Patrick TRAU
http://pat.fr.st

Le GRAFCET
et sa mise en oeuvre

Université Louis Pasteur
Institut Professionnel des
Sciences et Technologies
15 rue du Maréchal Lefèbvre
67100 STRASBOURG (F)

Cours GRAFCET – I – introduction

Table des matières

Table des matières

Table des matières

I – introduction

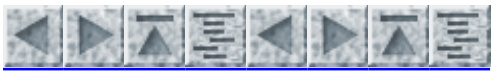
Le Grafcet est un outil graphique de définition pour l'automatisme séquentiel, en tout ou rien. Mais il est également utilisé dans beaucoup de cas combinatoires, dans le cas où il y a une séquence à respecter mais où l'état des capteurs suffirait pour résoudre le problème en combinatoire. Il utilise une représentation graphique. C'est un langage clair, strict mais sans ambiguïté, permettant par exemple au réalisateur de montrer au donneur d'ordre comment il a compris le cahier des charges. Langage universel, indépendant (dans un premier temps) de la réalisation pratique (peut se "câbler" par séquenceurs, être programmé sur automate voire sur ordinateur).

Ce document précise le langage Grafcet. Vous n'y trouverez pas d'exemples simples (qui font pourtant partie de la majorité des applications réelles), il y en a un tas à l'~~université de BREST~~.

Voici d'autres liens importants sur le Grafcet :

- un serveur Québecois extraordinaire : ~~DSC~~
- le serveur "officiel" du ~~Grafcet~~ (AFCET, au LURPA à Cachan)
- un autre cours, par ~~Emmanuel Geveaux~~

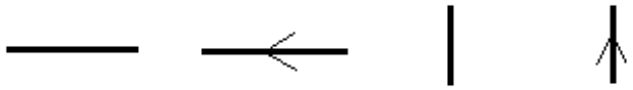
Remarque : le Grafcet est un langage d'origine française, et certains pays pensent que ce qu'ils n'ont pas inventé ne peut pas être de haut niveau. Regardez ce très bon site expliquant ~~comment on programme un automate sans Grafcet~~.



II – définitions

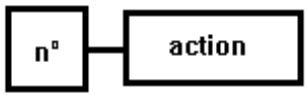
Un Grafcet est composé d'étapes, de transitions et de liaisons.

Une LIAISON est un arc orienté (ne peut être parcouru que dans un sens). A une extrémité d'une liaison il y a UNE (et une seule) étape, à l'autre UNE transition. On la représente par un trait plein rectiligne, vertical ou horizontal. Unå verticale est parcourue de haut en bas, sinon il faut le préciser par une flèche. Une horizontale est parcourue de gauche à droite, sinon le préciser par une flèche.

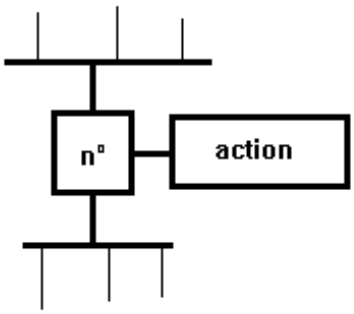


Une ETAPE correspond à une phase durant laquelle on effectue une ACTION pendant une certaine DUREE (même faible mais jamais nulle). L'action doit être stable, c'est à dire que l'on fait la même chose pendant toute la durée de l'étape, mais la notion d'action est assez large, en particulier composition de plusieurs actions, ou à l'opposé l'inaction (étape dite d'attente).

On représente chaque étape par un carré, l'action est représentée dans un rectangle à gauche, l'entrée se fait par le haut et la sortie par le bas. On numérote chaque étape par un entier positif, mais pas nécessairement croissant par pas de 1, il faut simplement que jamais deux étapes différentes n'aient le même numéro.



Si plusieurs liaisons arrivent sur une étape, pour plus de clarté on les fait arriver sur une barre horizontale, de même pour plusieurs liaisons partant de l'étape. Cette barre horizontale n'est pas une nouvelle entité du Grafcet, elle fait partie de l'étape, et ne représente qu'un "agrandissement" de la face supérieure (ou inférieure) de l'étape. On accepte de remplacer cette barre par un point si cela ne crée aucune ambiguïté.

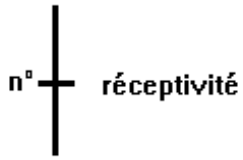


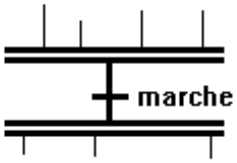
Une étape est dite active lorsqu'elle correspond à une phase "en fonctionnement", c'est à dire qu'elle effectue l'action qui lui est associée. On représente quelquefois une étape active à un inótant donné en dessinant un point à l'intérieur.

Une TRANSITION est une condition de passage d'une étape à une autre. Elle n'est que logique (dans son sens Vrai ou Faux), sans notion de durée. La condition est définie par une RECEPTIVITE qui est généralement une expression booléenne (c.à.d avec des ET et des OU) de l'état des CAPTEURS.

On représente une transition par un petit trait horizontal sur une liaison verticale. On note à droite la réceptivité, on peut noter à gauche un numéro de transition (entier positif, indépendant des numéros d'étapes). Dans le cas de plusieurs liaisons arrivant sur une transition, on les fait converger sur une grande double barre

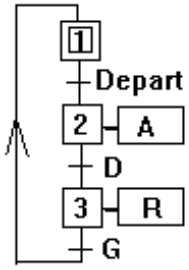
horizontale, qõi n'est qu'une représentation du dessus de la transition. De même pour plusieurs liaisons partant sous une transition.





III – exemple simple

Supposons un chariot pouvant avancer (A) ou reculer (R) sur un rail limité par deux capteurs G et D, Un cahier des charges pourrait être : Quand on appuie sur le bouton DEPART, on avance jusqu'en D, puis on revient. Ce C.d.C. est incomplet et imprécis. La réalisation du Grafcet permet de remarquer : Que fait−on avant l'appui de DEPART, jusqu'où revient−on, quelles sont les conditions initiales ? On réécrit le C.d.C. en 3 phases : Attendre jusqu'à l'appui de DEPART, avancer jusqu'en D, reculer jusqu'en G, attendre à nouveau DEPART et recommencer. On suppose le chariot initialement en G (sinon faire un cycle l'amenant en G).



IV – règles d'évolution

La modification de l'état de l'automatisme est appelée évolution, et est régie par 5 règles :

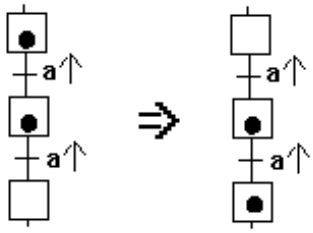
R1 : **Les étapes INITIALES sont celles qui sont actives au début du fonctionnement.** On les représente en doublant les côtés des symboles. On appelle début du fonctionnement le moment où le système n'a pas besoin de se souvenir de ce qui c'est passé auparavant (allumage du système, bouton "reset",...). Les étapes initiales sont souvent des étapes d'attente pour ne pas effectuer une action dangereuse par exemple à la fin d'une panne de secteur.

R2 : **Une TRANSITION est soit validée, soit non validée** (et pas à moitié validée). **Elle est validée lorsque toutes les étapes immédiatement précédentes sont actives** (toutes celles reliées directement à la double barre supérieure de la transition). **Elle ne peut être FRANCHIE que lorsqu'elle est validée et que sa réceptivité est vraie. Elle est alors obligatoirement franchie.**

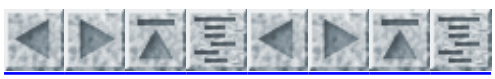
R3 : **Le FRANCHISSEMENT d'une transition entraîne l'activation de TOUTES les étapes immédiatement suivante et la désactivation de TOUTES les étapes immédiatement précédentes** (TOUTES se limitant à 1 s'il n'y a pas de double barre).

R4 : **Plusieurs transitions SIMULTANEMENT franchissables sont simultanément franchies** (ou du moins toutes franchies dans un laps de temps négligeable pour le fonctionnement). La durée limite dépend du "temps de réponse" nécessaire à l'application (très différent entre un système de poursuite de missile et une ouverture de serre quand le soleil est suffisant).

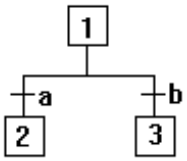
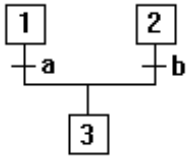
R5 : **Si une étape doit être à la fois activée et désactivée, elle RESTE active.** Une temporisation ou un compteur actionnés par cette étape ne seraient pas réinitialisés. Cette règle est prévue pour lever toute ambiguïté dans certains cas particuliers qui pourraient arriver dans certains cas :



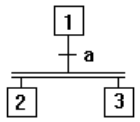
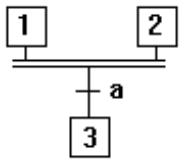
La partie COURS s'arrête ici. Toute autre règle que vous auriez pu entendre autre part ne fait pas partie du Grafcet. Il faudra TOUJOURS que votre Grafcet vérifie ce qui a été dit ci dessus (sinon ce n'est pas du Grafcet). Je tiens à préciser que le Grafcet devra être mis en oeuvre (câblé ou programmé) et donc une traduction de ce Grafcet en un schéma ou une suite d'instructions sera nécessaire. Le résultat de cette traduction, même s'il ressemble quelquefois à un Grafcet, ne peut pas imposer de nouvelles règles au Grafcet (qui dirait par exemple que le cas proposé après la règle 5 est interdit en Grafcet)



V – configurations courantes

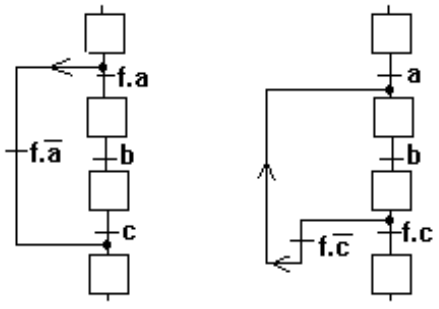
divergence en OU :	Convergence en OU :
	
si 1 active et si a seul, alors désactivation de 1 et activation de 2, 3 inchangé. si a et b puis 1 active alors désactivation 1, activation 2 et 3 quel que soit leur état précédent. (règle 4)	Si 1 active et a sans b, alors activation de 3 et désactivation de 1, 2 reste inchangé Si 1 et 2 et a et b alors 3 seule active

On appelle BARRE DE OU la barre symbolisant les entrées / sorties multiples d'étapes.

Divergence en ET :	Convergence en ET :
	
si 1 active et si a, alors désactivation de 1 et activation de 2 et 3.	Si 1 active seule et a alors aucun changement. Si 1 et 2 et a, alors activation de 3 et désactivation de 1 et 2.

On appelle couramment BARRE DE ET la double barre, mais attention ce n'est pas une entité à part mais une partie d'une transition.

Détaillons également le saut avant (si a alors ...) et les boucles (répéter ... jusqu'à c). Ce sont les deux seules possibilités avec des OU: il ne peut y avoir de divergence en ou après une transition

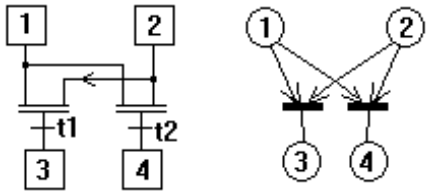


Passons maintenant à quelques problèmes plus complexes (tirés de "Comprendre et maîtriser le Grafcet, Blanchard, ed. Capadues"):

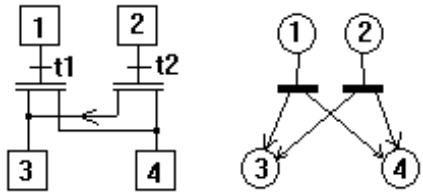
1– soient 4 étapes 1 à 4 et deux transitions de réceptivité t1 et t2. Construire la portion de Grafcet réalisant :
Quand 1 ET 2 actifs alors
si t1 passer en 3 (et désactiver 1 et 2),
si t2 passer en 4 (et désactiver 1 et 2),

sinon rester en 1 et 2

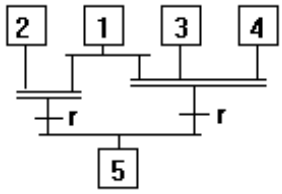
La solution ci–dessous est accompagnée d'une représentation de type "réseau de Petri" pour bien montrer où doivent se placer les convergences et divergences (à quoi doit être reliée 1?, à quoi doit être reliée t1? ...). En fait on trouve la solution facilement en analysant les cas d'évolution (quand franchit t'on t1 ?). Il faut souligner que l'ajout d'une étape intermédiaire n'est pas une bonne solution car tout passage d'une étape dure un laps de temps (donc discontinuité sur les sorties = aléa technologique)..



2 – Problème du même ordre : Quand (étape 1 et t1) OU (étape 2 et t2) alors passer en 3 ET 4:



3 – si {étape 1 et [étape 2 ou (étapes 3 et 4)]} et transition t alors activer l'étape 5 (et désactiver les autres).

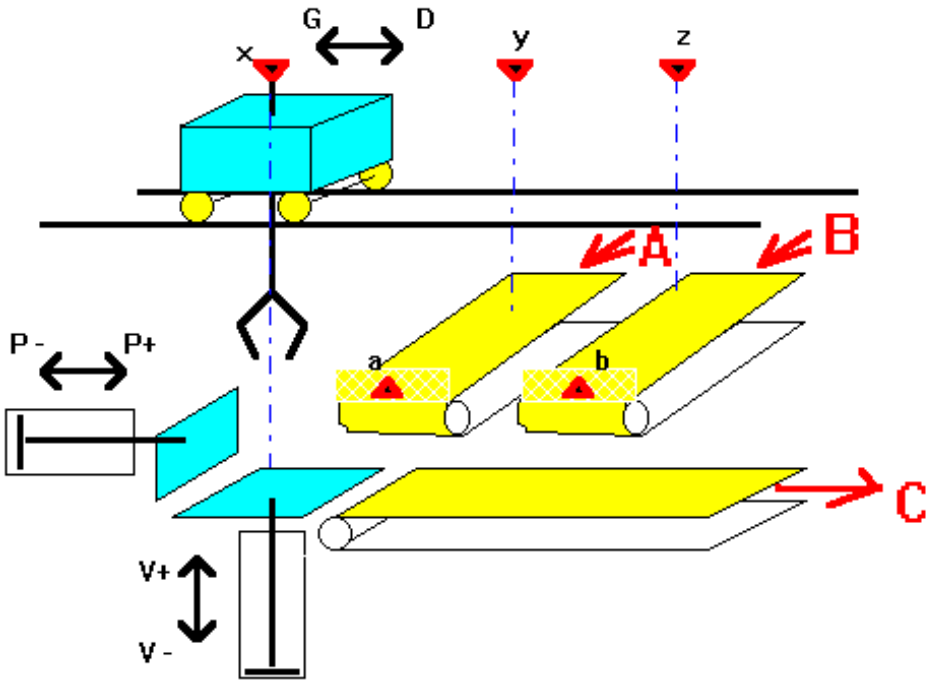


VI Cas génériques

Nous traitons ici des exemples génériques, c'est à dire que les problèmes évoqués ici se posent assez souvent, et la méthode utilisée pour les résoudre pourra être réutilisée.

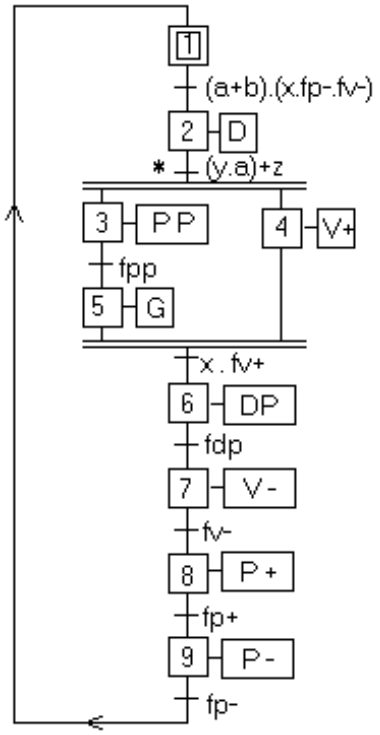
1 – priorité

Soit un chariot se déplaçant sur deux rails (action D vers la droite, G vers la gauche). Il comporte une pince pouvant prendre une pièce (PP, fin quand fpp) s'il se trouve sur le tapis A (capteur y) et qu'une pièce est présente (capteur a) (idem en z si b). Puis il retourne en x, pose la pièce (action DP, fin quand fdp) sur le plateaux supposé en position haute (fv+). Celui−ci descend (V−, jusqu'à fv−), un second vérin pousse la pièce (P+, fin quand fp+), puis le pousseur recule en fp−, le plateau remonte en fv+ Le tapis de sortie C est supposé toujours en mouvement. Les tapis A et B sont commandés par des systèmes non traités ici.



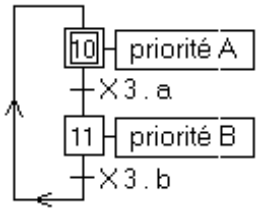
Effectuer d'abord un Grafcet linéaire comprenant une seule voie d'arrivée A. Puis l'améliorer en prévoyant les retours des actionneurs en temps masqué (attention toutefois de ne pas endommager le pousseur). Puis prévoir deux tapis d'alimentation A et B (en cas de pièces en a ET b, prendre celle en a). Puis prévoir une priorité tournante (en cas de conflit, prendre la voie qui n'a pas été servie la fois précédente) attention, si plusieurs pièces arrivent sur la même voie et aucune sur l'autre, ne pas bloquer le système. Puis modifier la règle de priorité en donnant en cas de conflit la priorité à celui qui n'en a pas profité lors du dernier conflit.

