Capítulo 2

Algoritmos y Lógica
Introducción al lenguaje del Robot

Objetivos

En este capítulo se verán con mayor profundidad algunos de los conceptos utilizados anteriormente para la definición de algoritmos.

Además se introducirá el concepto de lenguajes de expresión de problemas y los tipos de lenguajes existentes. Se realizará la presentación de la herramienta DaVinci Concurrente que tiene un lenguaje especial, con el que comenzaremos a trabajar en la resolución de problemas utilizando la computadora.

Este capítulo les permitirá aplicar lo visto sobre estructuras de control, pero en el lenguaje previsto para el robot.

Además se introducirán y repasarán algunos conceptos básicos de la lógica proposicional que permitirán representar condiciones complejas utilizadas en las estructuras del robot, aplicadas específicamente a problemas con el robot.

Temas a tratar

- Lenguajes de Expresión de Problemas. Tipos de Lenguajes. Sintaxis y semántica en un Lenguaje.
- DaVinci Concurrente. Operaciones sobre el Robot. Estructura general de un programa. Estilo de Programación. Ambiente de programación.
- Estructuras de control con el Robot
- Revisión del tema: Proposiciones atómicas y moleculares, simbolización y tablas de verdad
- Conectivos lógicos: Conjunción, Disyunción y Negación. Utilización del paréntesis.
- Conclusiones
- Ejercitación

2.1 Lenguajes de Expresión de Problemas. Tipos de Lenguajes. Sintaxis y semántica en un Lenguaje.

En el capítulo anterior se ha utilizado un lenguaje casi natural para especificar las instrucciones que debían llevarse a cabo. Esto, si bien facilita la escritura del algoritmo para quien debe decir como resolver el problema, dificulta la comprensión de dicha solución por

parte de quien debe interpretarla.

En algunos de los ejemplos presentados hasta el momento, seguramente el lector debe haber tenido diferentes interpretaciones. ¿Por qué?

Fundamentalmente porque el lenguaje natural tiene **varios significados para una palabra (es ambiguo) y porque admite varias combinaciones para armar un enunciado**. Estas dos condiciones son "indeseables" para un lenguaje de expresión de problemas utilizable en Informática.

En el ejemplo 1.7:

- ¿Qué sucede si la lámpara está en el centro de la habitación y la escalera no es de dos hojas?
- ¿Dónde se asegura que se dispone de lámparas nuevas?
- ¿"Alcanzar la lámpara" equivale a "tomar la lámpara con la mano para poder girarla"? ¿Cuándo se deja la lámpara usada y se toma la nueva para el reemplazo?

Por medio de las preguntas anteriores nos damos cuenta que el significado de cada instrucción del lenguaje debe ser exactamente conocido y como consecuencia no se pueden admitir varias interpretaciones.

Un lenguaje de expresión de problemas contiene un conjunto finito y preciso de instrucciones o primitivas utilizables para especificar la solución buscada.

Nótese que desde el punto de vista del diseño del algoritmo, el contar con un número finito de instrucciones posibles termina con el problema de decidir, de una forma totalmente subjetiva, el grado de detalle necesario para que los pasos a seguir puedan ser interpretados correctamente. El conjunto de instrucciones determinará cuales son los pasos elementales posibles que se utilizarán para el diseño de la solución.

Un lenguaje de expresión de problemas debe reunir las siguientes características:

- **Debe estar formado por un número de instrucciones finito.**
- **Debe ser completo, es decir que todas las acciones de interés deben poder expresarse con dicho conjunto de instrucciones.**
- **Cada instrucción debe tener un significado (efecto) preciso.**
- **Cada instrucción debe escribirse de modo único.**

2.1.1 Tipos de Lenguajes

No siempre los problemas se expresan con primitivas que representen un subconjunto preciso del lenguaje natural: se puede utilizar un sistema de símbolos gráficos (tales como los de los diagramas de flujo que se observarán en algunos textos de Informática); puede emplearse una simbología puramente matemática; puede crearse un lenguaje especial orientado a una aplicación; pueden combinarse gráficas con texto, etc.

De todos modos, cualquiera sea la forma del lenguaje elegido éste siempre respetará las

características mencionadas anteriormente. ¿Por qué?

Porque si se quiere que una máquina interprete y ejecute las órdenes del lenguaje, por más sofisticada que sea, requerirá que las órdenes diferentes constituyan un conjunto finito; que cada orden pueda ser interpretada de un modo único y que los problemas solubles por la máquina sean expresables en el lenguaje.

2.1.2 Sintaxis y Semántica en un Lenguaje

La forma en que se debe escribir cada instrucción de un lenguaje y las reglas generales de expresión de un problema completo en un lenguaje constituyen su sintaxis.

Por ejemplo hay lenguajes que en su sintaxis tienen reglas tales como:

- Indicar el comienzo y fin del algoritmo con palabras especiales.
- Indicar el fin de cada instrucción con un separador (por ejemplo punto y coma).
- Encerrar, entre palabras clave, bloques de acciones comunes a una situación del problema (por ejemplo todo lo que hay que hacer cuando la condición es verdadera dentro de la estructura de control de selección).
- Identar adecuadamente las instrucciones.

El significado de cada instrucción del lenguaje y el significado global de determinados símbolos del lenguaje constituyen su semántica.

Dado que cada instrucción debe tener un significado preciso, la semántica de cada instrucción es una indicación exacta del efecto de dicha instrucción. Por ejemplo, ¿Cuál sería el efecto de una instrucción del tipo:

si (condición)
……..
sino
.....
como la vista en el Capítulo 1?

El efecto sería:

1. Evaluar la condición.
2. Si la condición es Verdadera, realizar las acciones indicadas antes del **sino**.
3. Si la condición es Falsa realizar las acciones identadas indicadas a continuación del **sino**.

2.2 DaVinci Concurrente. Operaciones sobre el Robot. Estructura general de un programa. Estilo de programación. Ambiente de programación.

A lo largo de este curso se trabajará con una máquina abstracta simple, un robot móvil controlado por un conjunto reducido de primitivas, que permite modelizar recorridos y tareas dentro de una ciudad.

El robot que se utilizará posee las siguientes capacidades básicas:

1. Se mueve.
2. Se orienta hacia la derecha, es decir, gira 90 grados en el sentido de las agujas del reloj.
3. Dispone de sensores visuales que le permiten reconocer dos formas de objetos preestablecidas: flores y papeles. Los mismos se hallan ubicados en las esquinas de la ciudad.
4. Lleva consigo una bolsa donde puede transportar flores y papeles. Está capacitado para recoger y/o depositar cualquiera de los dos tipos de objetos en una esquina, pero de a uno a la vez. La bolsa posee capacidad ilimitada.
5. Puede realizar cálculos simples.
6. Puede informar los resultados obtenidos.

