REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Mohammed Seddik Benyahia -Jijel

Faculté des Sciences et de la Technologie

Département d'Automatique

Cours

Automates Programmables Industriels

Spécialité : Licence 3 Automatique

Présenté par l'enseignant :

Dr. Sofiane Doudou

Table des matières

Introduction

Description générale

Ce cours est destiné aux étudiants de niveau licence 3 option automatique. Il s'adresse également aux personnes voulant s'initier aux techniques d'automatisation de processus industriels.

Objectifs

L'automatique est un ensemble de théories, de techniques, de composants utilisés pour rendre les machines autonomes, qui fonctionnent tout seul ou sans intervention humaine. L'intervention de l'automatique dans les systèmes à événements discrets s'appel automatisme. Automatisme est partout dans notre environnement quotidien par exemples : les distributeurs automatiques, les ascenseurs, le montage automatique dans le milieu plusieurs industriel, les feux de croisement…etc.

Les objectifs poursuivis par une automatisation peuvent être assez variés. On peut retenir quelques uns :

- La réduction des coûts de production par réduction des frais de main-d'œuvre, d'économie de matière, d'économie d'énergie,…
- L'amélioration de la qualité du produit lié à la précision des actionneurs,
- Remplacer l'homme dans des travaux pénibles et dangereux et l'amélioration des conditions de travail (Réduction des accidents de travail).
- L'amélioration de la disponibilité des produits.
- La réalisation d'opérations impossibles à contrôler manuellement

La commande des systèmes automatisés industriels est assurée par un ensemble électronique programmable s'appel l'Automate Programmable Industriel (noté API en abrégé). Ces API sont apparus à la fin des années soixante pour remplacer avantageusement les technologies de la réalisation des parties commandes par les relais électromagnétiques et par les séquenceurs pneumatiques.

Ce cours à pour but principal d'initier les étudiants à la programmation des API afin de piloter le fonctionnement et de faire un suivi d'un système automatisé.

Plan de cours

L'organisation de ce cours comporte cinq chapitres :

Dans la première partie de, on donne des généralités sur les systèmes automatisé et on définit les différents types de commande de ces systèmes.

Dans la deuxième partie, on présente les notions de base de la modélisation des systèmes automatisés par les réseaux de Petri (Rdp) et on définit les différentes propriétés d'un Rdp.

L'objet du troisième chapitre est de présenter l'outil graphique GRAFCET. Cet outil permet à l'automaticien de décrire graphiquement l'évolution des systèmes automatisés.

La présentation des automates programmables industriels est proposée en quatrième chapitre.

Le dernier chapitre est dédié à la présentation des différents langages utilisés pour la programmation des API. Une grande importance est donnée à la programmation en langage à contact (Ladder).

Généralités sur les systèmes automatisés

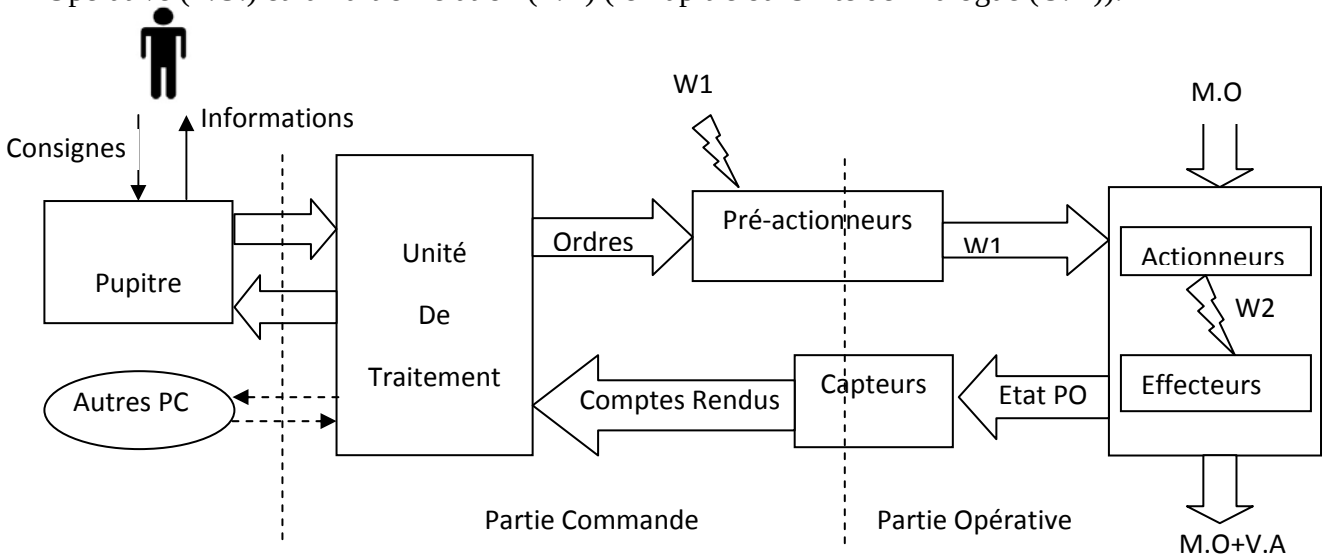
I.1. Introduction

Un système automatisé est un ensemble d'éléments organisés pour réaliser de manière autonome des opérations, qui exigeaient auparavant l'intervention humaine, dans un but précis : agir sur une matière d'œuvre afin de lui donner une valeur ajoutée. Ces opérations s'exécutent après avoir reçu les consignes d'un opérateur comme les ordres de départ et si besoin d'arrêt.

Ce chapitre permet de présenter les différentes parties d'un système automatisé : partie opérative, partie commande et partie relation. Il permet également de définir les différentes fonctions de la chaine d'énergie et la chaine d'information. En suite, la commande des systèmes automatisés par la logique câblée et la logique programmée, est exposée brièvement. En fin, le chapitre donne une idée générale sur les différents niveaux du cahier de charge.

I.2. Différentes parties des systèmes automatisés

Un Système Automatisé est composé de trois parties : la Partie Commande (P.C.), la Partie Opérative (P.O.) et la Partie Relation (P.R) (le Pupitre ou Unité de Dialogue (U.D)).



W1: Energie du réseau, **W2**: Energie utile, **M.O**: Matière Oeuvre, **V.A**: Valeur Ajoutée.

Figure I.1. Structure globale d'un système automatisé.

La **P.O**. est en général mécanisée, c'est elle qui assure la conversion de puissance et agit sur le processus automatisé ou sur la matière d'œuvre (M.O).

La **P.C.** traite les informations envoyées par les capteurs et par la P.R afin d'envoyer des ordres qui vont être exécutés par la P.O. par l'intermédiaire des pré-actionneurs. Elle envoie également des messages de communication vers l'opérateur et les autres systèmes.

Le **P.R.** est l'organe servant d'interface Homme Machine (HMI). L'opérateur est amené à donner des ordres (consignes) à la P.C. (par bouton, clavier, etc...) et reçoit en retour des informations (par voyants, indicateurs, alarme, Ecran, etc…).

Le tableau suivant montre les différents éléments d'un système automatisé.

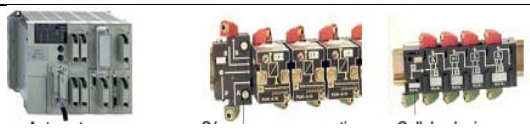
Eléments	Exemples	Commentaires
PRE-ACTIONNEUR	 Variateu Contacteur Distributeur	Distribuent l'énergie aux actionneurs à partir des ordres émis par la PC.
ACTIONNEUR	 Moteur Vérin	Convertissent l'énergie qu'ils reçoivent des pré-actionneurs en une autre énergie utilisée par les effecteurs. Ils peuvent être Pneumatiques, Hydrauliques ou Electriques
EFFECTEUR	Fraise, Foret, Mors d'étau, pince de robot…. 	Les effecteurs sont multiples et variés et sont souvent conçus spécialement pour s'adapter à l'opération qu'ils ont à réaliser sur la Matière d'œuvre. Ils reçoivent leur énergie des actionneurs.
CAPTEUR		Renseignent la PC sur l'état de la PO. Ils peuvent détecter et mesurer des grandeurs physiques. Ils Peuvent être électriques ou pneumatiques. Signaux du type TOR, Analogique ou Numérique.
TRAITEMENT	 Automates (API) Séquenceurs	Dans les systèmes modernes, l'API assure de plus en plus cette fonction. Certains systèmes purement pneumatiques peuvent être contrôlés par des séquenceurs ou des fonctions logiques
DIALOGUE		L'unité de dialogue permet à l'opérateur d'envoyer des consignes à l'unité de traitement et de recevoir de celle-ci des informations sur le déroulement du processus.

Tableau I.1. Différents éléments d'un système automatisé.

I.3. Chaîne fonctionnelle d'un système automatisé

Une chaîne fonctionnelle est un sous-ensemble d'un système automatisé. On peut distinguer au sein des systèmes deux chaînes, l'une agissant sur les flux de données, appelée chaîne d'information (Chaîne d'acquisition), l'autre agissant sur les flux de matières et d'énergies, appelée chaîne d'énergie (Chaîne d'action)

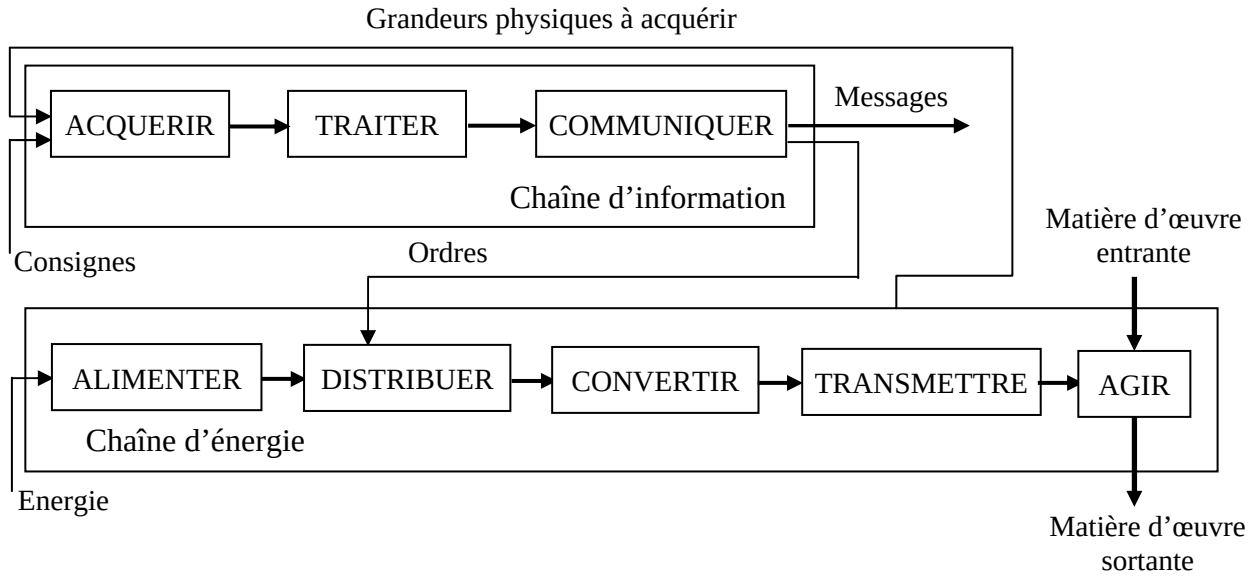


Figure I.2. Chaîne d'information et chaîne d'énergie d'un système automatisé.

I.3.1. Chaîne d'information

La chaîne d'information est composée des fonctions suivantes : Acquérir des informations, Traiter ces informations et communiquer les informations traitées.

- **Acquérir :** Les capteurs et les interfaces homme/machine (Bouton poussoir, bouton coup de poing, potentiomètre, interrupteur de position, clavier...etc) permettent de prélève des informations sur le comportement de la partie opérative ou du milieu extérieur et d'envoyer des consignes au système, respectivement. Les informations capturées et les consignes sont transformées en des informations exploitables (généralement de nature électrique ou pneumatique car ils seront portées par un support physique) par la partie commande.

- **Traiter :** L'unité de traitement est le plus souvent dans l'industrie une carte électronique ou un automate programmable industriel (API). C'est un ensemble électronique qui gère et assure la commande du système automatisé en fonction des informations recueillies par les capteurs et boutons.

- **Communiquer :** Les interfaces machine/homme (voyant, alarme sonore, écran….) permettent à l'utilisateur d'être informé sur l'état du système et les réseaux) réalisent une communication avec d'autres systèmes.

I.3.2. Chaîne d'énergie

La chaîne d'énergie assure la réalisation d'une fonction dont les caractéristiques sont spécifiées dans le cahier des charges. Elle est constituée des fonctions génériques : Alimenter, Distribuer, Convertir, Transmettre qui contribuent à la réalisation d'une action (Agir) sur la matière d'œuvre.

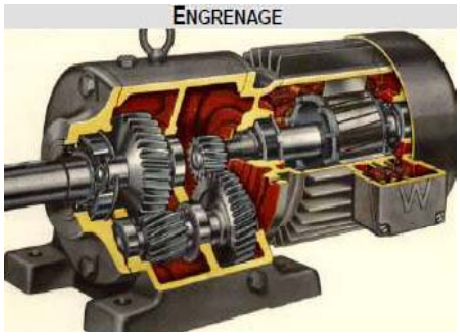
- **Alimenter :** Elle fournit au système l'énergie (électrique, pneumatique, hydraulique) dont il a besoin pour fonctionner. L'énergie électrique est générée par des batteries ou des piles. Elle peut-être produite par des réseaux électrique ou à partir d'énergie solaires ou du vent. Pour les systèmes qui nécessitent d'autres énergies comme de l'air comprimé il faut utiliser des compresseurs ou des pompes.

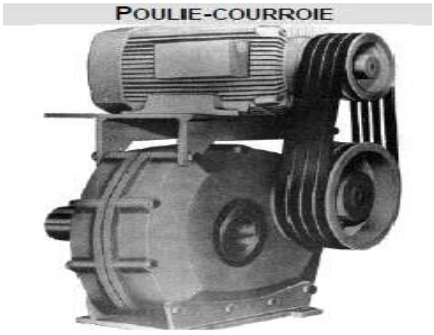
- **Distribuer :** En général, l'énergie l'ordre issue de la partie commande n'est pas suffisante pour être utilisable directement par les actionneurs. Alors le rôle du pré-actionneur est de distribuer ou non une énergie importante en attente, sous l'action d'une énergie de commande plus faible. Les pré-actionneurs les plus utilisés sont des contacteurs ou des relais si l'énergie est électrique. Si l'énergie est pneumatique, les pré-actionneurs sont des distributeurs.

➢ **Convertir :** La fonction des actionneurs, est de convertir une énergie de puissance provenant du pré-actionneur en une énergie adaptée à l'exécution de la tâche à effectuer. En général les actionneurs sont des moteurs (l'énergie électrique est convertie en une énergie mécanique de rotation) ou des vérins (l'énergie pneumatique est convertie en une énergie mécanique de translation).

➢ **Transmettre :** A la sortie de l'actionneur, l'énergie mécanique est disponible. Cependant, pour un grand nombre de systèmes, cette énergie mécanique n'est pas directement utilisable par l'effecteur. Alors il faut transmettre cette énergie à l'effecteur. Cette fonction peut être se réalisée en plusieurs actions :

- Transmettre l'énergie à l'effecteur par exemple par les engrenages et les poulies-courroies.
- Transformer l'énergie mécanique de rotation en translation ou vice-versa : cette fonction est réalisée par exemple par les guidages de rotation, les guidages de translation et les freins.
- Adapter l'énergie : cette fonction est réalisée par exemple par les systèmes vis-écrou.





➢ **Agir :** Les effecteurs sont les éléments terminaux de la chaîne d'action. Ils agissent directement sur la matière d'œuvre en vue de lui apporter une valeur ajoutée. Ils convertissent l'énergie reçue de l'adaptateur en une opération ou un effet sur la matière d'œuvre.

I.4. Différents types de commande

Les systèmes automatisés sont constitués d'un ensemble d'opérations élémentaires effectuées sans intervention humaine excepté l'ordre de marche. Ces systèmes se trouvent en deux types: combinatoires et séquentiels

Les systèmes automatisés combinatoires sont des systèmes dont les sorties dépendent des entrées seulement. Chaque combinaison des variables d'entrée fait correspondre un seul état de sortie. L'outil mathématique utilisé pour décrire les systèmes combinatoires est l'algèbre de Boole qui est basée sur la construction de la table de vérité et l'établissement des équations logiques des actionneurs par l'utilisation des règles de simplification comme le tableau de Karnaugh.

Par opposition, les sorties à l'instant t des systèmes séquentiels sont déterminées par les entrées à cet instant et les états antérieurs de ce système, ce qui implique qu'une même combinaison d'entrée ne générera pas toujours les mêmes sorties. Ces systèmes sont les plus répandus dans le domaine industriel. Plusieurs approches sont utilisées pour faire la synthèse des systèmes en vue de concevoir la partie commande. Parmi elles, on peut citer : la méthode de Huffman, le GRAFCET et les réseaux de Pétri.

Deux techniques sont utilisées pour mise en œuvre la partie commande synthétisé : la logique câblée et la logique programmée. Le choix d'une logique dépend de plusieurs critères : complexité, coût, évolutivité et la rapidité.

I.4.1. Logique câblée : Commande électronique ou pneumatique

La logique câblée est utilisée si le fonctionnement du système est prédéfini, figé, et simple (machine à laver par exemple). L'élément principal s'appelle module séquenceur et l'association de modules constitue un ensemble appelé séquenceur. Le séquenceur peut être électronique (portes et bascules logiques) ou pneumatique (distributeur).

- **Avantage**

- Automatisme simple et rapide à mettre en œuvre

- **Inconvénients**

- Volume du contrôleur proportionnel à la complexité du problème.
- Des modifications de la commande impliquent des modifications de câblage.

1.4.2. Logique programmée : commande électrique

La logique câblée est choisie 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). La programmation est réalisée directement en différent langages à l'aide d'une console de programmation (actuellement on utilise un PC, ce qui permet de saisir et simuler l'automatisme au calme, avant le test in situ). L'élément principal de cette technique s'appelle l'Automate Programmable Industriel ou l'API. Le pilotage des actionneurs se fait par l'intermédiaire de relais.

- **Avantages**

- Souplesse et adaptabilité de l'installation (Remplacement des fonctions combinatoires et séquentielles par un programme).
- Facilité de modification de la loi de contrôle: il suffit de modifier le programme.
- Simplification de la maintenance.
- Solution plus compacte : Faible liaison entre le volume matériel et la complexité du problème (effet simplement sur les entrées / sorties et taille mémoire).

- **Inconvénients**

- Vitesse inversement proportionnelle à la complexité du problème. Ceci peut être une limitation pour des processus électromécaniques rapides.
- Le coût est plus cher.

I.5. Cahier des charges

C'est un contrat entre le fournisseur (le concepteur-fabricant de l'automatisme) et le client (utilisateur de l'automatisme) définissant de façon exhaustive ce que le commanditaire attend de la réalisation d'un produit ou d'un service. Il est composé de plusieurs niveaux, trois au moins, correspondant aux différents aspects du projet :

a. Premier niveau

Il correspond aux spécifications fonctionnelles qui décrivent l'automatisme par son fonctionnement couplé au processus, indépendant de la technique utilisée (électronique, à relais, hydropneumatique). Ces spécifications doivent permettre au concepteur de l'automatisme de comprendre le rôle (les fonctions), de définir les actions à effectuer et leur enchainement séquentiel.

b. Deuxième niveau

Il correspond aux spécifications techniques et aux spécifications opérationnelles qui précisent la manière dont se font les échanges d'informations et d'énergies entre l'automatisme et le processus.

