SKAN^2: Una Notación Algorítmica basada en GCL

Octubre, 2003

Resumen

En este documento se presenta SKAN^2, una notación algorítmica en la que se expresarán los algoritmos a lo largo del curso de Algoritmos y Programación. La definición de SKAN^2 está basada en GCL (*Guarded Commands Language*) y otras notaciones de programación utilizadas para describir algoritmos. SKAN^2 es un lenguaje algorítmico, es decir, un lenguaje de alto nivel para expresar algoritmos, independientemente del lenguaje de programación en el que se piense implementar.

1. Constantes

Las constantes son entes cuyo valor es fijo, es decir, el valor asociado a una constante no cambia. Cada constante tiene asociado un nombre o identificador único. El identificador de una constante se expresa utilizando solamente letras mayúsculas.
La sintaxis de una declaración de constante tiene la siguiente estructura:

```
const
  ident_1 = val_1;
  ident_2 = val_2;
        ⋮
  ident_k = val_k.
```

Donde:

- `ident`$_i$ es el identificador (nombre) asociado a la i-ésima constante ($1 \leq i \leq k$).
- `val`$_i$ es el valor asociado a la i-ésima constante ($1 \leq i \leq k$).

2. Variables

Las variables son entes sobre los cuales se efectúan las acciones de un algoritmo, y el efecto (de las acciones) se percibe a través de un cambio de estado (en las variables).
Cada variable tiene asociado un nombre o identificador único, y puede tomar ciertos valores pertenecientes a un dominio.
La sintaxis de una declaración de variable tiene la siguiente estructura:

```
var
  ident_11, ident_12, ..., ident_1k_1 : T_1;
  ident_21, ident_22, ..., ident_2k_2 : T_2;
                  ⋮
  ident_n1, ident_n2, ..., ident_nk_n : T_n.
```

Donde:

- $\texttt{ident}_{ij}$ es el identificador (nombre) asociado a la j-ésima variable del i-ésimo tipo ($1 \leq i \leq n$).
- $\texttt{T}_i$ es el identificador (nombre) asociado al i-ésimo tipo ($1 \leq i \leq n$).

3. Tipos de Datos

El tipo de dato T de una variable x define el conjunto de valores que x puede tomar y determina las operaciones que se pueden realizar sobre x. Se escribe $x : T$ para denotar que una variable x es de tipo T.

3.1. Tipos elementales o primitivos

Los tipos elementales o primitivos se consideran pre-definidos, es decir, conocidos *a priori*. A partir de los tipos elementales se pueden construir otros más complicados. En esta sección se presentan los tipos elementales de SKAN^2.

3.1.1. Tipo Lógico

- Identificador: **logical**
- Dominio: *false*, *true*
- Operaciones: $\neg$ (negación), $\vee$ (conjunción), $\wedge$ (disjunción)
- Relaciones: $\Leftrightarrow$ (equivalencia)

3.1.2. Tipo Entero

- Identificador: **integer**
- Dominio: $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$
- Operaciones: $+$ (suma), $-$ (diferencia), $*$ (producto), **div** (cociente), **mod** (resto), **abs** (valor absoluto)
- Relaciones: $=$ (relación de igualdad), $\neq$ (relación de desigualdad), $<$ (menor), $>$ (mayor), $\leq$ (menor o igual), $\geq$ (mayor o igual)

3.1.3. Tipo Real

- Identificador: **float**
- Dominio: $\mathbb{R}$
- Operaciones: $+$ (suma), $-$ (diferencia), $*$ (producto), / (división)
- Relaciones: $=$ (relación de igualdad), $\neq$ (relación de desigualdad), $<$ (menor), $>$ (mayor), $\leq$ (menor o igual), $\geq$ (mayor o igual)

3.1.4. Tipo Caracter

- Identificador: **char**
- Dominio: {'A', ..., 'Z'} ∪ {'a', ..., 'z'} ∪ {'0', ..., '9'} ∪ {'#', '$', '%', '&' ... }
- Operaciones: Las funciones ordinales
- Relaciones: = (relación de igualdad), ≠ (relación de desigualdad), < (menor), > (mayor), ≤ (menor o igual), ≥ (mayor o igual)