	Pour gérer la priorité tournante, remplacer la réceptivité de la deuxième transition (notée *) par : $y.a.(\overline{b}+X10)+z$



priorité voie A, retour V masqué

qui signifie : arrivé en y avec une pièce en a et soit pas de pièce en b, soit priorité. sinon on continue et quoi qu'il arrive on s'arrête en z (le chariot n'a pas de parachute), en rajoutant un second Grafcet définissant quelle voie est prioritaire :

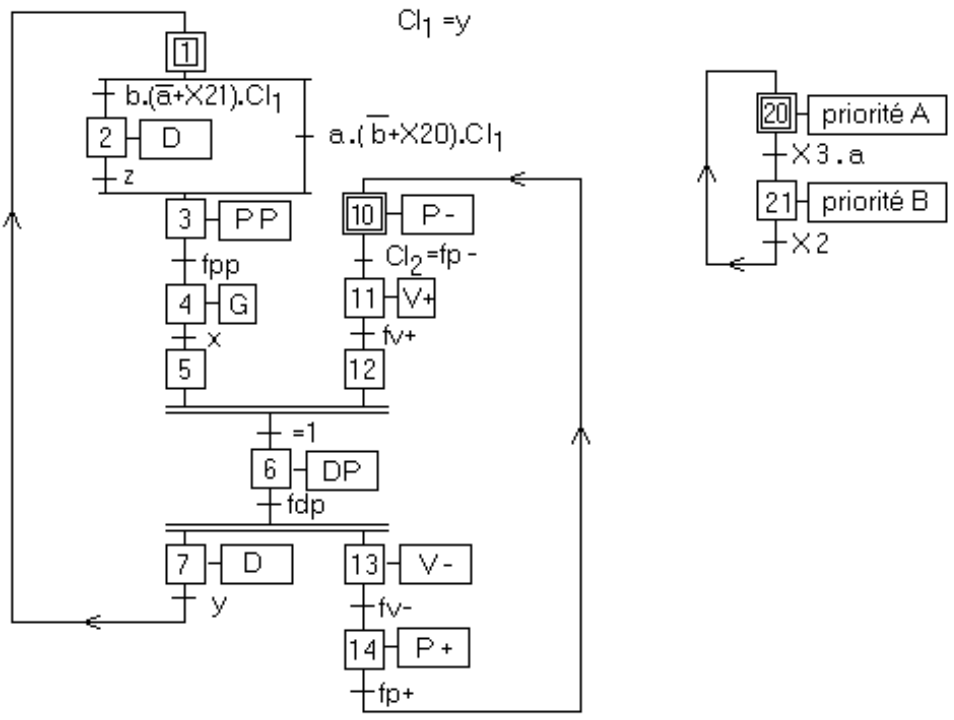


Chaque fois qu'une condition séquentielle (dépendant de ce qui s'est passé auparavant) intervient dans une réceptivité, il vaut mieux ne pas compliquer le Grafcet, mais "calculer" cette condition par un petit Grafcet annexe.

Améliorations :

a) permettre au chariot de rechercher une pièce dès qu'il a posé la précédente : séparer le problème en deux : chariot et partie basse.Prévoir deux Grafcet différents, pouvant évoluer simultanément, mais synchronisés pour le dépose de la pièce (par des Xi ou une ressource)

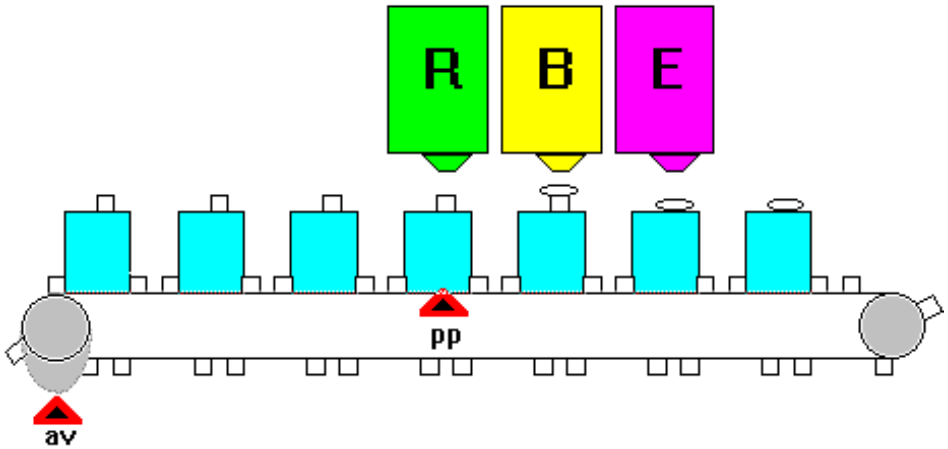
b) faire attendre le chariot en y plutôt qu'en x (pour améliorer le temps de réponse). Pour la partie basse, l'attente se fait plateau en haut, mais ce ne peut pas être l'état initial (il risque de descendre pendant la nuit). Prendre celà en compte :



Les deux étapes initiales ne testent que leurs conditions initiales respectives (partie haute ou partie basse).

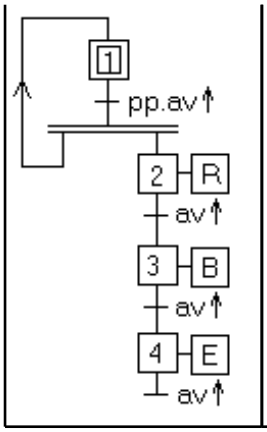
2 – travail à la chaîne

Soit une chaîne de remplissage de bidons d'huile. Un tapis roulant se déplaçant par saccades (cadencé par un système supposé externe à notre Grafcet, s'arrêtant à chaque nouvel appui de la came sur le capteur av) est alimenté manuellement (de temps en temps il manque des bidons). Trois postes sont prévus : remplissage (R), bouchage (B) et enfoncement (E).



Un seul capteur détecte la présence d'un bidon en début de chaîne : pp. On désire faire les 3 opérations simultanément, sauf s'il n'y a pas de bidon sous le poste. S'il vous semble obligatoire de rajouter des capteurs, vous n'avez RIEN compris au Grafcet puisqu'il vous faut un système combinatoire (il vaut mieux alors câbler en combinatoire chaque poste : avance tapis ET présence bidon => effectuer l'action). On suppose que le tapis est vide lors de l'initialisation.

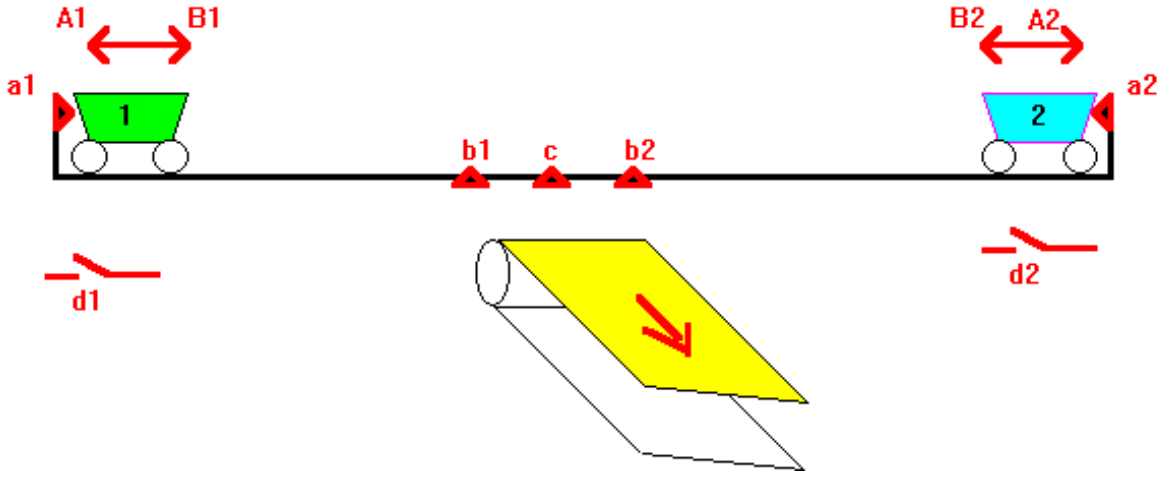
	L'étape 1 est constamment active. La dernière transition est appelée "transition puits", mais il était possible de la relier à l'étape 1. En fonctionnement normal, toutes les étapes du Grafcet sont actives. Du point de vue commande, chaque opération



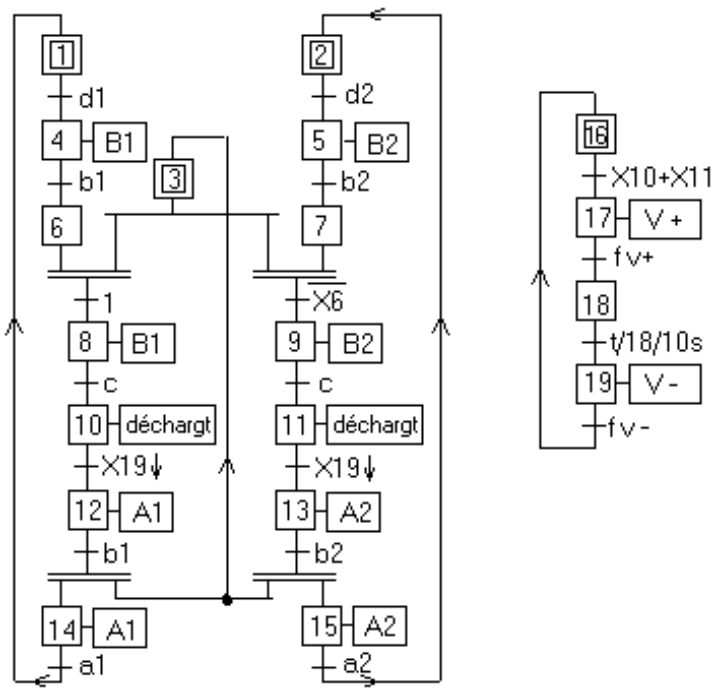
comportera plusieurs étapes (R = descendre l'entonnoir, ouvrir le robinet,...) dont une seule sera active à la fois). Chaque activation représente un bidon dans le circuit.

Cette méthode utilise au mieux le séquencement du Grafcet, on peut maintenant rajouter des capteurs, mais qui n'auront pour fonction que de vérifier le bon fonctionnement du système. Dans tous les cas similaires, on utilisera cette démarche : faire le Grafcet pour une pièce seule, puis le modifier pour gérer l'ensemble des pièces, en vérifiant bien que jamais une même étape ne corresponde à 2 pièces, on décompose donc le système en tronçons et on ne laisse entrer dans un tronçon que s'il est libre. Exemples : atelier flexible (on suit la pièce pour chaque opération jusqu'au produit fini), montage (monter 2 pièces ensemble correspond à une convergence en ET : de 2 étapes actives on arrive à 1), chariots filo−guidés (si un tronçon est occupé, essayer de le contourner par une voie libre)...

3 – ressource (ou sémaphore)



Au fond du puits de mine ndeg. i, un mineur remplit un chariot X_i. Quand il est plein (le chariot), il (le mineur) appuie sur un bouton d_i. Immédiatement, le chariot se déplace dans la direction B_i jusqu'au poste de déchargement, composé d'un tapis roulant en mouvement continu, et d'un vérin V qui retourne la benne. Si le poste de déchargement est libre, le chariot avance jusqu'au capteur c, est déchargé puis s'en retourne en a_i. Si le poste est occupé, il attend son tour en b_i. Le poste de déchargement, commun à plusieurs voies, n'est utilisable que par une voie à la fois. On l'appelle une "ressource physique". Traiter le cas de 2 voies (pas nécessairement de la même longueur).



Supposer que la ressource est occupée en utilisant le capteur c est **IDIOT** : et s'il est entre b et c ? Et si le temps de freinage l'a arrêté juste à côté de c ? Il faut utiliser les facilités séquentielles du Grafcet autant que possible (ne tester un capteur que quand c'est nécessaire). Un capteur ne doit servir que comme condition de passage d'une étape à une autre, mais pas pour vérifier un état du système qui découle du séquencement effectué (par exemple, une transition vérifie la présence d'une pièce, aucune action ne déplace la pièce puis on re−vérifie la présence : Ce n'est censé que si l'on prévoit dans le Grafcet ce qu'il faut faire si la pièce a disparu). Ici, on utilise donc une étape (la ressource), qui est active quand la ressource physique est disponible. Dès utilisation, on la désactive, pour la réactiver quand on libère la ressource physique.

On pouvait également résoudre le problème par des Grafcets séparés (un pour chaque chariot, un pour le déchargement) synchronisés par des Xi. La seule différence est que n'ayant plus de divergence sous l'étape 3, on risque d'oublier de traiter le cas d'arrivée simultanée en b1 et b2, cas arrivant assez rarement pour que l'on ne détecte pas le problème en phase d'essais, mais se produira de temps en temps en fonctionnement réel sans que l'on puisse reproduire le problème lorsqu'un spécialiste sera présent (seule solution : graphe des états accessibles).

Mise en Oeuvre du GRAFCET

Mise en oeuvre du GRAFCET

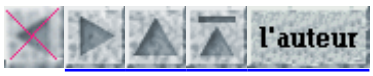
Patrick TRAU,ULP − IPST,Août 97

Un des nombreux avantages du Grafcet est sa facilité de mise en oeuvre. Non seulement la réalisation pratique de l'automatisme est facile et rapide, mais de plus on peut s'arranger pour que la réalisation pratique soit disposée de façon similaire au Grafcet, ce qui permet une maintenance facilitée (le mot est faible) par rapport aux autres méthodes.

Ce document est composé de deux tomes : celui−ci présente les généralités, et détaille la solution par câblage. Le second tome traite des solutions par programmation, en particulier les Automates Programmables Industriels (API), et peut être lû avant d'aborder le câblage.

Prérequis : la lecture de ce document nécessite une connaissance de base en automatisme combinatoire et séquentiel (cours disponible ici), et bien sûr du Grafcet.

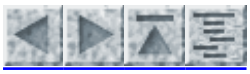
Patrick TRAU,ULP − IPST,Août 97


l'auteur

Quelle technologie choisir pour mettre en oeuvre un Grafcet ?

Tout est possible. Le choix ne dépend que de critères économiques, le Grafcet n'imposant aucune solution. Nous allons traiter les cas les plus courants :

- En cas de réalisation unitaire, comportant de nombreuses entrées / sorties, nécessitant des modifications de temps en temps (par exemple partie de ligne de production automatique), on choisira un automate programmable (API), programmé directement en Grafcet à l'aide d'un console de programmation (actuellement on utilise un PC, ce qui permet de saisir et simuler l'automatisme au calme, avant le test in situ). Cette solution semble assez chère dans l'absolu, mais reste économique relativement au prix des parties opératives mises en oeuvre. C'est de plus la solution qui minimise le prix des modifications (tant que la partie opérative n'a pas à être fortement modifiée).
- En cas de réalisation en petite série, de matériels qui devront être personnalisés pour chaque client (par exemple machine à emballer), on choisira comme précédemment un automate, programmable en Grafcet à l'aide d'une console (ou même par une connexion réseau). Ceci permet une production de matériels uniformes (ou en tous cas moins variables), la personnalisation se fera uniquement sur la console. On pourra vendre le produit avec l'automate seul (sans la console, qui vaut en général très cher), assorti d'un service après−vente pour les modifications ou évolutions.
- En cas de réalisation unitaire d'un petit automatisme (par exemple un transfert de pièces à l'aide d'un tapis et quelques vérins), on choisira un automate d'entrée de gamme, programmable uniquement dans un langage simple (ET, OU , mémoires...) La programmation sera aisée (voir mon document sur la ~~programmation d'un Grafcet~~) mais la modification sera souvent plus simple en réécrivant complètement le nouveau programme.
- Si l'on produit en série un système automatique, avec un fonctionnement prédéfini et figé mais pas trop compliqué (machine à laver par exemple), la solution la plus économique est le câblage : une carte électronique avec une bascule par étape, les seuls éléments coûteux sont les interfaces (donc le prix dépendra surtout du nombre d'entrées − sorties). Si un fabriquant de machines à laver me lit, qu'il m'explique pourquoi ils continuent à utiliser des programmateurs mécaniques qui sont plus chers et plus fragiles.
- Pour un système avec un fonctionnement complexe, où la rapidité est nécessaire, ou bien s'il nécessite également un peu de calcul numérique, on choisira une carte avec un micro−contrôleur (ou microprocesseur si les besoins sont plus importants), assorti d'une interface comme dans le cas précédent. La programmation est aisée (voir mon document sur la ~~programmation d'un Grafcet~~), une fois que l'on connaît bien le matériel. Le coût du matériel est dérisoire (quelques dizaines de francs pour un micro−contrôleur ST62 avec un peu de RAM, de ROM, une vingtaine d'entrées − sorties ToR et un port analogique), par contre le matériel de développement revient très cher, son acquisition n'est envisageable que si l'on prévoit de créer un certain nombre de systèmes (sinon on se rabattra sur le câblage ou l'API, ou la sous−traitance).
- Dans les cas où le système est incompatible avec l'électronique (champs magnétiques, parasites, ambiance humide...), on peut utiliser une partie commande dans la même énergie que celle de la partie opérative : pneumatique ou électrique. Dans ce cas une seule solution est possible, le câblage. On utilisera la même méthode que pour le câblage électronique, il suffit de savoir réaliser les fonctions combinatoires (ET, OU...) et un bistable équivalent à une bascule (distributeur bistable, relais auto−alimenté...). Un séquenceur est une telle bascule prévue pour représenter une étape, avec un brochage facilitant le chaînage d'étapes. Ce n'est presque jamais la solution la plus économique.



~~Patrick TRAU~~,~~ULP – IPST~~,Août 97

Réalisation par câblage

Réalisation par câblage

Le but de ce chapitre est de vous montrer comment mettre en oeuvre un Grafcet à l'aide de composants d'électronique ToR (portes et bascules). Vous pouvez, si vous ne l'avez pas encore fait, consulter mon ~~document~~ traitant de ces composants et des bases théoriques nécessaires à leur utilisation (algèbre de Boole,...).

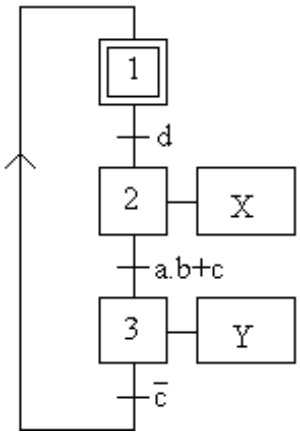
Cas sans problèmes

Nous allons d'abord voir les cas simples, par une méthode qui ne vérifie pas intégralement toutes les règles du Grafcet. Si j'en parle, c'est parce que les cas nécessitant plus de précautions sont rares et faciles à identifier.

Grafcet linéaire

Il suffit d'utiliser une bascule RS par étape. Une étape est allumée si l'étape précédente est active et que la réceptivité d'entrée est vraie. Dans le cas d'un Grafcet linéaire, on désactivera une étape quand la suivante est active. Ceci simplifie le câblage, mais ne respecte pas toutes les règles du Grafcet (en fait cette méthode fonctionne dans une très grande majorité de cas, nous traiterons les cas litigieux plus loin dans ce document).