Les spécifications techniques énumèrent et expliquent tous les renseignements sur la nature et les caractéristiques physiques des capteurs, des actionneurs et des circuits électroniques (les contraintes, les performances, les limitations). On y ajoute les conditions d'environnement (température, humidité, parasites électrostatiques ou électromagnétiques….etc.) et les conditions de fiabilité pour assurer du fonctionnement.

Les spécifications opérationnelles précisent d'une manière très pratique la mise en ouvre (réalisation pratique) de tout l'automatisme dans les circonstances de sa vie, c'est.-à-dire., assurent la facilité de modifications, dépannage et entretien, mode de marche et d'arrêt, absence de pannes dangereuses.

c. Troisième niveau

Il concerne la documentation décrivant le fonctionnement, l'utilisation, l'entretien et le dépannage du matériel. Cette documentation doit être constamment à jour au fur et à mesure de l'évolution des modifications. Ainsi les opérations de maintenance ou de dépannage peuvent-elles être simplifiées.

d. Autres considérations

Elles ne sont pas techniques mais sont très importantes : clauses juridique (responsabilités, accidents, pénalités,...), conditions commerciales (prix, garanties, …) ou financières.

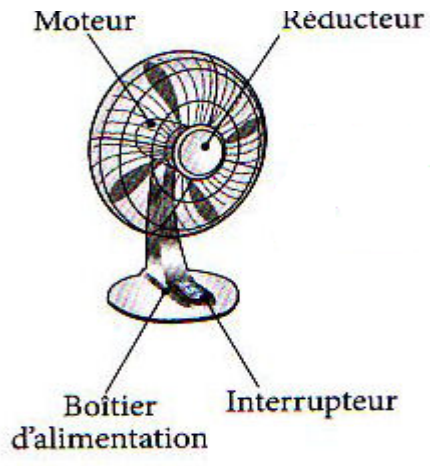
I.6. Exercices

Exercice 01

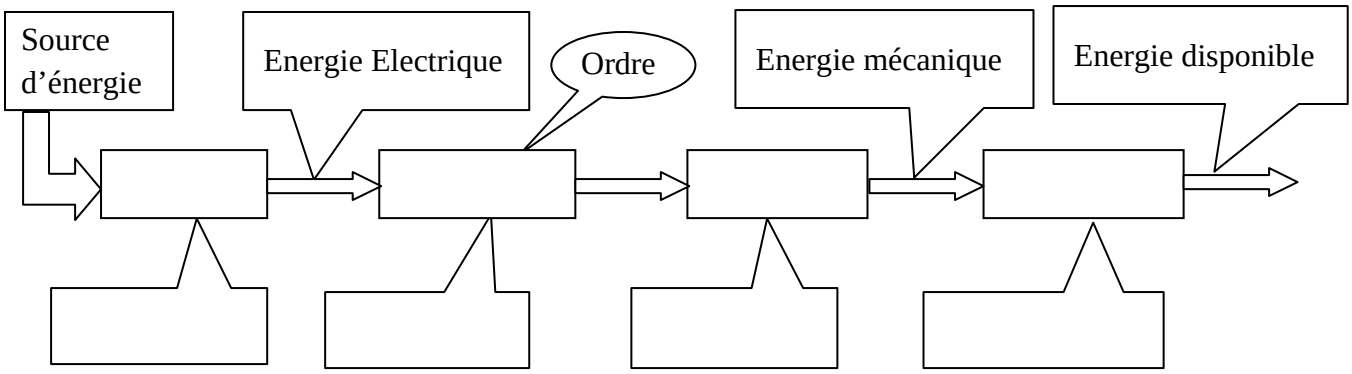
Soient les systèmes techniques suivants « Presse orange électrique », « Téléviseur avec support » et " Ponceuse vibrante"

1. Quelles sont les matières d'œuvre entrantes et sortantes pour chaque système technique ?
2. Quel est le type d'énergie utilisé par chaque système ?

Exercice 02



Compléter la chaine d'énergie de la perceuse et du ventilateur.

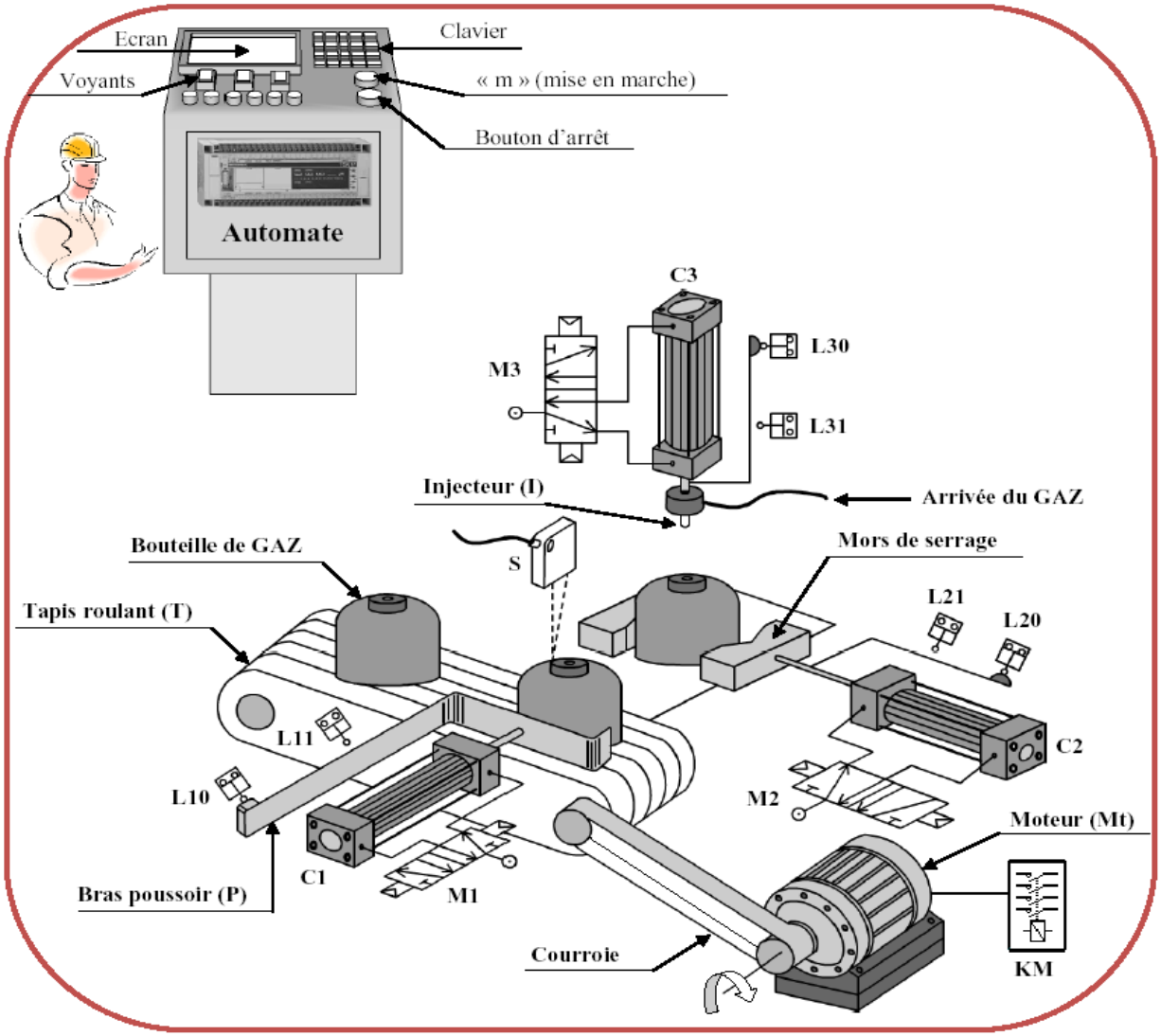


Exercice 03 Système technique : " Unité de remplissage automatique de bouteille de gaz "

I- FONCTIONNEMENT

L'appui sur le bouton (m) de mise en marche provoque le départ du cycle de la façon suivante :

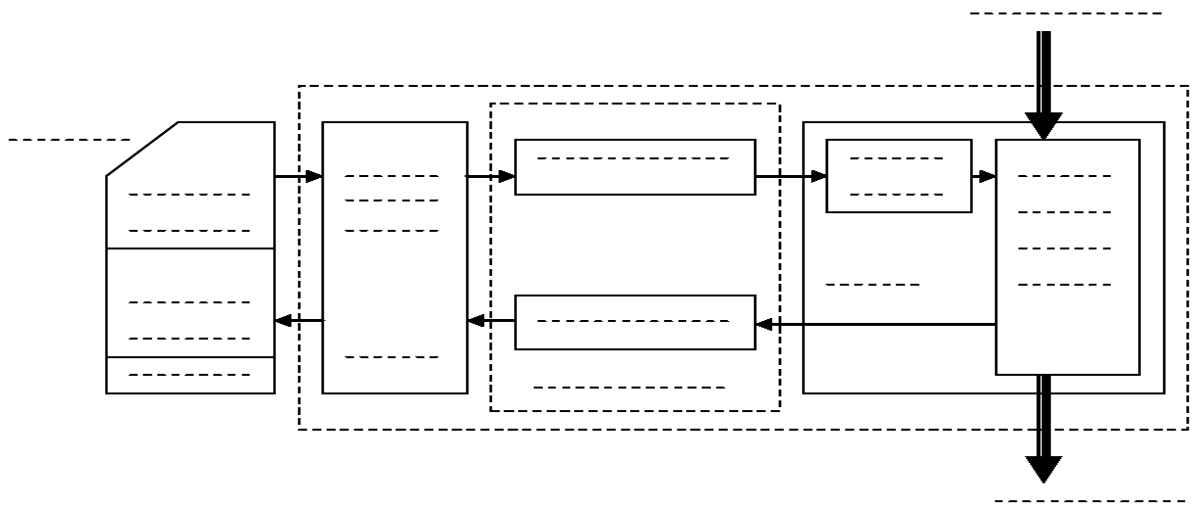
- L'amenée de la bouteille de gaz vide par le tapis (T) devant le bras poussoir (P).
- La poussée de la bouteille sous l'injecteur (I) de gaz par le bras poussoir (P).
- Le serrage de la bouteille réalisé grâce au vérin (C2).
- L'injection du gaz dans la bouteille par l'injecteur (I) donc la bouteille devient pleine.
- Desserrage de la bouteille.



1- Identifier la partie commande de ce système (P.C) :……
2- Identifier les éléments de sa partie opérative (P.O) :
Actionneurs : ……...
Effecteurs : ……..
3- Identifier les éléments d'interfaces de ce système :
Pré- actionneurs : ……..
Capteurs :…….
4- Compléter le tableau par les termes suivants :
Capteur à contact – Distributeur – Contacteur – Capteur sans contact.

Eléments	Désignation
S	
L31	
M1	
KM	

5- Compléter la chaîne fonctionnelle de ce système :



Réseaux de Petri

II.1. Introduction

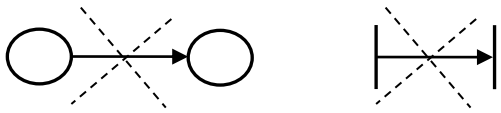
Les réseaux de Petri constituent un outil graphique et mathématique qui permet de décrire les relations existant entre des conditions et des événements et de modéliser le comportement de systèmes dynamiques à événements discrets. C'est Carl Adam Petri qui a inventé ce formalisme en 1962.En 1972-1973, cet outil est utilisé pour la description d'automatismes logiques, ce qui a débouché sur le GRAFCET. Les principaux utilisateurs de ces réseaux sont les informaticiens et les automaticiens.

L'objectif de ce chapitre est de donner les notions de base de la modélisation par les réseaux de Petri. Les différentes règles d'évolution et les différentes propriétés, des réseaux de Petri sont présentées par la suite. En fin, les abréviations et les extensions sont exposés brièvement .

II.2. Concepts de base

Un réseau de Petri est un modèle défini par :

- Un ensemble de places, $P = \{P_1, P_2, \dots, P_n\}$, notées graphiquement par des cercles et représentant des conditions qui traduisent les états d'une ressource du système modélisé (machine libre, une machine est au repos, une machine est en réparation, une commande est en attente, stock vide, convoyeur à l'arrêt, …etc.).
- Un ensemble de transitions (d´événements), $T = \{T_1, T_2, \dots, T_n\}$, notées graphiquement par des barres ou des rectangles et représentant l'ensemble des événements (les actions se déroulant dans le système : début de traitement sur une machine, panne sur une machine, début de traitement d'une commande) dont l'occurrence provoque la modification de l'état du système.
- Un ensemble d'arcs, notés par des flèches qui joignent les places aux transitions et les transitions aux places. ces arcs ont par défaut à un poids égal à 1. Ainsi les situations suivantes sont interdites.



- Un nombre m_i de jetons dans les places. On appelle M le vecteur des marquages défini par les m_i, $M = \{m_1, m_2, \dots, m_n\}$.

Exemple (l'atelier de coupe de bois)

Un atelier est constitué d'une machine de coupe et d'un stock. Quand une commande arrive et que la machine de coupe est disponible, la commande peut être traitée (découpe). Une fois le traitement terminé, la commande qui a été traitée est stockée. Sinon, la commande doit attendre que la machine de coupe se libère avant de pouvoir être traitée.

Conditions :	Evénements :
La machine de coupe est libre (P1) ; Une commande est en attente (P2) ; La commande est traitée (P3); La commande est en stock (P4)	Arrivage d'une commande (T1); Traitement de la commande (T2) ; Stockage de la commande (T3).

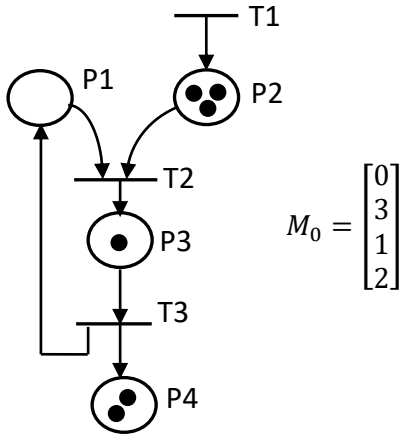


Figure II.1. Réseau de Petri de l'atelier de coupe de bois.

- On dit que la place P3 est en amont ou est une entrée de la transition T3.
- On dira que la place P3 est en aval ou est une sortie de la transition T2.

➢ **Cas particuliers**

- Transition source pas de place en entrée de la transition.
- Transition puits pas de place en sortie de la transition.

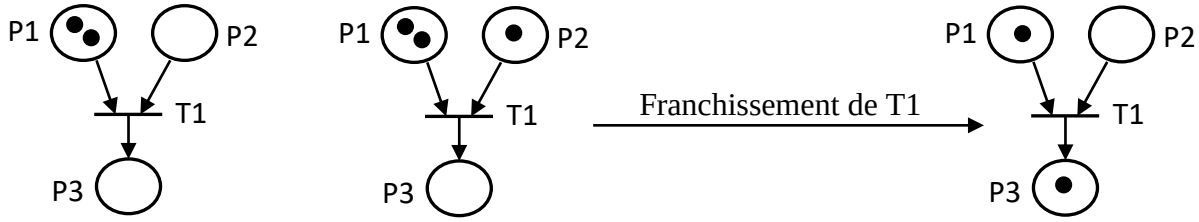
(a) (b)

Figure II.2. Transition: (a) source, (b) puits.

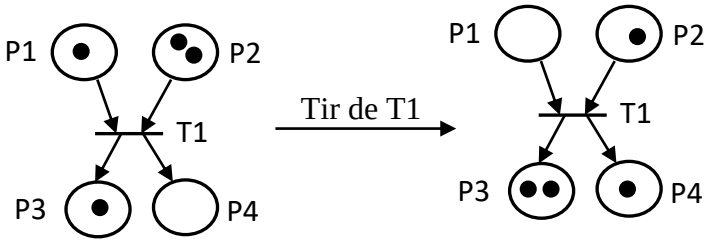
II.3. Franchissement de transition

Les règles générales d'évolution des réseaux de Petri marqués simples sont les suivantes :

- Le franchissement (Tir) d'une transition ne peut s'effectuer que si la transition est franchissable.
- Une transition est franchissable ou sensibilisée ou encore validée lorsque chacune des places en amont possède au moins un jeton.



Transition non validée. Transition validée.



Transition validée

Figure II.3. Exemples de franchissement d'une transition.

- Une transition source est par définition toujours validée.
- Le réseau ne peut évoluer que par franchissement d'une seule transition à la fois, transition choisie parmi toutes celles qui sont validées à cet instant.
- Le franchissement d'une transition est indivisible et de durée nulle (sauf dans les RdP temporisés).
- Le franchissement d'une transition consiste à retirer un jeton dans chacune des places en amont de la transition et à ajouter un jeton dans chacune des places en aval de celle-ci.

II.4. Évolution des réseaux de Petri

L'évolution d'un RdP peut être décrite par un graphe de marquage représentant l'ensemble des marquages accessibles et d'arcs correspondant aux franchissements des transitions faisant passer d'un marquage à l'autre pour un marquage initial M0.

➢ **Exemple**

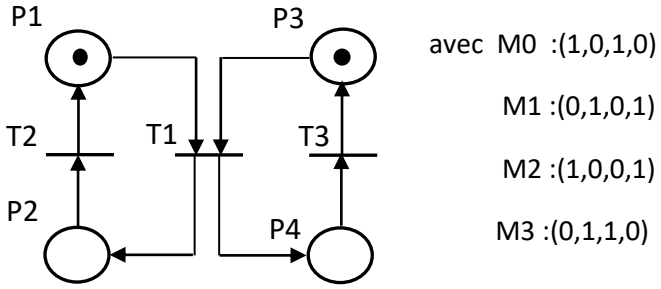


Figure II.4. Exemple d'un graphe de marquage.

➢ **Notations et définitions**

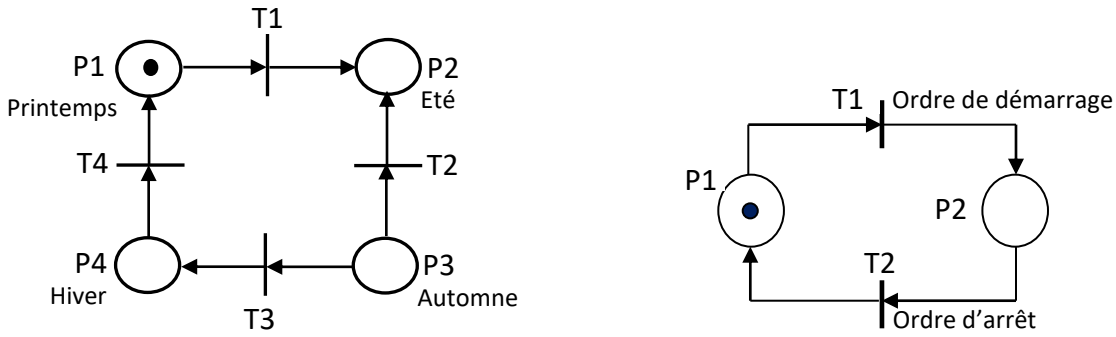
- Le passage du marquage **Mk** au marquage **Ml** par franchissement de la transition Tj est noté: **Mk(Tj > Ml**. **Exemple** : M0(T1 > M1 ; M1(T2 > M2 ; M3(T2 > M0.
- Le nombre de marques dans la place Pi pour le marquage Mk est noté **Mk(Pi)**.
  Exemple : M0(P1)=1 ; M1(P2)=1 ; M3(P3)=0.
- L'ensemble des marquages accessibles à partir du marquage M0 est l'ensemble des marquages obtenus à partir de M0 par franchissements successifs d'une ou plusieurs transition(s). Cet ensemble est noté ***M0**. **Exemple :** *M0={M0,M1,M2,M3}.
- Une séquence de franchissement (noté **S**) est une suite de transitions qui sont successivement franchies pour passer d'un marquage à un autre. **Exemple :** M0(S> M2 avec S=T1T2 ; M0(S> M3 avec S=T1T3 ; M0(S> M0 avec S=T1T2T3+T1T3T2.
- Un marquage **Mk** couvre un marquage **Ml** (noté **Mk** $\geq$ **Ml**). si, pour chaque place, le nombre de marques de Mk est supérieur ou égal au nombre de marques de Ml : $\forall i, Mk(Pi) \geq Ml(Pi)$.

