Implementación de automatismos secuenciales.

Contenido

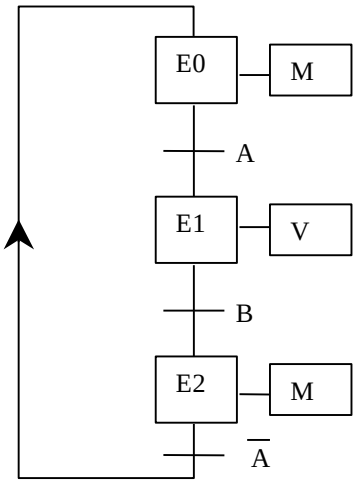
Implementación de automatismos secuenciales. 1
Contenido 1
1 Diagramas etapa transición 2
1.1 Elementos del Grafcet. 3
1.2 Estructuras en el Grafcet. 4
1.3 Reglas de evolución del Grafcet. 6
2 Programación de diagramas etapa transición 8
2.1 Inicialización 9
2.2 Lectura de entradas 10
2.3 Detección de flancos de las entradas 10
2.4 Desactivación/activación de etapas 10
2.5 Activación de las salidas 13
2.6 Definición de temporizadores 12
2.7 Actualización de salidas 13
2.8 Ejemplo 14

1 Diagramas etapa transición

Un diagrama de etapa-transición (GRAFCET) es un grafo orientado formado por 2 tipos de nodos, las **etapas** (simbolizadas por un cuadrado) y las **transiciones** (simbolizadas por una línea recta horizontal) unidos alternativamente por otras líneas rectas perpendiculares a las transiciones. Se pueden unir etapas con transiciones y viceversa, pero nunca elementos iguales. La orientación del grafo es siempre de arriba abajo (se pasa de una etapa a la transición de abajo y de ésta a la etapa de abajo), salvo excepciones en las que la línea que une la etapa y la transición debe tener una flecha indicando el sentido de evolución.

En un momento determinado cada etapa puede tener una marca (representada por un punto) o no tenerla, indicando que la etapa está activa o no. El conjunto de marcas del Grafcet (de etapas activas) define el estado del sistema.

El ejemplo de la embotelladora se podría modelar mediante el siguiente Grafcet:



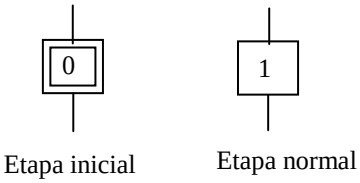
Los rectángulos asociados a cada etapa representan las acciones a realizar cuando la etapa está activa. Se indican únicamente las señales de salida que se tienen que poner a 1 (las que no están incluidas en el rectángulo están a 0 durante esa etapa).

La evolución del grafcet es como sigue: Cuando todas las etapas anteriores (por tanto dibujadas encima) a una transición están activas y la condición asociada a la transición está también activa, la transición se dispara y el sistema evoluciona desactivando las etapas anteriores y activando las etapas posteriores. En el ejemplo, si está activa la etapa 0 (lo que significa que no hay ninguna botella en la zona de llenado, y el motor está en marcha), cuando se activa el sensor A que detecta una botella, se debe desactivar la etapa 0 y activarse la etapa 1, con lo que el motor se parará, y se abrirá la válvula. Se estará en esa etapa hasta que se active el sensor B que detecta que la botella está llena.

Se dice que una transición está validada (o está habilitada) si todas las etapas anteriores están activas. Se denomina receptividad asociada a una transición a la condición lógica asociada a su disparo.

1.1 Elementos del Grafcet.

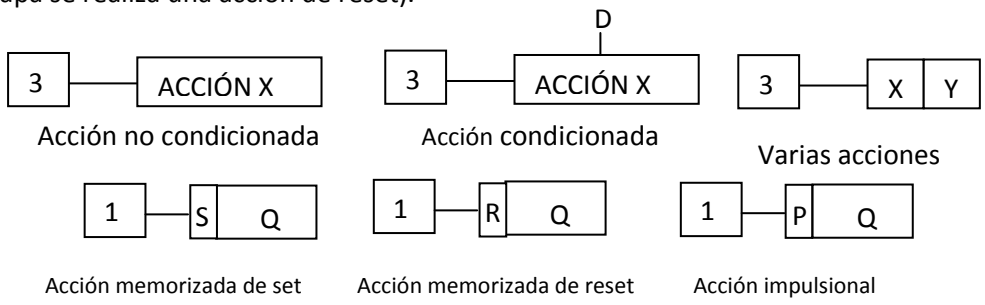
Etapa.



- Se representa por un cuadrado.
- Puede estar activa o inactiva (con marca o sin ella).
- Si es una etapa inicial tiene doble cuadrado. En ese caso se activa cuando se inicializa por primera vez el grafcet.

Acción asociada a una etapa.