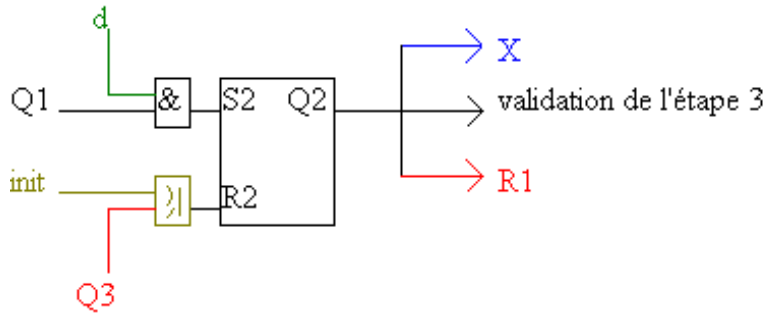
Soit le Grafcet :



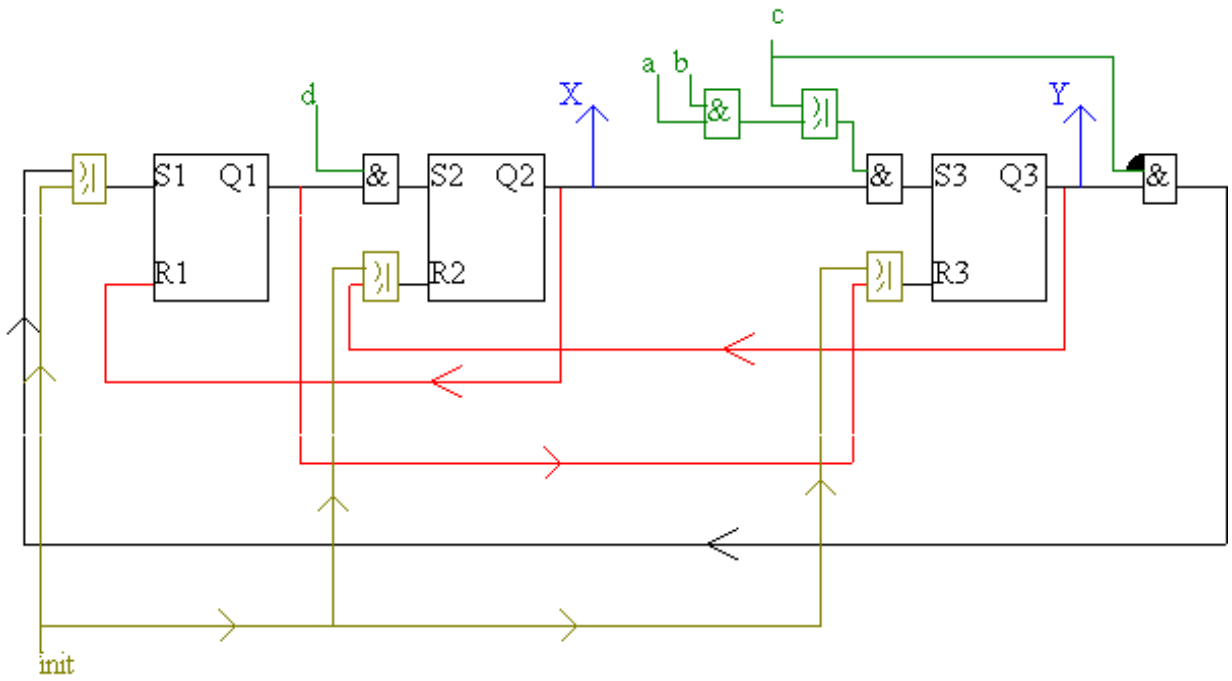
On peut gérer de différentes manières l'étape initiale. Dans la plupart des cas, le plus simple est d'utiliser des bascules se mettant à 0 à la mise sous tension, et d'initialiser l'automatisme à l'aide d'un bouton que je noterai ici "init", qui peut également servir à réinitialiser le Grafcet en cours de fonctionnement sans éteindre le système.

Notons, pour l'étape numéro i, son entrée Set par Si, son entrée Reset par Ri, sa sortie Qi. Etudions l'étape 2. Elle s'allume si l'étape 1 est active et d est vrai (S2=Q1.d). Tout le temps quelle est active, la sortie X est

allumée (X=Q2). Elle s'éteint normalement quand la réceptivité de sortie est vraie, mais (comme précisé plus haut) nous allons attendre pour éteindre l'étape 2 que l'étape 3 soit active (donc R2=Q3), et donc être sûr que l'étape 3 a eu le temps de prendre en compte l'information. Elle peut également être éteinte par init, puisqu'elle n'est pas initiale.



Il suffit de répéter cela pour chaque étape et relier le tout. Le schéma de câblage du système complet sera donc (j'ai gardé la même disposition que le Grafcet, mais retourné de 90 degrés, les électroniciens préfèrent les entrées à gauche et les sorties à droite) :



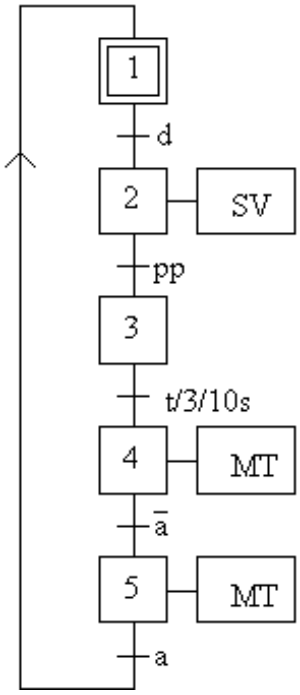
L'étude de chaque étape est simple, la principale difficulté est le routage (c'est à dire relier le tout), surtout si l'on veut faire un circuit imprimé (où les croisements de pistes sont impossibles). D'autant plus que chaque composant doit être alimenté, mais je n'ai pas représenté ici les alimentations. Mais il existe désormais de bons logiciels de routage.

On peut déjà conclure que si la mise en oeuvre d'un Grafcet par câblage n'est pas très compliquée, la modification est pour le moins difficile. En général, on préférera refaire un nouveau câblage si l'on désire modifier le Grafcet. De même, le câblage a intérêt à être complètement testé dès sa réalisation, la recherche d'erreurs après coup étant bien plus difficile.

~~**Exercice :**~~ câbler ce Grafcet de 5 étapes gérant une amenée de pièces :

Cahier des Charges :

à l'appui de d (départ), on actionne un vérin monostable par l'action SV, jusqu'à ce

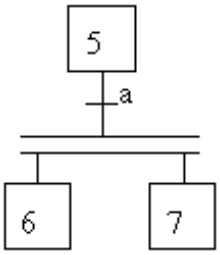


que la pièce soit positionnée sur le tapis. Puis on attend 10 secondes (retour du vérin), puis on enclenche le moteur du tapis roulant (MT) jusqu'à ce que la pièce arrive sur le capteur a. Comme la pièce précédente était peut−être en a au début du cycle, il faut attendre un front montant de a, que je gère en attendant que a soit d'abord relâché puis à nouveau appuyé. La temporisation sera réalisée par un composant réglable (en fait un circuit RC avec une résistance variable), qui donne 1 à sa sortie si son entrée est à 1 pendant au moins le temps requis.

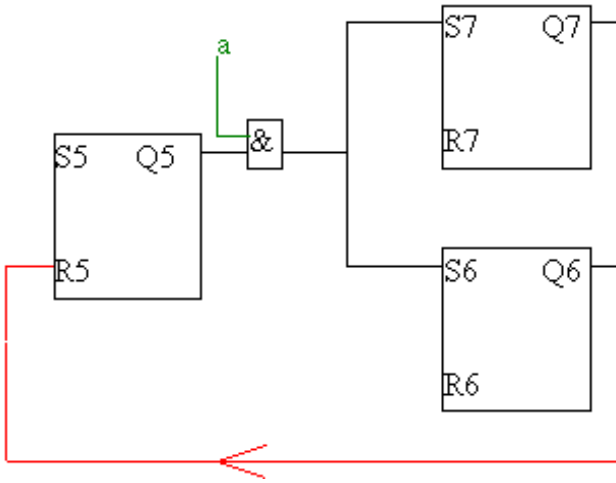
cliquez ~~ici~~ pour la solution

Divergence simple en ET

Quand la transition est franchissable, il suffit d'allumer deux étapes au lieu d'une. Le seul problème est la désactivation de l'étape précédente : il faut être sûr que les deux étapes suivantes ont eu le temps de prendre en compte l'information d'activation avant de désactiver la précédente (si l'on désactive dès qu'une des deux est active, la seconde ne s'activera plus).



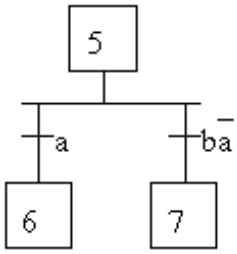
je ne traite ici ni l'amont, ni l'aval, ni les actions, uniquement les liaisons entre 5 et ses suivantes.

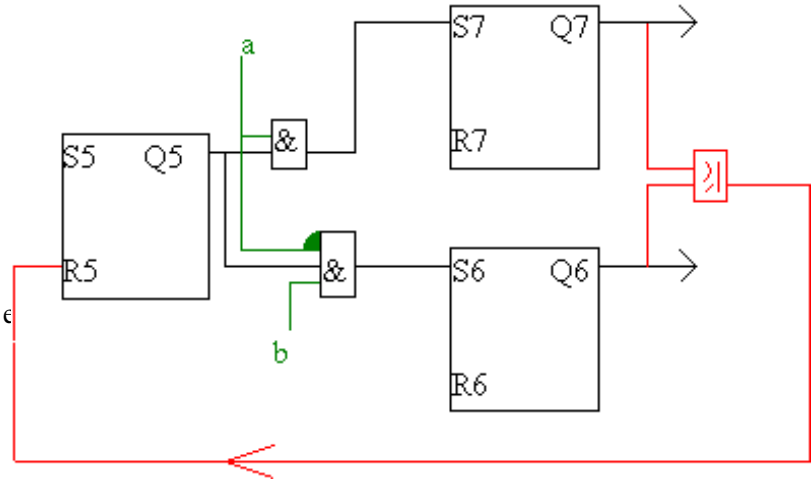


Ce câblage simple ne répond pas aux règles du Grafcet si 5 peut être réactivé avant que 6 et 7 n'aient été désactivées. Il en est de même si l'étape 7 par exemple peut être activée d'une autre manière (convergence en OU). Ces cas sont cependant très rares dans la pratique.

Divergence exclusive en OU

Il n'y a aucun problème particulier.





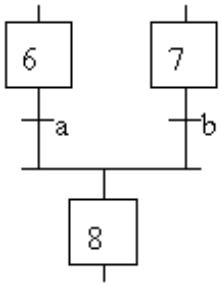
Comme au dessus, je ne traite ici que les liaisons é
et ses suivantes.

Si la divergence n'est pas exclusive (les deux réceptivités peuvent être vraies en même temps), c'est un peu plus compliqué, le mieux est de traiter les trois cas (l'une seule, l'autre seule, les deux).

Convergence en ET

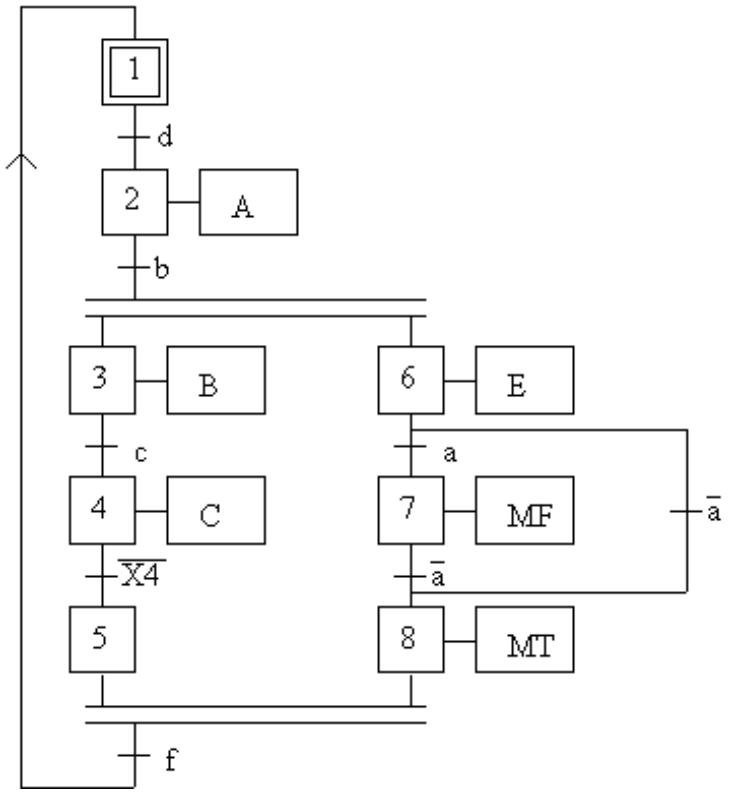
Je ne fais pas le schéma, il est évident : il faut que les (deux en général) étapes précédentes soient actives, et la réceptivité vraie, pour activer l'étape suivante, celle ci désactivant les étapes précédentes.

Convergence simple en OU



Vu le temps que je mets pour faire un schéma (le seul outil dont je dispose est paintbrush, et comme je suis en vacances je ne dispose que d'un portable à écran monochrome, nom parfaitement choisi puisqu'il n'est même pas noir et blanc mais gris et gris), je me contente de l'expliquer (ça vous fera un bon exercice).
On allume 8 si (6 et a) ou (7 et b). On éteint 6 et 7 tant que l'on a 8. Evidement ceci ne fonctionne que si l'on ne peut pas avoir simultanément 6 et 7 actives, mais j'ai bien dit (dans le titre ci−dessus) que je ne traite que le cas simple, qui de plus se trouve être aussi le plus courant.

Exercice récapitulatif



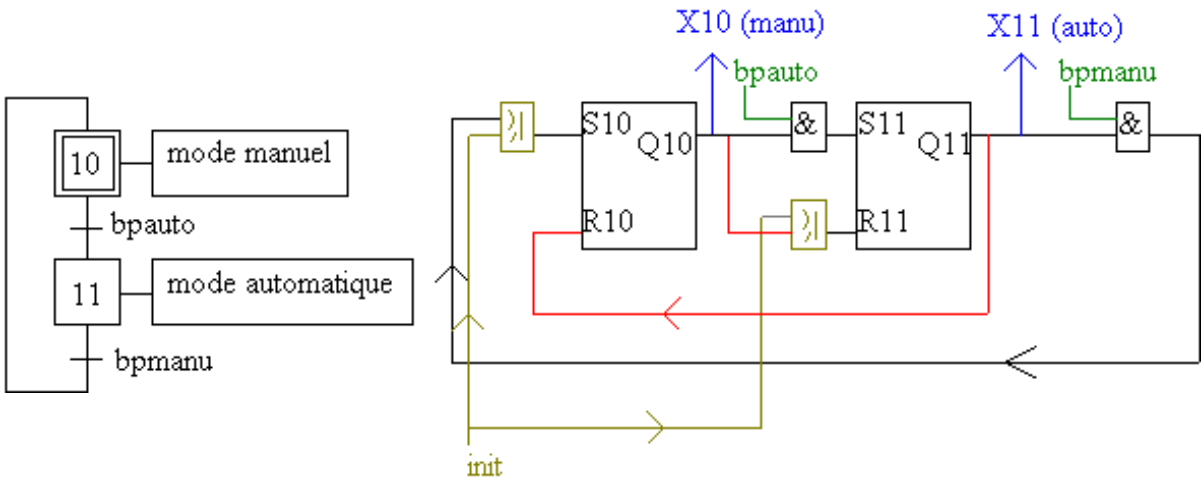
Câblez ce Grafcet (il ne pose pas de problème particulier). Ce Grafcet regroupe les différents cas de divergence – convergence.

cliquez ~~ici~~ pour la solution

Cas où cette méthode est mauvaise

Grafcet à deux étapes

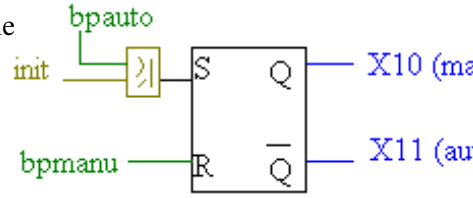
Soit le Grafcet suivant, et sa réalisation d'après la méthode précédente :



Quand 10 est actif (Q10) et bpauto vrai, en même temps on essaie d'allumer 11 par S11 et de l'éteindre par R11. Même en prenant une bascule à priorité déclenchement, l'état de 11 sera celui du dernier signal sur ses broches, ce qui risque d'être aléatoire.

Ici, la solution est simple : une seule bascule suffit. Mais cet exemple montre bien le

problème de ces câblages : une étape désactive la précédente tant qu'elle même lieu de ne le faire qu'au moment du franchissement de la transition.

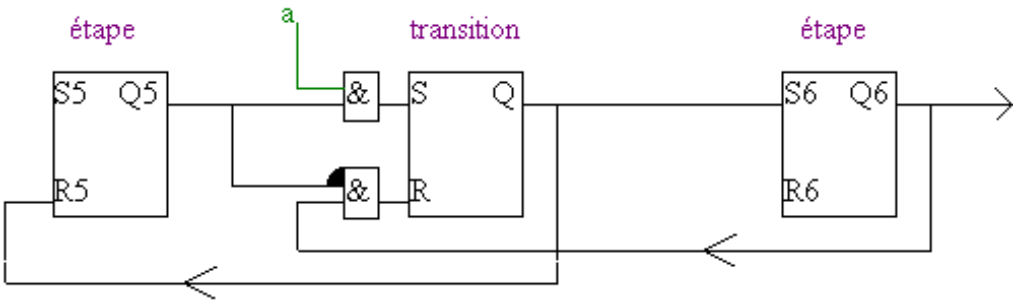


Le problème vient de la désactivation. Tous les composants ne peuvent pas avoir tous exactement un même temps de réponse. Donc puisqu'on active une étape quand la précédente est active et la réceptivité est vraie, si simultanément on désactivait la précédente il est possible que la suivante n'ai pas eu le temps de s'activer avant que le signal ne disparaisse. La solution choisie est sure, mais l'information de désactivation est envoyée bien plus longtemps que nécessaire. Pour être sûr du résultat il faudrait mémoriser (dans une bascule) l'état de chaque transition. En réalisation électronique ce n'est pas le prix qui poserait problème mais la complication du circuit (déjà assez complexe sans cela). En réalisation pneumatique ou électrique s'ajouterait le prix des composants.

mémorisation de la transition

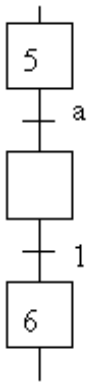
Donc une solution respectant mieux les règles du Grafcet consisterait à utiliser une bascule pour chaque transition. Elle est allumée quand l'étape précédente et la transition sont vraies, sa sortie active l'étape suivante et désactive la précédente. Quand doit on éteindre la bascule représentant la transition ? Le problème reste donc entier. Une bonne solution est de le faire quand le franchissement a été effectué, c'est à dire quand la suivante est active et que la précédente ne l'est pas. Attention, ce cas peut arriver sans que l'on soit passé par cette transition (convergence en OU par exemple), mais dans ce cas on éteint une transition qui l'était déjà, ce qui n'est pas grave.