La couverture est stricte si de plus (noté **Mk** $> Ml$) : $\exists m, Mk(Pm) > Ml(Pm)$.

II.5. Réseaux de Petri autonomes et non autonomes

Un RdP autonome décrit le fonctionnement d'un système qui évolue de façon autonome, c'est à dire, dont les instants de franchissement ne sont pas connus, ou pas indiqués. (Exemple : Le cycle des 4 saisons).

Un RdP non autonome décrit le fonctionnement d'un système dont l'évolution est conditionnée par des événements externes ou par le temps (Exemple : Démarrage d'un moteur). Un RdP non autonome est Synchronisé et/ou Temporisé.



(a) (b)

Figure II.5. RdP: (a) autonomes, (b) non autonomes.

II.6. Structures particulières des réseaux de Petri

a. Graphe d'états

Un RdP est un graphe d'états si est seulement si toute transition a exactement une place d'entrée et une place de sortie.

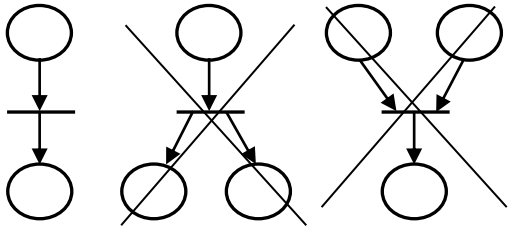


Figure II.6. Graphe d'états ou pas.

b. Graphe d'événements

Un RdP est un graphe d'événement si et seulement si toute place a exactement une transition d'entrée et une transition de sortie.

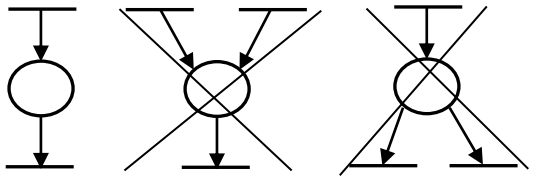
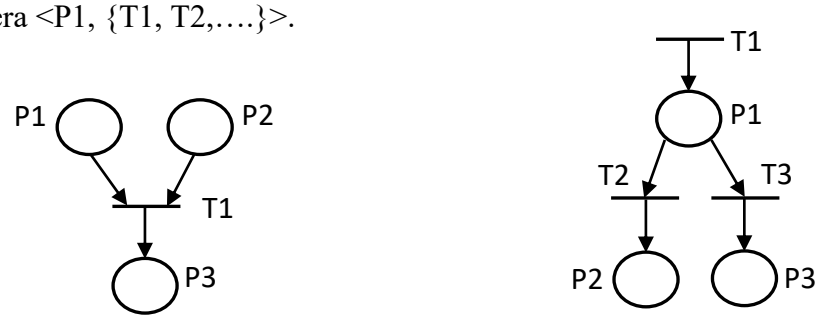


Figure II.7. Graphe d'événements ou pas.

c. RdP sans conflit

Un RdP dans lequel toute place a au plus une transition de sortie. Un conflit (ou conflit structurel) correspond à l'existence d'une place P1 qui a au moins deux transitions de sortie T1, T2,…On notera <P1, {T1, T2,….}>.



(a) (b)

Figure II.8. RDP : (a) Sans Conflit (b) Avec conflit< P1,{T2,T3}>.

d. RdP à choix libre

Un RdP à choix libre est un RdP dans lequel pour tout conflit <P1, {T1, T2,….}> s'il exisite, aucune des transitions T1, T2,…ne possède une autre place d'entrée que P1.

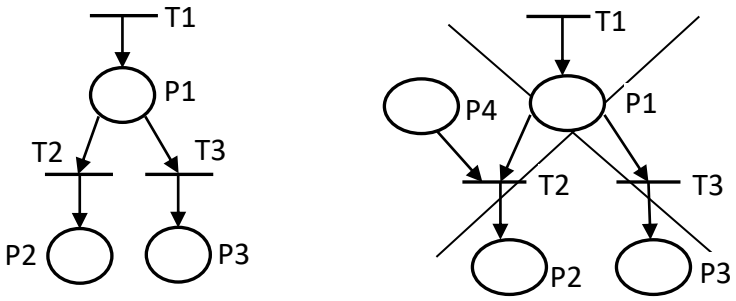
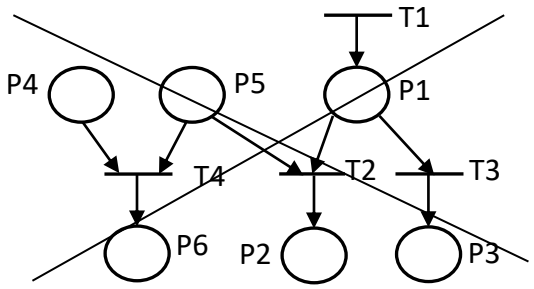


Figure II.9. RdP à choix libre ou non.

e. RdP simple

C'est un RdP dans lequel chaque transition ne peut être concernée que par un conflit au plus. S'il existe une transition T1 et deux conflits <P1, {T1, T2,…}> et <P2, {T1, T3,…}>, alors le RdP n'est pas simple.



T2 est concernée par deux conflits < P1,{T2,T3}> et < P5,{T2,T1}>.

Figure II.10. RdP n'est pas simple.

Remarque

- L'ensemble des RdP simples inclut l'ensemble des RdP à choix libre, qui inclut l'ensemble des RdP sans conflit, qui inclut lui-même l'ensemble des graphes d'événements.
- L'ensemble des graphes d'états est inclus dans l'ensemble des RdP à choix libre.

f. RdP pur

C'est un RdP dans lequel il n'existe pas de transition ayant une place d'entrée qui soit également place de sortie de cette transition. Dans le cas contraire on parle de RdP impur.

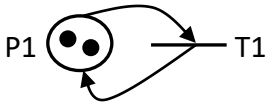


Figure II.11. Rdp impur.

II. 7. Modélisation par réseau de Petri

Les RdP permettent de modéliser un certain nombre de comportements importants dans les systèmes : le parallélisme, la synchronisation, le partage de ressources, la mémorisation et la lecture d'information, la limitation d'une capacité de stockage.

a. Parallélisme

Le parallélisme représente la possibilité que plusieurs processus évoluent simultanément au sein du même système. On peut provoquer le départ simultané de l'évolution de deux processus à

l'aide d'une transition ayant plusieurs places de sortie. Il est ensuite possible de synchroniser l'achèvement des deux processus.

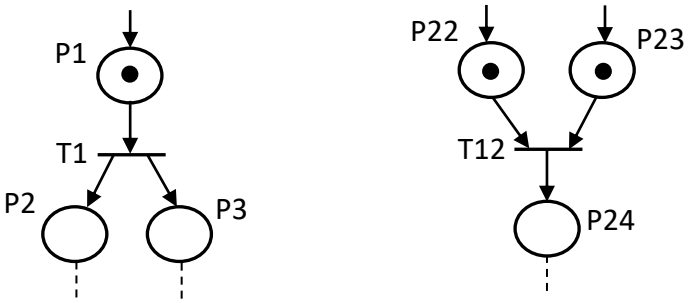
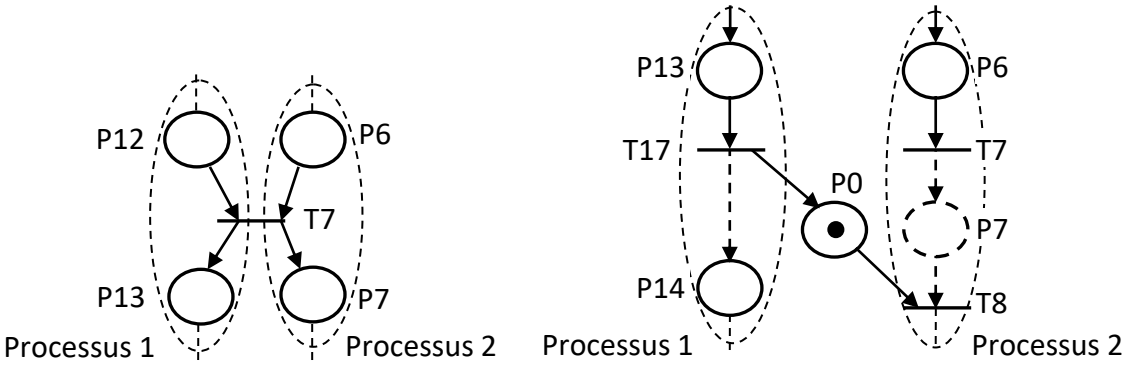


Figure II.12. Parallélisme.

b. Synchronisation

- **Mutuelle** : La synchronisation mutuelle ou rendez-vous permet de synchroniser les opérations de deux processus.
- **Sémaphore** : Les opérations du processus 2 ne peuvent se poursuivre que si le processus 1 a atteint un certain niveau dans la suite de ses opérations. Par contre, l'avancement des opérations du processus 1 ne dépend pas de l'avancement des opérations du processus 2.



(a) (b)

Figure II.13. Synchronisation : (a) Synchronisation mutuelle, (b) Sémaphore.

c. Partage de ressource

Cette structure va modéliser le fait qu'au sein du même système plusieurs processus partagent une même ressource.

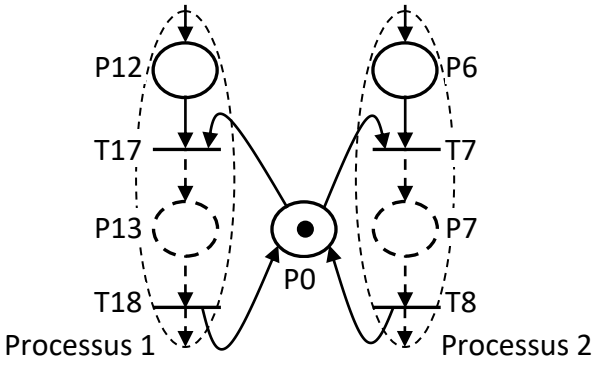
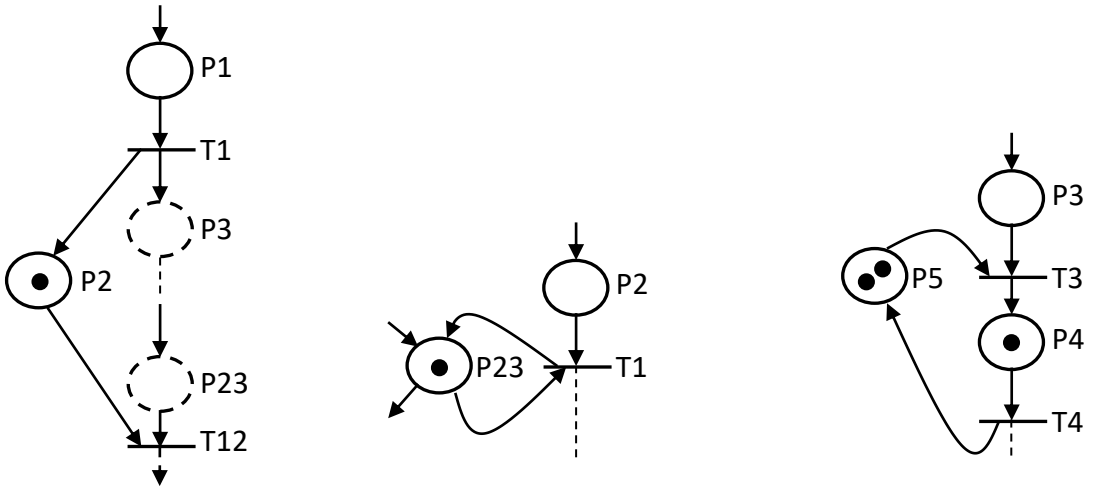


Figure II.14. Partage de ressource.

d. Mémorisation

– du franchissement d'une transition, c'est-à-dire de l'occurrence d'un événement.
– d'un nombre, par exemple de la quantité d'une ressource donnée dans un stock.



(a) (b) (c)

Figure II.15. Structures :(a) Mémorisation, (b) Lecture, (c) Capacité limitée.

e. Lecture

Le franchissement de la transition T1 est li´e au marquage de P23. Lors du franchissement, son marquage n'est pas modifié. On fait alors "une lecture" de ce marquage.

f. Capacité limitée

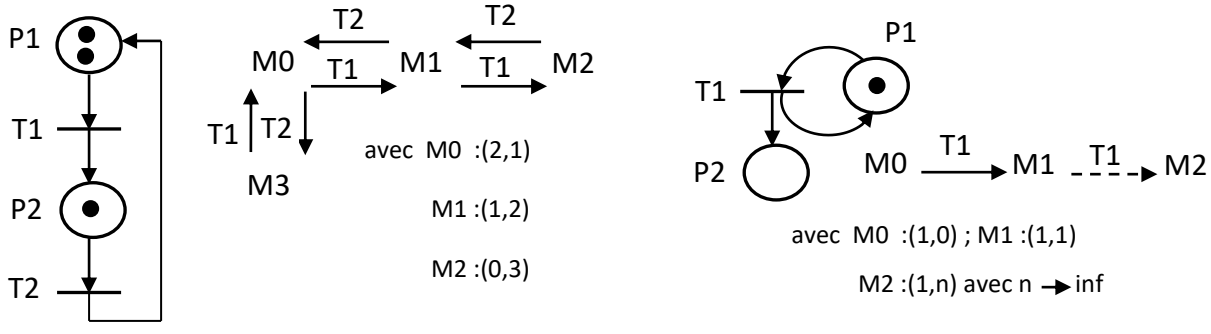
Cette partie de RdP peut modéliser un stock de capacité total égale à 3 : d'où le nom capacité limitée.

II.8. Propriétés des réseaux de Petri ordinaires

Un RdP ordinaire est un RdP autonome marqué fonctionnant selon les règles prédéfinies.

a. RdP borné et Réseau sauf

- Une place Pi est dite bornée pour un marquage initial M0 si pour tout marquage accessible à partir à partir de M0 (noté *M0) le nombre de jetons dans Pi est fini.
- Un RdP est borné pour un marquage initial M0 si toutes les places sont bornées pour M0.
- Si pour tout marquage M appartenant .000à l'ensemble des marquages *M0, on a M(Pi) ≤K où K est un nombre entier naturel, on dit que Pi est K-borné. Si la propriété est vraie pour toute place on dit que ce RdP est K-borné.



(a) (b)

Figure II.16. Rdp :(a) borné, (b) non borné.

- **RdP sauf** ou binaire est un RdP 1-borné. **Exemple** : le Rdp des 4 saisons.

b. Vivacité

- Une transition Tj est quasi-vivante pour un réseau R avec un marquage M si elle peut être validée au moins une fois. C'est-à-dire, il existe une séquence de franchissement à partir M qui contient Tj.

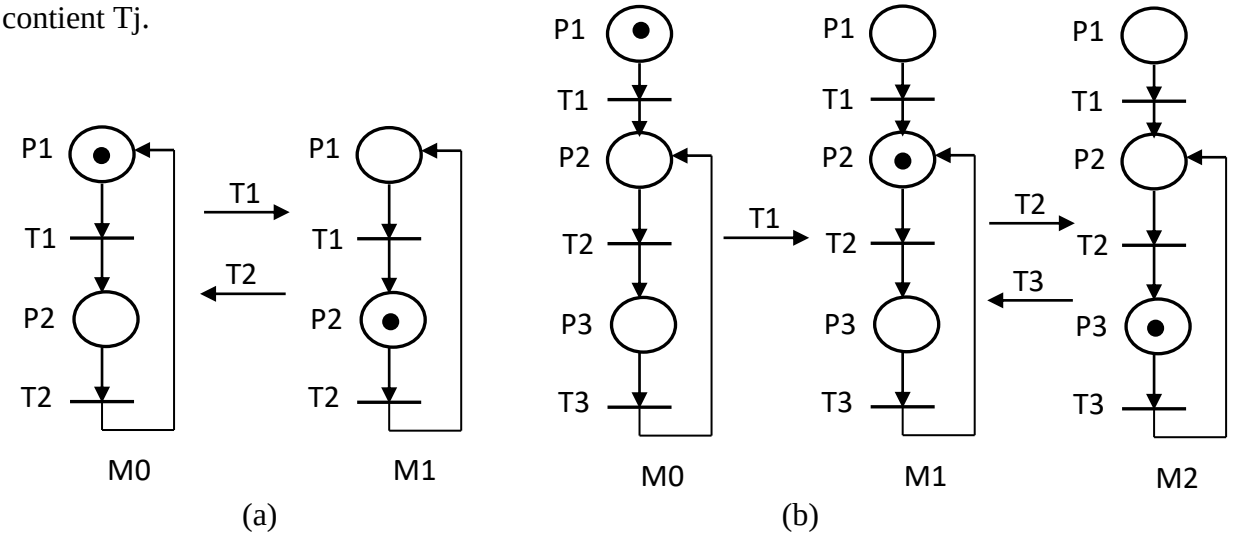


Figure II.17. Rdp :(a) vivant, (b) non vivant (T1 n'est pas vivante).

- Un réseau R avec un marquage M est quasi-vivant si toutes ses transitions sont quasi-vivantes.
- Une transition Tj est vivante pour un marquage initial M0 si pour tout marquage accessible Mi ∈*M0, il existe une séquence de franchissement S qui contient Tj à partir de Mi.
- Un RdP est vivant pour un marquage initial M0 si toutes ses transitions sont vivantes pour M0 (c-à-d aucune transition ne sera jamais définitivement infranchissable).

Propriétés

- Si la transition Tj est quasi vivante pour un marquage initial M0 alors elle est quasi vivante pour **Mk ≥ M0.**
- Si la transition Tj est vivante pour un marquage initial M0 alors elle n'est pas nécessairement vivante pour **Mk ≥ M0**.
- Un RdP est dit conforme s'il est sauf et vivant.

c. Blocage

- Un blocage (ou état puits) est un marquage tel qu'aucune transition n'est validée.
- Un RdP est dit sans blocage pour un marquage initial M0 si ∀ marquage accessible Mi ∈*M0, il est sans blocage.