3.1.5. Tipo Cadena

- Identificador: **string**
- Dominio: Secuencias de caracteres
- Operaciones: **head** (primer caracter de una cadena), **tail** (elimina el primer caracter de una cadena y retorna el resto, del segundo caracter en adelante), & (concatenación), **length** (número de caracteres en una cadena)
- Relaciones: = (relación de igualdad), ≠ (relación de desigualdad), < (menor), > (mayor), ≤ (menor o igual), ≥ (mayor o igual)

3.2. Tipos compuestos o estructurados

Los tipos de datos compuestos o estructurados están constituidos por otros tipos de datos (primitivos o estructurados) definidos previamente. En esta sección se presentan los tipos compuestos de SKAN[2].
Un tipo compuesto debe declararse. Una declaración de un tipo compuesto especifica el identificador del tipo y su definición, es decir, cómo se construye.
La sintaxis para la declaración de un tipo compuesto tiene la siguiente estructura:

```
type
  T1 = <def1>;
  T2 = <def2>;
         ⋮
  Tm = <defm>.
```

Donde:

- $\texttt{T}_i$ es el identificador del i-ésimo tipo compuesto ($1 \leq i \leq m$)
- $\texttt{def}_i$ es la definición del i-ésimo tipo compuesto ($1 \leq i \leq m$)

3.2.1. Tipos Ordinales

Definición 1 (Tipo Ordinal)
Un tipo T es ordinal si y sólo si sus elementos están organizados de manera tal que verifican las siguientes propiedades:

- Existen dos elementos notables en T: un primer elemento y un último elemento.
- Para cada elemento en T, excepto el último, existe un único elemento que le sigue llamado sucesor.
- Para cada elemento en T, excepto el primero, existe un único elemento que le precede llamado predecesor.

- Cada elemento de T tiene un valor entero asociado a él, llamado número ordinal, que representa la posición relativa del elemento en la secuencia de valores que define a T.

Observación
Todos los tipos de datos primitivos, con excepción de los tipos **float** y **string**, son tipos ordinales.

Definición 2 (Funciones ordinales)
Sea T un tipo de dato ordinal. Se definen entonces las funciones ordinales como sigue:

- **ord** : $T \to \mathbb{Z}$; retorna el número ordinal de su argumento.
- **pred** : $T \to T$; retorna el predecesor de su argumento.
- **suc** : $T \to T$; retorna el sucesor de su argumento.

3.2.2. Tipo Enumerado

Un enumerado es una secuencia ordenada y finita de valores referenciados por identificadores. Este tipo está definido por la enumeración explícita de los identificadores que lo conforman. La sintaxis para la definición de un tipo Enumerado tiene la siguiente estructura:

```
type
  T = (ident_1, ident_2, ..., ident_n)
```

Donde:

- `T` es el identificador del nuevo tipo
- Los `ident`$_i$ $(1 \leq i \leq n)$ son los identificadores que forman el tipo.
- El orden de los identificadores en `T` está definido por la siguiente regla:

$$\forall i \forall j : (1 \leq i, j \leq n) : [(ident_i < ident_j) \Leftrightarrow (i < j)]$$

Características

- Una variable x de tipo Enumerado T sólo puede tomar valores de la lista de identificadores que define a T.
- Los únicos operadores asociados al tipo Enumerado son el operador de asignación y los operadores de comparación.
- Un identificador no puede pertenecer a más de un tipo Enumerado.

3.2.3. Tipo Subrango o Intervalo

La sintaxis para la definición de un tipo Subrango tiene la siguiente estructura:

```
type
  T = min .. max
```

Donde:

- `T` es el identificador del nuevo tipo
- `min` y `max` son constantes del mismo tipo que especifican los límites inferior y superior del intervalo
- El tipo de las constantes `min` y `max` puede ser **integer**, **char** o un tipo Enumerado
- La definición de un tipo Subrango es aceptable si y sólo si se verifica que `min` $\leq$ `max`

Las operaciones que se pueden realizar sobre una variable de tipo Subrango son iguales que las del tipo a partir del cual está definido.