Faisons donc le schéma de passage entre une étape 5 et une étape 6, reliées par une transition de réceptivité a :



Cette méthode permet de régler le cas où l'étape 5 risque d'être réactivée avant la désactivation de 6.

On peut remarquer que l'on aurait obtenu à peu près le même schéma en modifiant le Grafcet pour qu'il soit compatible avec la première méthode, c'est à dire empêcher qu'il y ait deux étapes successives actives en même temps : il suffit d'intercaler une étape comme représenté ci–contre. C'est une méthode qui permet d'avoir un Grafcet plus proche du câblage, donc un câblage plus clair.

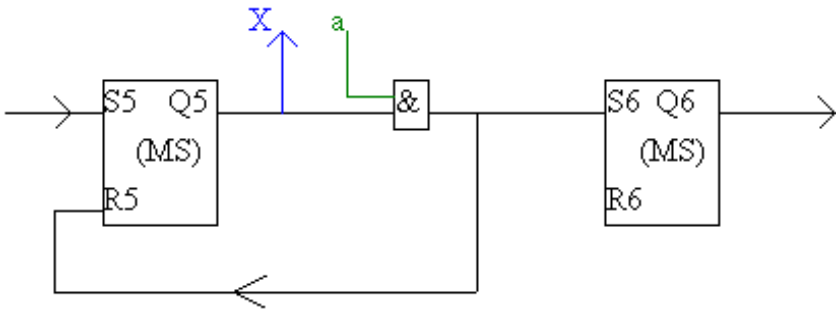


~~Exercice :~~ Câbler le ~~Grafcet~~ de la ~~chaîne de remplissage de bidons~~ proposé dans mon ~~document présentant le Grafcet~~. Attention, en fonctionnement normal (tous bidons présents), toutes les étapes du Grafcet sont actives ! Cliquez ~~ici~~ pour la solution.

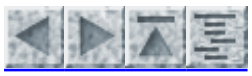
Bascules synchrones

La méthode précédente peut encore dans certains cas ne pas respecter la règle de simultanéité. Pour cela, une seule solution : synchroniser le fonctionnement des composants. Pour cela, il suffit de prendre la première méthode, mais d'utiliser des bascules MS (ou JK, voir mon ~~document sur les bascules~~ pour un peu plus de détails). Une bascule MS prend en compte les commandes Set et Reset qu'on lui applique non pas immédiatement, mais au prochain front montant de son entrée de synchronisation (horloge). La désactivation d'une étape se fait plus simplement : par la même information que celle qui active la suivante (les deux seront prises en compte en même temps : au prochain front de l'horloge. Il suffit de choisir une horloge suffisamment rapide pour ne pas ralentir l'automatisme (en général ce point ne pose pas de problème en P.C. électronique), mais plus lente que le temps de réaction du composant le plus lent.

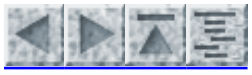
Faisons donc le schéma de passage entre l'étape 5 (d'action X) et l'étape 6, reliées par une transition de réceptivité a :



On peut immédiatement voir que le schéma résultant est grandement simplifié (je n'ai pas représenté l'horloge qui doit être reliée à chaque bascule, comme l'alimentation, plus la gestion de l'initialisation). On peut remarquer qu'une bascule MS est en fait composée de deux bascules RS, et que cette méthode revient à peu près au même que les autres modifications que j'ai proposées (en plus sûr et plus clair). La principale différence est que l'on fixe la durée de l'information de désactivation par un signal d'horloge.



~~Patrick TRAU~~,~~ULP – IPST~~,Août 97

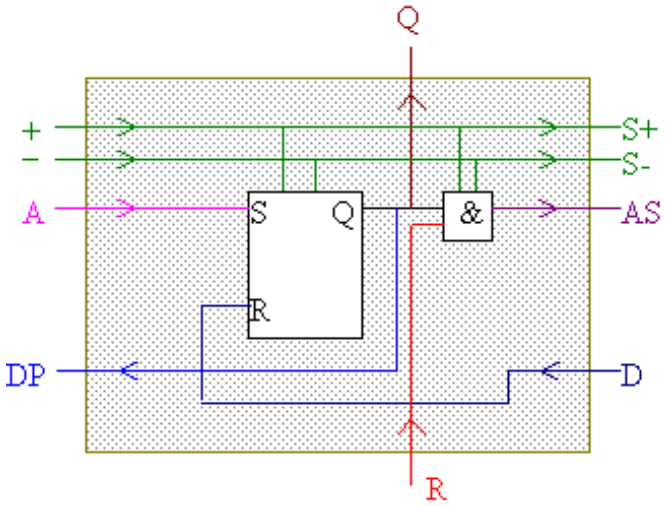


utilisation d'un séquenceur

utilisation d'un séquenceur

P.C. électronique

Supposons disposer du composant suivant, que j'appellerai un séquenceur :



il est alimenté par + et −, ce qui permet d'alimenter les composants à l'intérieur, mais aussi de transmettre cette alimentation (pour le séquenceur suivant). Le séquenceur représente une étape et sa transition de sortie. L'étape est activée par A (envoyé par l'étape précédente), et est désactivée par D (envoyé par l'étape suivante). Tant que l'étape est active, sa sortie Q est allumée, ainsi que DP qui servira à désactiver la précédente. Quand R est vrai (correspond à la réceptivité), le séquenceur envoie le signal AS (activation étape suivante), et ce jusqu'à ce qu'il soit éteint par D.

Donc ce séquenceur correspond exactement à la méthode présentée plus haut, dans les cas simples. Son seul avantage est de clarifier le câblage : en cas partie linéaire de Grafcet, les séquenceurs n'auront qu'à être mis côte à côte (on dit empilés), sans nécessiter de liaisons supplémentaires. Attention, ceci ne simplifie que les parties linéaires, pour les divergences, convergences et cas particuliers il faudra utiliser les câblages décrits précédemment, avec un résultat tout aussi confus.

Attention ! Ce séquenceur n'est sûrement pas disponible dans le commerce, et pour cause : je viens de l'inventer.

~~Exercice :~~ câblez le Grafcet de 5 étapes gérant une amenée de pièces proposé dans le chapitre ~~Grafcet linéaire~~ en utilisant ce séquenceur. Cliquez ~~ici~~ pour la solution.

P.C. pneumatique

Le câblage en pneumatique est exactement similaire au cas électronique (ce qui voulaient s'en passer devront néanmoins lire tout ce qui se trouve au dessus). On utilise des portes ET et OU, les bascules RS étant remplacées par des distributeurs bistables (à commande pneumatique évidement). Les différences sont :

- temps de réponse au minimum 1000 fois plus important,
- bruit important,
- maintenance préventive lourde (vérification tuyaux, connections...),
- prix des composants 10 à 100 fois supérieur,
- compatible avec des ambiances difficiles (humidité, parasites...)

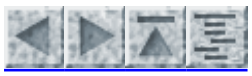
Je suis ouvert (un peu), ceux qui connaissent d'autres points (plus positifs) à rajouter ici, qu'ils m'envoient un

Vu le prix d'un câblage, on cherche toujours à minimiser le nombre de composants, et en général on cherchera à limiter le nombre d'étapes. Par exemple, on représentera par une seule étape une phase où l'on fait plusieurs actions successives, mais où les capteurs disponibles permettent de rendre ce "sous problème" combinatoire. Ce n'est qu'avec une grande expérience que l'on arrivera à optimiser le coût (mais qui restera en général bien supérieur à une solution électronique).

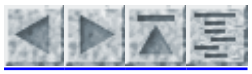
En pneumatique, il existe des séquenceurs (au fonctionnement exactement similaire à celui présenté en solution électronique) qui sont très utiles : ils simplifient grandement le câblage et réduisent de beaucoup le nombre de connexions à effectuer (par contre, ils sont chers).

P.C. électrique

On peut également directement câbler une Partie Commande en électrique (220 V par exemple). La bascule est réalisée à l'aide d'un relais auto–alimenté. Il a même existé des séquenceurs (RH je crois). Il est aujourd'hui hors de question de s'en servir (même dans l'éducation nationale on les a jetés, ce qui prouve bien que c'est totalement dépassé).



Patrick TRAU,ULP – IPST,Août 97



- Création d'une carte micro – programmée
- Utilisation d'un automate

Création d'une carte micro – programmée

Cette solution est très économique pour des systèmes nombreux mais modulables. Elle consiste en une carte comportant un microprocesseur (ou micro–contrôleur), et une interface de puissance pour toutes les entrées – sorties. La programmation d'un Grafcet est assez simple à réaliser (tous les détails sont dans mon document sur la ~~programmation d'un Grafcet~~) (Tome 2). Par contre cette programmation nécessite un matériel important (la réalisation aussi) qui est celui d'un département d'électronique numérique plutôt que celui habituellement disponible au sein d'un service automatisme.

Utilisation d'un automate

Un API (automate programmable industriel) est un matériel programmable pouvant être placé directement dans un environnement de production industrielle (sans aller jusqu'à accepter des jets de solvants). Ils possèdent un premier étage d'interfaçage (entres – sorties en 24 V par exemple, même quelquefois 220 V si l'on n'utilise qu'une faible intensité), un second étage spécifique à chaque actionneur devra être installé (mais disponible dans le commerce pour les cas habituels) : distributeurs , thyristors, relais... Les prix s'étalent de quelques kF à quelques centaines de kF.

En entrée de gamme, on trouve des automates avec quelques entrées – sorties ToR (Tout ou Rien), et un langage permettant de simuler les composants de base (ET, OU, NON, bascule, tempo, compteur). A un niveau de prix supérieur, on trouvera des systèmes avec un environnement de programmation de haut niveau (PC par exemple), simplifiant grandement la programmation et les tests (en particulier la possibilité de programmer directement l'automate en Grafcet). En haut de gamme on a des API modulaires : on les compose sur mesure, en ajoutant suivant les besoins des cartes d'E/S, des cartes numériques, des conversions numérique – analogique, des asservissements (DSP par exemple)... Excepté en entrée de gamme, les API permettent désormais des dialogues en réseaux d'automates, ce qui permet d'utiliser un superviseur qui ne fait que commander d'autres automates dans lesquels on a décentralisé les tâches (l'usage des réseau est grandement facilité par l'utilisation d'automates compatibles entre eux, en général de la même marque). On utilisera un API pour des automatismes unitaires, le câblage n'étant intéressant que pour une fabrication en série de produits automatisés.

Vous trouverez des détails sur la programmation des API dans mon document sur la programmation d'un Grafcet (Tome 2).

Patrick TRAU,ULP – IPST,Août 97

MISE EN OEUVRE DU GRAFCET SUR AUTOMATES

et autres systèmes programmables

Ce document est le deuxième tome traitant de la mise en oeuvre du Grafcet. La lecture du premier tome, traitant des généralités mais surtout du câblage, n'est pas nécessaire pour la compréhension de celui−ci.

Nous allons étudier dans ce document différents cas réels, de l'automate (appareil programmable permettant de gérer des automatismes en tout ou rien) d'entrée de gamme (Micro1, PB/April 15) en passant par le langage à contacts des TSX, jusqu'aux micro−contrôleurs (ST62xx) et ordinateurs (exemples en assembleur PC, en Pascal et en C). Les méthodes présentées ici s'adapteront à tout matériel programmable, c'est pour le prouver que l'on verra autant de langages.

---

quelques liens : la programmation des automates quand on est réfractaire au Grafcet (de l'autre côté de l'Atlantique)

Les fonctions de base d'un automate

Ces fonctions sont disponibles sur tous les automates, des plus simples aux plus évolués. Je vais définir un langage fictif (qui me servira pour la suite, dans la définition des méthodes). Ce chapitre continuera par une présentation succinte des instructions des différents matériels et langages utilisés dans la suite du document pour appliquer les méthodes présentées.

L'AF (automate fictif)

Je vais définir un automate ficitif (que j'apellerai par la suite AF), ainsi que son langage. Je fais ceci car les langages des automates sont très différents, pas très clairs, mais ils reviennent tous au même.

Un automate traite des variables booléennes (ne pouvant valoir que 0 ou 1). On distingue les entrées (en lecture seule), les sorties (en écriture mais généralement on peut les relire, au cas où on aurait oublié ce qu'on a allumé en sortie), et les variables internes (pour stocker les calculs intermédiaires) en lecture et écriture. Pour l'AF, je noterai les entrées Ei, les sorties Si, les variables internes Vi ou un nom en clair.

On dispose au moins des fonctions combinatoires ET, OU et NON, souvent plus (ou exclusif par exemple). Pour l'AF je supposerai pouvoir écrire une opération sous la forme : Résultat <= calcul, avec résultat pouvant être un sortie ou une variable interne, calcul une équation utilisant des entrées ou variables internes, les opérateurs ET, OU, / (complément) et parenthèses. Exemple : S3 <= (E1 ET /V3) OU V2 . On peut remarquer que dans la plupart des automates un tel calcul devrait se décomposer en plusieurs lignes de programme.

Pour gérer le séquentiel, on dispose de la bascule. Je la noterai : SI condition, résultat <= 1 ou 0. J'accepterai une condition complexe (comportant des opérateurs combinatoires), mais souvent dans la réalité il faudra passer par un calcul mémorisé dans une variable interne. Exemple : SI (E1 ET E2) ALORS S3 <= 1 .

La temporisation se notera : SI condition ALORS résultat <= 1 APRES n secondes. La temporisation est réinitialisée dès que la condition repasse à 0. Celle–ci doit donc rester validée tout le temps du comptage. Ce fonctionnement est admis sur tout les automates, mais souvent d'autres options sont possibles (déclenchement sur un front montant par exemple).

Certains automates autorisent des sauts (GOTO). Je les noterai SAUT. La destination du saut (en général un numéro de ligne) sera donné en AF par un label : un nom suivi du signe ":". Les automates qui n'acceptent pas les sauts bouclent continuellement sur l'ensemble du programme, mais certaines parties peuvent être invalidées suivant l'état de certaines variables, ce qui revient donc au même. Certains permettent également un saut conditionnel : SI condition SAUT. Exemple :

```
        hors boucle
label :
        dans boucle
        SI E1 SAUT label
```

Voyons quelques exemples d'automates réels :

Langage booléen du PB 100 ou April 15

Adresses

Le PB100 fonctionne en hexadécimal. On peut possèder par exemple 24 entrées appelées 000 à 017 et 24 sorties appelées 018 à 02F. Ces entrées−sorties peuvent être modifiées par groupe de 8 par adjonction de cartes (64 E−S maxi). Il possède 512 variables internes binaires (pouvant donc valoir soit 0 soit 1) appelées A00 à BFF. Les programmes doivent être écrits à partir des lignes 0C30, et peuvent aller jusqu'à 0FFF (sauf ajout de modules mémoire).

Le PB 15 par contre, possède les entrées 000 à 017, les sorties 020 à 02F et les variables internes sont limitées de A00 à A3F. Tout le reste fonctionne de manière analogue.

Langage booleen

Le langage est très simple : on peut simuler les fonctions ET, OU et les bascules bistables :

On désire allumer la sortie 020 si (l'entrée 000 est à 1 ET 001 est à 0) ou 002 est à 1 ou 003 est à 0. que je noterai en AF : S020<= (E000 ET /E001) OU E002 ou E003. Le programme correspondant sera :

```
0C30 SI  000
0C31 SI/ 001
0C32 ET  A00
0C33 SI  A00
0C34 SI  002
0C35 SI/ 003
0C36 OU  020
0C37 SAUT C30
```

La fonction ET possède ici 2 entrées : 000 et /001. Le résultat (0 ou 1) est mis dans la variable interne A00. La fonction OU possède ici 3 entrées : A00, 002, /003. Le résultat est mis sur la sortie 020.

Attention : un programme est une suite d'instructions, qui sont exécutées l'une après l'autre. Si une entrée change après le passage sur l'instruction qui la prend en compte et que l'on ne repasse plus sur les instructions, la sortie n'est pas modifiée. C'est la raison de la dernière ligne du programme : repasser sans arrêt sur l'ensemble du programme.

Par rapport à un câblage, on a donc deux désavantages : temps de réponse (un changement des entrées sera pris en compte au maximum après le temps d'un passage sur l'ensemble du programme, c'est ce qu'on appele le temps de scrutation, qui sera ici inférieur à la milliseconde) et non simultanéité (on n'effectue qu'un instruction à la fois). Mais ces temps étant en général très inférieurs aux temps de réaction des capteurs et actionneurs (inertie d'un moteur par exemple), ceci n'est que rarement gênant. L'avantage est que c'est programmable, donc facilement modifiable.

Les fonctions ET et OU acceptent autant d'entrées que l'on désire. Si on n'en utilise qu'une, on a une simple recopie de valeur (ex: SI A00 − OU A01).

Le SAUT peut être précédé de plusieurs SI ou SI/. si la conjonction (ET) des conditions est vraie, alors le saut est fait, sinon on continue sur la ligne suivante. Sur PB 15, seuls les sauts vers l'avant (SAUT Axx avec

xx=nb de lignes à ne pas faire +1) sont possibles, seule la dernière ligne du programme peut contenir un saut en arrière (sans SI).

Les fonctions MU (mise à un) et MZ (mise à zéro) permettent de traiter les bistables. Il peuvent eux aussi être précédés par un ou plusieurs SI :

(1)	SI A00	(2)	SI A00
	SI B0C		SI B0C
	MU AF3		ET AF3

si A00=1 et B0C=1 alors (1) et (2) mettent AF3 à 1. Sinon, (2) met AF3 à 0, mais (1) ne le fait pas (AF3 garde la même valeur qu'avant, 0 ou 1).

Une autre fonction utile est le DE :

```
DE A00
MZ BFF
```

donnera le même résultat que **MZ A00 – MZ A01 – MZ A02 – ... – MZ BFE – MZ BFF**

la temporisation

Il faut avant tout donner la condition qui déclenchera la tempo, par un ou plusieurs SI. Puis TP et le nom de la variable qui sera mise à 1 au bout de la tempo. Puis donner la base de temps (sur PB15 09FF, 09FE, 09FD pour 1 sec, 1/10è, 1/100è; sur PB100 BT F, BT E pour 1 sec, 1/10è). Puis donner l'adresse où l'on stocke la durée, et l'adresse réservée au compteur. Ces adresses sont n'importe quelle ligne de programme (mais sur laquelle on ne devra pas passer) sur PB100, et entre 0800 et 0817 sur PB15).