- Se representa por un rectángulo unido a la etapa.
- La acción consiste normalmente en la activación de una salida. La salida determinada se activa mientras la etapa esté activa.
- Puede haber varias acciones a la vez.
- Una acción puede estar condicionada a una variable. Se representa por una línea perpendicular.
- Normalmente la acción se realiza mientras la estapa está activa. Sin embargo, puede haber acciones impulsionales (se realizan una sola vez cuando se activa la etapa) y acciones memorizadas (que permanecen activas cuando se desactiva la etapa hasta que en otra etapa se realiza una acción de reset).

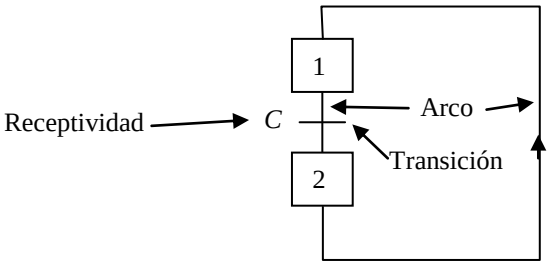


Transición y receptividad.

- Una transición se representa por un trazo horizontal.
- Se sitúa siempre entre dos etapas.
- Tiene asociada una **receptividad,** que puede ser una condición lógica de nivel o de flanco. Cuando es de flanco se representa con una flecha al lado de la condición (↑ si es de subida, ↓ si es de bajada). La condición lógica puede ser un simple sensor (como el detector A del ejemplo) o bien una expresión lógica compleja (A+B·C, por ejemplo).

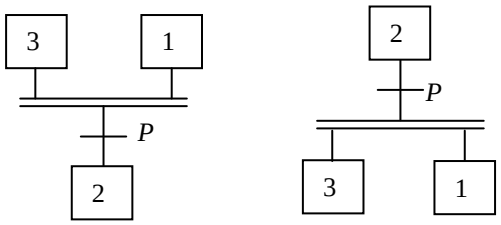
Arco

- Es una línea que une una transición con una etapa y viceversa.
- El sentido es siempre de arriba abajo. Cuando el sentido es el inverso (de abajo arriba) se indica con una flecha.



Trazos paralelos.

- Se utilizan para indicar que una transición se une a varias etapas.
- Sirven para definir secuencias paralelas o simultáneas y para sincronizar los finales de esas secuencias.

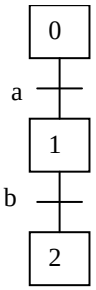


1.2 Estructuras en el Grafcet.

Las estructuras que se utilizan con más frecuencia en el diseño de automatismos con Grafcet son:

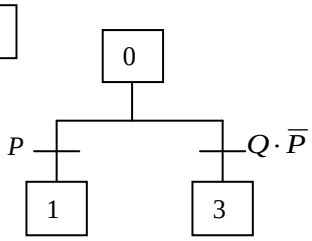
Secuencia.

- Es una sucesión alternada de etapas y transiciones.
- Representa una serie de operaciones secuenciales.
- Se dice que una secuencia está activa si lo está alguna de sus etapas.



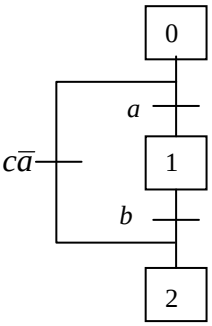
Selección entre secuencias

- Representa una bifurcación en la que según la condición se activa una secuencia u otra.
- Para evitar problemas suelen ser secuencias excluyentes (nunca se activan simultáneamente).



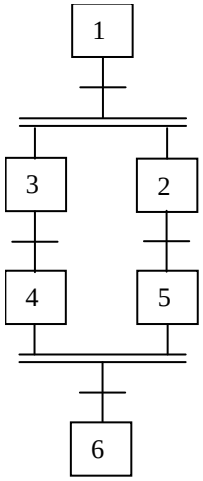
Salto

- Es una selección de secuencias.
- También suelen ser excluyentes para evitar problemas.
- El salto puede ser hacia adelante o hacia atrás.



Paralelismo de secuencias

- Se utiliza cuando una transición hace que dos o más secuencias se activen al mismo tiempo **y por tanto evolucionen de forma simultánea.**
- Se representa mediante dos trazos paralelos después de la transición.
- Al finalizar las secuencias simultáneas se suele poner una transición de sincronización, formada también por dos trazos paralelos. Indica que sigue la secuencia original solo cuando han terminado todas las secuencias simultáneas.
- Las secuencias simultáneas también podrían producirse con una selección de secuencias que no sea excluyente, aunque esto podría producir problemas al finalizar esas secuencias, por lo que no es aconsejable.