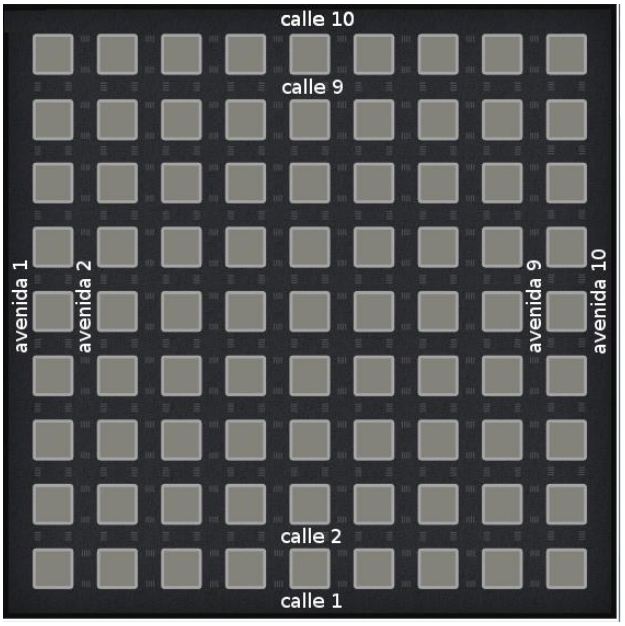


Figura 2.1: La ciudad del robot

La ciudad, en la que el robot se desplaza, está formada por calles y avenidas. Se denominan avenidas a las arterias verticales y calles a las arterias horizontales.

Como lo muestra la figura 2.1, la ciudad está formada por 10 calles y 10 avenidas.

Cada una de las esquinas está determinada por la intersección de una avenida y una calle. **Debe considerarse a las arterias como rectas y a la esquina como el punto de intersección entre dichas rectas.** La esquina se representará por dos coordenadas: la primera indicará el número de avenida y la segunda el número de calle. Por ejemplo, la esquina (2,4) es la intersección de la avenida 2 y la calle 4.
Las flores y los papeles se encuentran siempre en las esquinas. Pueden existir varias flores y

varios papeles en cada esquina. En la búsqueda de reducir el problema del mundo real a los aspectos básicos que debe cubrir el robot en el ambiente DaVinci Concurrente, se han realizado las siguientes abstracciones:

- La ciudad queda reducida a un ámbito cuadrado de 10 calles y 10 avenidas;
- El andar del robot queda asociado con un paso que equivale a una cuadra de recorrido;
- Se reducen los datos en el modelo para tratar sólo con flores y papeles;
- Se aceptan convenciones (el robot solo inicia sus recorridos en la posición (1,1) de la ciudad);
- Se supone que el robot ve y reconoce las flores y los papeles. No es de interés de este curso discutir cómo lo puede hacer.

Es interesante analizar el grado de exactitud del modelo y su relación con los objetivos a cumplir. Está claro que en este ejemplo no se modelizan exactamente la ciudad ni los objetos que están en ella. Tampoco se representa adecuadamente el movimiento de un robot real, que posiblemente tenga que dar un número grande de pasos para recorrer una cuadra. Se ignoran los detalles del proceso de reconocimiento de los objetos e incluso no se considera la posibilidad de que el robot confunda objetos.

Sin embargo, dado que el objetivo planteado es escribir programas que permitan representar recorridos con funciones simples (contar, limpiar, depositar) el modelo esencial es suficiente y funciona correctamente.

2.2.1 Operaciones sobre el Robot

El conjunto de acciones que el robot puede realizar es muy reducido. Cada una de estas acciones corresponde a una instrucción, entendible por el robot y que debe tener un modo unívoco de expresión, para que la máquina la interprete correctamente, y un significado único, a fin de poder verificar que el resultado final de la tarea se corresponde con lo requerido. De esta manera se desplaza, recoge, deposita, evalúa algunas condiciones sencillas y hasta puede visualizar información.

Este conjunto de instrucciones elementales que se detallan en la tabla 2.1 permite escribir programas con un objetivo bien definido, que tendrán una interpretación y una ejecución única por el robot. En dicha tabla se indica para cada instrucción su sintaxis, es decir, cómo debe escribirse, y su semántica, esto es como se interpreta esa orden en el lenguaje del robot.

Sintaxis	Semántica
iniciar	Instrucción primitiva que posiciona al robot en la esquina (1,1) orientado hacia el norte.
derecha	Instrucción primitiva que cambia la orientación del robot en 90° en sentido horario respecto de la orientación actual.

mover	Instrucción primitiva que conduce al robot de la esquina en la que se encuentra a la siguiente, respetando la dirección en la que está orientado. *Es responsabilidad del programador que esta instrucción sea ejecutada dentro de los límites de la ciudad. En caso contrario se producirá un error y el programa será abortado*.
tomarFlor	Instrucción primitiva que le permite al robot recoger una flor de la esquina en la que se encuentra y ponerla en su bolsa. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos una flor en dicha esquina. En caso contrario se producirá un error y el programa será abortado.
tomarPapel	Instrucción primitiva que le permite al robot recoger un papel de la esquina en la que se encuentra y ponerlo en su bolsa. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos un papel en dicha esquina. En caso contrario se producirá un error y el programa será abortado.
depositarFlor	Instrucción primitiva que le permite al robot depositar una flor de su bolsa en la esquina en la que se encuentra. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos una flor en dicha bolsa. En caso contrario se producirá un error y el programa será abortado.
depositarPapel	Instrucción primitiva que le permite al robot depositar un papel de su bolsa en la esquina en la que se encuentra. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos un papel en dicha bolsa. En caso contrario se producirá un error y el programa será abortado.
PosAv	Identificador que representa el número de avenida en la que el robot está actualmente posicionado. *Su valor es un número entero en el rango 1..10 y no puede ser modificado por el programador*.
PosCa	Identificador que representa el número de calle en la que el robot está actualmente posicionado. Su valor es un número entero en el rango 1..10 y no puede ser modificado por el programador.
HayFlorEnLaEsquina	Proposición atómica cuyo valor es V si hay al menos una flor en la esquina en la que el robot esta actualmente posicionado, ó F en caso contrario. *Su valor no puede ser modificado por el programador*.
HayPapelEnLaEsquina	Proposición atómica cuyo valor es V si hay al menos un papel en la esquina en la que el robot está actualmente posicionado, ó F en caso contrario.
HayFlorEnLaBolsa	Proposición atómica cuyo valor es V si hay al menos una flor en la bolsa del robot, ó F en caso contrario. Su valor no puede ser modificado por el programador.

HayPapelEnLaBolsa	Proposición atómica cuyo valor es V si hay al menos un papel en la bolsa del robot, ó F en caso contrario. Su valor no puede ser modificado por el programador.
Pos	Instrucción que requiere dos valores Av y Ca, cada uno de ellos en el rango 1..10, y posiciona al robot en la esquina determinada por el par (Av,Ca) sin modificar la orientación del robot.
Informar	Instrucción que permite visualizar en pantalla un conjunto de valores.

Tabla 2.1: Sintaxis del robot

Es importante remarcar que la sintaxis en este lenguaje no es sensible a mayúsculas y minúsculas. Es lo mismo escribir “depositarFlor”, “DepositarFlor” ó “depositarflor”.

2.2.2 Estructura general de un programa

Un programa escrito en el lenguaje del robot comienza con la palabra clave **programa**, la cual debe estar seguida por un identificador que determina el nombre del programa.

