Lección 1
Lenguaje algorítmico para programas secuenciales

- Descripción de datos
- Descripción de acciones
- Composición de acciones
- Abstracción de acciones
- Especificación de algoritmos
- Semántica axiomática del lenguaje secuencial
- Reglas de inferencia

Notación algorítmica para datos

predefinidos

```
Bool
Ent
Real
Car
String
```

dec. de variables

```
Vars
    i:Ent := 0
    x:Real
    s:String(10):= "hola"
    d:dias
    m:vector(1..n,1..n) de Real
    tenedores: vector(1..5) de Bool
                := (1..5,false)
```

inicialización

def. de constantes

```
Consts
    n:=100
    hoy: dias := viernes
```

Notación algorítmica para datos

dec. de nuevos tipos

```
Tipos
  ptTenedores= vector(1..5) de Bool
                      := (1..5,false)
  cola= Reg
    primero: Ent :=1
    ultimo: Ent :=1
    tamaño: Ent := 0
    contenido: vector(1:n) de Ent
  FReg

dias=(lun,mar,mie,jue,vie,sab,dom)
valorAceptable = 1..10
```

también inicialización en tipos

enumeración

subrangos

Notación algorítmica para acciones

asignación

```
variable := expresión
```

asignación múltiple

```
<v1...vn> := <exp1...expn>
```

intercambio

```
variable :=: variable
```

inst. nula

```
seguir
```

selección

indeterminista

```
Sel
    C1:  A1
    ......
    Cn:  An
FSel
```

iteración

indeterminista

```
Mq
    C1:  A1
    ......
    Cn:  An
FMq
```

Notación algorítmica para acciones

iteración indexada

```
Pt [v := expresion..expresion]+
                  [ tq expresion]
   A

FPt
```

- *declaraciones implícitas*
- *alcance: bucle*

```
Pt i:=1..n
   v(i) := v(i)+1
FPt
```

```
Pt i:=1..n;j:=i..n
      tq v(i)>v(j)
   v(i) := v(j)
FPt
```

```
Pt i:=1..n;j:=i..n
   v(i,j) := v(j,i)
FPt
```

Notación algorítmica para acciones

ejemplo de función

```
Funcion fact(E n:Ent)
                Dev (f:Ent)
Principio
  Sel
    n=0: f := 1
    n=1: f := 1
    n>1: f := f*fact(n-1)
  FSel
  Dev(f)
Fin
```

ejemplo de proced.

```
Accion incr(ES x:Ent)
Principio
   x:= x+1
Fin
```

***Nota**: normalmente, cuando por el contexto esté claro, omitiremos el tipo de las variables*

Sobre el paso de parámetros

- **in** — within the subprogram the parameter acts as a local constant — a value is assigned to the formal parameter on entry to the procedure or function. This is the only mode allowed for functions. It is the default mode.

- **out** — within the subprogram the value of the parameter can be written and read — a value is assigned to the calling parameter (which must be a variable) upon termination of the subprogram.

- **in out** — within the subprogram the parameter acts as a variable — upon entry to the subprogram a value is assigned to the formal parameter; upon termination the value attained is passed back to the calling parameter.

Interestingly, these parameter modes do not dictate the method of implementation, which could be either by copy or by reference. There are, however, some restrictions. Non-composite types (for instance, scalars such as integers and booleans) must be passed "by copy". Also, certain more complex types (for example, tasks and protected types) must be passed "by reference".

especificación

implementación

Sobre el paso de parámetros

```
Accion incr(ES x:Ent)
Principio
   x:= x+1
Fin
```

```
y := 27
inc(y)
--y=28
```

valor

```
y := 27
  --x=27 ∧ y = 27
  x := x+1
  --x=28 ∧ y = 27
--y=28
```

referencia

```
y := 27
  --x=27 ∧ y = 27
  x := x+1
  --x=28 ∧ y = 28
--y=28
```

Especificación y verificación de algoritmos

- **Estado de un proceso**: tupla de valores de sus variables
- Para decir algo sobre el estado
  - daremos una **aserción** (predicado) sobre sus variables
  - usaremos para ello la **Lógica de Primer Orden**
- Una acción
  - cambia el estado del proceso
  - se puede ver como un **transformador de predicados**