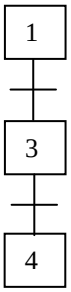


Transición Fuente

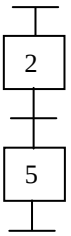
Etapas y transiciones fuente y sumidero

- Una etapa fuente no tiene ninguna transición ni etapa anterior. Solo se puede activar al inicializar (o mediante forzado).
- Una etapa sumidero no tiene ninguna transición ni etapa posterior. Una vez activada no se puede desactivar (salvo por forzado).

Etapa Fuente



Etapa Sumidero



Transición Sumidero

- Una transición fuente no tiene ninguna etapa anterior. Esta transición está siempre validada, por lo que solo tiene sentido con receptividad activa por flanco.
- Una transición sumidero no tiene ninguna etapa posterior.

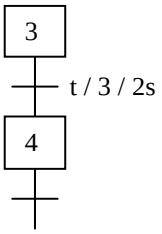
Acciones y receptividades temporizadas

Existen varias formas estándar de representar la medición de tiempos en Grafcet, aunque hay dos formas que son las más utilizadas.

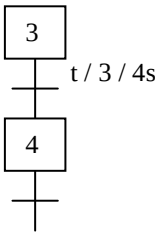
Una de las posibles notaciones tiene la forma: **t / Nºetapa / tiempo (s)**

Por ejemplo: t/4/2s es una señal que se activa dos segundos después de que se active la etapa 4**. Esta señal permanece activa aunque se desactive la etapa 4**. Se desactivará cuando la etapa 4 vuelva a activarse.

Las condiciones de temporización se suelen utilizar en las receptividades, para lograr que una etapa esté activa durante un tiempo determinado. En ese caso, las dos notaciones tienen un efecto similar:

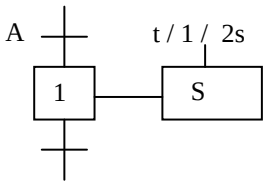


La etapa 3 estará activa durante 2 s

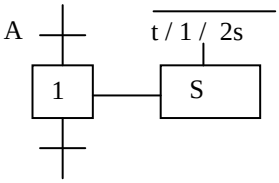


La etapa 3 estará activa durante 4 s

También se pueden utilizar las condiciones de temporización para condicionar una salida. Por ejemplo:



La salida S se activará 2 segundos después de que se active la etapa 1, desactivándose cuando se desactive la etapa 1.



La salida S se activará en cuanto se active la etapa 1, desactivándose 2 segundos después (si no se ha desactivado antes la etapa 1).

Representación de acciones según IEC848

La acción normal asociada a una etapa representa una variable que estará activa mientras esté activa la etapa. La IEC848 contempla otros 5 tipos de acciones, además de la acción normal. Se representan por una letra a la izquierda del rectángulo de la acción. Estas son:

- **C**: Acción condicionada. Se puede representar también con una línea perpendicular y la condición.
- **D:** Acción retardada (Delayed). Se activa un tiempo después de la activación de la etapa. Esta se puede representar también con una acción normal y un temporizador.
- **L**: Acción limitada en tiempo. Se activa en cuanto se activa la etapa, pero solo está activa un tiempo limitado. Se puede representar también con temporizadores.
- **P:** Acción impulsional. Se activa cuando se activa la etapa y dura un pulso muy corto. En la práctica debe ser suficiente para producir el efecto deseado. Este tipo de acción no se puede representar con la notación normal.

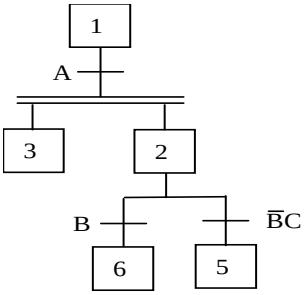
- **S**: Acción memorizada (Set). Se activa cuando se activa la etapa, y no se desactiva aunque se desactive ésta. Para desactivarse se necesita otra acción memorizada (de Reset) en una etapa posterior. Este tipo de acción no se puede representar con la notación normal.

En el primer caso, la salida Q se activa y permanece activa aunque se desactive la etapa 1. En el segundo caso, la salida Q se activa durante un pulso muy corto cuando la etapa 1 pasa de inactiva a activa.

Utilizando este tipo de acciones el diseño del automatismo es más flexible.

Combinación de paralelismo y selección de secuencia

Cuando se quiere una selección de secuencia inmediata a un paralelismo es necesario introducir una etapa auxiliar intermedia que no tiene ninguna acción. La situación es que estando en una etapa, al activarse A se quiere empezar dos secuencias, una fija y otra seleccionada entre dos en función del valor la variable B. En este caso hay que hacer primero una transición paralela con A, y después en una de las secuencias poner una etapa intermedia y hacer la selección de secuencias en función de B.



1.3 Reglas de evolución del Grafcet.

Las siguientes reglas de evolución permiten interpretar cómo evolucionará el sistema modelado por un Grafcet conforme se vayan activando las diversas señales que intervienen.

Inicialización

En la inicialización del sistema se tienen que activar todas las etapas iniciales (marcadas con doble cuadrado) y solo las iniciales.