El cuerpo del programa principal es una secuencia de sentencias, delimitada por las palabras claves **comenzar** y **fin**.

La sintaxis a utilizar es la siguiente:

programa *nombre_del_programa*
comenzar
sentencias que indican el recorrido del robot
fin

2.2.2.1 Comentarios Lógicos

Los problemas para poder ser resueltos, deben entenderse, es decir interpretarse adecuadamente. Esto requiere un enunciado preciso de los mismos.
A su vez las soluciones que se desarrollaron y que se expresaron en un lenguaje preciso y riguroso, también deben entenderse por otros. ¿Por qué?

- Porque no siempre se ejecuta la solución.
- Porque a veces se debe modificar la solución por un pequeño cambio del problema, y para esto se debe “leer” rápida y correctamente la solución anterior.
- Porque en ocasiones se comparan soluciones de diferentes pensadores y se debe tratar de entenderlas en un tiempo razonable.

Para que esta claridad y legibilidad se logre, se deben utilizar comentarios aclaratorios adecuados.

Un comentario dentro de un algoritmo no representa ni un dato, ni una orden. Sin embargo

quienes desarrollan sistemas informáticos coinciden en que un algoritmo adecuadamente escrito debería interpretarse sólo leyendo los comentarios que el autor intercaló a medida que construía la solución.

El término comentario lógico se refiere a que no se debe escribir un texto libre, sino expresar en forma sintética la función de una instrucción o un bloque del algoritmo, tratando de reflejar la transformación ó procesamiento de los datos que se logra con el mismo.

Normalmente los comentarios se intercalan en el algoritmo con algún símbolo inicial que indique que se trata de un comentario. En DaVinci Concurrente los comentarios multi-línea se indican por medio de llaves, ej. {Esto es un comentario}. Los comentarios de una línea se indican por medio de dos barras inclinadas hacia la derecha, ej: //esto es otro comentario.

Con la experiencia en Informática se aprende que el mayor esfuerzo y costo asociado con los sistemas de software es su mantenimiento, es decir corregir y ajustar dinámicamente el algoritmo ó programa inicial a nuevas situaciones. Para reducir el costo de este mantenimiento es fundamental documentar adecuadamente nuestros desarrollos, y un punto importante de esta documentación consiste en escribir comentarios lógicos adecuados a medida que se desarrolla la solución.

2.2.3 Estilo de programación

La programación en DaVinci Concurrente utiliza ciertas reglas sintácticas y delega la responsabilidad al programador aquellas que se encuentran relacionadas con la indentación y el uso de mayúsculas y minúsculas.

Estas reglas, aunque puedan resultar algo incómodas para el programador, buscan formar en el estudiante un buen estilo de codificación.

El objetivo de un estilo de programación es mejorar, en todo lo posible, la legibilidad del código de forma tal que resulte sencillo entenderlo, modificarlo, adaptarlo y reusarlo, ayudando así a maximizar la productividad y minimizar el costo de desarrollo y mantenimiento.

Un ejemplo sería aplicar a cada palabra compuesta la siguiente regla: comenzar cada palabra posterior a la primera con una letra mayúscula y las demás minúsculas. Ej: posAv, hayFlorEnLaBolsa.

También se recomienda la utilización frecuente de comentarios lógicos que aclaren el funcionamiento del algoritmo.

2.2.4 Ambiente de programación

Se denomina ambiente de programación a la herramienta que permite cubrir las distintas etapas en el desarrollo de un programa, que van desde la codificación del algoritmo en un

lenguaje de programación hasta su ejecución, a fin de obtener los resultados esperados.

Cada ambiente de programación trabaja sobre un conjunto de lenguajes específicos. En particular el ambiente EDIS (Entorno de Desarrollo Integrado Simple) trabaja sobre el lenguaje DaVinci Concurrente y reconoce la sintaxis descripta previamente.

Para codificar un algoritmo en el lenguaje del robot es necesario realizar las siguientes tres etapas:

1. Escribir el programa utilizando el Editor de código. El resultado de esto será un archivo de texto.
2. Para lograr que la computadora ejecute el programa escrito en la etapa anterior es necesario traducirlo a un lenguaje que la computadora comprenda. Esto se realiza mediante la verificación de las reglas sintácticas. En esta etapa se detectan los errores sintácticos.
3. Una vez que el programa ha sido verificado, puede ejecutarse. El ambiente DaVinci Concurrente permite visualizar durante la ejecución, el recorrido que realiza el robot.

La 2da etapa puede variar si se utiliza un intérprete o un compilador.

A continuación se describe el funcionamiento del ambiente a fin de poder realizar cada una de las etapas mencionadas anteriormente.

La figura 2.2 muestra el ingreso al ambiente formado por varias secciones que iremos detallando.

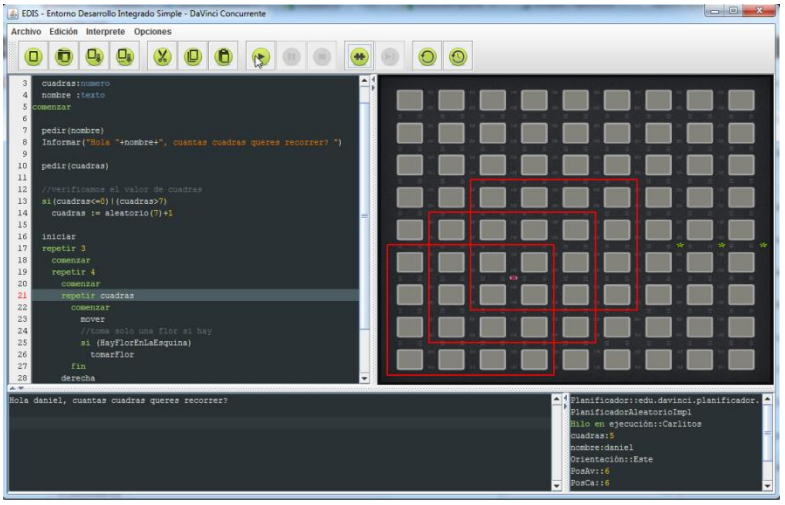


Figura 2.2: Ambiente de programación EDIS

Para comenzar a trabajar, debe ingresarse el programa que se desea ejecutar en el Editor de Código, como lo muestra la figura 2.3