3.2.4. Tipo Arreglo

La sintaxis para la definición de un tipo Arreglo de r dimensiones ($r \geq 1$) tiene la siguiente estructura:

```
type
  T = array [m1..n1,m2..n2,...,mr..nr] of T0
```

Donde:

- `T` es el identificador del nuevo tipo.
- Los intervalos $\texttt{m}_1\texttt{..n}_1\texttt{,m}_2\texttt{..n}_2\texttt{,...,m}_r\texttt{..n}_r$ son los índices del arreglo. El tipo del índice debe ser un tipo ordinal
- $\texttt{T}_0$ es el tipo base del arreglo. El tipo base de un arreglo puede ser primitivo o estructurado

3.2.5. Tipo Tupla

Los componentes de una tupla se conocen como campos. Los nombres de los campos se conocen como identificadores de campo.
La sintaxis para la definición de un tipo Tupla tiene la siguiente estructura:

```
type
  T = tuple
        ident11, ident12, ..., ident1k1 : T1;
        ident21, ident22, ..., ident2k2 : T2;
                 ⋮
        identn1, identn2, ..., identnkn : Tn
      end
```

Donde:

- `T` es el identificador del nuevo tipo.
- $\texttt{ident}_{ij}$ es el identificador (nombre) del j-ésimo campo del i-ésimo tipo ($1 \leq i \leq n$).
- $\texttt{T}_i$ es el tipo del i-ésimo campo ($1 \leq i \leq n$).

3.2.6. Tipo Apuntador

La sintaxis para la definición de un tipo Apuntador tiene la siguiente estructura:

```
type
  PTR_T = pointer to T0
```

Donde:

- `PTR_T` es el identificador del nuevo tipo
- $\texttt{T}_0$ es el tipo referenciado por `PTR_T`. El tipo referenciado puede ser primitivo o estructurado

4. Acciones

4.1. Asignación

La sintaxis de una acción de asignación simple tiene la siguiente estructura:

$$\texttt{var} \leftarrow \texttt{expr}$$

Donde `expr` es el valor asociado a la variable `var`.

4.2. Condicional

La sintaxis de una acción condicional tiene la siguiente estructura:

```
if cond then
    S1
else
    S2
endif
```

Donde:

- `cond` es una condición que retorna un valor de tipo **logical**.
- $\texttt{S}_1$ es el conjunto de acciones a ejecutar en caso de que la condición `cond` sea verdadera.
- $\texttt{S}_2$ es el conjunto de acciones a ejecutar en caso de que la condición `cond` sea falsa.

4.3. Selección

La sintaxis de una acción de selección tiene la siguiente estructura:

```
selection (var)
{
 val1: S1
 val2: S2
     ⋮
 valr: Sr
 default: S
}
```

Donde:

- `var` es una variable que se llama selector de la sentencia
- $\texttt{val}_i$ $(1 \leq i \leq r)$ es un valor constante de tipo ordinal que se denominan etiquetas de la sentencia
- $\texttt{S}_i$ $(1 \leq i \leq r)$ es el conjunto de acciones a ejecutar en caso de que `var =` $\texttt{val}_i$
- La variable `var` toma uno y sólo uno de los valores $\texttt{val}_i$ $(1 \leq i \leq r)$
- Si $\forall i : 1 \leq i \leq r :$ `var` $\neq$ $\texttt{val}_i$ entonces se ejecutará el conjunto de acciones `S` correspondientes a la cláusula `default`

4.4. Iteración

La sintaxis de una acción iterativa tiene alguna de las siguientes estructuras:

```
while cond do
{
    S
}
```

Donde:

- `cond` es una condición que retorna un valor de tipo **logical**

- `S` es el conjunto de acciones a ejecutar en caso de que la condición `cond` sea verdadera

```
for k in [min .. max] do
{
    S
}
```

Donde:

- `k` es una variable denominada contador de iteraciones
- `[min .. max]` define el intervalo de valores posibles para la variable `k`
- `S` es el conjunto de acciones a ejecutar `max - min + 1` veces
- La variable `k` puede realizar un recorrido ascendente sobre el intervalo (tomando como valor inicial `min` y valor final `max`, e incrementando de uno en uno) o un recorrido descendente (tomando como valor inicial `max` y valor final `min`, y decrementando de uno en uno)

4.5. Acciones de Entrada y Salida

4.5.1. Entrada / Salida (Estándar)

Entrada: La sintaxis de una acción de entrada tiene la siguiente estructura:

$$\underline{\texttt{read}}(x_1, x_2, \ldots, x_n)$$

La semántica de esta acción es la lectura del valor de las variables x_1, x_2, ..., x_n de un dispositivo de entrada.