Evolución de las transiciones

Una transición está validada si todas las etapas inmediatamente anteriores están activas. Si además de estar validada, la receptividad asociada es cierta, se dice que la transición es franqueable. Una transición franqueable es disparada (franqueada) inmediata y obligatoriamente.

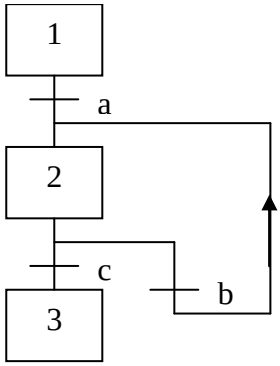
Evolución de las etapas

Cuando se dispara (o franquea) una transición todas las etapas inmediatamente anteriores se desactivan y simultáneamente se activan todas las posteriores.

Simultaneidad en el disparo

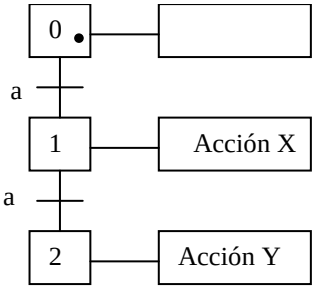
Si dos transiciones se disparan al mismo tiempo, las activaciones y desactivaciones de etapas se producen de forma simultánea.

Prioridad de la activación



Si en el disparo de una transición, una etapa debe activarse y desactivarse a la vez, quedará activa. En el ejemplo de la figura, si la receptividad b es cierta hay que volver a la etapa 2. Cuando esto ocurre, se deberá desactivar y activar la etapa 2 simultáneamente. Si no se cumpliera la regla de "prioridad de la activación", el GRAFCET se quedaría sin ninguna etapa activa, pero al cumplirse, la etapa 2 queda activa.

Estados estables e inestables



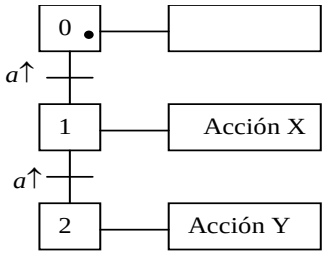
Un estado de un Grafcet es estable cuando no cambia ninguna etapa mientras no haya cambios en las entradas. Un estado es inestable si se produce alguna evolución de etapas sin que cambie ninguna entrada.

Los estados inestables duran muy poco tiempo, pues rápidamente pasan al estado siguiente. Por ejemplo, como en la figura.

El estado en que solo la etapa 0 está activa es estable, pues mientras a=0 no hay evolución. Cuando a=1, se produce la evolución desactivándose la etapa 0 y activándose la etapa 1. Ese estado es ahora inestable, pues aunque no haya cambios en las entradas se debe producir la evolución de forma que se desactive la etapa 1 y se active la etapa 2. El grafcet, por tanto, debe evolucionar siempre desde un estado estable hasta otro estado estable.

En teoría, el sistema está en la etapa 1 durante un pulso de duración muy pequeña. El dilema aparece con las acciones asociadas a esa etapa. **Si son acciones de nivel no deberían activarse**. En cambio, **si son acciones impulsionales o memorizadas sí deben activarse**, pues la etapa 1 se activa durante un pulso que es suficiente para producir el efecto de esas acciones. Por ejemplo, si la acción X consiste en hacer el SET de una variable, ese SET debe producirse. Si la acción X es de nivel (poner en marcha un motor mientras la etapa 1 esté activa) no tiene sentido que se active durante un pulso muy pequeño (idealmente casi nulo), por lo que no habría que activarla.

Transiciones activas por flanco



Por otra parte hay que tener en cuenta las situaciones especiales que pueden suceder cuando existen transiciones activas por flanco. Por ejemplo, en la figura de la derecha.

Cuando se produce un flanco de subida en *a* el grafcet debe evolucionar activándose la etapa 1 y desactivándose la etapa 0. La siguiente transición no está activa, pues el flanco ya se ha producido. Es decir, un flanco solo puede hacer avanzar una etapa. La etapa 1 es en este caso, estable. Para evolucionar a la etapa 2 la variable *a* debe bajar a 0 y volver a 1. Para evitar problemas en la programación de las transiciones por flanco, se debe limpiar la señal que indica el flanco nada más utilizarse.

2 Programación de diagramas etapa transición

Se trata de desarrollar un programa que reproduzca la forma de evolucionar de las distintas variables del Grafcet, siguiendo todas sus reglas de evolución. En particular hay que poner especial atención en 2 situaciones especiales, que son la existencia de etapas inestables, y la existencia de transiciones activas por flanco, en las que hay que modificar el Grafcet para evitarlas.