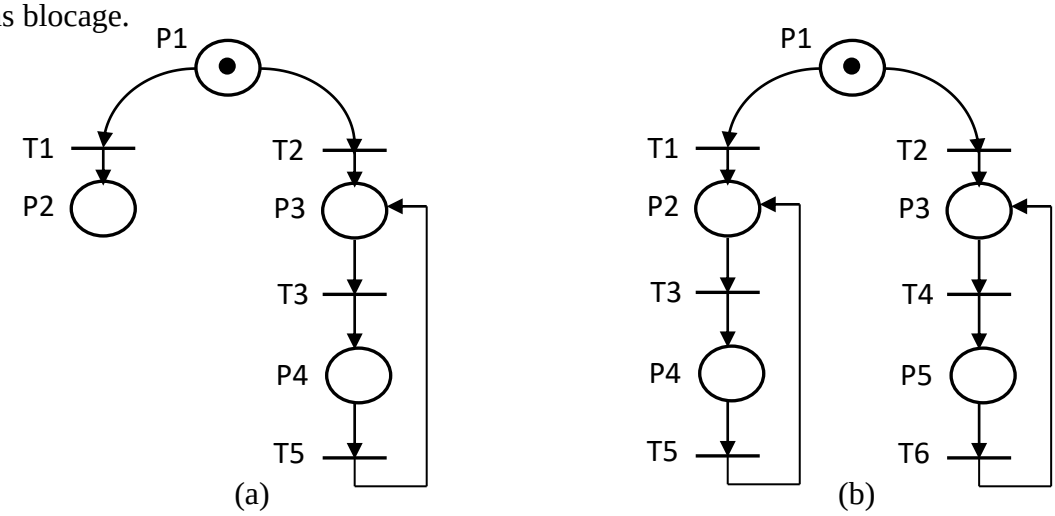


Figure II.18. Rdp : (a) avec blocage pour M=(0,1,0,0), (b) sans blocage.

Propriété

- Si un RdP marqué est sans blocage pour un marquage initial M0 alors il n'est pas nécessairement sans blocage pour **Mk ≥ M0.**

d. Conflits

- Un conflit structurel a précédemment été défini comme l'existence d'une place Pi qui a au moins deux transitions de sortie Tj, Tk, …,etc : Notation : < Pi,{Tj, Tk,…} >.

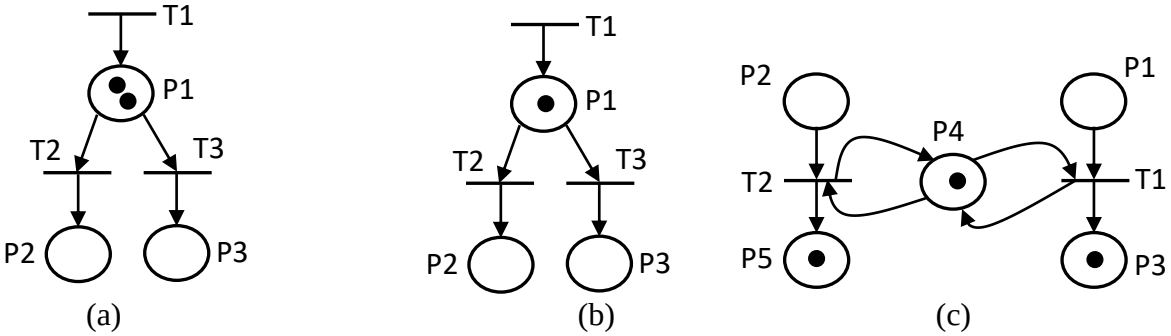


Figure II.19. Rdp : (a) sans conflit effectif, (b) avec conflit effectif, (c) persistant.

- Un conflit effectif est l'existence d'un conflit structurel < Pi,{Tj, Tk,…} > et d'un marquage M tels que le nombre de marques dans la place Pi est strictement inférieur au nombre des transitions de sortie de Pi validées par le marquage M.
- Lors qu'un conflit effectif se produit, il est nécessaire de choisir la transition qui va être effectivement franchie.
- Un RdP est persistant pour un marquage initial M0 si pour tout marquage Mi accessible à partir de M0, on a : si Tj et Tk sont validées par le marquage Mi alors TjTk(TkTj) est une séquence de franchissement à partir de Mi.
- Si le RdP est persistant alors il n'est pas nécessaire d'effectuer un choix lors d'un conflit effectif.

e. Invariants

Des invariants permettent de caractériser certaines propriétés des marquages accessibles et des transitions franchissables, quelle que soit l'évolution.

➢ **Composantes conservatives**

Soit **R** un RdP et **P** l'ensemble de ses places. On a un invariant de marquage s'il existe un ensemble de places $\mathbf{P}' \subseteq \mathbf{P}$ et un vecteur d'entiers naturels appelé vecteur de pondérations **q** tels que : $\forall M \in {}^{*}M0, \sum_{P_i \in P} q_i M(Pi) =$constante.

$\mathbf{P}'$ est appelé composante conservative. Un RdP est conservatif si et seulement si $P' = P$.

➢ **Exemples**

- Les composantes conservatives du Rdp représenté par la figure en face, sont :

•P'={P1, P3, P4} : m1 + m3 + m4=1.
• P'={P2, P4} : m2+ m4= 1.
• P'={P1, P2, P3, P4} : m1+m2+m3+2.m4 = 2.

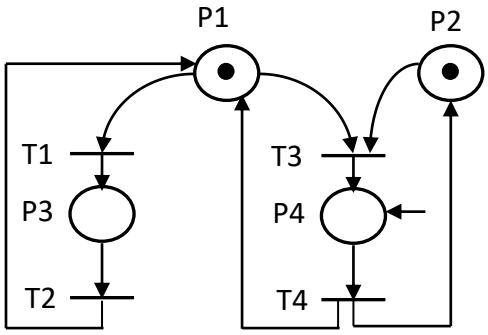


Figure II.20. Exemple 1.

- Le RdP des 4 saisons est conservatif. P'={P1, P2,P3, P4} =P: m1 +m2+ m3 + m4=1.

Remarques

- La propriété d'être composante conservative est indépendante du marquage initial. Par contre, la constante de l'invariant dépend du marquage initial.
- En règle générale, une composante conservative a une signification physique. Elle peut signifier soit qu'un système est dans un seul état à la fois, soit la conservation du nombre d'entités.

➢ **Composantes répétitives**

- On appelle séquence répétitive, une séquence de franchissements **S** telle que **M0(S> M0**.
- Une séquence répétitive **S** est dite séquence répétitive complète si elle contient toutes les transitions du RdP.
- On appelle composante répétitive l'ensemble **T'** des transitions de **T** apparaissant dans la séquence répétitive **S**.
- Le RdP est dit répétitif si **T'** = **T**.

➢ **Exemple**

La Le RdP des 4 saisons a une séquence de franchissement répétitive complète **S**=T1T2T3T4.

Propriété

- Si **S** est une séquence répétitive pour la condition initiale M0 alors c'est aussi une séquence répétitive pour la marquage initial $\mathbf{M'0 \geq M0.}$

II.9. Recherche des propriétés des réseaux de Petri

Pour pouvoir trouver si tel RdP présente telle ou telle propriété, il existe principalement 3 classes de méthodes :

a. Établissement du graphe de marquage ou de l'arbre de couverture
b. Utilisation des méthodes basées sur l'algèbre linéaire : résultats puissants.
c. Les méthodes de réduction des RdP.

Dans ce chapitre, on s'intéresse à l'étude des propriétés des RdP par graphe de marquage ou de l'arbre de couverture.

a. Graphe de marquage

Il est composé de nœuds qui correspondent aux marquages accessibles et d'arcs correspondant aux franchissements de transition faisant passer d'un marquage à l'autre. Sur le graphe des marquages, on peut trouver toutes les propriétés des RdP.

➢ **Exemple**

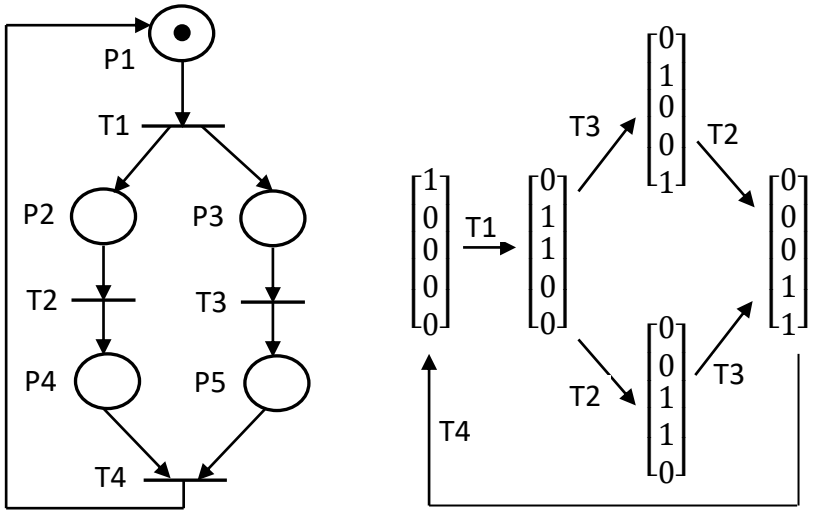


Figure II.21. Exemple2.

On voit de la figure (II.2) que ce RdP est borné, sauf, vivant, sans blocage, 2.m1+m2+m3+m4+m5 = 2 (donc le RdP est conservatif), et que T1T3T2T4 et T1T2T3T4 sont des séquences répétitives (donc le RdP est répétitif).

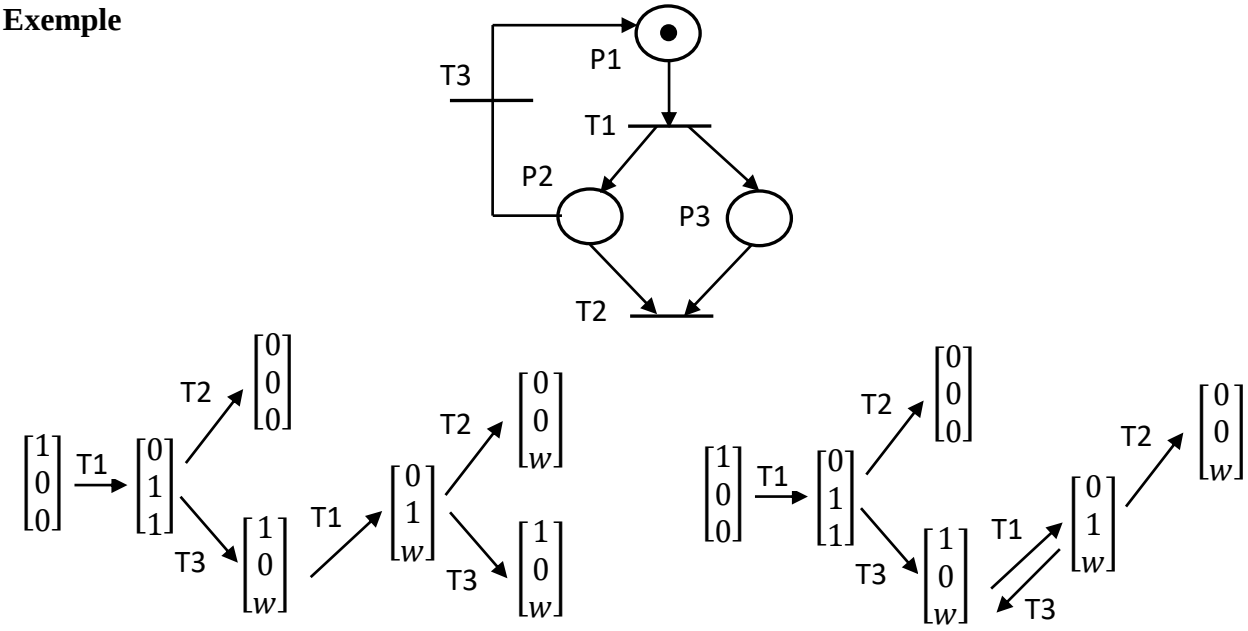
b. Arbre de couverture

Quand on ne peut pas construire le graphe des marquages (RdP non borné), on construit un arbre de couverture qui possède un nombre fini de nœuds. Un arbre est un graphe particulier dans lequel il n'y a pas de boucle ni de circuit.

- **Algorithme de construction de l'arbre de couverture**

Pas 1 : A partir du marquage initial M0, on indique toutes les transitions validées et les marquages successeurs correspondants. Si un de ces marquages est strictement supérieur à M0, on met **w** pour chacune des composantes supérieures aux composantes correspondantes de M0.
Pas 2 : Pour chaque nouveau marquage Mi de l'arbre, on fait soit le pas 2.1 soit le pas 2.2
Pas 2.1 : S'il existe sur le chemin de M0 à Mi (ce dernier exclu) un marquage Mj = Mi, alors Mi n'a pas de successeur.
Pas 2.2 : S'il n'existe pas de marquage Mk = Mi sur le chemin de M0 à Mi, alors on prolonge l'arbre en ajoutant tous les successeurs de Mi. Pour chaque successeur Mk de Mi : (a) une composante w de Mi reste une composante w de Mk ; (b) s'il existe un marquage Mj sur le chemin de M0 à Mk tel que Mk>Mj, alors on met w pour chacune des composantes supérieures aux composantes de Mj.

- **Exemple**



Arbre de couverture. Graphe de couverture.

Figure II.22. Exemple de l'arbre et de graphe de couverture.

- **Remarques**

- Si arbre de couverture contient toujours un nombre fini de nœuds, on peut obtenir un graphe de couverture (GC) en fusionnant les nœuds de l'arbre de couverture qui correspond aux mêmes marquages.
- Le graphe de couverture décide si un réseau est borné ou non : une place « p » d'un réseau marqué N est non-bornée si il existe un sommet M de GC que M(p) = ω.
- Si un réseau marqué N est borné, le graphe de couverture et le graphe des marquages sont identiques.

- D'une manière générale, le graphe couverture ne permet pas d'étudier la vivacité du réseau parce que le symbole ω correspond à une perte d'information.

II.9. Abréviations et extensions

- **Abréviations**: Des représentations simplifiées utiles pour alléger le graphisme mais auxquelles on peut toujours faire correspondre un RdP ordinaire (c.-à-d. un RdP autonome marqué fonctionnant selon les règles prédéfinies).
- **Extensions**: Des modèles auxquels des règles de fonctionnement ont été ajoutées afin d'enrichir le modèle initial pour aborder un plus grand nombre d'applications.

a. RdP généralisé

Un réseau de Petri généralisé est un RdP dans lequel des poids (nombres entiers strictement positifs) sont associés aux arcs. Tous les arcs dont le poids n'est pas explicitement spécifié, ont un poids de 1. Lorsqu'un arc reliant une place Pi à une transition Tj possède un poids p, cela signifie que la transition Tj ne sera validée que si la place Pi contient au moins p marques. Lors du franchissement de cette transition, p marques sont retirées de la place Pi .Lorsqu'un arc reliant une transition Tj a une place Pj possède un poids q, cela signifie que lors du franchissement de Tj, q marques seront ajoutées à la place Pj.

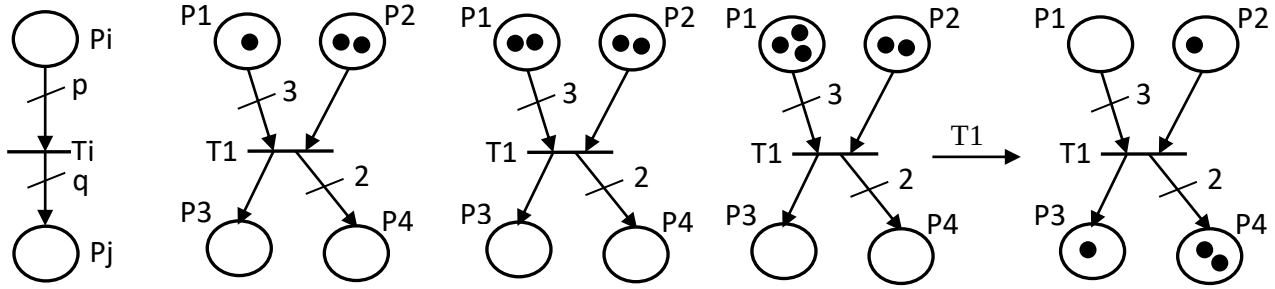


Figure II.23. Exemple de franchissement d'une transition d'un Rdp généralisé.

b. RdP à prédicats

Les réseaux de Petri à prédicats utilisent des variables pour inclure deux autres propriétés :

- des "gardes", variables ou expressions booléennes qui rendent infranchissables les transitions tant qu'elles ne sont pas vérifiées (=1). Dans ce cas le RdP est appelé RdP interprété.
- des "affectations", attributions qui modifient les valeurs de variables lors du franchissement des transitions. Dans l'exemple, on considère une famille dans laquelle A est le père de B.

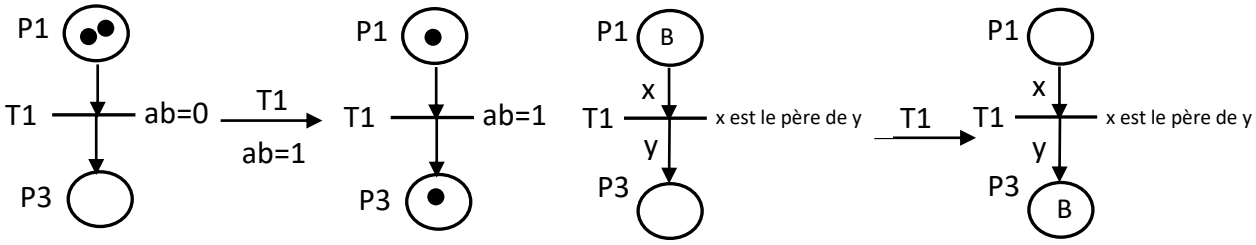
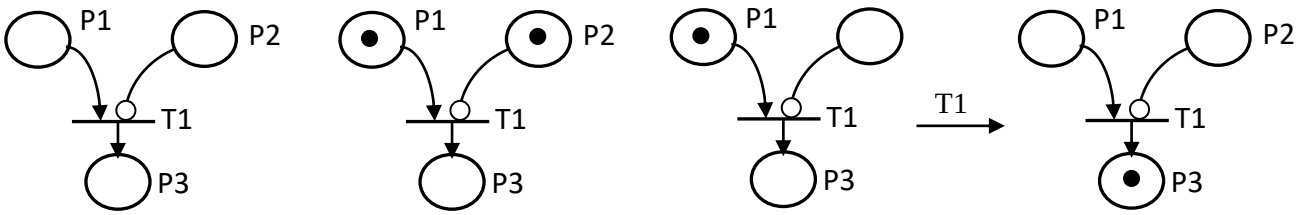


Figure II.24. Exemples de franchissement d'une transition d'un Rdp à prédicats.

C. RdP à arcs inhibiteurs

Un arc inhibiteur est un arc orienté qui part d'une place pour aboutir à une transition (et non l'inverse). Son extrémité est marquée par un petit cercle. La présence d'un arc inhibiteur entre une place Pi et une transition Tj signifie que la transition Tj n'est validée que si la place Pi ne contient

aucun jeton. Le franchissement de la transition Tj consiste à retirer un jeton dans chaque place située en amont de la transition à l'exception de la place Pi , et à ajouter un jeton dans chaque place située en aval de la transition.



T1 non validée. T1 non validée. T1 validée.

Figure II.25. Exemple de franchissement d'une transition d'un Rdp à arcs inhibiteurs.

d. RdP à capacité

Un RdP dans lequel des capacités (nombres entiers strictement positifs) sont associés aux places (cap(Pi)). Le nombre de jetons dans Pi ne dépasse pas cette capacité.

