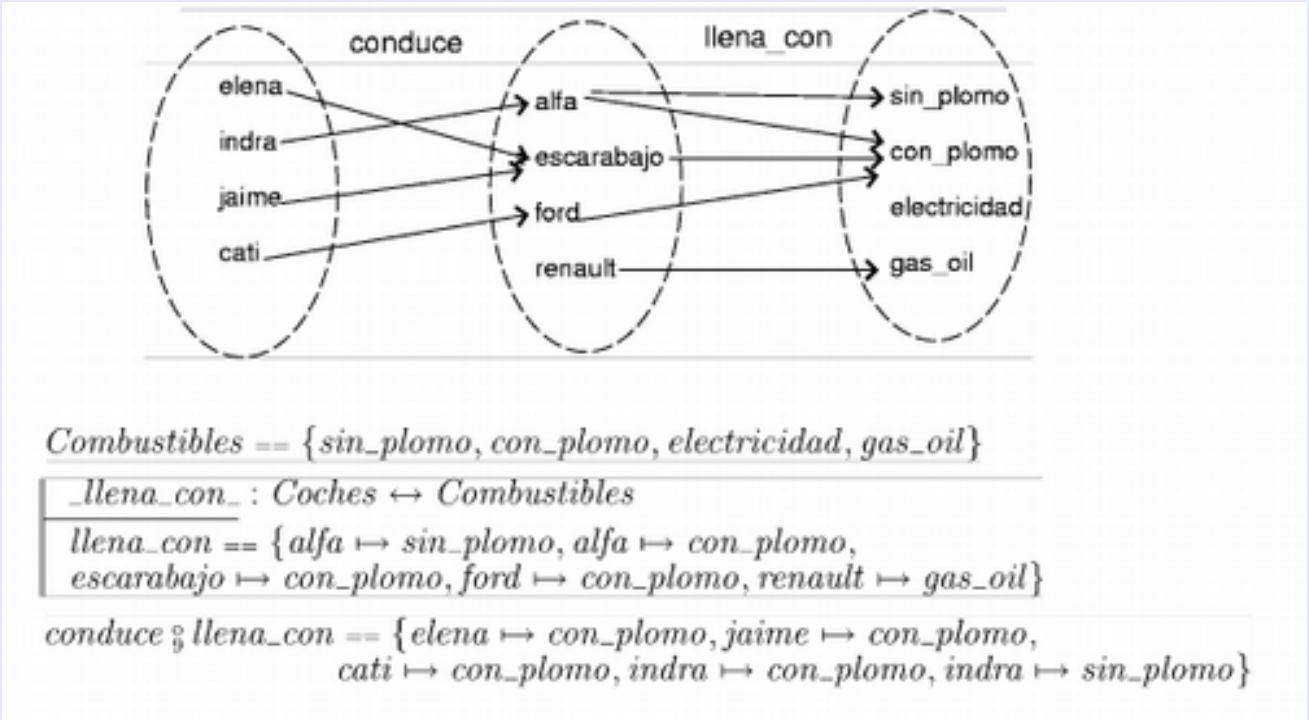
- Obtener información de las relaciones
  - Substracción de dominio, equivalente a $(X \backslash A) \lhd R$
    - $A \lhd R = \{x:X, y:Y \mid x \mapsto y \in R \land x \notin A \bullet x \mapsto y\}$
      - *{jaime}◄ conduce={elena ↦ escarabajo, indra ↦alfa,cati ↦ ford}*
  - Substracción de rango
    - $R \rhd B = \{x:X, y:Y \mid x \mapsto y \in R \land y \notin B \bullet x \mapsto y\}$
      - *conduce ▻ {escarabajo} = {indra ↦alfa, cati ↦ ford}*
  - Imagen relacional
    - $R (|A|) = \text{ran} (A \lhd R)$
      - *conduce({jaime,cati}) = {escarabajo,ford}*

# Relaciones

- Obtener información de las relaciones
  - Inversa de una relación
    - Dada una relación R de tipo X ↔ Y
      - X → origen
      - Y → destino
    - La operación inversa (~) intercambia dichos conjuntos
      - ∀ x:X, y:Y • x ↦y ∈ R ⇒ y ↦x ∈ R ~
    - Ejemplo
      - Conduce ~ = {escarabajo ↦ elena, alfa ↦ indra, escarabajo ↦ jaime, ford ↦ cati}