Salida: La sintaxis de una acción de salida tiene la siguiente estructura:

$$\underline{\texttt{write}}(x_1, x_2, \ldots, x_n)$$

La semántica de esta acción es la escritura del valor de las variables x_1, x_2, ..., x_n en un dispositivo de salida.

4.5.2. Entrada / Salida (Archivos)

Apertura de un archivo: La sintaxis de una operación de apertura de un archivo tiene la siguiente estructura:

$$\underline{\texttt{openfile}}(\texttt{f,mode,``name''})$$

Inicializa el mecanismo de acceso (lectura / escritura) al archivo identificado por `‘‘name’’`, asociándolo con el identificador lógico `f`. Los valores legítimos de `‘‘mode’’` son los siguientes: `r` (lectura), `w` (escritura), `rw` (lectura y escritura). En caso de no existir el archivo, la acción lo crea.

Lectura de un archivo: La sintaxis de una operación de lectura de un archivo tiene la siguiente estructura:

$$\underline{\texttt{readfile}}(\texttt{f,x})$$

Copia la información contenida en el registro actual del archivo `f` sobre la variable `x`.

Escritura en un archivo: La sintaxis de una operación de escritura en un archivo tiene la siguiente estructura:

```
writefile(f,x)
```

Copia la información contenida en la variable `x` sobre el registro actual del archivo `f`.

Cierre de un archivo: La sintaxis de una operación de clausura de un archivo tiene la siguiente estructura:

```
closefile(f)
```

Cierra el archivo y, en consecuencia, no es posible realizar más operaciones de lectura / escritura hasta una nueva apertura.

Final de un archivo: La sintaxis de una operación de final de un archivo tiene la siguiente estructura:

```
eof(f)
```

Esta operación permite detectar si se ha llegado o no al final de todos los registros de un archivo.

5. Funciones y Procedimientos

5.1. Declaración de Funciones

```
func ident_func (esp_param_1; esp_param_2; ...; esp_param_r): T_1, T_2, ..., T_q
{
  cuerpo_func
  return(x_1, x_2, ..., x_q)
}
```

Donde:

- `ident_func` es el identificador de la función
- `esp_param`$_i$ tiene la forma siguiente: `ident`$_{i1}$`, ident`$_{i2}$`, ..., ident`$_{ip}$ `: T`$_i$
  `ident`$_{ij}$ es la j-ésima variable del i-ésimo tipo `T`$_i$ ($1 \leq j \leq p$, $1 \leq i \leq r$)
- `T`$_k$ ($1 \leq k \leq q$) es el tipo del k-ésimo valor retornado por la función `ident_func`
- `cuerpo_func` es el conjunto de acciones que definen el cuerpo de la función
- `x`$_i$ es el i-ésimo valor retornado por la función `ident_func`. `x`$_i$ debe ser de tipo `T`$_i$ ($1 \leq i \leq q$)

5.2. Declaración de Procedimientos

proc ident_proc (esp_param$_1$; esp_param$_2$; ...; esp_param$_r$)
{
  cuerpo_proc
}

Donde:

- ident_proc es el identificador del procedimiento
- esp_param$_i$ ($1 \leq i \leq r$) tiene una de las tres formas siguientes:
  - in ident$_1$, ident$_2$, ..., ident$_k$: T$_i$ (Entrada)
  - out ident$_1$, ident$_2$, ..., ident$_k$: T$_i$ (Salida)
  - in-out ident$_1$, ident$_2$, ..., ident$_k$: T$_i$ (Entrada / Salida)
- cuerpo_proc es el conjunto de acciones que definen el cuerpo del procedimiento

6. Estructura de un Algoritmo

La sintaxis de un algoritmo está definida por la siguiente estructura:

```
{
  Declaración de constantes

  Declaración de tipos

  Declaración de variables

  Declaración de funciones y procedimientos

  Cuerpo del algoritmo
}
```

Donde:

- Declaración de constantes tiene la sintaxis especificada en la sección 1
- Declaración de tipos tiene la sintaxis especificada en la sub-sección 3.2
- Declaración de variables tiene la sintaxis especificada en la sección 2
- Declaración de funciones y procedimientos tiene la sintaxis especificada en la sección 5
- Cuerpo del algoritmo utiliza las acciones definidas en la sección 4