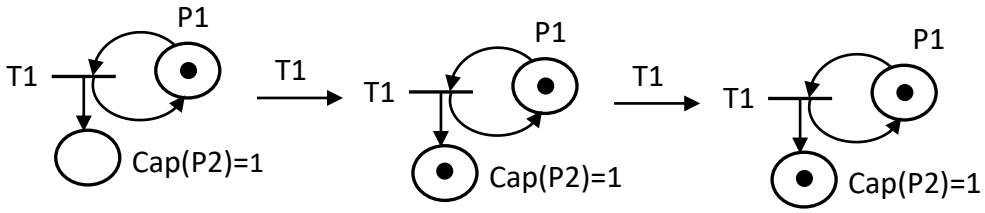


Figure II.26. Exemples de franchissement d'une transition d'un Rdp à capacité.

e. RdP coloré

Un RdP coloré comporte des jetons auxquelles on attribue des couleurs (possibilité de représenter des processus parallèles).

f. RdP FIFO

Dans un réseau de Petri FIFO (First In, First Out, premier entré, premier sorti), les marques sont différenciées de telle manière que l'on puisse modéliser différents messages et les règles de franchissement sont modifiées pour modéliser le mécanisme FIFO.

g. RdP à priorité

Un tel réseau est utilisé lorsque l'on veut imposer un choix entre plusieurs transitions validées.

h. RdP continu

Leur particularité est que le marquage d'une place est un nombre réel (positif) et non plus un nombre entier.

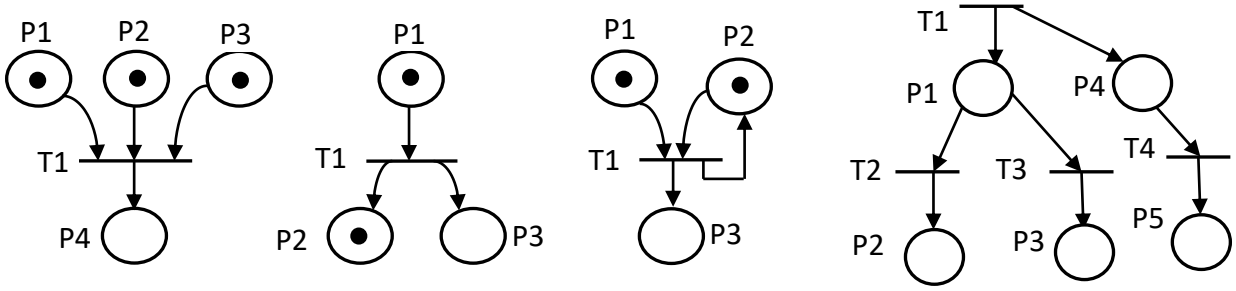
➢ **Remarques**

- Les RdP généralisés, à capacité, coloré sont des abréviations des RdP ordinaires. Toutes les propriétés que nous avons vues peuvent donc être adaptées à ces modèles.
- Les RdP à arcs inhibiteurs, les RdP à priorités, les RdP non autonomes et les RdP continus sont des extensions des RdP ordinaires. Certaines propriétés des RdP ordinaires sont applicables mais pas toutes.
- Toutes les propriétés des RdP ordinaires se conservent pour les abréviations, tandis que ces propriétés ne se conservent pas toutes pour les extensions.

II.10 Exercices

Exercice 01

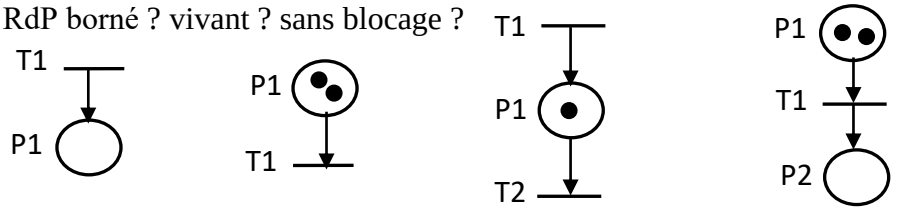
Les RdP suivants sont ils des graphes d'états ? des graphes d'événement ? sans conflit ? à choix libre ? simple ? pur ?



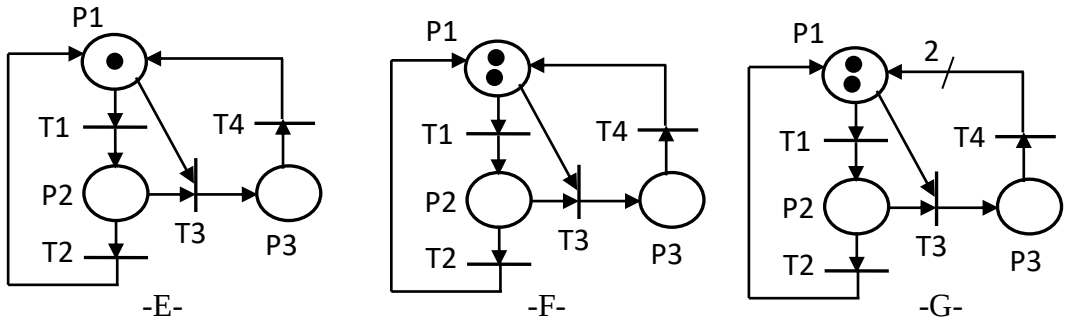
-A- -B- -C- -D-

Exercice 02

Pour chacun des RdP suivants dont certains sont des RdP généralisés, indiquer est qu'il s'agit d'un RdP borné ? vivant ? sans blocage ?



-A- -B- -C- -D-

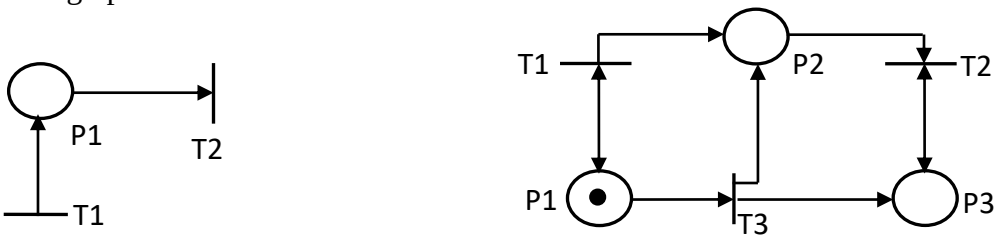


-E- -F- -G-

Les RdP A, B, C et D sont-ils conservatifs, répétitifs?

Exercice 03

Construisez le graphe de couverture des RdP suivants :



-A- -B-

Exercice 04

On considère maintenant la réaction chimique suivante qui se produit en présence de catalyseur (platine) :

$$H2 + C2H4 \xrightarrow{\text{platine}} C2H6$$

On suppose qu'au départ on a deux unités d'H2, une unité de C2H4, une unité de C2H6 et que le platine est libre. Cependant, quand la réaction chimique se produit, le platine étant un catalyseur, il n'est pas consommé.
Etablir le réseau de Petri correspond à cette réaction chimique.

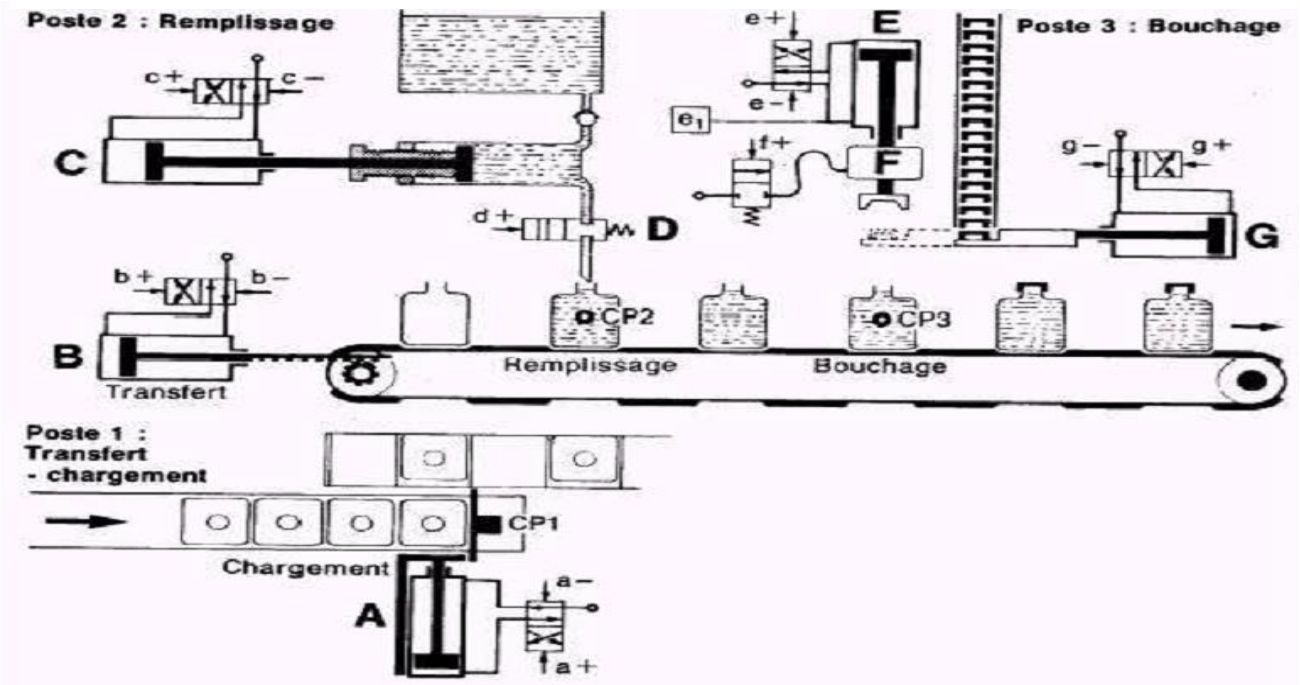
La réaction chimique a été supposée être un événement instantané. En réalité, la réaction a une certaine durée caractérisée par un début (où les réactifs se fixent sur le catalyseur afin de réagir) et par une fin (les réactifs ont fini de réagir et libère le catalyseur).
Rétablir le nouveau réseau de Petri.

Exercice 05

Un atelier est constitué d'une machine de coupe et d'un stock. Quand une commande arrive et que la machine de coupe est disponible, la commande peut être traitée (découpe). Une fois le traitement terminé, la commande qui a été traitée est stockée. Sinon, la commande doit attendre que la machine de coupe se libère avant de pouvoir être traitée.
Construisez le réseau de Petri.

On désire prendre en compte que l'atelier peut être ouvert ou fermé et que la machine de coupe ne peut traiter une commande que quand l'atelier est ouvert.
Reconstruisez le réseau de Petri.

Exercice 06



La machine à remplir et à boucher des bouteilles est composée de trois postes travaillant en parallèle.
– Le poste 1 sert au transfert et au chargement. Dans on premier temps, on sort le vérin de transfert B pour à décaler le convoyeur d'une position vers la droite. Ensuite, le vérin A sert au chargement d'une nouvelle bouteille vide.
– Le poste 2 sert au remplissage des bouteilles à l'aide de la vanne D.
– Le poste 3 est le poste de bouchage.

Les actions de chargement d'une bouteille, remplissage d'une bouteille et bouchage d'une bouteille sont effectuées en parallèle. Le transfert par le vérin B n'est effectué que lorsque ces trois opérations sont terminées.

Etablir le réseau de Petri correspond à cette machine.

GRAFCET

III.1. Introduction

Le GRAFCET (Graphe Fonctionnel de Commande Etape/Transition) a été proposé par ADEPA (agence pour le développement de la Productique Appliquée à l'industrie) et l'AFCET (Association Française pour la Cybernétique Economique et Technique) en 1977, et normalisé en 1982 par la NF C03-190. C'est un outil graphique destiné à décrire les différents comportements de l'évolution d'un automatisme séquentiel. Il est très utilisé pour la programmation des automates programmables industriels. Lorsque le mot grafcet est écrit en minuscule, il fait alors référence à un modèle obtenu à l'aide des règles du GRAFCET.

Le GRAFCET possède les avantages suivants :

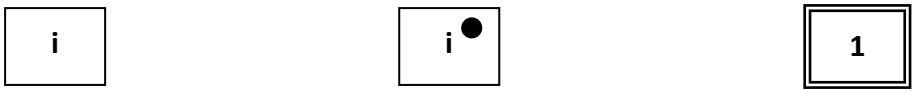
- Il est indépendant de la matérialisation technologique.
- Il traduit de façon cohérente le cahier des charges.
- Il est bien adapté à la complexité des systèmes automatisés et à la conception et réalisation.

Dans ce chapitre, on commence par donner les bases nécessaires de la modélisation des systèmes automatisés en langage GRAFCET. Ensuite, on illustre ces différentes règles et on présente les différentes structures du grafcet. Les différentes actions associées aux étapes sont définies par la suite. En fin, on montre la transformation du modèle grafcet en des équations.

III.2. Eléments graphiques de base

III.2.1. Etape

Une étape symbolise un état stable ou une partie stable de l'état du système automatisé. L'étape i est représentée par un carré et elle est associée à la variable binaire Xi, dite variable de l'étape. L'étape possède deux états possibles : active, représentée par un jeton (marque) dans l'étape, ou inactive. La situation initiale d'un système automatisé est indiquée par une étape dite étape initiale et représentée par un carré double.



Etape i inactive (Xi=0). Etape i active (Xi=1). Etape initiale.

Figure III.1. Etapes.

- **Remarque III1** *:* Dans un grafcet il doit y avoir au moins une étape initiale.

- **Actions associées aux étapes**

Chaque étape est associée une action (qui s'effectuera quand l'étape sera active) ou plusieurs, c'est à dire un ordre envoyé vers la partie opérative ou vers d'autres grafcets. L'action est représentée dans un rectangle à gauche.

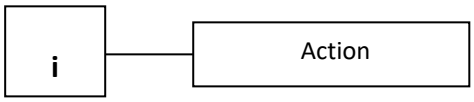


Figure III.2. Action d'une étape.

➢ **Remarque III.2**

- On peut rencontrer une étape vide (sans action).
- Plusieurs actions peuvent être associées à une même étape.
- On peut rencontrer une même action associée à plusieurs étapes.

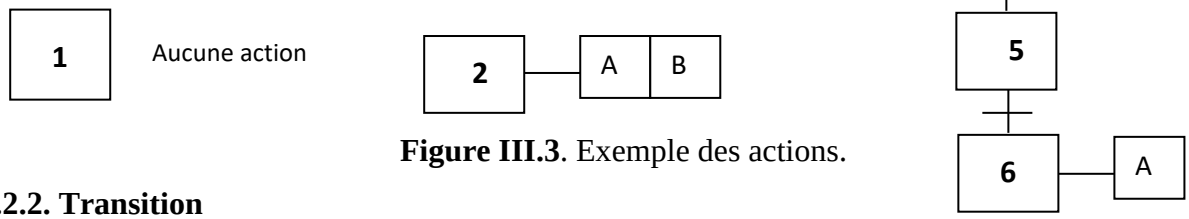


Figure III.3. Exemple des actions.

III.2.2. Transition

Une transition indique la possibilité d'évolution entre deux ou plusieurs étapes (passage d'une (ou plusieurs) étape(s) à une (ou plusieurs) autre(s) étape(s)). On représente une transition par un petit trait horizontal sur une liaison verticale. La condition d'évolution est définie par une réceptivité (Vraie ou Fausse) qui est inscrite de façon littérale ou symbolique ou par une expression booléenne, à la droite de la transition.

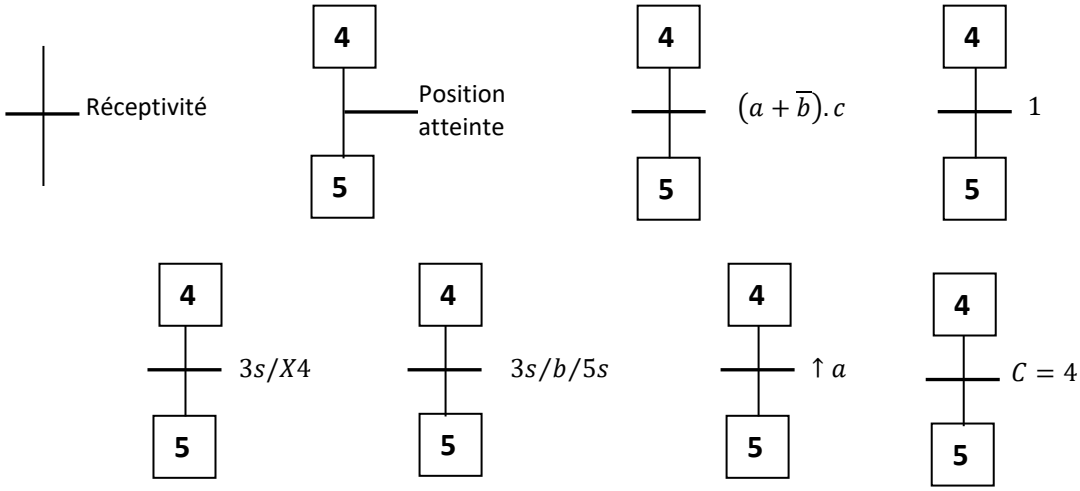


Figure III.4. Exemple des transitions.

➢ **Notations particulières**

↑ a : front montant de la variable a ;
1 : réceptivité toujours vraie ;
3s/X4 : temporisation de 3s après activation de l'étape 4 ;
3s/b/5s : est vraie 3s après ↑ b et devient fausse 5s après ↓ b ;
[C=4] : valeur booléenne du prédicat "C=4".

III.2.3. Liaisons (arcs) orientées

Elles relient les étapes et les transitions. Chaque liaison est représentée par un trait plein rectiligne (droit), vertical ou horizontal. Par convention, les évolutions se font du haut vers le bas. Dans le cas contraire, il est nécessaire d'indiquer le sens de l'évolution par une flèche.

III.3. Règles de syntaxe

Deux étapes ou deux transitions ne doivent jamais être reliées directement : elles doivent être séparées par une transition ou une étape, respectivement. Pour les séquences simultanées, on a une transition unique et deux traits parallèles. Pour les séquences sélectionnées on a : une transition au début de chaque séquence pour la divergence en OU, et une transition à la fin de chaque séquence pour la convergence en OU.

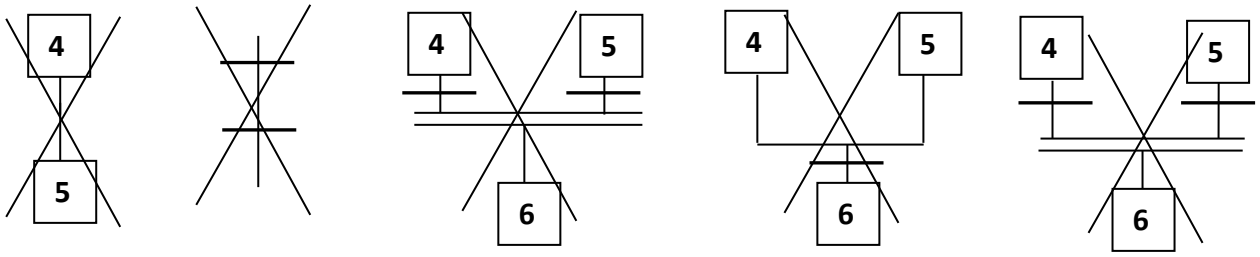


Figure III.5. Erreurs de syntaxe.

III.4. Exemple de grafcet

Une fraiseuse est une machine utilisée pour usiner tous types de pièces mécaniques à l'aide d'un outil coupant nommé fraise.

Figure III.6. Fraiseuse.