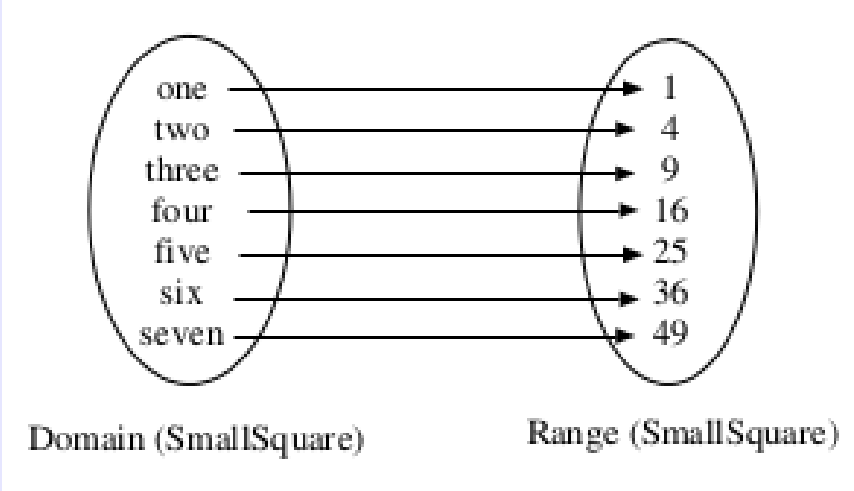
# Relaciones

- Composición de relaciones
  - Se puede construir una relación a partir de otras dos
  - Sintaxis: R; S
    - Dadas R de tipo X ↔Y  y S de tipo Y↔ Z
      - x ↦ z ∈ R;S ⇔  ∃y:Y • x↦y ∈ R ∧  y↦z ∈ S



# Funciones

- Tipo especial de relaciones
  - Cada objeto origen tiene como máximo una imagen



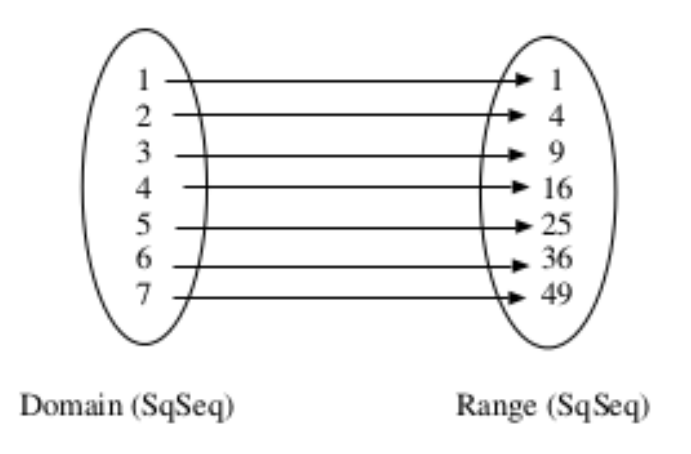

  - Funciones parciales
    - $f \in S \nrightarrow T \Leftrightarrow f \in S \leftrightarrow T \land \forall a: S, b:T, b':T \mid a \mapsto b \in f \land a \mapsto b' \in f \bullet b=b'$
  - Funciones totales
    - $f \in S \rightarrow T \Leftrightarrow f \in S \nrightarrow T \land \forall a:S \bullet \exists b:T \bullet a \mapsto b$

# Funciones

- Operadores sobre funciones
  - Los mismos que se definieron para las relaciones
  - Sobrecarga de funciones ($\oplus$)
    - Añade y/o actualiza un $x \mapsto y$ a los pares de la función (F)
    - $F \oplus x \mapsto y$
      - Si $x \notin dom(F) \rightarrow$ añade $x \mapsto y$ a F
      - Si $x \in dom(F) \rightarrow$ actualiza el valor de y en F
  - Aplicación de una función
    - F(a)
      - Nos da el único objeto que es imagen de a