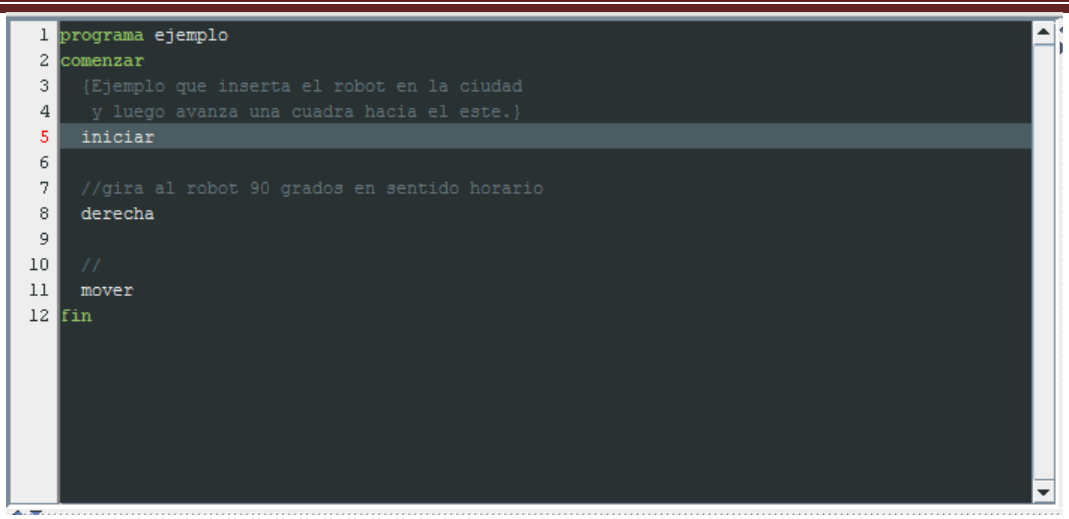


Figura 2.3: Editor de Código

Si bien no es imprescindible, se recomienda salvar el programa ingresado mediante la opción Archivo \ Guardar Como. Allí se deberá indicar el nombre que se desea dar al programa. Los programas que se ejecutan dentro del ambiente pueden poseer cualquier extensión aunque se recomienda utilizar la extensión dvc.

Puede ejecutarse mediante la opción Intérprete \ Ejecutar del menú. Previo a su ejecución el intérprete verificará si se encuentra correctamente escrito, en caso de encontrar un error se mostrará un mensaje marcando la posición del mismo y porque se produjo.

En la ventana correspondiente a la Ciudad es posible ver cómo el robot efectúa el recorrido indicado.

En el inferior izquierdo de la pantalla aparece una ventana en la cual se mostrarán los mensajes del usuario o mensajes producidos por el intérprete (ver Figura 2.4), y en la parte derecha se informan las coordenadas y la dirección del robot así como la cantidad de objetos que lleva en su bolsa. Esta información será visible cuando se ejecute la interpretación del programa en modo depuración.

De ser necesario, la ejecución del programa puede abortarse en forma manual usando la opción Ejecutar\ Abortar. Esta opción sólo se encuentra habilitada cuando el programa está corriendo.

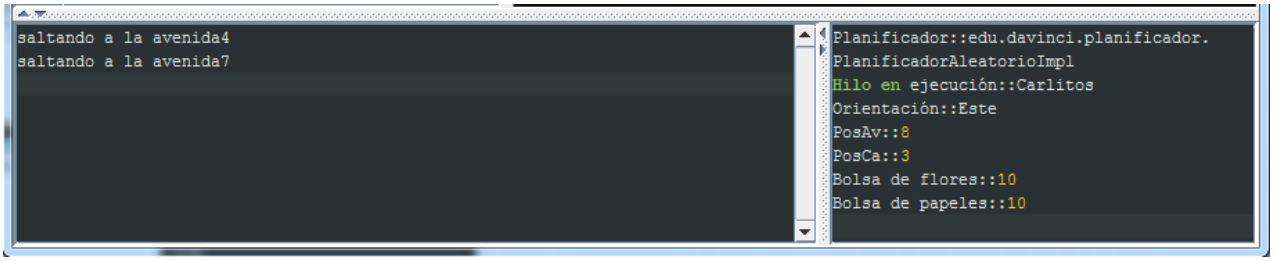


Figura 2.4: salida y depuración

Es posible indicar la cantidad inicial de flores y papeles tanto para la ciudad como para la bolsa del robot. Para ello se utiliza la opción Opciones \ Configurar ciudad. La figura 2.5 muestra la distribución de las flores. Puede procederse de la misma forma para manejar la distribución de papeles.

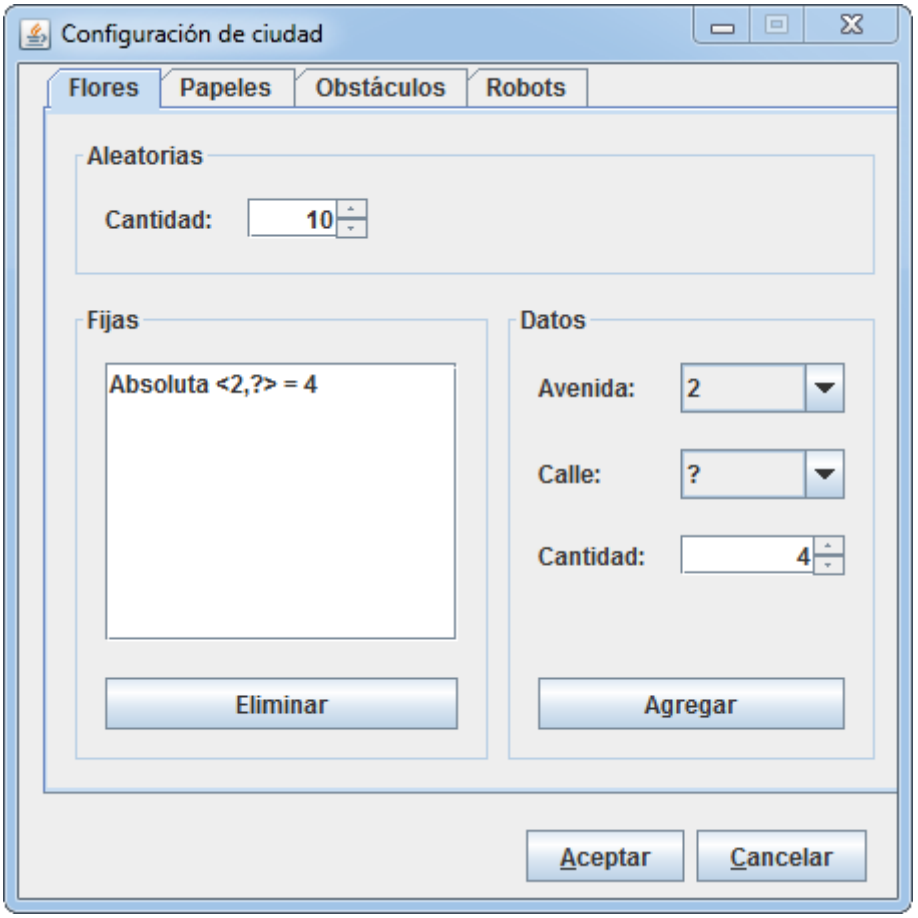


Figura 2.5: Opciones de la Ciudad

Esta configuración se hará efectiva hasta que se configure nuevamente.

2.5 Estructuras de Control

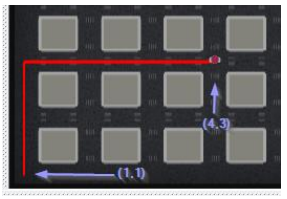
En esta sección se detallará la sintaxis correspondiente a las estructuras de control utilizadas por el robot. Las mismas, han sido explicadas en forma detallada en el capítulo 1.

2.5.1 Secuencia

Está definida por un conjunto de instrucciones que se ejecutarán una a continuación de otra.