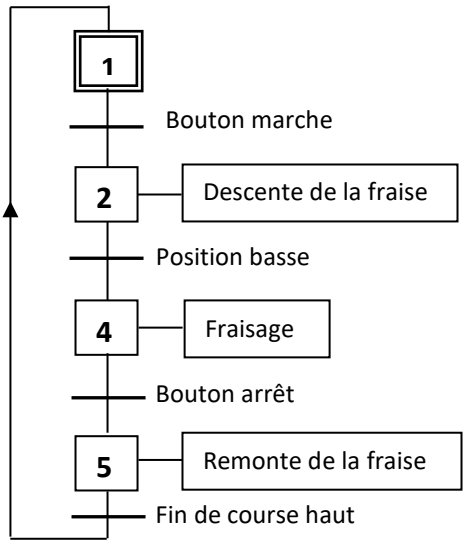


Figure III.7. Grafcet de la fraiseuse.

Le fonctionnement est donné comme suit :
• On appuie sur le bouton marche de la fraiseuse, la fraise descend.
• Une fois la position basse atteinte le fraisage s'effectue.
• On appuie sur le bouton arrêt, le fraisage s'arrête et la fraise remonte.
• Une fois la position haute atteinte, la fraiseuse est en position initiale.

III.5. Règles d'évolution

- **Règle 1 : Situation initiale**

L'étape initiale caractérise le comportement de la partie commande d'un système en début de cycle. Elle correspond généralement à une positon d'attente. L'étape initiale est activée sans condition en début de cycle (fonctionnement). Il peut y avoir plusieurs étapes initiales dans un même grafcet.

- **Règle 2 : Franchissement d'une transition**

Une transition est validée si toutes les étapes immédiatement précédentes sont actives. L'évolution du grafcet correspond au franchissement d'une transition qui se produit sous deux conditions :

• si cette transition est validée
• si la réceptivité associée à cette transition est vraie
Si ces deux conditions sont réunies, la transition devient franchissable, elle est alors obligatoirement franchie.

➢ **Règle 3 : Evolution des étapes actives**

Le franchissement d'une transition entraîne simultanément l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes celles immédiatement précédentes.

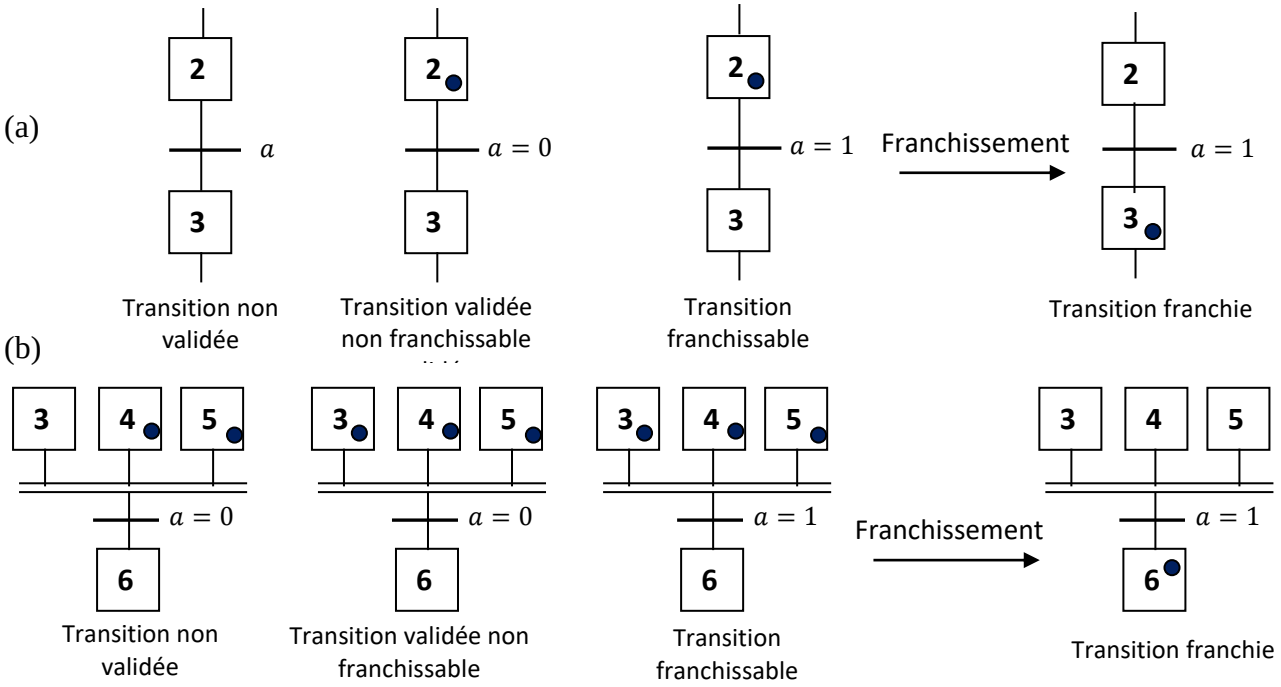


Figure III.8. Franchissement d'une transition :(a) séquence unique, (b) séquences simultanées.

➢ **Règle 4 : Evolutions (Franchissements) simultanées**

Plusieurs transitions simultanément franchissables sont simultanément franchies.

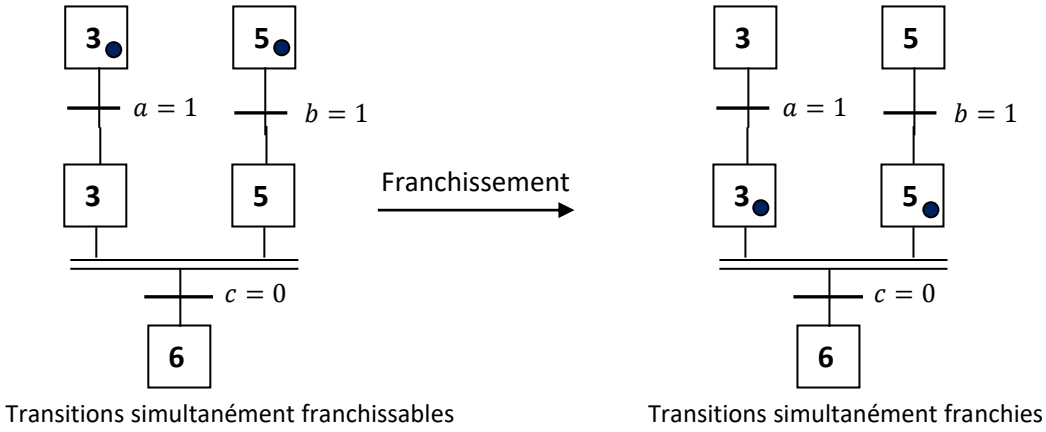


Figure III.9. Franchissements simultanées.

➢ **Règle 5 : Activation et désactivation simultanée**

Si, au cours du fonctionnement, une même étape doit être désactivée et activée simultanément, elle reste active.

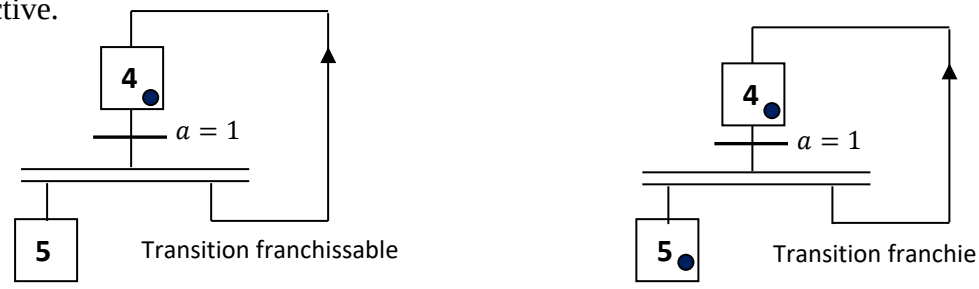
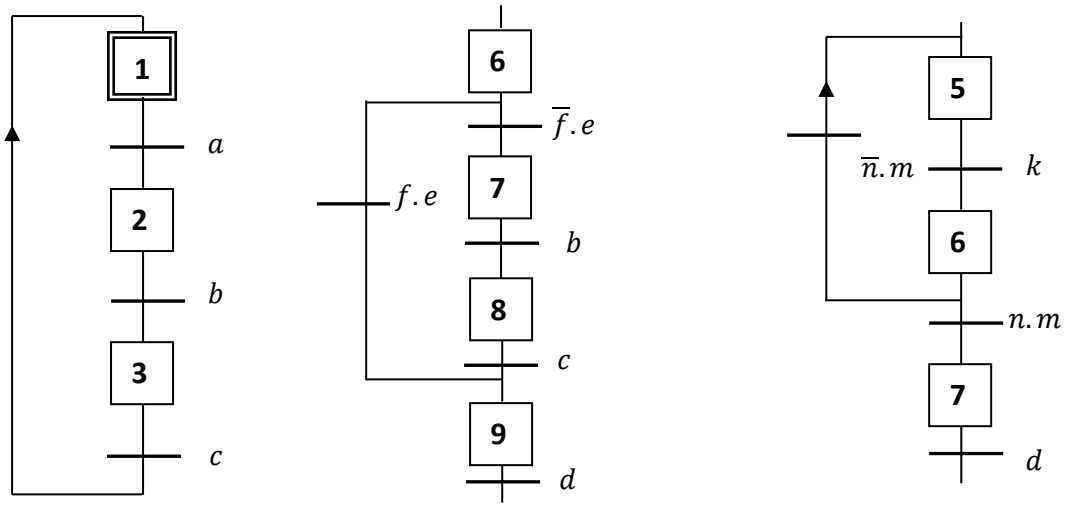


Figure III.10. Activation et désactivation simultanée.

III.6. Structures de base

- **Séquence unique :** C'est une suite d'étapes pouvant être activées les unes après les autres.



(a) (b) (c)

Figure III.11. Structure : (a) séquence unique, (b) Saut d'étapes, (c) Reprise de séquence.

- **Saut d'étapes :** Il permet de sauter une ou plusieurs étapes : Boucle Si Alors.

- **Reprise de séquence :** Il permet de répéter une séquence des étapes : Boucle Répéter Tant que.

- **Séquences simultanées (ET) :** Il s'agit des plusieurs séquences qui se déroulent en même temps. On distingue 2 structures des séquences simultanées :

  - Convergence en ET (ou Synchronisation de séquences): La transition n'est validée que lorsque toutes les étapes en amont immédiatement sont actives.
  - Divergence en ET (ou Séquences parallèles) : il s'agit d'une "distribution", le franchissement de la transition entraîne l'activation des étapes immédiatement suivantes.

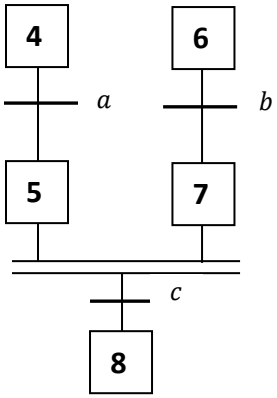


Figure III.12. Convergence en ET.

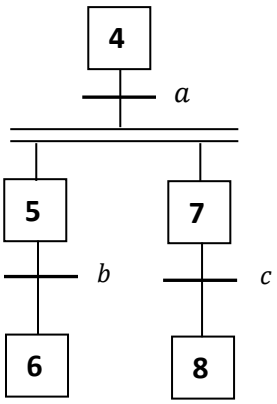


Figure III.13. Divergence en ET.

- **Séquences sélectionnées (alternatives) :** Il s'agit d'une "sélection" parmi les séquences. On peut avoir deux structures des séquences sélectionnées:

  - Convergence en OU : Après l'évolution dans une branche, il y a convergence vers une étape commune.
  - Divergence en OU: L'évolution du système vers une branche dépend des réceptivités associées aux premières transitions de chaque séquence.

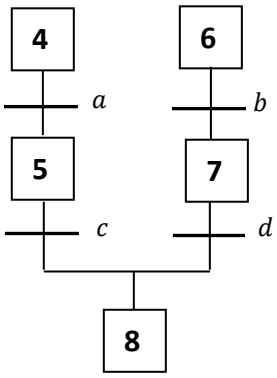


Figure III.14. Convergence en OU.

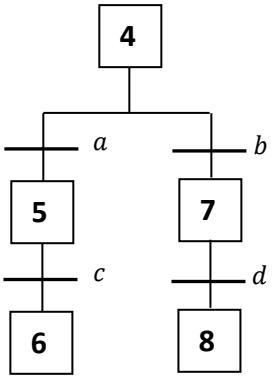


Figure III.15. Divergence en OU.

➢ **Remarque III.3**

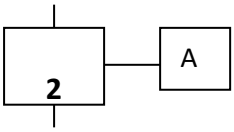
- Après une convergence en ET, on trouve une divergence en ET.
- La réceptivité associée à la convergence ET peut être de la forme =1. Dans ce cas la transition est franchie dés qu'elle elle est valide.
- Après une convergence en OU, on trouve une divergence en OU.
- On ne peut pas utiliser simultanément les 2 branches d'un grafcet avec choix de séquences.

III.7. Classification des actions associées aux étapes

L'action associée à l'étape peut être de 3 types : continue, conditionnelle ou mémorisée.

III.7.1. Action continue

L'ordre est émis, de façon continue, tant que l'étape, à laquelle il est associé, est active.



On note : A = X2

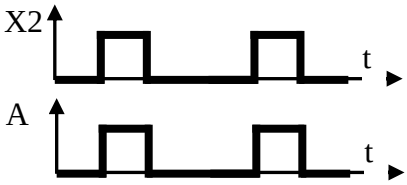
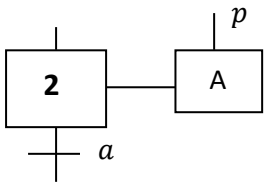


Figure III.16. Chronogramme de l'action continue.

III.7.2. Action conditionnelle

Une action conditionnelle n'est exécutée que si l'étape associée est active et si la condition associée est vraie. Elle peut être décomposée en 3 cas particuliers:

III.7.2.1. Action conditionnelle simple



On note : A = X2. p

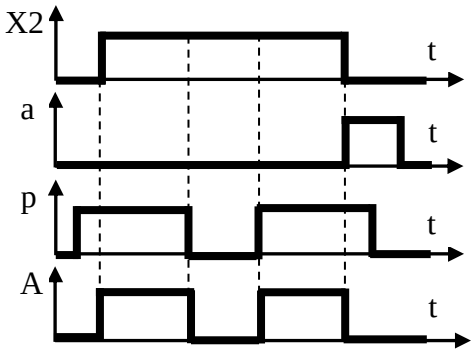
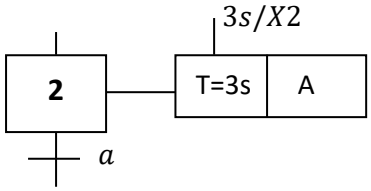


Figure III.17. Chronogramme de l'action conditionnelle simple.

III.7.2.2. Action retardée



On note : A = 3s/X2

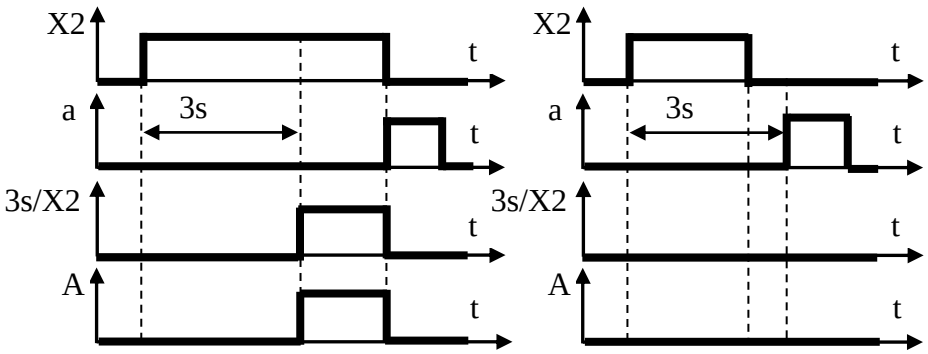
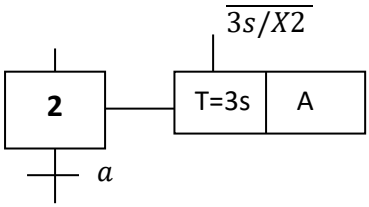


Figure III.18. Chronogramme de l'action retardée.

- L'ordre "A" prendra la valeur logique 1, 3s après l'activation de l'étape 2.
- Si la durée d'activité de l'étape 2 est inférieure à 3s, la sortie A ne sera pas assignée à la valeur vraie.

III.7.2.3. Action de durée limitée



On note : A = X2. $\overline{3s/X2}$

L'ordre "A" est limité à 3s après l'activation de l'étape 2.

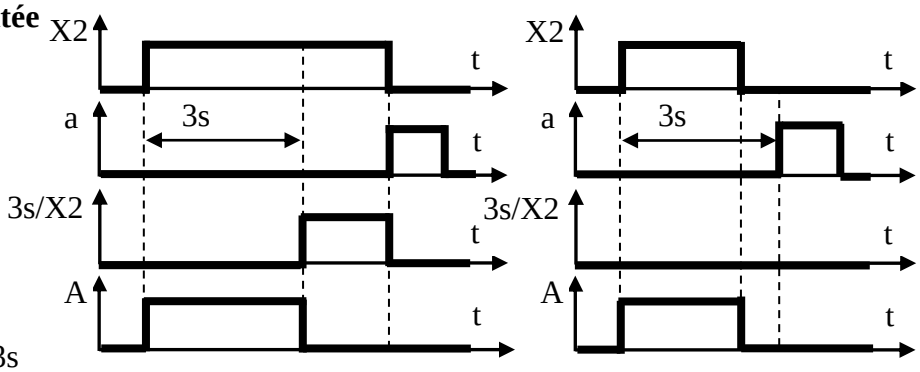


Figure III.19. Chronogramme de l'action de durée limitée.

III.7.3. Action mémorisée

Afin de maintenir la continuité d'une action sur plusieurs étapes, il est possible de répéter l'ordre continu relatif à cette action dans toutes les étapes concernées ou d'utiliser une description sous forme de séquences simultanées.

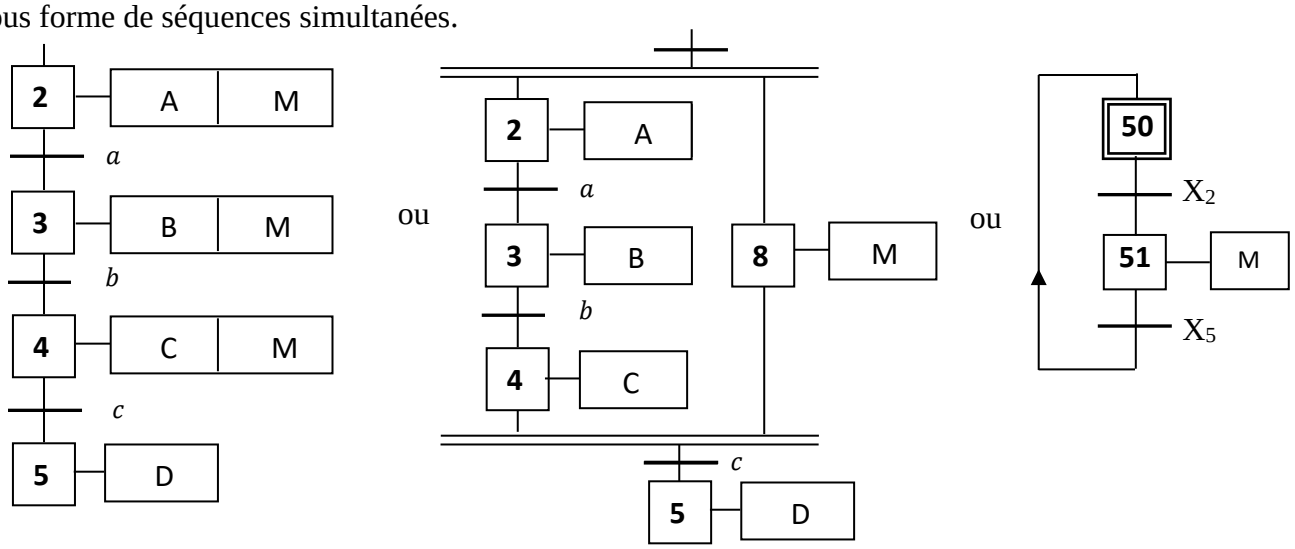


Figure III.20. Action mémorisée.