Para la programación se asocia una variable interna (1 bit) a cada etapa (el bit estará a 1 si la etapa está activa y a cero en caso contrario). Las ecuaciones lógicas tratarán de ir modificando esos bits según cambian las entradas de acuerdo a las reglas de evolución del grafcet.

El algoritmo se ejecuta en un bucle infinito, de forma que en cada ciclo de ese bucle se ejecuta el algoritmo, se refrescan las señales de entrada y de salida y se realizan las tareas necesarias de comunicación o de gestión de temporizadores.

Con respecto a las señales de entrada y salida, es conveniente tener una copia de las mismas en memoria, y trabajar con esas copias en las ecuaciones de activación, desactivación y salida, realizando una actualización de las entradas y salidas físicas cada bucle (igual que hacen los aut´omatas). Esto garantiza que se cumplen las reglas de evolución del Grafcet.

Se describirá un método que únicamente es válido cuando no se dan las 2 situaciones especiales comentadas anteriormente (la existencia de etapas inestables, y la existencia de transiciones activas por flanco), pero que resulta en un algoritmo muy sencillo. Se supondrá que la implementación se realiza en un microcontrolador Arduino. Su traducción a otro tipo de microcontroladores o PCs programados en C es inmediata.

Los bloques que contiene el programa son:

1. **Inicialización** de las etapas mediante el bit de inicio. Este bloque solo se ejecuta una vez, en el primer ciclo de scan, normalmente al mismo tiempo que se definen las variables. En este bloque se ponen a 1 las etapas iniciales de todos los Grafcets (las marcadas con doble cuadro), poniéndose a 0 las demás.
2. **Lectura de entradas.** Almacenamiento del valor actual de las entradas en variables.
3. **Detección de flancos de las entradas** que dan lugar a transiciones por flanco. En este bloque, de cada variable cuyo flanco se tenga que calcular se obtiene una variable de bit que valdrá 1 únicamente durante el siguiente ciclo de scan si la variable ha pasado de 0 a 1 o de 1 a 0 (según sea un flanco ascendente o descendente).
4. **Desactivación/activación de las etapas** anteriores/posteriores a las **transiciones** franqueables. En este bloque es donde se modifican las etapas, produciéndose la evolución del Grafcet. . Se incluye la **ejecución de acciones impulsionales**, como las acciones de **Set** y **Reset**, así como el incremento de contadores o la llamada a ejecución de subrutinas.
5. **Activación de las salidas**. Se ponen a 1 las salidas (acciones a nivel) asociadas a las etapas que están activas
6. **Definición de temporizadores y contadores**.
7. **Actualización de salidas**.

Funciona bien siempre que:

- **No existan estados inestables.** Si los hay, en un solo ciclo de scan pueden avanzarse varias etapas, por lo que las acciones impulsionales no se producirían.
- **No** haya transiciones por flanco consecutivas. Con este método una transición por flanco está activa en el autómata durante un ciclo completo de scan. Esto hace que un solo flanco pueda hacer evolucionar varias etapas consecutivas.
- **No existan transiciones que puedan desactivar y activar a la vez una etapa.**
- No haya bifurcaciones de selección no excluyentes. Si hay bifurcaciones no excluyentes la primera que se evalúe será la única que se active, ya que la etapa anterior quedará desactivada.
- No haya transiciones que dependan de etapas.

2.1 Inicialización

En la inicialización se deben definir las variables de etapa así como las diferentes variables utilizadas para copiar el estado de entradas y salidas. El siguiente código muestra un ejemplo en el que hay 3 etapas, siendo la etapa 0 la etapa inicial, y en el que hay un sensor de entrada P conectado al pin 2, y un actuador S conectado al pin 4.

```
boolean E[3]={1,0,0};
const int pinP=2;
int P;
const int pinS=4;
int S;
void setup(){
      pinMode(pinP,INPUT);
      pinMode(pinS,OUTPUT);
}
```

El resto del programa del automatismo (el resto de ítems del método) se debe ejecutar en un bucle infinito que, en el caso del microcontrolador Arduino se indica como

```
void loop(){
      // Bucle infinito. Ciclo de Scan
}
```

En la función de inicialización (setup) también se inicializarían los puertos de comunicaciones en caso de utilizarse. Por ejemplo, para poner en marcha el puerto serie con un baudrate de 9600 se escribiría

```
Serial.begin(9600);
```

2.2 Lectura de entradas

La lectura de entradas consiste en acceder a las entradas digitales y almacenar su valor en una variable. En el caso anterior, la ecuación sería

```
P=digitalRead(pinP);
```

Si se trata de una entrada analógica se utiliza el código

```
A=analogRead(pinA);
```

En general las entradas pueden ser de cualquier tipo. Puede ser incluso un comando que llega mediante un puerto de comunicaciones, en cuyo caso se incluiría aquí la lectura de dicha variable, si está. Por ejemplo, para recibir un comando mediante el puerto serie se escribiría

```
if (Serial.available() > 0){
      ByteEntrante = Serial.read();
}
```