Exemple: 0C30	SI 000	si appui sur le capteur 000
	TP 020	allumer la sortie 020 après un certain délai
	09FF	le délai sera donné en Secondes
	0800	adresse durée
	0801	adresse réservée compteur
	SAUT C30	
0800	0005	la durée sera de 5 secondes
0801	0000	

En appuyant le contact, la sortie sera allumée 5 secondes après. Elle restera allumée jusqu'à ce qu'on lâche l'entrée. le compteur sera alors automatiquement remis à zero. Si on appuie moins de 5 secondes, rien ne se passe. Cetteinstruction n'arrête pas le programme, il faut constamment passer sur les lignes d'instruction pour que la sortie soit bien affectée en bout de tempo.

Le langage à contacts du TSX

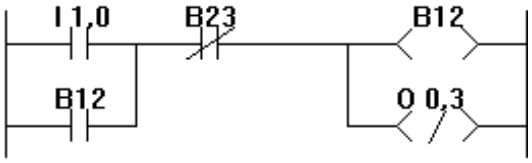
On peut trouver une documentation plus complète sur cet automate dans mon document ~~"description du TSX"~~

Sur un TSX, les sorties peuvent être appelées O0,0 à O0,F, les entrées I1,0 à I1,F (si le rack de 16 sorties est positionné en position 0, les 16 entrées en 1). Les variables internes sont notées en décimal de B0 à B255.

Les réseaux

Les schémas sont effectués l'un après l'autre, de haut en bas (et non suivant leur label). Chaque réseau est scruté par colonne de gauche à droite.

ex:



Dans ce cas l'entrée B12 est l'ancienne valeur de la bobine (variable interne) B12. Si l'on veut utiliser le résultat de ce réseau, il faut utiliser B12 dans le réseau suivant.

On note un capteur par le signe −−| |−−, un contact inverse par −−|/|−−.

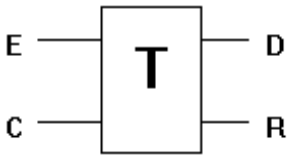
Une bobine est notée −−< >−− , une bobine inverse −−< / >−− (commandée par un niveau 0).

Un bistable est allumé par −−< S >−− , éteint par −−< R >−− .

Un saut à un autre réseau est noté −−< J >−−. On a intérêt de n'utiliser que des sauts avants (vers la fin du programme). L'automate fixe automatiquement les entrées au début de cycle et n'affecte les sorties qu'en fin de cycle (les variables internes sont évidement immédiatement modifiées).

Temporisation

On représente la tempo par le signe :



E correspond à l'armement de la tempo, C au contrôle. D passe à 1 en fin de tempo, R est à 1 tant que la tempo est en cours. En appuyant la touche ZM, on peut donner : TB: la base de temps, PRESET: la durée.

le Micro 1 de IDEC−IZUMI (distribué par CHAUVIN ARNOUX)

On peut trouver une documentation plus complète sur cet automate dans mon document ~~"description du Micro 1"~~

Cet automate est produit d'entrée de gamme : prix : environ 3000F, compact (environ 1 dm^3), avec 6 sorties (numérotées 200 à 205) comportant chacune un relais (maxi 2A sous 220V), 8 entrées (24V) (numérotées 0 à 7). Les variables internes vont de à . Les opérations sont font via une pile (un peu comme une calculatrice HP) :

Pour effectuer S203 <= (E00 ET /E01) OU (/E02 ET E03) on écrit :

LOD 0	charger 0 sur le sommet de la pile
AND NOT 1	opération ET entre le sommet de la pile et l'argument donné, le résultat remplace le précédent sommet de la pile
LOD NOT	empiler (au dessus du résultat précédent) une nouvelle valeur

2

AND 3	faire le calcul (le résultat remplace le sommet de la pile)
OR LOD	OU entre le sommet de la pile et le précédent, le résultat remplace le précédent et la pile a baissé d'un étage
OUT 203	Copie du sommet de la pile sur la sortie (sans changement dans la pile !!!)
END	fin du programme, saut automatique en première ligne

La bascule : si sommet de la pile (dernier calcul ou LOD) vaut 1, SET résultat ou RST résultat allumera ou éteindra le résultat, sinon il restera inchangé

le saut : si sommet de la pile (dernier calcul ou LOD) vaut 1, JMP sautera au prochain JEND (saut avant uniquement, pas d'imbrications)

micro contrôleur ST62xx

L'assembleur sera utilisé soit sur un ordinateur (plutôt PC) mais surtout si l'on a choisi d'utiliser un micro−contrôleur. Ceux−ci sont des composants comportant un micro−processeur, de la RAM et PROM, des ports d'E/S, et souvent un CAN. Ces composants sont la solution idéale en cas de production en série de produits automatiques (même pour une petite série) : ils coutent à peine plus de 10 F pièce (si on en achète beaucoup, évidement). Un bon exemple est le ST62 :

Il possède trois ports d'E/S 8 bits, un compteur/timer 16 bits et un CNA 8 bits. L'EPROM est de 2 ou 4 Ko, la RAM utilisable est cependant limitée à 64 octets (256 adressables dont les 3/4 utilisés par le système). Le minimum à savoir sur son langage est simple :

− chargement d'un registre : LD A,adresse ou LDI A,valeur (on dispose aussi d'autres registres et d'autres adressages)

− comparaison avec une valeur : CPI A,valeur

− masquage : ANDI A,valeur

− ET, complément : AND A,adresse , COM A

Mais surtout des instructions avec accès direct à un bit donné , ce qui facilite la programmation et évite les problèmes de masquages et décalages :

− saut si un bit d'une mémoire est à 1 : JRS num_bit,adresse,destination_du_saut (JRR saut si bit=0)

− mise à 1 d'un bit d'une mémoire : SET num_bit,adresse

− mise à 0 d'un bit d'une mémoire : RST num_bit,adresse

Il possède bien évidement toutes les possibilités habituelles des micro−processeurs : interruptions (5 vecteurs, reset compris), sous− programmes (mais petite pile : 4 ou 6 imbrications maxi)...

assembleur (PC)

L'assembleur ne sera que rarement utile : personellement je conseillerai plutôt C. Néanmoins ce n'est pas une possibilité à négliger, notament dans les cas nécessitant un traitement temps réel.

On trouvera un descriptif plus complet dans le document "mémento 8088"

On se limitera aux instructions suivantes :

LABEL: donne un nom à une ligne de programme

MOV reg,val met la valeur VAL dans le registre REG (ici DX ou AL)

OUT DX,reg envoie sur le port de sortie dont l'adresse est dans DX la valeur contenue dans le registre REG (ici AL)

IN reg,DX met dans le registre REG (ici AL) la valeur disponible sur le port d'entrée dont l'adresse est dans DX

TEST reg,val masque le contenu de REG par VAL (fonction ET) sans changer REG. Le(s) bit(s) de REG correspondant à un 1 de VAL seront inchangés, les autres sont mis à 0

JZ label saute à l'adresse LABEL si le résultat du test précédent était 0 (le bit non masqué valait 0)

JNZ label saute à LABEL si le résultat du test précédent était non nul.

JMP label saute à LABEL (sans condition)

Je pense que ces petites explications suffiront pour comprendre les programme que je donnerai plus bas

En langage évolué

Vous pouvez consulter si nécéssaire mon livre sur le ~~Language C~~ ou mon document sur le ~~Pascal~~.

Tous les langages conviennent : ils savent tous faire ET, OU et NON, mettre une mémoire à 1 ou à 0 sous condition (IF). Le Pascal permet un programme plus clair que tous les autres à condition d'utiliser la notion d'ensembles (est−ce que l'étape X appartient à l'ensemble des étapes actives ?). Dans les autres langages il faudra faire des boucles et des masquages, pour cela le C sera certainement le plus pratique (ainsi que pour gérer directement une carte d'entrées − sorties). Rappel : les sauts sont possibles dans tous les langages classiques, contrairement à ce que certains enseignants essayent de faire croire.

Conclusion

Ces fonctions de base sont présentes dans tous les automates (même si elles sont mises en oeuvre par des langages très différents, y compris graphiques), sauf les sauts qui peuvent être plus limités (au minimum bouclage automatique sur l'ensemble du programme, mais sans sauts dans le programme). Nous utiliserons ces seules fonctions pour voir comment programmer un grafcet, mais le principe reste valable quel que soit l'automate. Souvent, d'autres possibilités existent, en particulier temporisations, comptage, comparaisons,... Voir les documentations correspondantes si nécessaire.

Programmation d'un Grafcet dans le langage de base

Méthode globale

Cette méthode marche sur tout automate, pour tout Grafcet. Nous l'appliquerons au cas particulier de l'AF mais le principe est rigoureusement le même sur n'importe quel langage d'automate (y compris les langages graphiques comme le langage contacts du TSX), ainsi qu'en langage machine ou même langage évolué tel Pascal ou C

Principe

On utilise une variable binaire pour représenter l'état d'activation de chaque étape, et une variable pour le franchissement de chaque transition. De plus on fige les entrées pour une durée du cycle. Il est capital de bien préciser, sur une feuille de papier, à quoi correspond chaque entrée, sortie et variable interne.

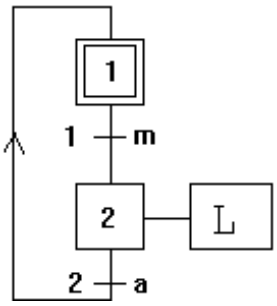
```
   - initialisation (mise à 1 étape(s) initiale(s), à
       0 les autres, mise à 0 des sorties,...)
+->- lecture des entrées    (et copie dans des variables internes)
|  - calcul des conditions d'évolution (quelles transitions seront franchies)
|  - désactivation des étapes à désactiver
|  - activation des étapes
|  - combinatoire (si nécessaire: tempos, comptage, actions conditionnelles...)
|  - affectation des sorties (en fonction des étapes actives)
+-------+ (saut après l'initialisation)
```

Remarque sur la phase de lecture des entrées : celle–ci est inutile sur la plupart des automates simples (ils bloquent les entrées le temps d'un cyle, ce qui les empêche des programmes du genre : boucler tant que capteur laché). C'est la seule solution pour respecter l'algèbre de Boole : a.(b+c) doit être équivalent à a.b + a.c même si a change au milieu de calcul.

L'ordre des phases est capital pour respecter les règles du Grafcet (surtout 4 et 5), il ne peut être modifié que dans les cas simples (en particulier quand on n'a pas besoin des règles 4 et 5).

Exemple simple : Grafcet 1

Appliquons la méthode au Grafcet le plus simple possible, dans le seul but de bien la comprendre. Que l'on soit bien d'accord, ce problème peut se résoudre bien plus simplement (à l'aide d'un interrupteur par exemple).



choix des adresses et variables internes :

– entrées :
m : entrée E00, variable interne V00
a : entrée E01, variable interne V01
– sortie L : S20
– étapes : 1=V21, 2=V22
– transitions : 1=V11, 2=V12

Programme en AF :

```
Début :
     V21<=1    initalisation : étape 1 active
     V22<=0                    étape 2 non active
Boucle :
     V00<=E00    lecture des entrées : copie de l'état du capteur m dans V00
     V01<=E01    copie de l'état du capteur a dans B01

     V11<=V00 ET V21   conditions d'évolution : transition 1 passante si étape 1 active et capt
     V12<=V01 ET V22       transition 2 passante si étape 2 active et capteur a

     SI V11 ALORS V21<=0   désactivation :si transition 1 passante, désactiver l'étape 1
     Si V12 ALORS V22<=0     si transition 2 passante, désactiver l'étape 2

     SI V11 ALORS V22<=1   activation : si transition 1 passante, activer l'étape 2
     SI V12 ALORS V21<=1   si transition 2 passante, activer l'étape 1

     S20<=V22   affectation des sorties : allumer L si étape 2 active  (éteindre sinon)

     SAUT Boucle  boucler (mais ne pas réinitialiser)
```

Quelques remarques : cette méthode marche quel que soit le nombre d'étapes actives simultanément. Le fait de bloquer les capteurs pour un cycle allonge le temps de réaction mais donne un résultat conforme au grafcet (si par exemple le capteur change entre le passage sur la désactivation et l'activation). La désactivation de TOUTES les étapes doit précéder l'activation : essayez 2 étapes qui se suivent, toutes les 2 actives, suivies de la même réceptivité (front montant sur un capteur par exemple). Le programme obtenu est long et lent, mais conçu rapidement (pas ou peu de réflexion).

langage booleen APRIL – PB :

Mettons en oeuvre cette méthode, pour ce Grafcet, en language booléen :

Choix des variables : entrées : m : 000, mémorisé dans B00; a: 001, mémorisé dans B01. Sortie L : 020, étapes : A01 et A02, transitions : A11 et A12. La phase de mémorisation des entrées (dans B00 et B01) n'est nécessaire que sur PB100, inutile sur PB15 qui bloque les entrées le temps d'un cycle.

```
     0C30 MU  A01   initalisation : étape 1 active
     0C31 MZ  A02                  étape 2 non active

     0C32 SI  000   lecture des entrées :
     0C33 ET  B00                  copie de l'état du capteur m dans B00
```

```
0C34 SI  001
0C35 ET  B01                  copie de l'état du capteur a dans B01

0C36 SI  A01   conditions d'évolution :
0C37 SI  B00       transition 1 passante si étape 1 active et capteur m
0C38 ET  A11
0C39 SI  A02
0C3A SI  B01       transition 2 passante si étape 2 active et capteur a
0C3B ET  A12

0C3D SI  A11   désactivation :
0C3E MZ  A01       si transition 1 passante, désactiver l'étape 1
0C3F SI  A12
0C40 MZ  A02       si transition 2 passante, désactiver l'étape 2

0C41 SI  A11   activation :
0C42 MU  A02       si transition 1 passante, activer l'étape 2
0C43 SI  A12
0C44 MU  A01       si transition 2 passante, activer l'étape 1

0C45 SI  A02   affectation des sorties :
0C46 OU  020       allumer L si étape 2 active (éteindre sinon)

0C47 SAUT C32  boucler (mais ne pas réinitialiser)
```

Application en ST62xx

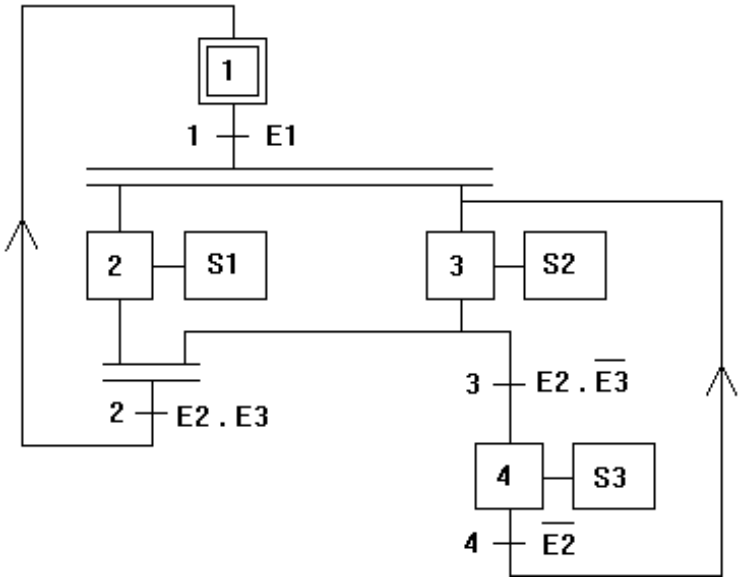
Toujours pour ce même cas, supposons :

- entrées : port A (bit 0 = m, bit 1 = a, bits 2 à 7 inutilisés)
- sorties : port B (uniquement bit 0 pour notre seule sortie)
- variables internes : capt : mémorisation des entrées, etap pour les étapes (bits 0 et 1 uniquement), trans pour les transitions (bits 0 et 1 uniquement).

```
;définition des adresses
PortA   .def    0C0h    ;registre de données du port A
PortB   .def    0C1h    ;registre de données du port B
DirA    .def    0C4h    ;registre de direction du port A
DirB    .def    0C1h    ;registre de direction du port B
capt    .def    0A0h    ;pour figer les entrées (adresse choisie parmi les 64 disponibles)
etap    .def    0A1h    ;étapes actives
trans   .def    0A2h    ;transitions (franchissables ou non)
;définition des constantes
BitM    .equ    00h     ;entrée m branchée sur le bit 0
BitA    .equ    01h     ;entrée a branchée sur le bit 1
;initialisation
        LDI DirA,00h    ;tout en entrée
        LDI DirB,01h    ;seul bit 0 en sortie, les autres inutilisés ici
        LDI PortB,00h   ;éteindre les sorties
        LDI etap,01h    ;étape 1 active
boucle :
;scrutation des entrées
        LD A,PortA
        LD capt,A
;conditions d'évolution
        LDI trans,00h
        JRR 0,etap,trans2       ;saut plus loin si étape inactive
        JRR BitM,capt,trans2    ;saut plus loin si capteur éteind
        SET 0,trans     ;allume la transition (bit 0)
trans2:
        JRR 1,etap,desact1
        JRR BitA,capt,desact1
```

```
        SET 1,trans     ; allume bit 1
;désactivations
desact1:
        JRR 0,trans,desact2     ;ne rien faire si transition non franchissable
        RST 0,etap
desact2:
        JRR 1,trans,act1        ;ne rien faire si non franchissable
        RST 1,etap
;activations
act1:
        JRR 0,trans,act2        ;ne rien faire si transition non franchissable
        SET 0,etap
act2:
        JRR 1,trans,sorties     ;ne rien faire si non franchissable
        SET 1,etap
;affectation des sorties
sorties:
        LDI A,00h
        JRR 1,etap,FinSorties   ;ne pas allumer la sortie si étape non active
        LDI A,01h;
FinSorties:
        LD PortB,A
;bouclage
        JP boucle
.END
```

Exemple complexe : grafcet 2



Ce grafcet a surtout un intérêt didactique : tous les ET et OU pour un minimum d'étapes.