Le maintien d'un ordre, sur la durée d'activation de plusieurs étapes consécutives, peut également être obtenu par la mémorisation de l'action, obtenue par l'utilisation d'une fonction auxiliaire appelée fonction mémoire.

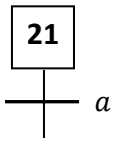
III.8. Etape source/puits et transition source/puits

- **Etape source**

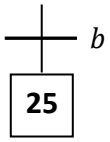
Etape non reliée à une transition amont. Elle ne peut être activée, que si elle est initiale ou que si elle est soumise à un ordre d'activation venant d'une autre partie de grafcet.

- **Etape puits**

Etape non reliée à une transition aval. Elle ne peut être désactivée, que si elle est soumise à un ordre de désactivation venant d'une autre partie de grafcet (forçage, étape encapsulante).

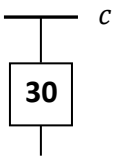


(a)

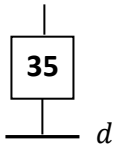


(b)

Figure III.21. Etape: (a) source, (b) puits.



(a)



(b)

Figure III.22. Transition: (a) source, (b) puits.

- **Transition source**

Transition non reliée à une étape amont. Par convention elle est toujours validée, et devient franchissable lorsque la réceptivité associée est vraie.

- **Transition puits**

Transition non reliée à une étape aval.

III.9. Différents points de vue d'un grafcet

La représentation d'un système automatisé par un grafcet prend en compte le "point de vue" selon lequel l'observateur s'implique au fonctionnement de ce système.
On distingue trois points de vue :

- Grafcet du point de vue système.
- Grafcet du point de vue partie opérative.
- Grafcet du point de vue partie commande.

a. Grafcet du point de vue système

C'est un graphe qui décrit le fonctionnement global du système. Il traduit le cahier des charges sans préjuger de la technologie (moyens techniques) adoptée. Il permet de dialoguer avec des personnes non spécialistes (fournisseurs, décideurs ...). Son écriture, en langage clair, permet donc sa compréhension par tout le monde.

b. Grafcet du point de vue partie opérative

Dans ce type de grafcet, on spécifie la technologie de la partie opérative (capteurs et actionneurs) ainsi que le type de ses informations reçues et envoyées. L'observateur de ce point de vue étant un spécialiste de la partie opérative, la partie commande ne l'intéresse que par ses effets.

c. Grafcet du point de vue partie commande

Ce grafcet est établi en spécifiant la technologie des éléments de dialogue :

- entre PC et PO ;
- entre PC et opérateur ;
- entre PC et autre système.

Ce point de vue de grafcet est réalisé par un spécialiste de la partie commande afin de lui permet d'établir les équations et éventuellement les schémas de réalisation (électronique, électrique, pneumatique).

➢ **Remarque III.4**

- Le grafcet du point de vue système est dit grafcet de niveau 1 du cahier des charges.
- Les grafcets du point de vue partie opérative et partie commande système sont des grafcets de niveau 2 du cahier des charges.
- Le grafcet point de vue partie commande ressemble au grafcet point de vue partie opérative à ceci près que les entrées et sorties sont écrites sous forme de mnémoniques (variables) et font référence aux entrées et sorties de l'automate. En fait, il y a d'autres différences mais nous n'en parlerons pas ici.
- La façon de décrire par GRAFCET un système n'est pas unique.

III.10. Mise en équations du grafcet

La plupart des automates ne se programment pas en GRAFCET directement. Mais, généralement ils peuvent être programmés en LADDER. Donc, Il faut transformer le grafcet en des équations afin de traiter les systèmes séquentiels en langage LADDER.

III.10. 1. Mise en équation d'une étape

Pour décrire l'activité d'une étape i par une équation, on exploite lés règles dévolution décrites précédemment**.**

➢ **La condition d'activation d'une étape Xi (CAX_i)**

D'après la règle 3, une étape Xi est activée ($\mathbf{CAX_i = 1}$) si la transition (les transitions) immédiatement précédente(s) soit (soient) franchissable(s).

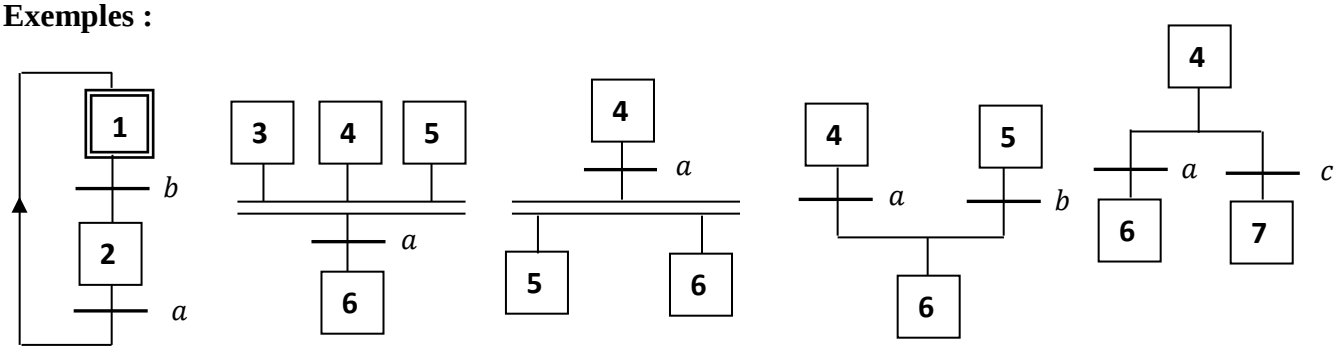
➢ **La condition de désactivation d'une étape Xi (CDX_i)**

D'après la règle 3, une étape Xi est désactivée ($CDX_i = 1$) si si la transition immédiatement suivante soit franchissable.

➢ **Equation d'une étape Xi** :

Une étape est active si elle est activée ou elle est active et la condition de désactivation est fausse. $X_i = CAX_i + \overline{CDX_i}\,.X_i$

Exemples :



$CAX_1 = X_2.a$ $CAX_6 = X_3.X_4.X_5.a$ $CAX_5 = X_4.a$ $CAX_6 = X_4.a + X_5.b$ $CAX_6 = X_4.a$

$CDX_1 = X_1.b$ $CDX_3 = X_3.X_4.X_5.a$ $CDX_4 = X_4.a$ $CDX_5 = X_5.b$ $CDX_4 = X_4.(a + c)$

III.10. 2. Gestion des modes Marche/Arrêt

A l'initialisation du grafcet, toutes les étapes autres que les étapes initiales sont désactivées. Seules les étapes initiales sont activées.

Soit la variable **Init** telle que :
Init= 1 : Initialisation du grafcet : mode Arrêt.
Init=0 : déroulement du cycle : mode Marche.

- **Mise en équation d'une étape initiale**

  - **La condition d'activation d'une étape initiale Xi (CAX_i)**

Une étape initiale Xi est activée ($CAX_i = 1$) si la transition (les transitions) immédiatement précédente(s) soit (soient) franchissable(s) ou si le grafcet est initialisé.

  - **La condition de désactivation d'une initiale Xi (CDX_i)**

Une étape initiale Xi est désactivée ($CDX_i = 1$) si si la transition immédiatement suivante soit franchissable et le mode en marche.

  - **Equation d'une étape initiale Xi :** $X_i = CAX_i + \overline{CDX_i} \,.\, X_i + Init = CAX_i + \overline{CDX_i} \,.\, X_i$

- **Mise en équation d'une étape non initiale**

  - **La condition d'activation d'une étape non initiale Xi (CAX_i)**

Une étape non initiale Xi est activée ($CAX_i = 1$) si la transition (les transitions) immédiatement précédente(s) soit (soient) franchissable(s) et le mode en marche.

  - **La condition de désactivation d'une non initiale Xi (CDX_i)**

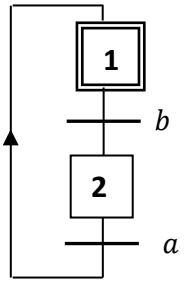
Une étape Xi est désactivée ($CDX_i = 1$) si si la transition immédiatement suivante soit franchissable ou si le grafcet est initialisé.

  - **Equation d'une étape non initiale Xi** : $X_i = \left(CAX_i + \overline{CDX_i} \,.\, X_i\right).\overline{Init} = CAX_i + \overline{CDX_i} \,.\, X_i$

Exemple :

$CAX_1 = X_2.a + Init \qquad CAX_2 = X_1.b.\overline{Init}$

$CDX_1 = X_1.b.\overline{Init} \qquad CDX_2 = X_2.a. + Init$



III.10.3. Gestion des arrêts d'urgences

Soient les variables arret d'urgences (AUdur et Audoux) telles que :
AUdur=1 : désactivation de toutes les étapes
Audoux=1 : désactivation des actions, les étapes restant actives.

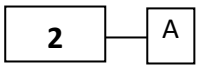
- **Equation d'une étape initiale Xi** :

$$X_i = \left(CAX_i + \overline{CDX_i} \,.\, X_i + Init\right).\overline{AUdur} = \left(CAX_i + \overline{CDX_i} \,.\, X_i\right).\overline{AUdur}.$$

- **Equation d'une étape non initiale Xi** :

$$X_i = \left(CAX_i + \overline{CDX_i} \,.\, X_i\right).\overline{Init}.\overline{AUdur} = \left(CAX_i + \overline{CDX_i} \,.\, X_i\right).\overline{AUdur}.$$

- **Equation des actions** :



$A = X_2.\overline{AUdoux}.$

III.11. Exercices

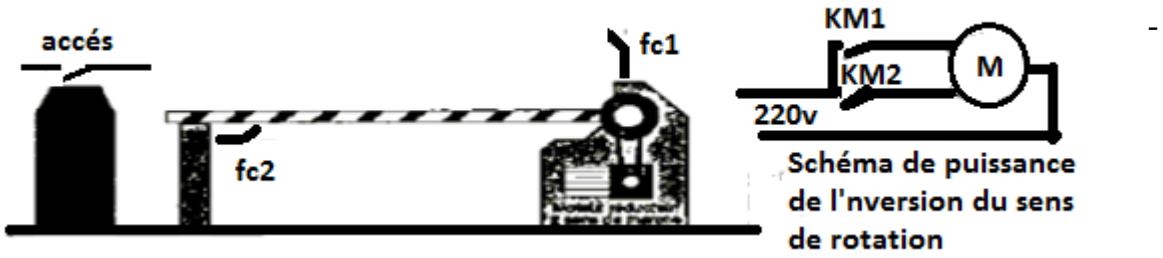
Exercice 01 : Perceuse automatisée

Une perceuse montée sur un support coulissant verticale doit être automatisée. En appuyant sur le bouton poussoir ‘Marche (m)’ (si une pièce est détectée (p)) le support entame sa descente jusqu’à la position milieu (me) en grande vitesse sans rotation de la forêt. L'arrivée en position milieu provoque la rotation de la forêt, le support continue sa descente en petite vitesse jusqu’en position basse (b).
Arrivé en position basse, le support remonte en petite vitesse jusqu’à la position milieu, la remontée s'effectue alors en grande vitesse et la rotation de la forêt est stoppée. En position haute (h) le cyle s'arrête et reprend si un nouvel ordre de ‘Marche’ est donné.

On appelle:
Le moteur du support coulissant: Mc. Le moteur de la broche: M
GVMc+: monter support grande vitesse ; GVMc-: descendre support grande vitesse ; PVMc+: monter support petite vitesse ; PVMc-: descendre support petite vitesse M: faire tourner broche
Construire le grafcet point de vue partie opérative.

Exercice 02: Barrière automatique

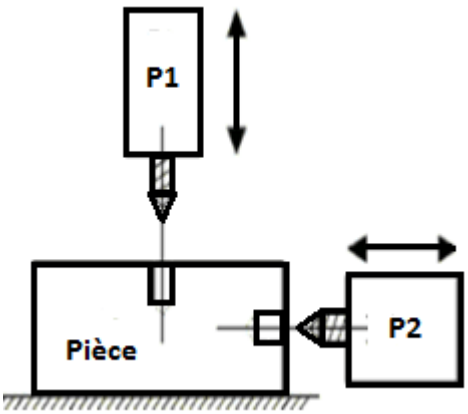
Une barrière automatique contrôle l’accès d’un parking. L’ouverture est autorisée si le conducteur présente une carte magnétique, une fois la barrière ouverte le conducteur dispose de 10 secondes pour entrer, passé ce délai la barrière se referme.
La position des capteurs est donnée ci dessous.



Construire le grafcet de point de vue commande.

Exercice 03: Deux perceuses

Deux perceuses effectuent chacune un perçage sur une même pièce simultanément comme le montre la figure ci contre.



La commande de mise en marche s'effectue grâce à un bouton poussoir.
L'avancement et le reculement de chaque perceuse sont effectués avec rotation.
Les différents capteurs et pré-actionneurs utilisés sont montrés par le tableau suivant :

Capteurs	Pré-actionneurs
Départ cycle : S1 Capteur de fin de course haut : S2 Capteur de fin de course bas : S3 Capteur de fin de course gauche : S4 Capteur de fin de course droite : S5	Rotation perceuse 1 : KM1 Rotation perceuse 2 : KM2 Avancer perceuse 1 : Y1+ Reculer perceuse 1 : Y1- Avancer perceuse 2 : Y2+ Reculer perceuse 2 : Y2-

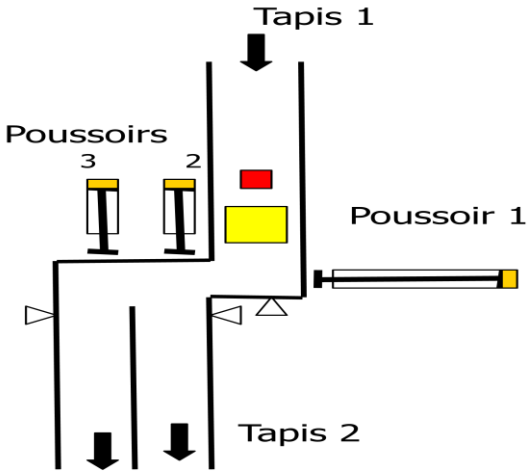
1- Construire le grafcet de point de vue opérative.
2- Construire le grafcet de point de vue commande.

Exercice 04 : Tri de caisses

Un dispositif automatique destiné à trier de deux tailles différentes se compose d'un tapis amenant les caisses, de trois poussoirs et deux tapis d'évacuation.
Le poussoir P1 pousse les petites caisses devant le poussoir P2 qui à son tour les transfère sur le tapis 2, alors que les grandes caisses sont poussées devant le poussoir P3, ce dernier les évacuant sur le tapis 3.

Pour effectuer la sélection des caisses, un dispositif de détection placé devant le poussoir P1 permet de reconnaître sans ambiguïté le type de caisse qui se présente (a=1 si petite caisse, b=1 si grande caisse).

Capteurs	Actionneurs
i1 :présence pièce a i2 :présence pièce b i10 :poussoir P1 rentré i11 : poussoir P1 position intermédiaire i12 : poussoir P1 sorti i20 ;poussoir P2 rentré i21 :poussoir P2 sorti i30 :poussoir P3 rentré i31 :poussoir P3 sorti	o10 : rentrer poussoir P1 o11 : sortir poussoir P1 o20 : rentrer poussoir P2 o21 : sortir poussoir P2 o30 : rentrer poussoir P3 o31 : sortir poussoir P3



Tracer le grafcet de point de vue commande.

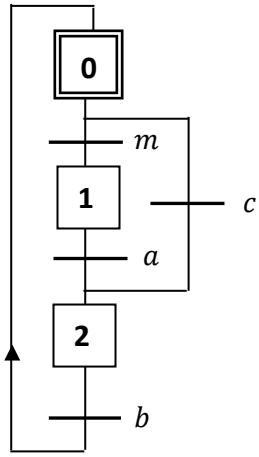
Exercice 05 : Unité de remplissage automatique de bouteille de gaz

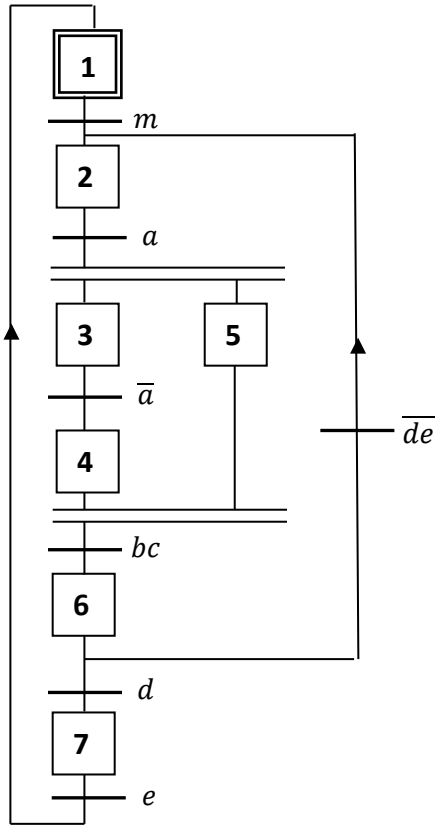
Pour le système technique donné en exercice 03-TD 01,

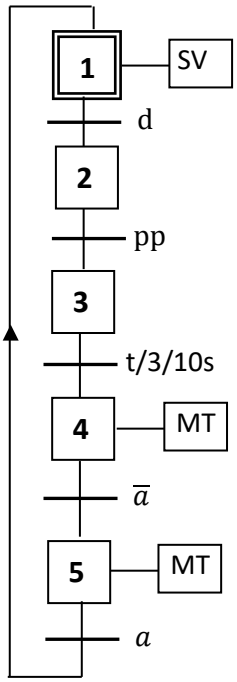
1- Construire le grafcet de point de vue système.
1- Construire le grafcet de point de vue opérative.
2- Construire le grafcet de point de vue commande.

Exercice 06

Traduire les grafcets suivants en des équations logiques







Architecture des API

IV.1. Introduction

Les Automates Programmables Industriels (API), ou en anglais, Programmable Logic Controller (PLC), sont apparus aux Etats-Unis vers 1969 (Modicon) où ils ont remplacés la logique a relais câblée et ils répondaient aux désirs des industries de l'automobile de développer des chaînes de fabrication automatisées qui pourraient suivre l'évolution des techniques et des modèles fabriqués.

Un automate programmable industriel est une machine électronique, programmable par un personnel non informaticien et destiné à commander au moyen de signaux d'entrées et de sorties, et d'un programme informatique, en temps réel, des procédés de processus industriels par un traitement séquentiel.

Ce chapitre d'écrit les automates programmables industriels. On débute par présenter l'environnement industriel et les différentes fonctions réalisés par les API. Ensuite, on définit les API compacts et modulaires. La structure interne des API est traitée par la suite. Les critères de choix d'un API sont donnés à la fin du chapitre.