2.3 Detección de flancos de las entradas

La detección de flancos requiere crear dos variables, una que copia la señal cuyo flanco se quiere detectar, y otra que indique si el flanco se ha producido o no. Por ejemplo, el código para detectar un flanco de subida en la variable A es

```
Flanco_A = (!Copia_A) && A;
Copia_A =A;
```

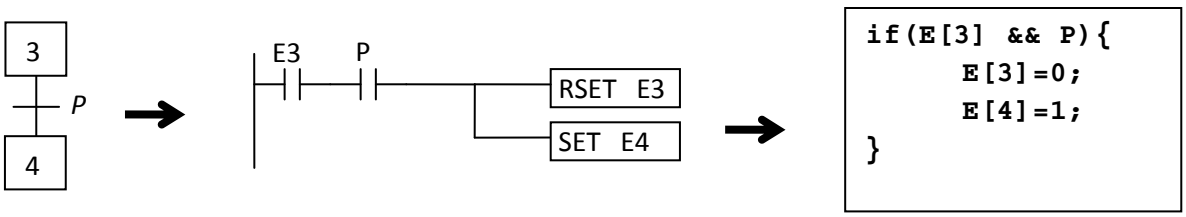
Para detectar un flanco de bajada se haría de forma similar

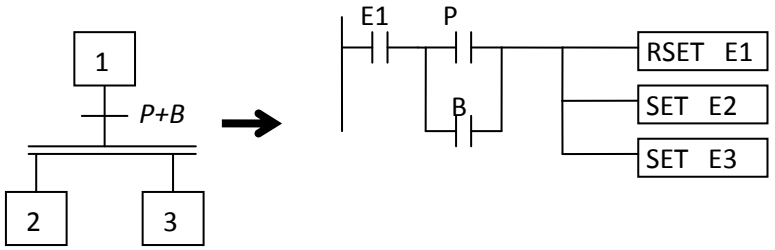
```
Flanco_bajada_A = (Copia_A) && !A;
Copia_A =A;
```

Las variables de tipo flanco se deben resetear nada más ser utilizadas, si no podría haber transiciones consecutivas franqueables con un único flanco.

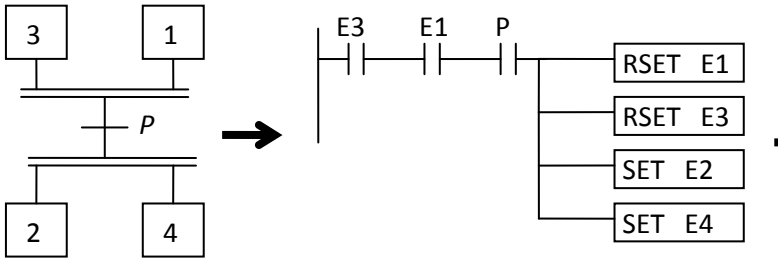
2.4 Desactivación/activación de etapas

En este bloque debe haber una ecuación para cada transición del grafcet. La ecuación de activación/desactivación de una transición debe tener en cuenta todas las etapas anteriores y todas las posteriores. Ejemplos:

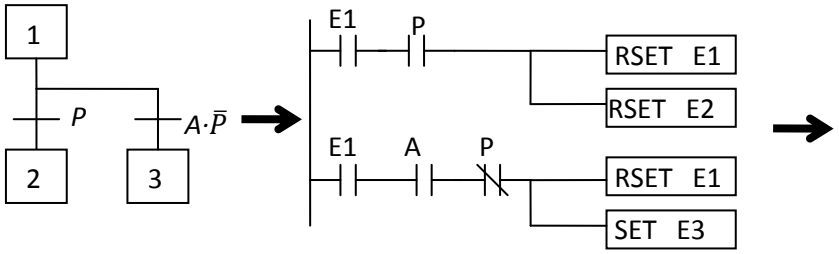





```
if(E[1] && (P || B)){
      E[1]=0;
      E[2]=1;
      E[3]=1;
}
```

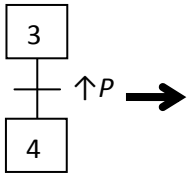



```
if(E[3] && E[1] && P){
      E[1]=0;
      E[3]=0;
      E[2]=1;
      E[4]=1;
}
```



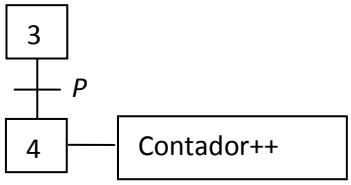

```
if(E[1] && P){
      E[1]=0;
      E[2]=1;
}
if(E[1] && A && !P){
      E[1]=0;
      E[3]=1;
}
```

Si la condición de activación fuera un flanco, se debería limpiar la señal de flanco nada más utilizarla, si no podría haber dos transiciones consecutivas con el mismo flanco:




```
if(E[3] && Flanco_P){
      Flanco_P;
      E[3]=0;
      E[4]=1;
}
```