# Secuencias

- Colección ordenada de objetos
  - A veces es relevante guardar el orden de los objetos
  - Los objetos son referenciados por la posición que ocupan
    - S(1) → primer elemento de la secuencia S
    - S(5) → quinto elemento de la secuencia S
  - Puede verse como una función donde el domino está formado por números naturales consecutivos



# Secuencias

- Notación
  - Secuencia vacía: <>
  - Otras secuencias: <a,b,c>
- Operadores
  - Concatenación
    - <a,b,c>^<d,e> = <a,b,c,d,e>
  - Inversa
    - rev <a,b,c> = <c,b,a>
  - Cardinalidad
    - # <a,b,c> =3

# Secuencias

- Operadores (descomposición de secuencias)
  - Head:
    - Primer elemento de la secuencia
      - *head <a,b,c> = a*
  - Last
    - Último elemento de la secuencia
      - *last <a,b,c> = c*
  - Tail
    - Secuencia sin el primer elemento
      - *tail <a,b,c> = <b,c>*
  - Front
    - Secuencia sin el último elemento
      - *front <a,b,c> <a,b>*

# Secuencias

- Operadores
  - Filtro (S ↾ U)
    - Extrae una secuencia que contiene únicamente los elementos de S que pertenecen al subconjunto U
      - *<a,b,c,d,e,d,c,b,a> ↾ {a,d} = <a,d,d,a>*
  - Extraccion (V 1 S)
    - Extrae una secuencia que contiene los elementos de S que ocupan las posiciones indicadas por V
      - *{1,3,5} 1 {a,b,c,d,e,d,c,b,a} = {a,c,e}*

# Secuencias

- Operadores
  - Concatenación distribuida ^/
    - Si q es una secuencia de secuencias ^/q es el resultado de concatenar todos los elementos de q, uno tras otro
      - *^/ <<a,b,c>,<d,e>,<f,g>> = <a,b,c,d,e,f,g>*

# Bolsas

- Si se desea reflejar multiplicidad pero no orden, se deben utilizar bolsas
- Notación
  - [a,a,b,b,c,c ] es la misma bolsa que [a,b,c,a,b,c]
  - [] bolsa vacía
- Operadores
  - count
    - Devuelve para cada elemento, el nº de veces que aparece
      - Count [a,a,b,b,c,c] = {a ↦ 2, b ↦ 2, c ↦ 2}

# Bolsas

- Operadores
  - *B#x*
    - Número de veces que un elemento aparece en la bolsa
      - [a,a,b,b,c,c] # c = 2
  - Pertenencia a una bolsa (*x* ∈ *B*)
    - x pertenece a B si aparece en la bolsa una o más veces
      - b ∈ [a,a,b,b,c,c] → verdadero
      - d ∈ [a,a,b,b,c,c] → falso
  - Sub-bolsa ( B ⊆ C)
    - B es una sub-bolsa de C si cada elemento aparece menos veces en B que en C

# Bolsas

- Operadores
  - Unión de bolsas (B∪+C)
    - Cada elemento aparece un numero de veces igual a la suma de las veces que aparece en B y en C
  - Diferencia de bolsas (B ∪- C)
    - Cada elemento aparece un numero de veces igual al número de veces que aparece en B menos el número de veces que aparece en C (0 si es negativo)
  - Items
    - Transforma una secuencia en bolsa

# Operaciones sobre esquemas

- Conjunción
  - Modo más simple de combinar dos esquemas
  - S ∧ T
    - Unión de las variables de los dos esquemas
    - Conjunción de los predicados de los mismos


  - Si una misma variable está en los dos esquemas debe tener el mismo tipo

# Operaciones sobre esquemas

- Disyunción
  - Permite definir alternativas en el comportamiento del sistema
  - $S \vee T$
    - Unión de las variables de los dos esquemas
    - Disyunción de los predicados de los mismos
  - Si una misma variable está en los dos esquemas debe tener el mismo tipo

# Operaciones sobre esquemas

- Composición
  - Dados dos esquemas que describan operaciones, la composición de los mismos describe el cambio de estado que se produce cuando la segunda operación se realiza a continuación de la primera
  - Para que esté definida los dos esquemas deben referirse al mismo estado
  - Notación
    - OpOne; OpTwo