Choix des variables (i entre 1 et 4) : étape i : ETi, transition i : TRi , entrée Ei mémorisée dans Vi

Programme correspondant en AF :

```
initialisation :
        ET1<=1
        ET2<=ET3<=ET4<=0
entrees:
        V1<=E1
        V2<=E2
        V3<=E3
evolution:
        TR1<=ET1 ET V1
```

```
        TR2<=ET2 ET ET3 ET V2 ET V3
        TR3<=ET3 ET V2 ET /V3
        TR4<=ET4 ET /V2
desactivation:
        SI (TR1) ALORS ET1<=0
        SI (TR2) ALORS (ET2<=0 , ET3<=0)
        SI (TR3) ALORS ET3<=0
        SI (TR4) ALORS ET4<=0
activation:
        SI (TR1) ALORS (ET2<=1 , ET3<=1)
        SI (TR2) ALORS ET1<=1
        SI (TR3) ALORS ET4<=1
        SI (TR4) ALORS ET3<=1
sorties:
        S1<=ET2
        S2<=ET3
        S3<=ET3
bouclage:
        SAUT entrees
```

Cas du langage Booléen

Les numéros de lignes n'ont été mis que pour les premières (pour savoir où l'on doit boucler), Les suivants sont faciles à calculer. Les colonnes sont à imaginer une en dessous de l'autre.

Choix des variables : étape i : A0i, transition i : A1i, entrée Ei : 00i (et mémorisation dans B0i), sortie Si : 02i

```
0C30 MU  A01      SI  A01        SI  A11       SI  A11        SI A02
0C32 DE  A02      SI  B01        MZ  A01       MU  A02        OU 021
0C33 MZ  A04      ET  A11        SI  A12       SI  A11        SI A03
                  SI  A02        MZ  A03       MU  A03        OU 022
0C34 SI  001      SI  A03        SI  A12       SI  A12        SI A04
     ET  B01      SI  B03        MZ  A02       MU  A01        OU 023
     SI  002      SI  B02        SI  A13       SI  A13
     ET  B02      ET  A12        MZ  A03       MU  A04        SAUT C34
     SI  003      SI  A03        SI  A14       SI  A14
     ET  B03      SI  B02        MZ  A04       MU  A03
                  SI/ B03
                  ET  A13
                  SI  A04       (à lire colonne après colonne)
                  SI/ B02
                  ET  A14
```

En langage évolué (pascal)

Le pascal est le seul langage qui permette de gérer les entrées−sorties sans avoir besoin de masquages. En effet, grâce aux ensembles (SET OF), voir si un capteur est allumé se réduit à demander si le capteur appartient à l'ensemble des entrées allumées.

```
{ Ce programme correspond au GRAFCET 2 du poly
  "mise en oeuvre du grafcet sur automate" }

PROGRAM grafcet_2 (input,output);

CONST adresse_port=$330;

TYPE liste_capteurs=(e1,e2,e3);
     ensemble_capteurs=SET OF liste_capteurs;
     liste_actions=(sortie1,sortie2,sortie3);
     ensemble_actions=SET OF liste_actions;
```

```
VAR {pour le programme principal}
      etape:array [1..4] of boolean;
      transition:array [1..4] of boolean;
      capteurs:ensemble_capteurs;
      sorties:ensemble_actions;
      i:integer;

  PROCEDURE lire_capteurs(VAR etat_actuel_capteurs:ensemble_capteurs);
  {cette procédure lit les capteurs et rend un ensemble contenant les
   capteurs à 1. Cette procédure dépend du type de machine }

  VAR {locale} etat:record case integer of
                      1: (compatible_port:byte);
                      2: (compatible_ensemble:ensemble_capteurs)
                    end;
  BEGIN
    etat.compatible_port:=port[adresse_port];
    etat_actuel_capteurs:=etat.compatible_ensemble
  END;

  PROCEDURE affecte_sorties(etat_sorties:ensemble_actions);
  {affecte les sorties}
  VAR etat:record case integer of
             1: (compatible_port:byte);
             2: (compatible_ensemble:ensemble_actions)
           end;
  BEGIN
    etat.compatible_ensemble:=etat_sorties;
    port[adresse_port]:=etat.compatible_port
  END;

BEGIN {programme principal}

                                          {initialisation}
  sorties:=[]; {ensemble vide}
  affecte_sorties(sorties);
  etape[1]:=true;
  for i:=2 to 4 do etape[i]:=false;
  REPEAT
                                          {lecture des entrées}
    lire_capteurs(capteurs);

{-----------------------------------------------
    write('capteurs : ');
    if e1 in capteurs then write('E1);
    if e2 in capteurs then write('E2 ');
    if e3 in capteurs then write('E3 ');
    writeln;
-----------------------------------------------}

                                          {conditions d'évolution}
    transition[1]:=etape[1] and (e1 in capteurs);
    transition[2]:=etape[2] and etape[3] and ([e2,e3]*capteurs=[]); {intersection vide}
    transition[3]:=etape[3] and (e2 in capteurs)
                            and not (e3 in capteurs);
    transition[4]:=etape[4] and not (c2 in capteurs);
                                          {désativation}
    if transition[1] then etape[1]:=false;
    if transition[2] then begin
                            etape[2]:=false;
                            etape[3]:=false
                          end;
    if transition[3] then etape[3]:=false;
    if transition[4] then etape[4]:=false;
                                          {activation}
```

```
    if transition[1] then begin
                           etape[2]:=true;
                           etape[3]:=true
                         end;
    if transition[2] then etape[1]:=true;
    if transition[3] then etape[4]:=true;
    if transition[4] then etape[3]:=true;
                                          {affectation sorties}
{---------------------------------------------
    write('étapes : ');
    for i:=1 to 4 do if etape[i] then write(i,' ');
    writeln;
---------------------------------------------}
    sorties:=[];
    if etape[2] then sorties:=sorties+[sortie1];
    if etape[3] then sorties:=sorties+[sortie2];
    if etape[4] then sorties:=sorties+[sortie3];
    affecte_sorties(sorties);
  UNTIL false; {boucler jusqu'à extinction}
END.
```

Méthode locale

Cette méthode est beaucoup plus rapide (à l'exécution), prend beaucoup moins de place, mais ne fonctionne que pour un grafcet à une seule étape active à la fois. De plus l'automate doit pouvoir faire des sauts en avant et en arrière (ce n'est pas le cas des automates d'entrée et moyenne gamme comme le Micro 1, APRIL 15, TSX 17 à 47...).

Principe

Supposons être dans l'étape I, les sorties étant déjà affectées. On attend alors (en fonction des capteurs) que l'on doive quitter l'étape. Puis on choisit quelle doit être la suivante (au cas où l'on avait un OU divergent), on modifie les sorties si nécessaire et on saute à l'étape suivante (qui sera traitée exactement de la même manière).

Exemple simple

On reprend le grafcet (1), avec la même affectation des entrées et des sorties.

Il ne faut plus figer les entrées, il n'est plus nécessaire de représenter les étapes par des variables puisque seule une étape est active, et elle correspond à l'endroit où l'on se trouve dans le programme.

```
initialisation:
        S20<=0;
etape1:
        SI (/E1) SAUT etape1  ;attendre capteur m
        S20<=1              ;mise à jour des sorties etape2:
        SI (/E2) SAUT etape2
        S20<=0
        SAUT etape1
```

Evidement, le programme est plus simple (mais c'est uniquement le cas dans les cas simples). Le programme est le plus rapide qui puisse exister : à un instant donné on ne teste que les capteurs nécessaires, sans aucun autre calcul. Lors d'un évolution on ne modifie que les sorties nécessaires. Le temps de réponse est donc minimal avec cette méthode. Son seul problème est qu'elle ne fonctionne qu'avec des Grafcets à une étape active à la fois (sans ET, tous les OU doivent être exclusifs) (mais voir plus bas pour résoudre ce problème)

mise en oeuvre sur PB 100

Le PB 100 accepte les sauts généralisés, contrairement au PB15

Je ne numérote plus les lignes, mais je leur donne un nom pour que ce soit plus clair (mais il faudrait évidement mettre ces numéros avant d'entrer le programme dans l'automate).

Choix des variables : entrées : m : 000, a: 001. Sortie L : 020,

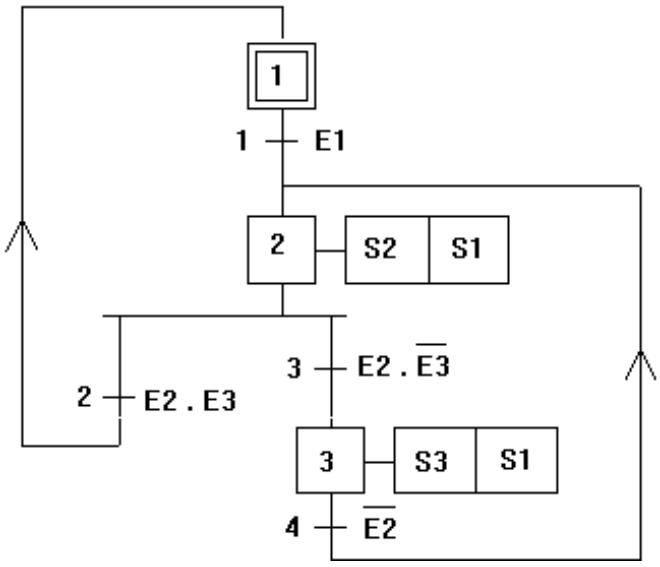
```
     MZ 020
et1  SI/ 000      étape 1 : attendre capteur m (rester dans ces 2 lignes
     SAUT et1               tant que m=0
     MU  020      passage étape 1 à 2 : allumer L puis aller à étape 2
et2  SI/ 001      étape 2 : attendre capteur a
     SAUT et2
     MZ  020      passage 2 à 1 : éteindre L puis aller à étape 1
     SAUT et1
```

Exemple complexe (Grafcet 3)

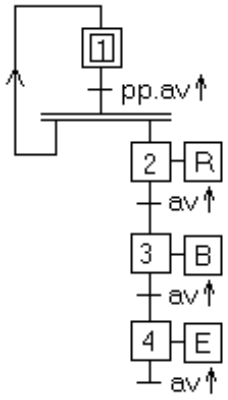
Le grafcet (2) ne convient pas pour cette méthode, il faut d'abord le transformer en un grafcet à une seule étape active à la fois. On fait donc la table des états accessibles puis le graphe des états accessibles :.

étapes actives avant	transition	étapes actives après	num d'état
1	E1	2 3	(1)
2 3	E2.E3 E2./E1	1 2 4	(2)
2 4	/E2	2 3	(3)

on obtient donc le grafcet (3) suivant (il est rare que le graphe des états accessibles soit plus simple que l'initial) :



Les OU divergents DOIVENT être exclusifs (sinon vous avez oublié un cas dans la table). Si vous désirez essayer de créer un graphe des états accessibles, je vous conseille d'essayer celui là :



(c'est le Grafcet du ~~remplissage de bidons~~, dans mon cours sur le ~~Grafcet~~

Vous devez arriver à un Graphe à 8 étapes (tout le monde sait faire un Grafcet à 8 étapes !). Je ne donne pas la réponse ici, ce serait trop facile (n'empêche que j'ai même réussi un jour à le faire sans intersection)

Programe en AF :

```
initialisation :
        S1<=S2<=S3<=0
etape1:
        SI (/E1) SAUT etape1
        S2<=S1<=1
etape2:
        SI (/E2) SAUT etape2  ;seul E2 nécessaire pour sortir de l'étape 2
        S2<=0                 ;dans les 2 cas éteindre S2
        SI (/E3) SAUT passage2a3    ;je traite ce cas plus loin
        S1<=0                 ;dernière sortie à mettre à jour
        SAUT etape1
passage2a3:
        S3<=1                 ;mise à jour sorties
etape3:
        SI (E2) SAUT etape3
        S3<=0                 ;mise à jour sorties, inutile de modifier S1 ici
        S2<=1
        SAUT etape2
```

Pour la mise à jour des sorties, on sait toujours d'où l'on vient et où l'on va, on ne modifie donc que celles qui doivent l'être.

cas du PB100

Choix des variables : entrée Ei : 00i, sortie Si : 02i

```
        DE  021
        MZ  023
et1     SI/ 001
        SAUT et1
p1-2    MU  022
        MU  021
et2     SI/ 002
        SAUT et2
        MZ  022
        SI/ 003
        SAUT p2-3
p2-1    MZ  021
        SAUT et1
p2-3    MU  023
```

```
     et3  SI  002
          SAUT et3
     p3-2 MZ 023
          MU 022
          SAUT et2
```

En assembleur PC (avec MASM ou TASM)

```
; programme en assembleur PC (sous DOS) pour le GRAFCET 3

; pour faire un .COM
    code segment
    assume cs:code,ds:code,es:code
    org 100H

; déclarations de constantes
    adresse_port_e EQU 300H ;c'est l'adresse de mon port d'entrées
    adresse_port_s EQU 301H ;c'est l'adresse de mon port de sorties
    c0 EQU 00000001B      ;capteur E1 et sortie S1
    c1 EQU 00000010B      ;E2,S2
    c2 EQU 00000100B      ;E3,S3

; programme
; je devrais commencer par définir la direction des ports, si ma carte le permettait
s_et1:   mov dx,adresse_port_s
         mov al,0
         out dx,al   ;sorties toutes à 0
         mov dx,adresse_port_e  ;je laisse cette adresse par défaut dans DX

et1:     in al,dx
         test al,c0
         jz et1

s_et2:   mov dx,adresse_port_s
         mov al,00000011B
         out dx,al        ;modif sorties
         mov dx,adresse_port_e

et2:     in al,dx
         test al,c1
         jz et2
         test al,c2
         jnz s_et1

s_et3:   mov dx,adresse_port_s
         mov al,00000101B
         out dx,al
         mov dx,adresse_port_e

et3:     in al,dx
         test al,c2
         jnz et3
         jmp s_et2
fin:     mov ax,4C00h
         int 21h
; fin du programme
    code ends
    end s_et1
```

application en C

```
#define port_e 0x300
#define port_s 0x301
#define e1 0x01 //sur quel bit ai-je branché l'entrée ?
```

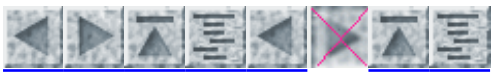
```
#define e2 0x02
#define e3 0x04 //le suivant serait 8 puis 0x10....
#define s1 0x01 //idem sorties
#define s2 0x02
#define s3 0x04

int lecture(void)
  {return(inport(port_e)); }
void ecriture(int i)
  {outport(port_s, i); }

void main(void)
 {
  int capteurs;
etape1:
  ecriture(0);
  do capteurs=lecture(); while (!(capteurs&e1));
etape2:
  ecriture(s1|s2);
  do capteurs=lecture(); while (!(capteurs&e2);
  if (capteurs&e3) goto etape1;
etape3:
  ecriture(s3|s1);
  co capteurs=lecture() while (capteurs&e2);
  goto etape2;
}
```

Remarque sur le masquage : capteurs & e2 fait un ET bit à bit entre les deux variables. Pour qu'un bit du résultat soit à 1, il faut que les bits du même niveau des DEUX variables soient à 1. Or e2 contient un seul bit à 1, celui correspondant à l'entrée E2. Si le résultat vaut 0 (Faux), c'est que E2 était à 0, sinon (différent de 0 mais pas nécessairement 1), c'est qu'il était allumé

En C (comme toujours) le code est compact, facile à écrire mais nécessite une conaissance plus poussée du fonctionnement de l'ordinateur (ici binaire et masquages).

Conclusions

Cette seconde méthode donne un programme est bien plus court dans ce cas, mais en général le graphe des états accessibles est bien plus complexe que le grafcet initial. On se débrouille en général dans ces cas complexes, de manière à minimiser la taille et le temps, en faisant un compromis entre les deux méthodes. Il n'empèche que faire par un programme la table des états accessibles est relativement aisé, et donc on peut en déduire immédiatement le programme résultant (toujours automatiquement) puisque la méthode est très simple (et correspond directement à la table)



Programmation directe en Grafcet

Certains automates sont prévus pour être programmés plus facilement à partir d'un Grafcet. C'est le cas du TSX (à l'aide d'une cartouche ROM) et du PB 15, mais pas du Micro1 par exemple. Mais ces implantations ne sont pas bien pratiques, c'est pourquoi on utilisera généralement (si on en a les moyens) une génération automatique du programme à l'aide d'un logiciel spécifique (en général sur PC).

PB APRIL 15

Chaque étape doit être représentée par une variable interne. On doit utiliser les premières (donc à partir de A00). Les variables internes suivantes peuvent être utilisées pour d'autres utilisations, par exemple pour un calcul de receptivité (on utilise en général les variables en décroissant à partir de A3F).

La première ligne du programme doit comporter l'instruction PRED, qui initialise le Grafcet. Puis on définit toutes les transitions : quelles est l'étape(s) précédente (antérieure), quelle est la variable contenant la receptivité puis quelle est l'étape(s) postérieure. Puis on affecte les sorties, et on boucle (sans bien sur repasser sur PRED qui réinitialiserait).

Exemple du grafcet 2 :

Choix des variables : étape i : A0i, entrée Ei : 00i , sortie Si : 02i

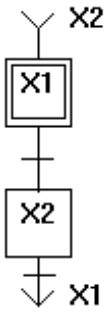
```
0C30 PRED
0C31 EANT A01     étape antérieure 1
     RCEP 001     réceptivité
     EPOS A02     2 étapes postérieures : 2 et 3
     EPOS A03

     EANT A03
     SI  002      calcul receptivité
     SI/ 003
     ET  A3F      mis dans une variable interne
     RCEP A3F
     EPOS A04

     EANT A04
     RCEP /002
     EPOS A03

     EANT A02     2 étapes antérieures
     EANT A03
     SI 003
     SI 002
     ET A3E
     EPOS A01

     SI A02       affectation des sorties
     OU 021
     SI A03
     OU 022
     SI A04
     OU 023

     SAUT C31     boucler apres PRED
```

Le langage vérifie toutes les règles du grafcet, et entre autre accepte plusieurs étapes actives simultanément, ou plusieurs grafcets simultanés (n'oubliez pas dans ce cas de numéroter différement les étapes). Par contre il nécessite une transcription du Grafcet, et n'est pas trop pratique pour le déboguage (quelles étapes sont actives à un instant donné ?)

Dernier point à élucider, PRED : Cet ordre remet les variables internes (et lesregistres de tempos et compteurs) dans le dernier état sauvé. Les variables internes sont mémorisées dans des mots 16 bits aux adresses 0C00 à 0C03. 0C00 contient A00 en bit de poids fort, jusqu'à A0F en bit de poids faible. Dans notre cas, mettre A01 à 1 et tout les autres à 0 revient donc à mettre 4000 (hexa) dans 0C00 et 0000 dans 0C01 à 0C03. On mémorise alors cet état en appuyant sur la touche MEM puis en validant par VAL/+.

sur TSX

Il faut tout d'abord dessiner le grafcet. Analysons le cas du grafcet 1 :



Les traits verticaux vont de haut en bas uniquement. Par contre on peut remplacer un trait par 2 flèches, en précisant l'étape d'où l'on vient et où l'on va. C'est ce que l'on utilise pour une remontée. (ici pour aller de l'étape 2 à l'étape1). Cette notation ne respecte pas les règles du Grafcet (une laison est entre une étape et une transition) et peut nécessiter de rajouter une étape au niveau d'une convergence ou divergence.