Si hay que implementar una acción impulsional, debe ejecutarse nada más activar la etapa. Por ejemplo, para incrementar un contador en la etapa 4 del primer ejemplo se pondría:




```
if(E[3] && P){
      E[3]=0;
      E[4]=1;
      Contador++;
}
```

2.5 Definición de temporizadores

Normalmente el microcontrolador dispone de un temporizador hardware capaz de contar un tiempo suficientemente largo, con lo que la implementación de las instrucciones de temporizador consisten en comparar el valor actual del temporizador con un valor de consigna, de forma que cuando el primero sea mayor que el segundo quiere decir que ha transcurrido el tiempo especificado (lo cual llevaría a poner a 1 un bit).

Por ejemplo, en el microcontrolador Arduino se tienen dos temporizadores hardware de 32 bits que se incrementan bien cada milisengundo, bien cada microsegundo desde que arranca el microcontrolador. Con el primero se pueden contar tiempos de hasta 50 días con una precisión de 1 ms, y con el segundo tiempos de hasta 1 hora. Cuando cuentan este tiempo (2^{32}**periodo,* con periodo=1ms ó 1μs, según el caso) , vuelven a 0 y empiezan de nuevo el contaje del tiempo.

Las instrucciones que indican el tiempo transcurrido en los temporizadores indicados anteriormente son

```
long tiempo_ms, tiempo_us; //variables enteras de 32 bits.
tiempo_ms=millis();        // contaje actual en ms
tiempo_us=micros();        // contaje actual en us
```

Para implementar la definición de dos temporizadores del tipo t/4/5s y t/6/70ms se escribiría inicialmente en la activación de etapas:

```
if(E[3]&&A){
      E[3]=0;    E[4]=1;
      fin_t_4_5s = millis()+5000; //5000*1ms=5s
      }
if(E[5]&&B){
      E[5]=0;    E[6]=1;
      fin_t_6_700ms = millis()+70; //70*1ms=70ms
      }
```

La definición del bit del temporizador se haría mediante las instrucciones

```
t_4_5s = E[4] && (millis()>=fin_t_4_5s);
t_6_700ms = E[6] && (millis()>=fin_t_6_70ms);
```

Para evitar problemas de desbordamientos, hay que pasar a valores con signo los valores utilizados en la comparación:

```
t_4_5s = E[4] && ((long) millis()-(long) fin_t_4_5s >=0);
```

Las variables fin_t_4_5s y fin_t_6_70ms se usan para contar el tiempo y se deben definir como enteros de 32 bits sin signo, mientras que t_4_5s y t_6_70ms hace referencia a la variable t/4/5s y t/6/70ms y se han de definir como un boolean:

```
unsigned long fin_t_4_5s=0, fin_t_6_70ms=0;
boolean t_4_5s =0, t_6_70ms =0;
```

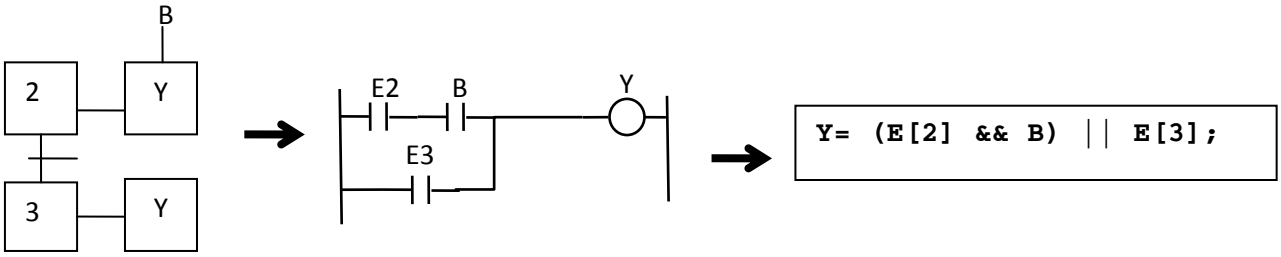
Si una transición o una acción están condicionadas a t/4/5s se utilizaría la variable `t_4_5s` para la validación. Por ejemplo, si la evolución de la Etapa 4 a la 5 se ha de hacer 5 segundos después de la activación de la etapa 4, se haría

```
if(E[4] && t_4_ts){E[4]=0; E[5]=1;}
```

2.6 Activación de las salidas

En este bloque se introduce **una red para cada salida**.