Ejemplo 2.1: Programe al robot para que camine desde (1,1) a (1,3) y desde allí a (4,3).

```
programa Cap2Ejemplo1
comenzar
  iniciar
  mover
  mover
  derecha
  mover
  mover
  mover
fin
```

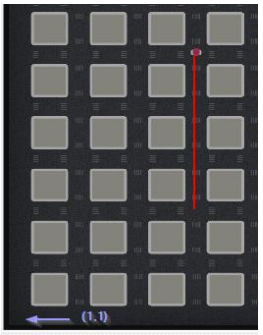


La instrucción *iniciar* ubica al robot en (1,1) orientado hacia el norte (hacia arriba). Luego avanza dos cuadras en línea recta por lo que queda posicionado en la calle 3. Dobla a la derecha para seguir avanzando por la calle 3 y camina tres cuadras por lo que finaliza el recorrido parado en (4,3). El robot queda mirando hacia el este.

Note que no existe en el lenguaje del robot una instrucción que permita detenerlo. Esto ocurrirá naturalmente al terminar el programa.

Ejemplo 2.2: Programe al robot para que recorra la avenida 4 desde la calle 3 hasta la calle 6. Al finalizar debe informar en qué esquina quedó parado.

```
programa Cap2Ejemplo2
comenzar
  iniciar
  Pos(4,3)
  mover
  mover
  mover
  Informar(PosAv, PosCa)
  fin
```



La instrucción *Pos(4,3)* permite que el robot "salte" desde (1,1) hasta (4,3). A partir de allí, camina tres cuadras en línea recta, realizando el recorrido solicitado. Al terminar el programa el robot quedará ubicado en la esquina (4,6). La instrucción *Informar(PosAv, PosCa)* muestra los valores retornados por las instrucciones *PosAv y PosCa*.

Ejercicios:

1. Programe al robot para que recorra la calle 6 desde la avenida 8 a la avenida 10.
2. Programe al robot para que recorra la avenida 7 desde la calle 8 hasta la calle 2.

2.5.2 Selección

Esta estructura permite al robot seleccionar una de dos alternativas posibles. La sintaxis es la que se detalla a continuación.

La estructura que se visualiza en el costado izquierdo se utiliza cuando la cantidad de acciones a realizar es exactamente una, cuando se requiere **un conjunto de acciones mayor a uno** se utilizan las palabras reservadas **comenzar y fin** como se muestra en el costado derecho.

si (condición)
acción en caso de que la condición sea verdadera
sino
acción en caso de que la condición sea falsa

si (condición)
comenzar
acción 1
acción 2 grupo de acciones en caso de que la condición
...
acción n
fin
sino
comenzar
acción 1
acción 2
...
acción n
fin

La estructura establece como opcional el camino que se selecciona si la condición se evaluó como falso.

acción 1
si (condición)
acción en caso de que la condición sea verdadera
acción 3

acción 1
si (condición)
comenzar
acción 2
acción 3 grupo de acciones en caso de que la condición
...
acción n
fin
acción n+1

Ejemplo 2.3: Programe al robot para que recorra la calle 1 desde la avenida 1 a la 2 depositando, si puede, una flor en cada esquina. Además debe informar el número de avenida en las que no haya podido depositar la flor.

programa Cap2Ejemplo3
comenzar

```
iniciar
derecha
{Evalúa la primera esquina}
si HayFlorEnLaBolsa
  depositarFlor
sino
  Informar(PosAv)
mover
{Evalúa la segunda esquina}
si HayFlorEnLaBolsa
  depositarFlor
sino
  Informar(PosAv)
fin
```

En caso de que haya flores en la bolsa, el robot ha depositado una sola flor en cada esquina.

Ejemplo 2.4: Programe al robot para que recorra la avenida 5 desde la calle 2 a la calle 4 recogiendo, de ser posible, un papel en cada esquina.

```
programa Cap2Ejemplo4
comenzar
 iniciar
 Pos(5,2)
 si HayPapelEnLaEsquina
   tomarPapel
 mover
 si HayPapelEnLaEsquina
   tomarPapel
 mover
 si HayPapelEnLaEsquina
   tomarPapel
fin
```

Ejercicios:

1. Programe al robot para que, si puede, deposite un papel en (1,2) y una flor en (1,3).
2. Programe al robot para que intente recoger una flor de la esquina determinada por la calle 5 y la avenida 7. Solo si lo logra debe ir a la calle 6 y avenida 8 e intentar recoger allí otra flor. Al finalizar debe informar en que esquina quedó parado.

Es posible que dentro del bloque de acciones a realizar tanto por **si** o por **sino** se incluya una nueva selección.
Suponga la siguiente modificación del Ejemplo 2.4:

Ejemplo 2.4 con Modificación:
Programe al robot para que recorra la avenida 5 desde la calle 2 a la calle 4 recogiendo, de ser posible, un papel en cada esquina que tenga al menos una Flor

```
programa  Cap2Ejemplo2.4 modificado
comenzar
  iniciar
  Pos(5,2)
  si HayFlorEnLaEsquina
     si HayPapelEnLaEsquina
       tomarPapel
  mover
si HayFlorEnLaEsquina
     si HayPapelEnLaEsquina
        tomarPapel
  mover
si HayFlorEnLaEsquina
     si HayPapelEnLaEsquina
       tomarPapel
fin
```

Veremos más adelante que, este ejercicio también puede ser resuelto utilizando condiciones lógicas compuestas.

2.5.3 Repetición

Cuando se desea realizar una acción o un conjunto de acciones un número fijo de veces, N, puede utilizarse la siguiente estructura:

```
repetir N
  acción
```

```
repetir N
   comenzar
   acción 1
   acción 2
      ...
   acción n
   fin
```

Es importante remarcar que **la cantidad de veces que se repite el bloque de acciones debe ser conocida de antemano.** Una vez iniciada la repetición la ejecución no se detendrá hasta no haber ejecutado el conjunto de acciones la cantidad de veces indicada por N.
Si se analizan con más detalle algunos de los ejemplos anteriores se verá que pueden resolverse utilizando una repetición.

Ejemplo 2.5: Programe al robot para que camine desde (1,1) a (1,3) y desde allí a (4,3).

Este problema fue resuelto utilizando una secuencia en el ejemplo 2.1. Ahora será

implementado utilizando la repetición.

```
programa Cap2Ejemplo5
comenzar
  iniciar
  repetir 2
    mover
  derecha
  repetir 3
    mover
fin
```

Comparemos los programas Cap2 Ejemplo 1 y Cap2 Ejemplo 5 verificando que el recorrido realizado en ambos casos es el mismo. A continuación se muestra una segunda implementación del ejemplo 2.3 utilizando una repetición:

El Ejemplo 2.3 decía: Programe al robot para que recorra la calle 1 depositando, si puede, una flor en cada esquina. Además debe informar el número de avenida de aquellas esquinas en las que no haya podido depositar la flor.

```
programa Cap2Ejemplo6
comenzar
  iniciar
  derecha
  repetir 2
    comenzar
    si HayFlorEnLaBolsa
      depositarFlor
    sino
      informar(PosAv)
      mover
    fin
fin
```

- ¿Por qué al finalizar el recorrido, el robot no queda posicionado en el mismo lugar que en el ejemplo 2.3? ¿Perjudica en algo este hecho a lo que debe ser informado?