Une fois le grafcet entré, on définit les transitions et les actions correspondant aux étapes. Pour ceci, placer le curseur sur une transition à définir, appuyer la touche ZM (zoom). Un réseau à contacts apparaît, avec une bobine représentant la transition. Il faut alors représenter le schéma qui, en fonction des capteurs, "allumera" la receptivité. On valide le réseau par ENT (touche ENTER). Pour les actions, il faut pointer une étape, appuyer ZM, donner le schéma qui allumera les bobines de sortie. On entre un réseau complet, on peut donc faire du combinatoire sur les sorties (test arrêt d'urgence) ou déclencher une tempo. Mais les sorties ne peuvent être activées que par une bobine <S>, ce qui force à désactiver la sortie par une bobine <R> l'étape suivante. Une autre possibilité (que je conseille fortement) est de gérer les actions grâce au "traitement postérieur", voir document complet sur le ~~TSX~~

Soient : Capteurs m=I1,0, a=I1,1 Sortie L=O0,0. Les réseaux à entrer sont donc:

```
transition 1                          transition 2

    |  I1,0                    |          |  I1,1                    |
    |---||----------------< # >--|          |---||----------------< # >--|
    |                           |          |                           |

étape 1                               étape 2

    |                    O 0,0  |          |                    O 0,0  |
    |--------------------< R >--|          |--------------------< S >--|
    |                           |          |                           |
```

L'automate MICRO1

L'automate MICRO 1

P.Trau – 24/3/97

1 – Description générale

L'automate programmable IDEC MICRO1 (distribué par Chauvin Arnoux) est un automate de faible prix (4000F) permettant de traiter des petits problèmes d'automatisme à faible coût (même prix que 10 relais et 3 tempos, chez le même fabriquant). Il se compose d'une unité centrale (UC) comportant le contrôleur, la mémoire (EEPROM : ne s'efface pas quand elle n'est pas alimentée), 8 entrées (extensible à 16 voire 64), 6 sorties (extensible à 12) à relais acceptant du 220V à 2A par commun, c'est à dire 2A totales pour les 3 premières sorties, 2A pour chacune des 3 autres. Le programme est mis au point sur la console de programmation, puis est transféré dans l'UC, qui ne nécessite plus la présence de la console pour fonctionner.

2 – Connexions

Entrées : La borne +COM délivre du 24V–, permettant d'alimenter 8 capteurs (interrupteurs), numérotés 0 à 7.

Sorties : La première borne COM permet d'amener l'énergie de sortie pour les bornes 200, 201 et 202. Les bornes 203, 204 et 205 possèdent chacune sa propre alimentation (borne de gauche). Les différentes alimentations ne sont pas nécessairement identiques (par exemple, une en 220V~, une en 5V–, une en 24V–).

Console : elle est branchée sur la prise située à droite de l'UC (sous une petite trappe). Elle peut être clipsée sur l'UC (glisser vers le haut pour déclipser). Si la console est absente, l'UC, à son allumage, met toutes les variables internes et sorties à 0 puis démarre automatiquement le dernier programme entré (à condition d'avoir ôté la console en mode RUN).

3 – Adresses

- Entrées : 0 à 7
- Sorties : 200 à 205
- Mémoires internes : 400 à 597, sauf les nombres se terminant par 8 ou 9 (400 à 407, 410 à 417,...)
- Mémoires à usage spécifique : 600 à 717 (ex 714 vaut 1 durant 1/2s toutes les secondes, 715 toutes les 0,1s, 713 mis à 1 éteint toutes les sorties...)
- Compteurs : 0 à 46 (45 et 46 spécifiques)
- Temporisations : 0 à 79
- Sorties impulsionnelles : 0 à 95
- Lignes de programme : 0 à 599

4 – Structure du programme

L'automate exécute les lignes du programme de manière séquentielle (l'une après l'autre, en commençant par la ligne numéro 0. Le programme se termine par l'instruction END. Aucune ligne de ce programme, avant le END, ne peut être vide. Au passage sur une ligne de programme, les variables internes sont immédiatement mises à jour, mais les sorties ne sont réellement modifiées que sur le passage de l'instruction END. Comme en informatique, chaque instruction décrit une action à effectuer à l'instant de son exécution uniquement. Ceci implique, pour les cas d'automatismes habituels, où l'état des sorties doit continuellement être recalculé en fonction des entrées, de toujours boucler sur l'ensemble du programme. C'est ce que fait l'instruction END, qui, après la mise à jour effective des sorties, lit l'état des entrées puis remonte au début du programme (ligne 0). Les entrées seront figées le temps d'un cycle du programme (jusqu'au END), ce qui permet d'évaluer des expressions logiques complexes sans qu'une entrée ne change d'état en cours de calcul.

5 – Langage

Le MICRO1 fonctionne en notation polonaise, avec une pile (LIFO) de 8 registres (en cas d'introduction de plus de 8 valeurs dans la pile, les plus anciennes sont perdues).

Les paragraphes 5.1 à 5.6 présentent les fonctions combinatoires de base. Elles permettent de traiter simplement tout problème combinatoire. Les fonctions de base du séquentiel (bascules et timer) sont traitées de 5.7 à 5.9, puis sont traitées des fonctions plus puissantes ou plutôt informatiques (mais aussi plus complexes).

5.1 LOD (load – charger)

Syntaxe : LOD numéro d'entrée, de mémoire interne ou de sortie

Fonction : charge la valeur de son argument au sommet de la pile

Exemple : LOD 0 met au sommet de la pile l'état de l'entrée 0

5.2 OUT (sortir)

Syntaxe : OUT numéro de sortie ou de mémoire interne

Fonction : met la valeur du sommet de la pile dans son argument. La pile reste inchangée

Exemple :
```
LOD 0
OUT 400
OUT 200
```

Ceci met l'état de l'entrée 0 dans la mémoire 400 et sur la sortie 200

5.3 AND (et)

Syntaxe : AND numéro de sortie ou de mémoire interne

Fonction : effectue un ET logique (vrai si les deux entrées sont vraies, faux sinon) entre le sommet de la pile et son argument. Le sommet de la pile est remplacé par le résultat de l'opération.

Exemple :

```
LOD 0
AND 1
OUT 200
```

la sortie 200 s'allume quand l'entrée 0 et l'entrée 1 sont allumées, s'éteint sinon

5.4 OR (ou)

Idem AND, mais effectue un OU logique

5.5 NOT (non)

Syntaxe : Opérateur NOT argument

Fonction : complémente (1 si 0, 0 si 1) l'argument d'un opérateur LOD, AND ou OR

Exemple :

```
LOD NOT 0
AND NOT 1
OUT 200
```

La sortie 200 s'allume quand ni l'entrée 0 ni l'entrée 1 ne sont allumées

5.6 AND LOD / OR LOD

Syntaxe : AND LOD (resp. OR LOD)

Fonction : effectue un ET logique (resp. OU) entre le sommet de la pile et son suivant. Les deux arguments sont supprimés du sommet de la pile et remplacés par le résultat (la hauteur de la pile baisse donc d'un élément).

Exemple :

LOD 1	pile : [1]	(hauteur 1)
AND 2	pile : [1.2]	(hauteur 1)
LOD 3	pile : [1.2] [3]	(hauteur 2)
AND 4	pile : [1.2] [3.4]	(hauteur 2)
OR LOD	pile : [(1.2)+(3.4)]	(hauteur 1)
OUT 200	pile inchangée	

effectue (écriture type "algorithmique") : 200<=(1.2)+(3.4)

Remarque : appuyer sur la touche SHF (shift=majuscule) avant LOD pour ne pas entrer un 1 (qui est sur la même touche).

5.7 SET (allumer)

Syntaxe : SET numéro de mémoire interne, de sortie ou de registre à décalage

Fonction : Si le sommet de la pile vaut 1, met à 1 son argument. Sinon laisse son argument dans son état précédent, qu'il ait été à 1 ou à 0. Le sommet de la pile reste inchangé.

5.8 RST (reset – éteindre)

Syntaxe : RST numéro mémoire ou sortie ou registre à décalage

Fonction : met son argument à 0 si le sommet de la pile vaut 1, le laisse inchangé sinon.

```
Exemple :  LOD 1      (entrée "marche")
           SET 200
           LOD 2      (entrée "arrêt")
           RST 200
```

Si l'entrée 1 est validée, allume la sortie 200 jusqu'à la validation de l'entrée 2. Si 1 et 2 sont à 0, 200 reste dans son état précédent (mémorisation). Si 1 et 2 sont simultanément à 1, 200 est tout d'abord mis (de manière interne) à 1 puis à 0. Au END du programme, la sortie sera effectivement modifiée d'après le dernier état de 200 donc éteinte. C'est une bascule à priorité déclenchement. On obtient une priorité enclenchement en changeant l'ordre des instructions.

5.9 TIM (timer – temporisation)

Syntaxe : TIM numéro de timer (0 à 79)
durée (en 1/10ème de seconde, entre 0 et 9999)

Fonction : à compter du passage à 1 de son entrée (sommet de la pile), met sa sortie à 1 lorsque la durée est écoulée. Lorsque l'entrée est mise à 0, le timer est réinitialisé (compteur de temps remis à 0 et sortie à 0). L'entrée doit donc rester à 1 tout le temps du comptage et ensuite suffisamment pour que l'on ait pu lire la sortie. La sortie peut soit être récupérée au sommet de la pile (ordre OUT par exemple dans l'instruction suivant la durée) soit, n'importe où dans le programme, par lecture de la mémoire interne TIM numéro (LOD TIM numéro, AND TIM numéro ou OR TIM numéro)

```
Exemple :  LOD NOT 200
           TIM 0
           20
           SET 200
           LOD 200
           TIM 1
           30
           RST 200
```

fait clignoter la sortie 200 (allumé 3s, éteint 2s)

Remarque : un timer n'arrête pas le programme (une autre partie de l'automatisme peut fonctionner pendant ce temps) mais met à 1 sa sortie, au bout du temps prédéfini, à condition d'avoir maintenu son entrée à 1 et de repasser à chaque cycle sur l'instruction TIM.

5.10 JMP (jump – saut avant) et JEND (fin de saut)

Syntaxe :

JMP
instructions
JEND
Fonction : si le sommet de la pile vaut 1 au JMP, le programme passe directement au JEND sans effectuer les instructions intermédiaires. Sinon JMP et JEND sont ignorés. Il est impossible d'imbriquer des sauts.

Remarque : On obtient la fonction JEND en appuyant deux fois la touche JMP

5.11 MCS (Master Control Set) et MCR (Master Control Reset)

Syntaxe :

MCS
instructions
MCR

Fonction : Si le sommet de la pile vaut 1 à l'arrivée au MCS, celui–ci est ignoré (le MCR également). Par contre si le sommet de la pile vaut 0, toutes les entrées des instructions suivantes (jusqu'au MCR) seront considéré comme valant 0. Donc un OUT sortira un 0 quel que soit l'état du sommet de la pile, un SET ne changera rien (ne met à 1 que si son entrée est à 1), un TIMer sera réinitialisé, les compteurs et registres à décalage seront figés. Plusieurs MCS peuvent se terminer par un MCR unique.

Exemple :	LOD 0	l'entrée 0 contrôle une grosse partie du programme
	MCS	
	instructions	
	LOD 1	l'entrée 1 contrôle une plus petite partie du programme
	MCS	
	instructions	
	MCR	terminaison des 2 MCS

5.12 SOT (Single OutpuT – sortie impulsionnelle)

Syntaxe : SOT numéro de sot (0 à 95)

Fonction : Si son entrée (sommet de la pile) valait 0 au cycle de programme précédent, et vaut 1 désormais, sa sortie (mise au sommet de la pile) vaudra 1 durant un cycle. Cette fonction ne doit pas être utilisée pour une sortie car le temps d'un cycle est inférieur au temps de réaction des relais de sortie, mais pour des variables internes

Exemple : voir CNT

5.13 CNT (counter – compteur)

Syntaxe : CNT numéro compteur (0 à 44)
valeur finale à atteindre (0 à 9999)

Fonction : la fonction possède deux arguments : empiler en premier son entrée Reset puis son entrée Pulse. A chaque front montant (passage de 0 à 1) du sommet de la pile (Pulse), ajoute 1 au compteur. Quand la valeur finale est atteinte, met sa sortie à 1 (sommet de la pile, et variable CNT numéro utilisable dans LOD CNT numéro, AND CNT numéro ou OR CNT numéro). La sortie reste à 1 jusqu'au passage à 1 de la première entrée (Reset), qui remet également le compteur à 0, et autorise le comptage dès qu'elle passe à 0.

Exemple :

```
LOD 2
SOT 0      création d'un front montant sur l'entrée 2,résultat sur la pile
AND 200    entrée "Reset" : si sortie allumée et front montant sur 2
LOD 2      entrée "Pulse"
CNT 0
5
OUT 200
```

la sortie 200 est allumée après 5 fronts montants sur l'entrée 2, s'éteint au 6ème

Remarque : Le compteur 45 possède une troisième entrée : décomptage. Ceci permet de gérer une quantité de pièces, en comptant à l'entrée et décomptant à la sortie
Le compteur 46 possède une troisième entrée : Up qui, s'il vaut 1 considère l'entrée Pulse (deuxième) comme impulsion de comptage, s'il vaut 0 comme décomptage.
Ces deux compteurs/décompteurs nécessitent également une valeur, mais qui est sera mise dans le compteur lors du Reset, la sortie sera mise à 1 en arrivant à la valeur 0. En décomptant après 0, on passe à 9999.

5.14 Comparateurs (associés aux compteurs CNT)

Syntaxe A: FUN 100+numéro de compteur (donc 100 à 146)
valeur de comparaison

Syntaxe B: FUN 200+numéro de compteur (donc 200 à 246)
valeur de comparaison

Fonction : ne possède aucune entrée. Quand le compteur vaut la valeur (A) ou le compteur est supérieur ou égal à la valeur (B), la sortie (sommet de la pile) vaut 1, sinon la sortie vaut 0. En stockant la sortie dans une mémoire interne (OUT), l'utilisation du LOD NOT permet de traiter les autres cas (différent, inférieur,...)

5.15 SFR (ShiFt Register – registre à décalage)

Syntaxe : SFR numéro du premier bit utilisé (0 à 127)
nombre de bits utilisés (1 à 128)

Fonction : possède 3 entrées, empilées avant l'instruction SFR : Reset, Pulse puis Data (au sommet de la pile). Reset met tous les bits utilisés à 0. A chaque front montant de Pulse, la valeur disponible en Data est entrée dans le premier bit utilisé, l'ancienne valeur du premier est décalée dans le second,..., la valeur du dernier est mise au sommet de la pile. On peut lire l'état de chaque bit du registre à décalage par LOD SFR numéro, AND SFR numéro ou OR SFR numéro. On peut créer plusieurs registres à décalage indépendants parmi les 128 bits disponibles. On peut forcer l'état d'un bit par SET SFR numéro ou RST SFR numéro (mais

OUT SFR n'est pas possible).

Exemple :

```
LOD NOT 400
TIM 0
20
OUT 400          Timer : 400 vaut 1 un cycle toutes les 2 secondes
LOD NOT 597      Reset, uniquement lors du premier cycle
LOD 400          Pulse : un décalage toutes les 2 s
LOD SFR 12       Data : on entre en premier bit l'état du dernier
SFR 10           Notre registre commence en position 10
3                avec 3 bits (jusqu'au 12)
LOD SFR 10
OUT 200
LOD SFR 11
OUT 201
LOD SFR 12
OUT 202          affecter une sortie à chaque bit utilisé du registre
LOD NOT 597      comme toutes les mémoires internes, 597 vaut 0 au premier cycle
SET SFR 10       au premier cycle, mettre le 1er bit à 1
SET 597          mettre 597 à 1 définitivement, pour ne plus réinitialiser
```

Ceci crée un chenillard (sur 3 bits) qui boucle jusqu'à extinction de l'UC

Remarque : SFR peut être utilisé pour gérer un Grafcet linéaire, ou même plus complexe.
SFR NOT (mêmes entrées, même sortie) effectue un décalage en sens inverse. Il peut s'utiliser (et c'est là sa principale utilité) sur les mêmes bits qu'un SFR, permettant d'avancer et reculer.

6 – Entrée d'un programme

- Brancher l'automate, y relier (si nécessaire) la console de programmation.
- Effacer le programme en mémoire de la console **DELT END ENTR**. Toutes les lignes du programme contiennent END, on peut le vérifier par les flèches haut et bas.
- Se placer (si ce n'est pas le cas) en ligne 0 (par flèche haut ou trois fois **CLR**).
- Entrer les lignes du programme, validées par **ENTR**.
- En cas d'erreur :
  - avant validation par **ENTR** : **CLR.**
  - sélectionner une ligne validée à modifier (par flèches), taper la nouvelle ligne, valider par **ENTR.**
  - pour insérer une ligne, se placer sur la ligne N, entrer la nouvelle instruction puis **INST**, l'ancienne est décalée en N+1.
  - pour supprimer une ou N ligne(s), se placer sur la première à effacer, appuyer **DELT** puis le nombre de lignes à effacer puis **ENTR**. les lignes suivantes sont remontées.
  - pour accéder rapidement à une ligne, **ADRS** puis numéro de ligne puis **READ**. Ou bien taper une instruction (ligne complète, avec son argument) puis **READ** : recherche la prochaine ligne contenant cette instruction.
- Transférer le programme de la console à l'UC :
  - mettre l'UC en mode STOP (interrupteur sur la console, voyant RUN de l'UC éteint).
  - appuyer **TRS ENTR** puis confirmer par **ENTR**. Attendre l'affichage de END
  - mettre l'UC en mode RUN. On peut désormais débrancher la console, l'UC continue à

fonctionner.
Si l'UC est éteinte en mode RUN, à son rallumage le programme débutera automatiquement (au bout de quelques secondes), avec toutes les mémoires mises à 0.

La console, contrairement à l'UC, perd le contenu de sa mémoire lorsqu'elle est débranchée. On peut recharger un programme dans la console depuis une UC par **TRS READ** puis confirmation par **ENTR**. **TRS VERI** permet de vérifier la similitude du programme de la console avec celui de l'UC.