Cuando son acciones a nivel, la red debe incluir todas las etapas en las que la salida está activa, incluyendo además las condiciones lógicas correspondientes si la salida está condicionada. Por ejemplo, si la salida Y debe estar activa en la etapa 3 y en la etapa 2 condicionada a B, la ecuación de salida sería:



2.7 Actualización de salidas

Una vez finalizado el ciclo de scan se debe actualizar la tensión de los pines de salida que actúan sobre el proceso. Para ello se utiliza la función de escritura de salidas digitales digitalWrite. Por ejemplo para actualizar la salida con aquello que hayan definido las ecuaciones que implementan el Grafcet se escribe:

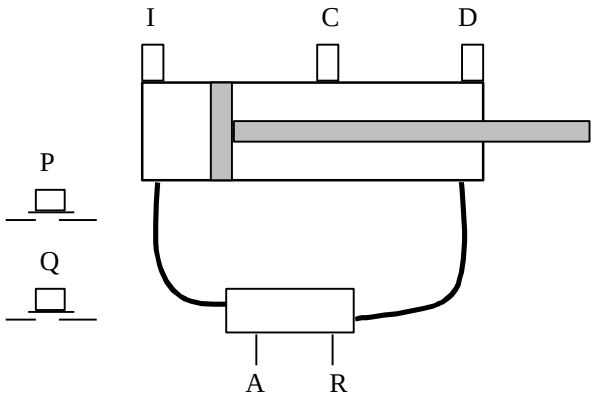
```
digitalWrite(pinS,S);
```

Si se tienen salidas analógicas, se escribe

```
analogWrite(pinS,valor);
```

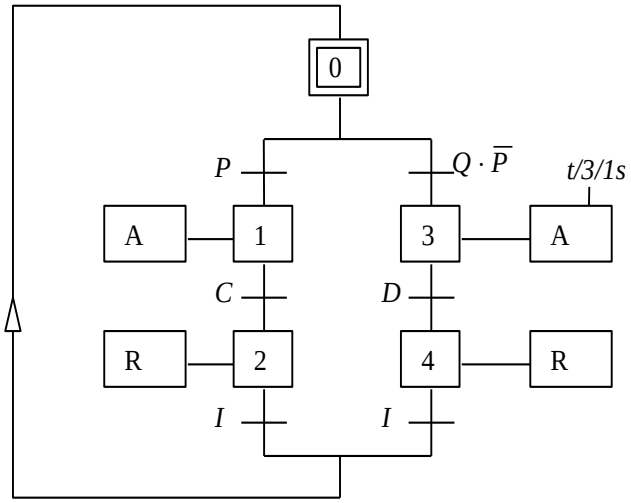
2.8 Ejemplo

Sea el cilindro de la figura.



Inicialmente se supone que el cilindro está en la posición más a la izquierda. Al pulsar P se debe mover hacia la derecha hasta que se active el detector C, para volver después a la izquierda. Si se pulsa Q debe hacer lo mismo, pero esperando 1 segundo antes de empezar a moverse, y llegando a D en lugar de C. Para mover el cilindro se actúa sobre las electroválvulas A y R.

Una posible solución es:



El código que implementa el Grafcet anterior en un microcontrolador Arduino, queda:

```
// Definición de variables e inicialización
const int pinP=0, pinQ=1, pinI=2, pinC=3, pinD=4, pinA=5,
pinR=6;
boolean P=0, Q=0, I=0, C=0, D=0, A=0, R=0;
boolean E[5]={1, 0, 0, 0, 0},
boolean t_3_1s=0;
unsigned long fin_t_3_1s=0;
```

```
void setup(){
    // Configuración de entradas y salidas
    pinMode(pinP,INPUT);
    pinMode(pinQ,INPUT);
    pinMode(pinI,INPUT);
    pinMode(pinC,INPUT);
    pinMode(pinD,INPUT);
    pinMode(pinA,OUTPUT);
    pinMode(pinR,OUTPUT);

    // Inicialización de las salidas
    digitalWrite(pinA,0);
    digitalWrite(pinR,0);
}

void loop(){
    // Lectura de entradas
    P=digitalRead(pinP);
    Q=digitalRead(pinQ);
    I=digitalRead(pinI);
    C=digitalRead(pinC);
    D=digitalRead(pinD);

    // Desactivación/Activación de etapas
    if(E[0] && P){ E[0]=0; E[1]=1; }
    if(E[1] && C){ E[1]=0; E[2]=1;}
    if(E[0] && !P &&Q){E[0]=0;E[3]=1; fin_t_1_3s=millis()+3000;}
    if(E[3] && D){ E[3]=0; E[4]=1;}
    if(E[2] && I){ E[2]=0; E[0]=1;}
    if(E[4] && I){ E[4]=0; E[0]=1;}

    // Definición de temporizadores
    t_1_3s=E[1] && ((long) millis()-(long) fin_t_1_3s >= 0);

    // Activación de salidas
    A = (E[1] || (E[3] && t_3_1s));
    R = (E[2] || E[4]);

    // Actualización de salidas
    digitalWrite(pinA,A);
    digitalWrite(pinR,R);
}
```