A continuación se muestra una variante del ejemplo 2.4:

Ejemplo 2.7 (variante del 2.4): Programe al robot para que recorra la avenida 5 desde la calle 2 a la 4 recogiendo, de ser posible, un papel en cada esquina.

```
programa Cap2Ejemplo7
comenzar
  iniciar
  Pos(5,2)
  repetir 2
    comenzar
    si HayPapelEnLaEsquina
      tomarPapel
```

```
    mover
    fin
    si HayPapelEnLaEsquina    (*)
      tomarPapel
  fin
```

- ¿Las acciones realizadas por el robot en los programas Cap 2 Ejemplo 4 y Cap 2 Ejemplo 7 son iguales?
- ¿Es posible incluir la instrucción (*****), del ejercicio anterior, en la repetición? Si es así indique la manera de hacerlo y que diferencias encuentra con el programa Cap 2 Ejemplo 7.
-

Ejemplo 2.8: Programe al robot para que recorra la calle 4 dejando una flor en cada esquina.

Para resolver este problema es necesario analizar las 10 esquinas que forman la calle 4. El recorrido será efectuado de la siguiente forma:

```
programa Cap2Ejemplo8
comenzar
  {ubicar al robot en (1,4) orientado hacia la derecha}
  {Recorrer las primeras 9 esquinas de la calle 4}
    {depositar la flor (solo si tiene)}
    {avanzar a la próxima esquina}
  {Falta ver si se puede depositar la flor en la esquina (1,10)}
fin
```

En la sintaxis del robot, este algoritmo se traduce en el siguiente programa:

```
programa Cap2Ejemplo8
comenzar
  iniciar
  derecha                  {ubicar al robot en (1,4) orientado hacia la derecha}
  Pos(1,4)
  repetir 9                {recorrer las primeras 99 esquinas de la calle 4}
    comenzar
    si HayFlorEnLaBolsa
      depositarFlor
    mover
    fin
  si HayFlorEnLaBolsa      {falta la esquina (1,10)}
      depositarFlor
fin
```

Notemos que la sentencia *Pos(1,4)* no modifica la orientación del robot.

2.5.4 Iteración

Cuando la cantidad de veces que debe ejecutarse una acción o bloque de acciones depende del valor de verdad de una condición, puede utilizarse la siguiente estructura de control.

```
mientras (condición)
  acción que se ejecuta mientras
        la condición sea verdadera
```

```
mientras (condición)
   comenzar
   acción 1
   acción 2 bloque que se ejecuta mientras
        la condición sea verdadera

      ...
   acción n
   fin
```

A continuación se muestran algunos ejemplos que permiten representar el funcionamiento de esta estructura.

Ejemplo 2.9: Escriba un programa que le permita al robot recorrer la avenida 7 hasta encontrar una esquina que no tiene flores. Al finalizar debe informar en qué calle quedó parado. Por simplicidad, suponga que esta esquina seguro existe.

Tenga en cuenta que no se conoce la cantidad de cuadras que hay que recorrer para poder llegar a la esquina buscada. Para resolver este recorrido es preciso inspeccionar las esquinas una a una hasta lograr hallar la que no tiene flores. La solución es la siguiente:

```
programa Cap2Ejemplo9
comenzar
  iniciar
  Pos(7,1)
  mientras HayFlorEnLaEsquina
    mover
  Informar( PosCa )
fin
```

En este último ejemplo, para visualizar el número de calle donde el robot quedó parado debe utilizarse *PosCa* ya que no es posible calcular el valor a priori. Note que la ubicación de las flores en las esquinas de la ciudad puede variar entre una ejecución y otra.
Además del ejemplo se puede ver que por cada vez que el robot evalúa la condición *"HayFlorEnLaEsquina"* avanza una cuadra, si la condición es verdadera.

Ejemplo 2.10: Programe al robot para que deposite en (1,1) todas las flores que lleva en su bolsa.

```
programa Cap2Ejemplo10
comenzar
  iniciar
  mientras HayFlorEnLaBolsa
    depositarFlor
fin
```

Ejemplo 2.11: Programe al robot para que recoja todas las flores y todos los papeles de la

esquina determinada por la calle 7 y avenida 3. El algoritmo a utilizar es el siguiente:

```
programa Cap2Ejemplo11
comenzar
  {Ubicar al robot en la esquina que se quiere limpiar}
  {Recoger todas las flores}
  {Recoger todos los papeles}
fin
```

Dado que en una esquina no se conoce a priori la cantidad de flores y/o papeles que puede haber será necesario utilizar dos iteraciones: una para recoger las flores y otra para recoger los papeles, de la siguiente forma:

```
programa Cap2Ejemplo11
comenzar
  iniciar
  Pos(3,7)
  {Recoger todas las flores}
  mientras HayFlorEnLaEsquina
    tomarFlor
  {Recoger todos los papeles}
  mientras HayPapelEnLaEsquina
    tomarPapel
fin
```

Ejemplo 2.12: Programe al robot para que camine desde (4,2) hasta (4,4) y luego hasta (7,4). El algoritmo es de la forma:

```
programa Cap2Ejemplo12
comenzar
  {Posicionar al robot }
  {Avanzar dos cuadras }
  {Doblar a la derecha }
  {Avanzar tres cuadras }
fin
```

Este algoritmo puede ser implementado de diferentes formas. Analice estas dos opciones:

```
programa Cap2Ejemplo12
comenzar
  iniciar
  Pos(4,2)
  repetir 2
    mover
  derecha
  repetir 3
    mover
fin
```

```
programa Cap2Ejemplo12
comenzar
  iniciar
  Pos(4,2)
  mientras (PosCa <= 4)
    mover
  derecha
  mientras (PosAv <= 6)
    mover
fin
```

- ¿El robot realiza la misma cantidad de pasos en ambos programas?

- ¿Cuál solución prefiere Ud.? Justifique su respuesta pensando en que ahora debe hacer que el robot camine desde (3,5) hasta (3,7) y desde allí hasta (6,7).

Análisis de la sintaxis del robot

La tabla 2.2 resume la sintaxis del robot ejemplificada hasta el momento. Explique:

Sintaxis del robot	
iniciar	HayFlorEnLaBolsa
mover	HayPapelEnLaBolsa
derecha	HayFlorEnLaEsquina
tomarFlor	HayPapelEnLaEsquina
tomarPapel	
depositarFlor	
depositarPapel	posAv
pos	posCa
informar	

Tabla 2.2: Resumen de la sintaxis del robot

- ¿Qué función cumple la instrucción iniciar?
- ¿Por qué todos los programas del robot deben comenzar con esta instrucción?
- ¿Qué diferencia hay entre las instrucciones PosAv y PosCa y la instrucción Pos?
- Suponga que el robot se encuentra en (1,1) y se desea que salte a (3,4). ¿Es posible realizar las siguientes asignaciones para lograrlo?

  PosAv := 3
  PosCa := 4

Justifique su respuesta. Indique la manera correcta de resolver este problema.