Especificación y verificación de algoritmos

Las FBF

cualquier fórmula atómica es una FBF

* True,False
* una variable booleana
* cualquier expresión booleana

(P) siendo P una FBF

¬P siendo P una FBF

P ∧ Q siendo P,Q FBF

P ∨ Q siendo P,Q FBF

P → Q siendo P,Q FBF

P ↔ Q siendo P,Q FBF

∀ α∈D.P siendo P FBF

∃ α∈D.P siendo P FBF

D: dominio

Especificación y verificación de algoritmos

Leyes conmutativas

$(E1 \wedge E2) = (E2 \wedge E1)$

$(E1 \vee E2) = (E2 \vee E1)$

$(E1 \leftrightarrow E2) = (E2 \leftrightarrow E1)$

Leyes asociativas

$E1 \wedge (E2 \wedge E3) = (E1 \wedge E2) \wedge E3$

Leyes distributivas

$E1 \wedge (E2 \vee E3) = (E1 \wedge E2) \vee (E1 \wedge E3)$

$E1 \vee (E2 \wedge E3) = (E1 \vee E2) \wedge (E1 \vee E3)$

Leyes de Morgan

$\neg(E1 \wedge E2) = \neg E1 \vee \neg E2$

$\neg(E1 \vee E2) = \neg E1 \wedge \neg E2$

Ley de la negación

$\neg(\neg E)=E$

Ley del "medio" imposible

$E \vee \neg E = T$

Ley de la contradicción

$E \wedge \neg E = F$

Especificación y verificación de algoritmos

Ley de la implicación

$E1 \rightarrow E2 = \neg E1 \vee E2$

Ley de la igualdad

$(E1 \leftrightarrow E2) = (E1 \rightarrow E2) \wedge (E2 \rightarrow E1)$

Leyes de simplificación del OR

$E1 \vee E1 = E1$

$E1 \vee T = T$

$E1 \vee F = E1$

$E1 \vee (E1 \wedge E2) = E1$

Leyes de simplificación del AND

$E1 \wedge E1 = E1$

$E1 \wedge T = E1$

$E1 \wedge F = F$

$E1 \wedge (E1 \vee E2) = E1$

Leyes de $\forall$ y $\exists$

$(\forall \alpha \in D.P) = T$ si $D = \varnothing$

$(\exists \alpha \in D.P) = F$ si $D = \varnothing$

$\neg(\forall \alpha \in D.P) = (\exists \alpha \in D.\neg P)$

$\neg(\exists \alpha \in D.P) = (\forall \alpha \in D.\neg P)$

$(\blacklozenge \alpha \in D.P) = (\blacklozenge \beta \in D.P[\beta/\alpha])$

Si $\beta \notin$ variables(P)

$\blacklozenge \alpha \in D_1. \blacklozenge \beta \in D_2.P =$

$\blacklozenge \beta \in D_2. \blacklozenge \alpha \in D_1.P$

$\blacklozenge$ *: cuantificador*

Especificación y verificación de algoritmos

- **Prueba:** hacer "algunas" pruebas
- **Razonamiento operacional**: análisis exhaustivo
- **Razonamiento axiomático:**
  - un "lenguaje formal" para especificar las propiedades
  - una definición axiomática del lenguaje
  - **acción** = transformador de predicados

```
{Pre}
acción
{Post}
```

sobre el programa o un modelo

- Necesitamos extender el lenguaje para tratar la concurrencia

Especificación y verificación de algoritmos

- Especificación **Pre/Post:**

```
--Pre
acción
--Post
```

- Significado:
  - ***Si*** *al invocar la ejecución de la acción el estado cumple Pre*
    entonces *la acción termina*
    el estado al que llega verifica el predicado Post
    si no *no podemos decir nada*
- Ejemplo:

```
Accion niSeSabe(E  x:....
                S  y:....
                ES z:....)
-- Pre:  Q(x,z)
-- Post: R(x,y,z)
```

Anotación de algoritmos

- Habitualmente, anotaremos los algoritmos

```
Accion swap(ES x,y:Ent)
-- Q: x=X ∧ y=Y
-- R: x=Y ∧ y=X
Principio
                --Q:  x=X ∧ y=Y
  x := x+y
                --Q1:  x=X+Y ∧ y=Y
  y := x-y
                --Q2:  x=X+Y ∧ y=X
  x := x-y
                --R: x=Y ∧ y=X
Fin
```

completamente anotado

esquema de demostración

Semántica axiomática

- Para cada instrucción "A",
  ¿cuándo se puede afirmar que **{Q} A {R}** es correcto?

- Regla de la **instrucción nula**: **{Q} seguir {Q}**

- Regla de la **asignación**: **$\{R_x^E\}$ x := E {R}**

- Regla del **intercambio**: **$\{R_{x1}^{x2}{}_{x2}^{x1}\}$ x1 :=: x2 {R}**

- Regla de la **asignación múltiple**:

$$\{R_{x_1}^{e_1}{}^{\ldots}{}_{x_n}^{e_n}\}$$
$$\langle v_1 \ldots v_n \rangle := \langle exp_1 \ldots exp_n \rangle$$
$$\{R\}$$

Semántica axiomática

- Regla de la **composición secuencial**:

$$\frac{\{Q\}\ A1\ \{Q'\},\{Q'\}\ A2\ \{R\}}{\{Q\}\ A1;A2\ \{R\}}$$

- Regla de la **selección**:

$$\frac{Q \wedge \neg(C_1 \vee \ldots \vee C_n) \rightarrow R,\quad \forall i \in \{1..n\}.\{Q \wedge C_i\}\ A_i\ \{R\}}{\{Q\}\ Sel...FSel\ \{R\}}$$

- Regla de la **iteración**:

corrección parcial

$$\frac{\forall i \in \{1..n\}.\{I \wedge C_i\}\ A_i\ \{I\}}{\{I\}\ Mq...FMq\ \{I \wedge \neg(C_1 \vee \ldots \vee C_n)\}}$$

I: invariante

- ¿Cuándo es correcto?

$$\{Q\}\ Mq...FMq\ \{R\}$$

Semántica axiomática

- Regla de la invocación a una acción

```
accion miAcc(E x:tX;ES y:tY;S z:tZ)
--Pre:  Q(x,y)
--Post: R(x,y,z)
        A
```

Considerar **miAcc(a,b,c)** (b,c distintos))
Sea **I** un predicado que no depende de b,c .
Entonces

$$\{Q^{a,b}_{x,y} \wedge I\}\ \texttt{miAcc(a,b,c)}\ \{R^{a,b,c}_{x,y,z} \wedge I\}$$

I: invariante

* x no es asignable
* actuales distintos

Semántica axiomática

- Reglas de la invocación a una función

```
función F(E x:tX) dev (z:tZ)
--Pre_f(x)
--Post_f(x,z)
```

Regla 1

$$\frac{Q \rightarrow Pre_f\,{}^{E}_{x}\,, Post_f\,{}^{E,b}_{x,z} \rightarrow R}{\{Q\}\ b:= f(E)\ \{R\}}$$

Regla 2

```
e:= I(f(E))

es equivalente a

TEMP := f(E)
e:= I(TEMP)
```