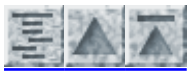
7 – Monitoring

En plus de la visualisation sur l'UC de l'état des entrées et des sorties, on peut visualiser, en mode RUN, l'état de l'UC par les fonctions MON :

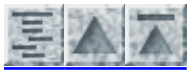
MON numéro mémoire **READ** : affiche l'état des 8 mémoires à partir du numéro donné. MON 0 pour les entrées, MON 200 pour les sorties, MON 400 pour les mémoires internes 400 à 407, les flèches haut et bas permettent de visualiser les mémoires suivantes. Une mémoire à 1 est visualisée par un rectangle plein, 0 par un vide. **MON TIM** numéro permet de visualiser le contenu d'un timer, **MON CNT** numéro pour un compteur, **MON SFR** numéro pour les registres à décalage. **CLR** quitte le mode Monitoring. Les changements d'état de moins de 0,1s ne sont pas visualisés. Remarque : MON TIM 47 affiche le temps d'un cycle du programme en mémoire (en ms).

On peut modifier la valeur limite d'un compteur ou timer par **TRS CNT** (ou **TIM**) numéro **READ**. La valeur est affichée. Entrez la nouvelle valeur puis **ENTR**. Cette valeur est perdue à l'extinction de l'UC (plus d'un jour). **TRS SET** numéro mémoire **ENTR** met à un une mémoire (RST, SET SFR, RST SFR sont également possibles).

FUN 93 **READ** 1 **ENTR** : affiche une ligne de programme, et l'état de sa sortie (ex LOD entrée à 1 ou LOD NOT entrée à 0 affichent un rectangle plein). On peut se déplacer dans le programme par les flèches ou par ADRS. On quitte ce monitoring par 0 puis **ENTR**.



~~P. TRAU~~, ~~ULP–IPST~~, 26/3/97



Description succincte du TSX

Description succincte du TSX

P.TRAU, 24/3/97.

Les fonctions de base d'un automate

Un automate programmable permet de remplacer une réalisation câblée comportant des composants combinatoires (portes) et séquentiels (bascules, séquenceurs,...) par un programme. Un programme est une suite d'instructions, qui sont exécutées l'une après l'autre. Si une entrée change alors qu'on ne se trouve pas sur l'instruction qui la traite et que l'on ne repasse plus sur ces instructions, la sortie n'est pas modifiée. C'est la raison de la nécessité de bouclage permanent sur l'ensemble du programme.

Par rapport à un câblage, on a donc deux désavantages : temps de réponse (un changement des entrées sera pris en compte au maximum après le temps d'un passage sur l'ensemble du programme, c'est ce qu'on appelle le temps de scrutation, qui sera souvent de l'ordre de la milliseconde) et non simultanéité (on n'effectue qu'un instruction à la fois). Mais ces temps étant en général très inférieurs aux temps de réaction des capteurs et actionneurs (inertie d'un moteur par exemple), ceci n'est que rarement gênant. L'avantage est que c'est programmable, donc facilement modifiable.

Tout automate programmable possède :

- des entrées, des sorties, des mémoires internes : toutes sont binaires (0 ou 1), on peut les lire (c.a.d connaître leur état) (même les sorties), mais on ne peut écrire (modifier l'état) que sur les sorties et les mémoires internes. Les mémoires internes servent pour stocker des résultats temporaires, et s'en resservir plus tard.
- des fonctions combinatoires : ET, OU, NON (mais aussi quelquefois XOR, NAND,...)
- des fonctions séquentielles : bascules RS (ou du moins Set et Reset des bascules), temporisations, compteurs/décompteurs mais aussi quelquefois registres à décalage, etc...
- des fonctions algorithmiques : sauts (vers l'avant mais aussi quelquefois saut généralisés), boucles, instructions conditionnelles...
- de plus il permet de créer, essayer, modifier, sauver un programme, quelquefois par l'intermédiaire d'une console séparable et utilisable pour plusieurs automates. Désormais cette fonctionnalité est également possible sur PC, permettant une plus grande souplesse, une assistance automatique, des simulations graphiques,... mais pour un prix supérieur.

Ce qui différencie les automates, c'est la capacité (entrées, sorties, mémoires internes, taille de programme, nombre de compteurs, nombre de temporisations), la vitesse mais surtout son adaptabilité (possibilité d'augmenter les capacités, de prendre en compte de l'analogique et numérique, de converser via un réseau...)

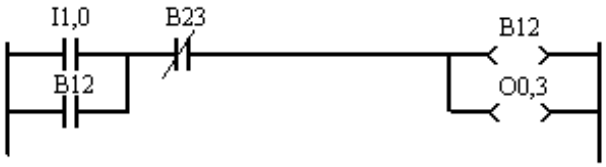
Le langage à contacts du TSX

C'est le langage de base des TSX. Il est nécessaire de le connaître même pour utiliser le langage PL7−2 (proche du Grafcet).

Sur le TSX, les sorties sont appelées Oi,0 à Oi,23 (i=numéro de carte d'entrée), les entrées Ii,0 à Ii,24. Les variables internes sont notées en décimal de B0 à B255 (B pour Bit interne ou Bobine).

La programmation se fait à l'aide de programmes graphiques : les réseaux. Ce sont des schémas qui sont exécutés l'un après l'autre, de haut en bas (et non suivant leur label). Chaque réseau est scruté par colonne de gauche à droite.

exemple :



Dans ce cas l'entrée B12 est l'ancienne valeur du bit interne (bobine) B12. Si l'on veut utiliser le résultat B12 de ce réseau, il faut utiliser B12 dans le réseau suivant.

On note un capteur par le signe , un contact complémenté (vrai si 0) par .

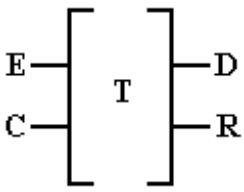
Un bit interne est notée , un bit interne inverse (commandée par un niveau 0).

Une bascule bistable est allumée par , éteinte par .

Un saut à un autre réseau est noté . Un saut est effectué immédiatement lors de son évaluation (les bobines en sortie dans le même réseau mais sur les lignes suivantes ne seront donc pas évaluées en cas de saut). On a intérêt de n'utiliser que des sauts avants (vers la fin du programme). L'exécution du dernier réseau sera automatiquement suivie de l'exécution du premier (sauf si sauts). L'automate fixe automatiquement les entrées au début de cycle et n'affecte les sorties qu'en fin de cycle (les variables internes sont évidement immédiatement modifiées). Il est nécessaire de refaire un cycle (c'est à dire passer du dernier réseau au premier) fréquemment (tous les 150 ms maximum).

Temporisation

On représente la tempo par le signe :

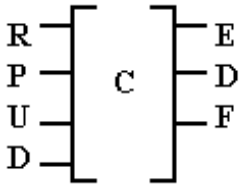


Il existe sur les TSX 17 32 tempos (T0 à T31). E correspond à l'armement de la tempo, C au contrôle. D passe à 1 en fin de tempo, R est à 1 tant que la tempo est en cours. En appuyant la touche ZM, on peut donner : TB: la base de temps (1mn, 1s, 100ms, 10ms), PRESET: la durée (0 à 9999).

E doit être à 1 tout le temps du comptage, son passage à 0 met D à 0 et réinitialise le compteur. C (que l'on peut brancher sur E) valide le comptage (si C=0;, le compteur est bloqué mais pas remis à 0)

On dispose également de 8 tempos monostables M0 à M7, avec une seule entrée S, une seule sortie R valant 1 à durant le temps présélectionné, partir du front montant de S, indépendamment du moment de passage à 0 de S. Un nouveau front montant de S en cours de comptage relance le compteur à 0.

Compteur / décompteur



Il existe sur les TSX 17 32 compteurs (C0 à C31). R (reset) met le compteur et les sorties à 0. P (preset) met le compteur à la valeur finale et la sortie D (done) à 1 (sauf si R=1). U (up) incrémente le compteur, D (down) le décrémente. La sortie F (full) vaut 1 lors du passage du compteur (par U) de 9999 à 0, E (empty) lors du passage (par D) de 0 à 9999. Si U=D=1, le compteur est inchangé.

La valeur de préselection (Ci,P, entre 0 et 9999) se définit en "zoomant" sur le compteur.

Les autres fonctions disponibles (comparateurs, opérations arithmétiques et logiques, piles, registres à

décalage, transcodage binaire, BCD, ASCII...) sont détaillées dans le chapitre 5 du document "Terminal TSX T407 Modes opératoires PL7−2", au chapitre 2 de "Langage PL7−2 Synthèse" ainsi qu'au chapitre B4 du manuel de référence du PL7−2.

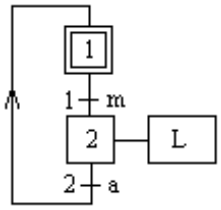
Conclusion

Ces fonctions de base (tempo, compteur) sont présentes dans tous les automates (même si elles sont mises en oeuvre par des langages très différents), sauf les sauts qui peuvent être plus limités (au minimum bouclage automatique sur l'ensemble du programme, mais sans sauts dans le programme). Mais le principe reste valable quel que soit l'automate. Souvent, d'autres possibilités existent, en particulier temporisations, comptage, comparaisons,...

Programmation directe en Grafcet

Certains automates sont prévus pour être programmés plus facilement à partir d'un Grafcet. C'est le cas du TSX (à l'aide d'une cartouche ROM) mais pas du MICRO1.

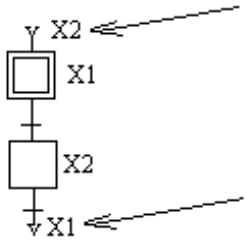
Il faut tout d'abord dessiner le Grafcet. Analysons le cas du Grafcet suivant :



choix des adresses et variables internes

- entrées :
  - m : entrée I1,0
  - a : entrée I2,0
- sortie L : O0,0

programme :



On vient de l'étape 2

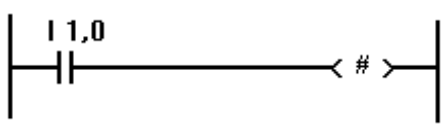
Les liaisons verticales vont de haut en bas uniquement. Par contre on peut remplacer une liaison par 2 flèches, en précisant l'étape d'où l'on vient et celle où l'on va. C'est ce que l'on utilise pour une remontée.

On va à l'étape 1

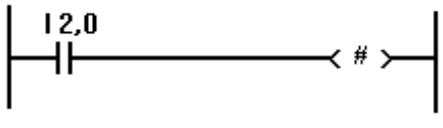
Une fois le grafcet entré, on définit les transitions et les actions correspondant aux étapes. Pour ceci, placer le curseur sur une transition à définir, appuyer la touche ZM (zoom). Un réseau à contacts apparaît, avec un bit interne représentant la transition. Il faut alors représenter le schéma qui, en fonction des capteurs, "allumera" la réceptivité. On valide le réseau par ENT (touche ENTER). Pour les actions, on peut (mais je ne le conseille pas) pointer une étape, appuyer ZM, donner le schéma qui allumera les bobines de sortie. Sur nos TSX, les sorties ne peuvent être activées que par un bit interne <S>, ce qui force à désactiver la sortie par un bit interne <R> l'étape suivante.

Soient : Capteurs m=I1,0, a=I2,0 Sortie L=O0,0. Les réseaux à entrer sont donc:

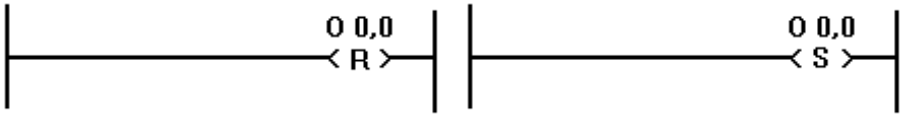
transition 1



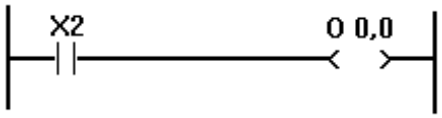
transition 2



étape 1 étape 2

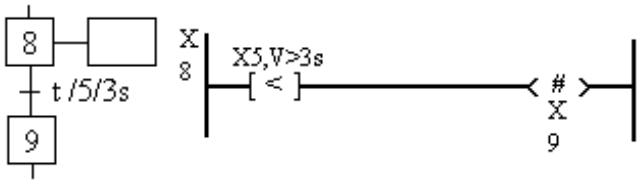


Une bien meilleure solution est de regrouper toutes les actions dans le "traitement postérieur". Attention, du fait que le TSX fige les entrées−sorties le temps d'un cycle, il ne faut mettre en place qu'une seule "équation" par sortie (sinon seule la dernière sera prise en compte). On n'oubliera donc pas de regrouper (en parallèle) les Xi allumant une sortie.



Une ~~**tempo**~~ (en secondes) est automatiquement liée à chaque étape, et permet de tester la durée depuis l'activation de l'étape par un opérateur de type −[<]− (comparaison) par la variable interne Xi,V (i numéro d'étape)

exemple de transition comportant une tempo :



Détails pratiques

Ce chapitre précise l'utilisation des claviers et les branchements à effectuer.

Allumez l'automate puis la console (touche ON). Mettez l'automate en mode STOP (sur la platine portant l'automate). Quand le menu principal est affiché, choisissez PRG (program) en appuyant la flèche qui se trouve sous cette option. Vous pouvez alors soit examiner le programme en mémoire, soit visualiser la suite des options du menu par la flèche sous le −/−. On choisit alors l'option CLM (clear memory), qui nous demande en quel langage on veut travailler (SEQ pour grafcet, LAD pour contacts). Puis après quelques secondes, on peut revenir au programme principal par la touche QUIT (qui remonte brutalement) ou 2 fois CLR (qui remonte avant la dernière commande effectuée).

Pour entrer le programme, choisir PRG puis SEQ ou LAD. On valide un réseau ou tout le programme par la touche ENT.

Pour faire tourner le programme, revenez au menu principal (QUIT), choisissez DBG (debug) puis R/S (run/stop), ou mettez en RUN par le contacteur RUN/STOP de la platine.. Quand le programme tourne, on peut revenir sous PRG pour le voir, mais pas le modifier. Les capteurs et les bobines sont alors représentées en pointillés si à 1, en trait plein si à 0.

Les modifications se font en revisualisant le programme sous PRG, choisir le bon réseau ou partie de Grafcet et choisir l'option MOD. En général, une modification nécessite d'effacer le capteur ou trait existant (touche SH + DEL) et remettre le nouveau, dans d'autres cas (traits horizontaux par exemple), on efface une entité en lui superposant une entité identique.

Un cycle de programme en langage Grafcet peut être précédé par un programme en langage à contacts (ladder) appelé traitement préliminaire, et suivi d'un taitement postérieur. La scrutation des entrées se faisant avant le traitement préliminaire, on peut y traiter des conditions sur les entrées préliminaires ou effectuer une

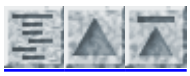
partie d'un calcul au cas où un réseau ne suffirait pas pour une réceptivité. Le traitement postérieur se fait avant l'affectation des sorties, et peut donc modifier des sorties avant leur affectation définitive. Ces traitements peuvent utiliser l'état du Grafcet (par l'intermédiaire des bits Xi). On peut également les utiliser pour implanter le GEMMA (modes de marche et d'arrêt) sans modifier le Grafcet. A la mise en route de l'automate, tous les bits internes (sauf indication contraire) sont mùis à 0, sauf les étapes initiales.

Description des menus (utiles) sur la console T407

- ~~Menu principal [TSX 17−20]~~
  - ADJ (adjust) permet de visualiser ou modifier toute variable.
  - DBG (debug) : mise au point : permet de visualiser le programme et voir l'état des capteurs, sorties, étapes actives... (trait plein dans le programme si actif, interrompu si 0) et mettre des points d'arrêt dans le programme.
  - PRG : créer ou modifier le programme.
  - TRF (transfert) pour mémorisation sur EEPROM et impression sur imprimante (RS232).
- ~~Menu PRG (dans tous les cas)~~
  - CLM (clear memory) efface le programme actuel, permet de définir si le nouveau programme sera en langage à contacts (LAD) ou Grafcet (SEQ).
  - CNF (config) configuration de l'automate, de la liaison RS232 pour l'imprimante (LINE), des bobines à sauvegarder même en cas de coupure de courant (SAV)...
  - NAME permet de donner un nom au programme.
  - LK vérifie si le programme en mémoire ne comporte pas d'erreur.
  - FREE retasse le programme (à faire après de nombreuses modifications).
- ~~Menu PRG en mode ladder (LAD)~~
  - TOP aller au premier réseau
  - BOT (bottom) aller après le dernier réseau (on passe ensuite au dernier par la flèche vers le haut)
  - LAB : donner un numéro de réseau (label) puis [ENT] pour le visualiser
  - INS insère un nouveau réseau (vide) devant le réseau actuel.
  - DEL (delete) supprime le réseau actuel.
  - SCH (search) permet de rechercher tous les réseaux comportant une bobine ou contact donné.
  - [ZM] (zoom) visualise l'adresse d'un contact ou bobine (exemple I1,2), on peut se déplacer dans le réseau par les flèches.
  - [CLR] (clear) retour au niveau supérieur (ZM−>LAD−>PRG−>principal)
  - [Quit] retour direct au menu principal.
- ~~en mode ZOOM (sous PRG en mode LADDER)~~
  - LAB donner au réseau actuel un numéro de label (0 à 999)
  - " " donner un commentaire au réseau actuel (15 caractères maxi, sera affiché au dessus du réseau).
  - MOD permet de modifier l'élément pointé (l'effacer par [DEL] par exemple), on valide le réseau modifié par [ENT].
- ~~Menu PRG en mode GRAFCET~~

  On dispose de 8 pages (0 à 7) de 14 lignes de 8 colonnes. On peut au maximum prendre en compte 96 étapes, les divergences et convergences sont limitées à 4 voies. L'écran ne montre qu'une petite partie de la page, mais le numéro de page (P), de ligne (L) et de colonne (C) sont toujours affichés. On se déplace par les flèches, ou en tapant P, L, C ou X (étape) suivi du numéro désiré. Les fonctions sont approximativement les mêmes qu'en mode ladder, hormis :
  - DLP : effacement d'une page complète
  - [ZM] face à une transition, la définit (si réseau vide, réceptivité toujours fausse)
  - [ZM] face à une étape, définit son action (étape d'attente si réseau vide)
  - MOVE : déplace l'élément actuel (par les flèches) puis valider par [ENT]
- ~~Menu DBG~~

- R/S passe de RUN à STOP et inversement (on peut aussi utiliser le contacteur sur la platine).
- PRG : visualiser le programme et l'état des variables (trait plein=1, pointillé=0), insertion de points d'arrêt.
- CY/ : exécution cycle par cycle
- STP : liste des étapes actives
- /L point d'arrêt sur un label, /o sur une étape.
- S/L et S/o : blocage sur un label ou une étape.



P. TRAU, ULP–IPST, 26/3/97