- ¿Es posible para el robot depositar una flor sin verificar primero si tiene al menos una flor en su bolsa?
- ¿El robot está capacitado para decir si en una misma esquina hay varios papeles?

2.6 Proposiciones atómicas y moleculares, simbolización y tablas de verdad

Cuando se emplearon condiciones para definir las acciones a tomar en la selección y la iteración no se indicó la forma en que pueden combinarse. Como el lector habrá notado, todos los ejemplos desarrollados hasta el momento fueron lo suficientemente simples como para poder ser resueltos con una pregunta sencilla. Sin embargo, en un problema real, esto no es así y se requiere combinar expresiones para poder representar una situación a evaluar. Por este motivo se introducirán y repasarán algunos conceptos básicos de la lógica proposicional que permitirán clarificar este aspecto, aplicados específicamente a problemas con el robot.

Dos de las estructuras de control ya vistas, selección e iteración, requieren para su funcionamiento, la evaluación de una condición. Estas condiciones se corresponden con lo que en términos de lógica se conoce como proposiciones.

Una proposición es una expresión de la cual tiene sentido decir si es verdadera o falsa, o sea es posible asignarle un valor de verdad (verdadero o falso, pero no ambos).

Ejemplos de proposiciones

- 1 + 4 = 5 (Verdad)
- La Pampa es una nación. (Falso)
- Un triángulo es menor que un círculo. (No se le puede asignar un valor de verdad, por lo tanto **no** es proposición)
- El color azul vale menos que una sonrisa (ídem anterior)
- Hay una flor en la esquina (será verdadero ó falso dependiendo de si la flor se encuentra o no en la esquina)

2.6.1 Proposiciones atómicas y moleculares

En Lógica, el término atómico se utiliza con su significado habitual: "algo que no puede ser dividido nuevamente".

Una proposición es considerada atómica si no puede ser descompuesta en otras proposiciones.

Ejemplos

- La casa es roja.
- Hoy es lunes.
- He llegado al final del recorrido.
- Estoy ubicado a 3 metros de altura.

Cuando en una expresión se unen varias proposiciones atómicas se forma una proposición molecular o compuesta. Dicha unión se realiza mediante conectivos lógicos ó términos de enlace.

Estos términos de enlace son de gran importancia. Tanto es así, que se estudiarán algunas reglas muy precisas para el uso de esta clase de términos.

Los términos de enlace a utilizar son los siguientes: "y", "o", "no". Los dos primeros se utilizan para conectar proposiciones atómicas; en tanto que el conectivo "no", solamente se coloca frente a una proposición atómica.

Ejemplos

- **No** es cierto que la luna esté hecha de queso verde.
- La vaca está cansada **y no** dará leche.
- Hace calor **ó** hay mucha humedad.
- Hay papel en la bolsa **y** hay papel en la esquina.

Resumiendo:

Una proposición es atómica si no tiene conectivos lógicos, en caso contrario es molecular

2.6.2 Simbolización

Así como en matemática se simbolizan las cantidades para facilitar el planteo y solución de problemas, también en este caso es importante simbolizar las proposiciones atómicas, las moleculares y los conectivos lógicos con el objeto de facilitar las operaciones.

Conectivo	Simbolización en DaVinci Concurrente
y	&
o	|
no	!

Tabla 2.3: Conectivos lógicos ó términos de enlace

Se utilizarán letras minúsculas para simbolizar las proposiciones atómicas:

Ejemplos de simbolización de proposiciones atómicas

> Ayer fue un día ventoso.
> Si se considera p = "ayer fue un día ventoso", esta proposición puede ser simbolizada como: p.
>
> Ese pájaro vuela muy alto.
> Si se llama q = "ese pájaro vuela muy alto", la proposición se simboliza como: q.
>
> PosCa < 10.
> Si se llama r = "PosCa < 10", la proposición se simboliza como: r.

A continuación se aplicará este mecanismo de simbolización a las proposiciones moleculares.

El proceso para simbolizarlas consiste en:

1. Determinar cuáles son las proposiciones atómicas que la componen.
2. Simbolizar las proposiciones como se explicó anteriormente.
3. Unir las proposiciones con los conectivos ya vistos. Por tal motivo, debe definirse un símbolo para cada uno de los conectivos. La tabla 2.3 muestra la simbolización a utilizar en cada caso.

Ejemplos de simbolización de proposiciones moleculares

Juan es estudiante y es jugador de fútbol.
p = "Juan es estudiante"
q = "Juan es jugador de fútbol"
Simbolizando *p & q*

No es cierto que PosCa = 10.
p = "PosCa=10"
Simbolizando !p

Hay flor en la esquina y hay papel en la bolsa, o hay papel en la esquina.
p = "Hay flor en la esquina"
q = "hay papel en la bolsa"
r = "hay papel en la esquina"

Analicemos, para resolver correctamente el ejemplo anterior debe tenerse en cuenta la aparición de la coma, la cual separa las dos primeras proposiciones de la tercera. Cuando se obtiene la simbolización debe respetarse ese orden. Por lo tanto la simbolización sería: (p & q) | r

No hay flor en la bolsa, pero hay flor en la esquina.
p = "hay flor en la bolsa"
q = "hay flor en la esquina"

Simbolizando (!p & q)

Notemos que la palabra **pero** actúa como el conectivo lógico "y".

2.6.3 Tablas de verdad. Repaso

Como se explicó previamente, una proposición es una expresión de la cual tiene sentido decir si es verdadera o falsa.

Para poder analizar cualquier proposición molecular y decir qué valor de verdad tiene, es usual hacerlo a través de lo que se conoce como tabla de verdad.

La tabla de verdad de una proposición es, como su nombre lo indica, una tabla donde se muestran todas las combinaciones posibles de los valores de verdad de dicha proposición.

2.6.3.1 Conjunción. Tabla de verdad

Dadas dos proposiciones cualesquiera p y q, la proposición molecular p & q representa la conjunción de p y q.

La conjunción de dos proposiciones es cierta únicamente en el caso en que ambas proposiciones lo sean.

Dadas dos proposiciones cualesquiera *p* y *q*, si ambas son verdaderas, entonces *p & q*, que representa la conjunción de p y q, es verdadera. Cualquier otra combinación da como resultado una proposición molecular falsa. La tabla 2.4 representa la tabla de verdad de la conjunción *p* & *q* utilizando las cuatro combinaciones posibles de valores de verdad para *p* y *q*. Por lo tanto, si *p* & *q* es una proposición verdadera entonces *p* es verdadera y q también es verdadera. En Lógica se pueden unir dos proposiciones cualesquiera para formar una conjunción. No se requiere que el contenido de una de ellas tenga relación con el contenido de la otra.

p	q	p & q
V	V	V
V	F	F
F	V	F
F	F	F

Tabla 2.4: Conjunción *(p & q)*

Ejemplos

6 es un número par y divisible por 3.
p = "6 es un numero par"
q = "6 es divisible por 3"
p es verdadera y q también, por lo tanto *p* & *q* es verdadera.

Suponiendo que el robot se encuentra situado en la esquina (1,1)
p = "PosCa=1"
q = "PosAv=2"
p es verdadera y q es falsa, por lo tanto *p* & *q* es falsa.

El siguiente ejemplo muestra la aplicación de la conjunción en un algoritmo:

Ejemplo 2.13: Hacer que el robot deposite, de ser posible, una flor en la esquina (4,7) en el caso que en dicha esquina no haya flores.

```
programa Cap2Ejemplo13
comenzar
  iniciar
  Pos(4,7)
  si ((hayFlorEnLaBolsa) & !(hayFlorEnLaEsquina))
    depositarFlor
fin
```

La selección utiliza la conjunción de dos proposiciones que, como se explicó anteriormente, para ser verdadera necesitará que ambas proposiciones lo sean simultáneamente. Esto es, basta con que una de ellas sea falsa para que no se deposite una flor en la esquina.

2.6.3.2 Disyunción. Tabla de verdad

Dadas dos proposiciones cualesquiera p y q, la proposición molecular p | q representa la disyunción de p y q.

La disyunción entre dos proposiciones es cierta cuando al menos una de dichas proposiciones lo es.

La disyunción utiliza el término de enlace "o" en su sentido incluyente. Esto significa que basta con que una de las dos proposiciones sea verdadera para que la disyunción sea verdadera.

Dadas dos proposiciones cualesquiera *p* y *q*, su disyunción, *p* | *q*, será falsa cuando ambas proposiciones sean falsas. La tabla 2.5 representa la tabla de verdad de la disyunción p | q utilizando las cuatro combinaciones posibles de valores de verdad para p y q.

p	q	p | *q*
V	V	V
V	F	V
F	V	V
F	F	F

Tabla 2.3: Disyunción *(p v q)*

Ejemplos

2 es primo o es impar
p = "2 es primo"
q = "2 es impar"
p es verdadera, q es falsa. Se deduce que p | q es verdadera

Suponiendo que el robot se encuentra situado en la esquina (1,1)
p = "PosCa=8"
q = "PosAv=2"
p es falsa, q es falsa. Se deduce que p | q es falsa.

El siguiente problema puede resolverse aplicando la disyunción:

Ejemplo 2.14: hacer que el robot se mueva desde la esquina (4,1) a la esquina (4,5) en el caso que la esquina (4,1) no esté vacía (se considera no vacía a aquella esquina que tiene al menos una flor o un papel).

```
programa Cap2Ejemplo14
comenzar
  iniciar
  Pos(4,1)
  si ((hayFlorEnLaEsquina) | (hayPapelEnLaEsquina))
    repetir 4
```

```
    mover
fin
```

La selección utiliza la disyunción de dos proposiciones que, como se explicó anteriormente, para ser verdadera solo requiere que una de ellas sea verdadera. Note que si la esquina (4,1) tiene flor y papel, el robot avanza hasta la esquina (4,5). La única forma de que el robot no avance es que la esquina esté vacía, sin flor ni papel.

2.6.3.3 Negación. Tabla de verdad

Dada una proposición *p,* su negación ~ *p,* permitirá obtener el valor de verdad opuesto.
El valor de verdad de la negación de una proposición verdadera es falso y el valor de verdad de la negación de una proposición falsa es verdadero.

Dada una proposición p, la tabla 2.4 representa los posibles valores de verdad de dos valores de verdad posibles de p.

p	!p
V	F
F	V

Tabla 2.4: Negación *(!p)*

Ejemplos

p = “El número 9 es divisible por 3”
La proposición p es verdadera.
La negación de p es:
!p = “El número 9 no es divisible por 3”
Se ve claramente que !p es falsa.

Suponiendo que el robot se encuentra situado en la esquina (1,1)
p = “PosCa=1”
La proposición p es verdadera.
La negación de p es: !p= “!PosCa=1”.
Se deduce que !p es falsa.

Ejemplo 2.15: el robot debe recorrer la calle 1 hasta encontrar una flor que seguro existe.

```
programa Cap2Ejemplo15
comenzar
  iniciar
  derecha
  mientras  !(hayFlorEnLaEsquina)
    mover
fin
```

Note que la iteración utiliza la negación de la proposición atómica “hay flor en la esquina”. De esta forma, el robot detendrá su recorrido cuando encuentre una flor.

2.6.4 Utilización del paréntesis

Es frecuente encontrar proposiciones que tienen más de un término de enlace. En estos casos, uno de los términos de enlace es el más importante, o el término dominante, porque actúa sobre toda la proposición.

El operador "!" es el que tiene mayor prioridad, es decir, es el que primero es evalúa; seguido por "&" y "|". Estos dos últimos poseen igual prioridad. Ante una expresión que utilice varias conjunciones y/o disyunciones, se evaluaran de izquierda a derecha.

Lo mismo ocurre en Matemática. Si se considera la siguiente cuenta: "2 + 3 * 5", el resultado final, como es fácil de deducir, es 17.

La primera operación que se resuelve es el producto entre 3 y 5, y luego a su resultado se le suma 2. Ocurre así porque el operando * (por), igual que el operando / (dividido), se resuelve antes que el operando + (mas), o que el operando - (menos).

Dada la siguiente proposición p & !*q*, primero se resuelve la negación y luego la conjunción. En determinados casos se tiene la necesidad de alterar este orden natural de resolución. Al igual que en matemática, el uso de paréntesis permite resolver este problema.

Ejemplo

!*p* | *q* es una disyunción entre ! *p y q*.

!(p | q) es la negación de una disyunción.

En el ejemplo anterior se muestra la manera de convertir una disyunción en una negación. Es decir, el operador de negación se resuelve en forma posterior a la disyunción.

2.7 Conclusiones

El uso de un lenguaje de programación elimina la ambigüedad que se produce al expresarse en lenguaje natural permitiendo que cada instrucción tenga una sintaxis y una semántica única.

Se ha presentado un lenguaje de programación específico (DaVinci Concurrente) que permite escribir programas que pueden ser ejecutados en una computadora.

Además, en este capítulo, se ha realizado una breve introducción a los conceptos de lógica proposicional y se han presentado numerosos ejemplos que muestran la utilidad de las proposiciones en el funcionamiento de las estructuras de control. Como pudo observarse, las proposiciones moleculares, formadas a través de los conectivos lógicos, poseen una mayor potencia de expresión ya que permiten definir claramente el rumbo de acción a seguir.

Por otra parte, estos ejemplos también muestran las distintas posibilidades de combinar las estructuras de control y encontrar soluciones más generales a los problemas que se han planteado.

Lograr adquirir el conocimiento y la habilidad para desarrollar algoritmos utilizando estas estructuras es una tarea que requiere práctica. El lector no debería desanimarse si inicialmente advierte cierta dificultad en su uso. Se requiere un cierto tiempo de maduración para poder expresarse a través de esta terminología. El mecanismo para poder lograr adquirir las habilidades necesarias se basa en la ejercitación por lo que se recomienda la resolución de los problemas que se plantean en los enunciados de la práctica respectiva.

Nota: para realizar este apunte se utilizó (con el permiso correspondiente) el material del curso de Ingreso a la Facultad de Informática de la UNLP. En el mismo se han realizado algunos agregados y modificaciones.