IV.2. Avantages et inconvénients des API

Les automates programmables industriels, avec leur solution programmée, présentent de nombreux avantages par rapport à la technologie de logique câblée. Parmi ces intérêts, on cite :

- la simplicité car avec le même API, il devient possible de traiter une variété d'applications qui, autrement, auraient chaque fois requis des matériels différents.
- la flexibilité : car le changement du mode de fonctionnement de la machine commandée s'obtient par simple modification du programme enregistré en mémoire.
- la réduction de des coûts de câblage et de maintenance.
- la réduction de beaucoup d'espace requis pour l'installation.
- Les éléments qui les composent sont particulièrement robustes leur permettant de fonctionner dans des environnements particulièrement hostiles.
- Ils permettent d'assurer un temps d'exécution minimal, respectant un déterminisme temporel et logique, garantissant un temps réel effectif (le système réagit forcément dans le délai fixé).

En contrepartie, ils présentent les inconvénients suivants :

- Le prix est cher : le prix est notamment dépendant du nombre d'entrées/sorties nécessaires, de la mémoire dont on veut disposer pour réaliser le programme, de la présence ou non de modules métier.
- La connaissance des langages de programmation.

IV.3. Environnement des API

Les API se fonctionnent dans un environnement industriel qui se trouve en trois types :

- Environnement physique et mécanique : vibrations, chocs, humidité, température.
- Environnement chimique : gaz corrosifs (CI2, H2S,SO2), vapeurs d'hydrocarbures, poussières métalliques (fonderies, aciéries, ...), poussières minérales (cimenteries, ...).

➢ Environnement électrique : les éléments perturbateurs sont : les forces électromotrices (fem), thermoélectriques (effet Peltier, quelques mV), les parasites d'origine électrostatiques, les interférences électromagnétiques (transformateurs, postes de soudure, ...).

Ces environnements peuvent provoquer des dégâts non négligeables pour les appareils et des dangers pour les utilisateurs. Il faut donc veiller à la sécurité en analysant les risques et les normes en vigueur pour adapter les automates parfaitement à l'environnement industriel : Entrées/Sorties conformes aux standards de signaux industriels, protection contre les parasites électromagnétiques, tenue aux chocs et aux vibrations, résistance à la corrosion qui les contacts et provoque des courts-circuits, des filtres pour éliminer les poussières ou gaz et recouvrent d'un enduit les circuits imprimés, des dispositifs de ventilation Pour les température élevée, dispositifs de sécurité en cas de panne ou de chute de tension, …etc.

IV.4. Fonctions principales réalisées par API

L'automate programmable Industriel effectue les fonctions principales suivantes :

1. La détection, depuis des capteurs de tous types répartis sur la machine.
2. La commande d'actions, vers les actionneurs et pré-actionneurs
3. Le dialogue d'exploitation : des dialogues hommes-machine sont nécessaires pour la conduite de la machine, pour ses réglages et pour ses dépannages.
4. Le dialogue de supervision de production; Les automates offrent un écran de visualisation où l'on peut voir l'évolution des entrées / sorties
5. Le dialogue de programmation pour la première mise en œuvre.

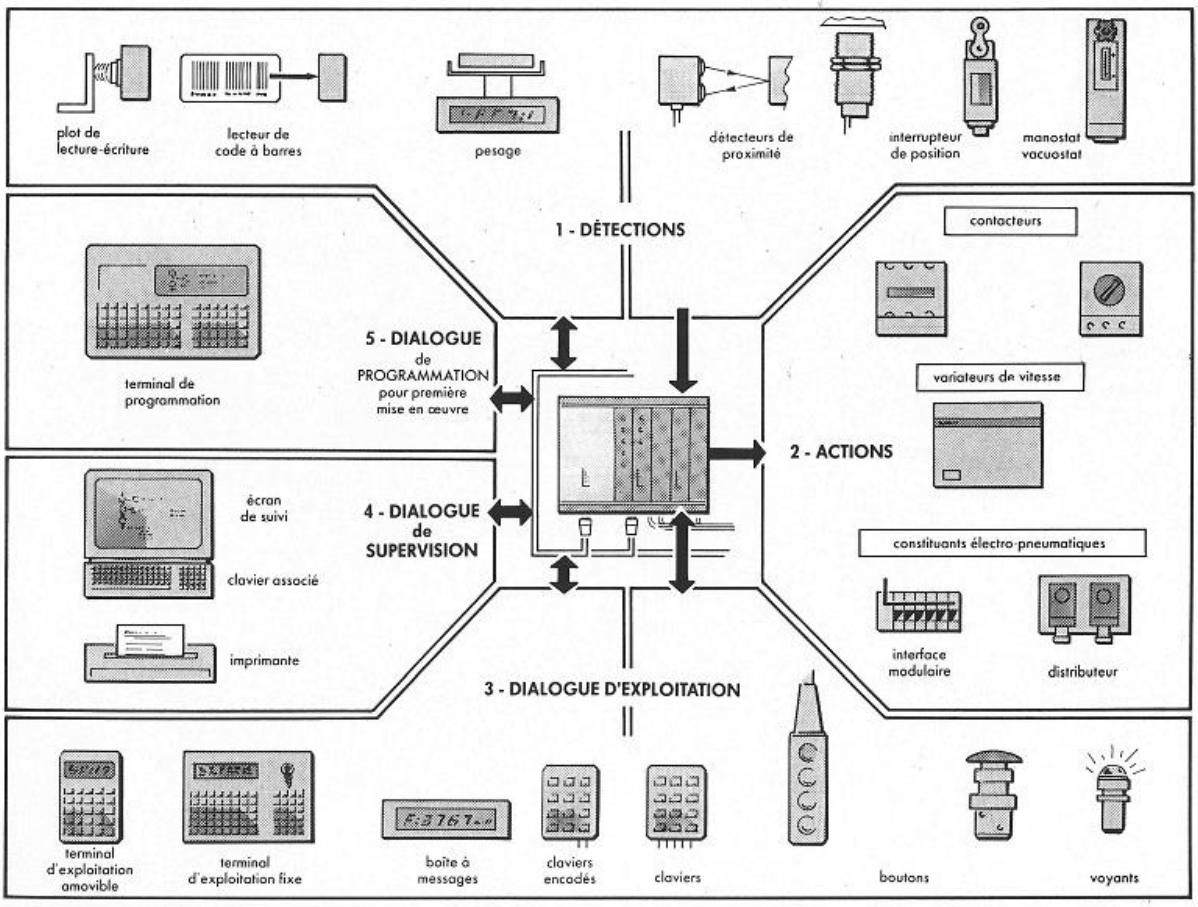


Figure IV.1. Fonctions principales réalisées par API.

IV.5. Aspect extérieur des API

Les automates programmables industriels peuvent être de type compact ou modulaire.

IV.5.1. Type compact (centralisé)

Il intègre le processeur, l'alimentation, les entrées et les sorties dans un seul boitier (rack). Selon les modèles et les fabricants, il pourra réaliser certaines fonctions supplémentaires (comptage, E/S analogiques ...) et recevoir des extensions en nombre limité. Ces automates, de fonctionnement simple, sont généralement destinés à la commande de petits automatismes.

SIEMENS LOGO	CROUZET MILLENIUM	SCHNEIDER ZELIO	SCHNEIDER TWIDO	MOELLER PS4
				

Figure IV.2. API compacts.

IV.5.2. Type modulaire

L'automate programmable se présente comme un ensemble de blocs fonctionnels. Généralement, chaque bloc est physiquement réalisé par un module spécifique (coffret, rack, baie ou cartes). Ces différents modules s'articulant autour d'un canal de communication: le bus interne. L'automate programmable est du type modulaire contenant un rack, un module d'alimentation, un processeur, des modules d'E/S, des modules de communication et de comptage. Cette organisation modulaire permet une grande souplesse de configuration pour les besoins de l'utilisateur, ainsi qu'un diagnostic et une maintenance facilités et elle destinée pour les automatismes complexes où puissance, capacité de traitement et flexibilité sont nécessaires.

SIEMENS S7-300	SCHNEIDER TSX 37	MOELLER	SCHNEIDER TSX 57
			

Figure IV.3. API modulaires.

IV.6. Structure interne des API

La structure interne d'un automate programmable industriel (API) est assez voisine de celle d'un système informatique simple. Cette structure comporte quatre parties principales : Une unité de traitement (un processeur CPU);Une mémoire ; des Interfaces et des modules d'entrées-sorties ;

Une alimentation. Un bus interne (liaisons parallèles) est utilisé pour échanger les informations entre les différents éléments de l'automate (entrées, sorties, mémoires).

IV.6.1. Processeur

Son rôle consiste d'une part à organiser les différentes relations entre la zone mémoire et les interfaces d'entrées et de sorties et d'autre part à exécuter les instructions du programme. Les instructions sont effectuées les unes après les autres, séquencées par une horloge.

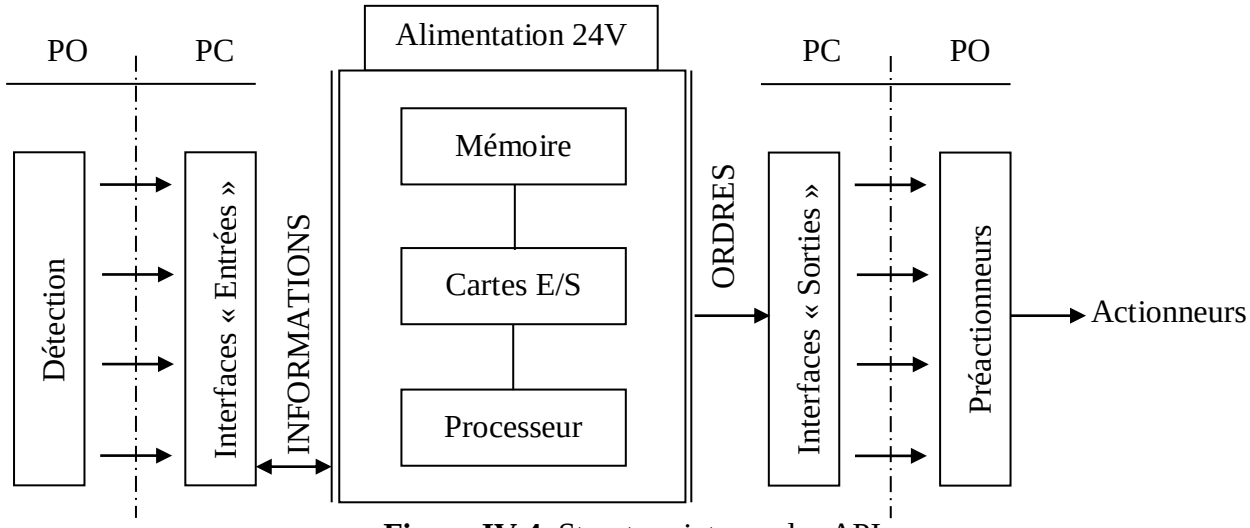


Figure IV.4. Structure interne des API.

IV.6.2. Mémoire

Elle est conçue pour :

- recevoir les informations issues des capteurs d'entrées
- recevoir les informations générées par le processeur et destinées à la commande des sorties.
- recevoir et conserver le programme du processus.

Il existe dans les automates trois types de mémoires qui remplissent des fonctions différentes :

• **Mémoire de programme :** Cette mémoire est utilisée pour stocké le programme. Elle est en général de type EEPROM (electrically erasable PROM : mémoires mortes reprogrammables effacement électrique).

• **Mémoire système :** Cette mémoire, présente dans le cas d'automates à microprocesseurs, est utilisée pour stocker le système d'exploitation et elle est programmée en usine par le constructeur. Elle peut donc sans problème être réalisée en technologie PROM (c'est-à-dire programmable une seule fois, sans possibilité d'effacement) voire ROM (mémoire morte accessible uniquement en lecture).

• **Mémoire de données** : Elle est utilisable en lecture-écriture des données pendant le fonctionnement. C'est une mémoire de type RAM (mémoire vive dans laquelle on peut lire, écrire et effacer) utilisant une technologie spéciale (CMOS) à très faible consommation électrique du moins, à l'état de repos et elle nécessite une batterie de sauvegarde.

IV.6.3. Interfaces et cartes d'Entrées / Sorties

Les entrées reçoivent des informations en provenance des éléments de détection (capteurs) et du pupitre opérateur (BP). Les sorties transmettent des informations aux pré-actionneurs (relais, électrovannes …) et aux éléments de signalisation (voyants) du pupitre. Le nombre de ces entrées est sorties varie suivant le type d'automate. Les cartes d'E/S ont une modularité de 8, 16 ou 32 voies. Les tensions disponibles sont normalisées (24, 48, 110 ou 230V continu ou alternatif ...).

L'interface réalise trois fonctions principales :
• Le découplage mécanique (borniers à vis par exemple) entre le câblage processus et le câblage interne de l'automate.
• Le découplage électrique (isolation galvanique) : Le problème est de se protéger contre les tensions de mode commun existant non seulement entre les signaux d'entrée et l'automate mais aussi entre les signaux d'entrée eux-mêmes.
• L'adaptation des niveaux de tensions (Par exemple, atténuer les entrées haut niveau hors standards, amplifier les entrées bas niveau, effectuer la transformation courant/tension)
• La conversion analogique/numérique.
• Filtrage des signaux parasites : Elimination des parasites industriels de fréquence supérieure à celles du signal utile.
• La synchronisation des transferts conformément aux procédures d'échange du BUS de l'automate.

IV.6.4. Alimentation électrique

Tous les automates actuels sont équipés d'une alimentation 240 V 50/60 Hz, 24 V DC. Les entrées sont en 24 V DC et une mise à la terre doit également être prévue.

IV.6.5. Modules complémentaires (spéciaux)

Les automates compacts permettent de commander des sorties en T.O.R et gèrent parfois des fonctions de comptage et de traitement analogique. Les automates modulaires permettent de réaliser de nombreuses autres fonctions grâce à des modules intelligents que l'on dispose sur un ou plusieurs racks. Ces modules ont l'avantage de ne pas surcharger le travail de la CPU car ils disposent bien souvent de leur propre processeur.

- **Principales fonctions**

• **Cartes de comptage rapide**: elles permettent d'acquérir des informations de fréquences élevées incompatibles avec le temps de traitement de l'automate. (Signal issu d'un codeur de position).
• **Cartes d'entrées / sorties analogiques**: Elles permettent de réaliser l'acquisition d'un signal analogique et sa conversion numérique (CAN) indispensable pour assurer un traitement par le microprocesseur. La fonction inverse (sortie analogique) est également réalisée. Les grandeurs analogiques sont normalisées : 0-10V ou 4-20mA.
• **Cartes de communication (RS485, Ethernet ...)** : Ils permettent d'établir des communications à distance avec d'autres systèmes de traitement par lignes séries: paires téléphoniques, coaxes, fibres optiques, ...
• **Cartes d'entrées / sorties déportées**: ils permettant de décentraliser des châssis entrées / sorties industrielles sur des distances importantes (ordre du km). Cette possibilité de décentralisation permet, dans de nombreux cas, de réduire substantiellement le volume de câblage entre le processus et l'automate.

- **Autres cartes**

•Cartes de régulation PID.
•Cartes de commande d'axe.
•Cartes de pesage.
•Cartes de surveillance et de contrôle.

IV.7. Critères de choix des API

Les critères de choix essentiels d'un automate programmable industriel sont :

• Le nombre et la nature des E/S ;
• Les capacités de traitement du processeur (vitesse, données, opérations, temps réel...).
• Fonctions ou modules spéciaux
• Les moyens de dialogue et le langage de programmation ;

•La communication avec les autres systèmes ;
•Les moyens de sauvegarde du programme ;
• La fiabilité, robustesse, immunité aux parasites ;
•La documentation, le service après vente, durée de la garantie, la formation.

Programmation des API

V.1. Introduction

La réalisation de tout ou partie d'une partie commande en logique programmée nécessite la traduction du modèle concerné (grafcet, équations, ...) en programme exécutable par l'API. L'élaboration d'un tel programme vise donc à écrire les équations d'activation de sorties de l'API et les conditions associées en un langage répond à la norme industrielle de la commission électrotechnique internationale IEC 1131-3. Cette norme des langages de programmation des automates programmables permet de définir cinq langages de programmation utilisables, qui sont:

Schéma à contacts (LD : Ladder Diagram)
Schéma par blocs (FBD : Function Block Diagram)
Langage SFC (Sequential Function Chart)
Langage liste d'instructions (IL : Instruction List)
Langage littéral structuré (ST : Structured Text).

Les langages LD, FBD et IL, sont disponibles pour toutes les gammes d'automates, alors que le ST et le SFC sont uniquement disponibles pour les modèles S7-300/400, WINAC et S7-1500.

Pour programmer l'automate, l'automaticien peut utiliser une console de programmation (portable) ou un PC doté d'un logiciel de programmation (exemple : Step 7 pout les API Siemens ou PL7 pour Schneider).

Pour envoyer le programme réalisé dans la mémoire de l'automate on utilise une liaison série entre l'automate et l'ordinateur ou un câble spécifique lors de l'utilisation d'une console.

Dans ce chapitre on présente le cycle d'exécution d'un paragramme par l'automate programmable industriel, en premier lieu. En second lieu on définit la notion d'dressage et affectation. En suite, on décrit les différents langages de programmation des API. Par la suite, on s'intéresse à la programmation des différentes opérations en langage à contact. La programmation en langages par blocs et par liste d'instructions est donnée en bref à la fin du chapitre.

V.2. Traitement du programme automate

Un automate exécute son programme de manière cyclique comme suit :

- **Traitement interne** : L'automate effectue des opérations de contrôle et met à jour certains paramètres systèmes (détection des passages en RUN / STOP, mises à jour des valeurs de l'horodateur, ...).

- **Lecture des entrées** : L'automate reçoit des données par ses entrées et les recopie dans la mémoire image des entrées.

- **Traitement du programme** : L'automate traite les données entrantes par un programme défini dans sa mémoire et écrit les sorties dans la mémoire image des sorties.

- **Ecriture des sorties** : L'automate bascule les différentes sorties aux positions définies dans la mémoire image des sorties pour commander les actionneurs et dialoguer avec l'opérateur et les autres systèmes automatisés.

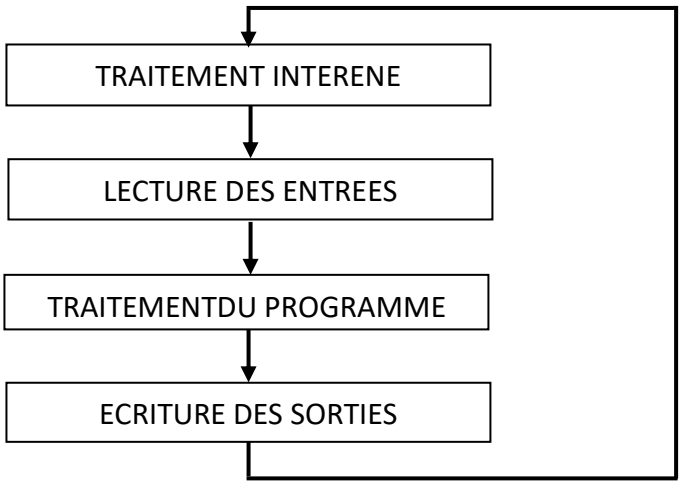


Figure V.1. Cycle de fonctionnement d'un API.

On appelle scrutation l'ensemble des quatre opérations réalisées par l'automate et le temps de scrutation est le temps mis par l'automate pour traiter la même partie de programme. Ce temps varie selon la taille du programme et la puissance de l'automate et il est de l'ordre de la dizaine de millisecondes pour les applications standards.

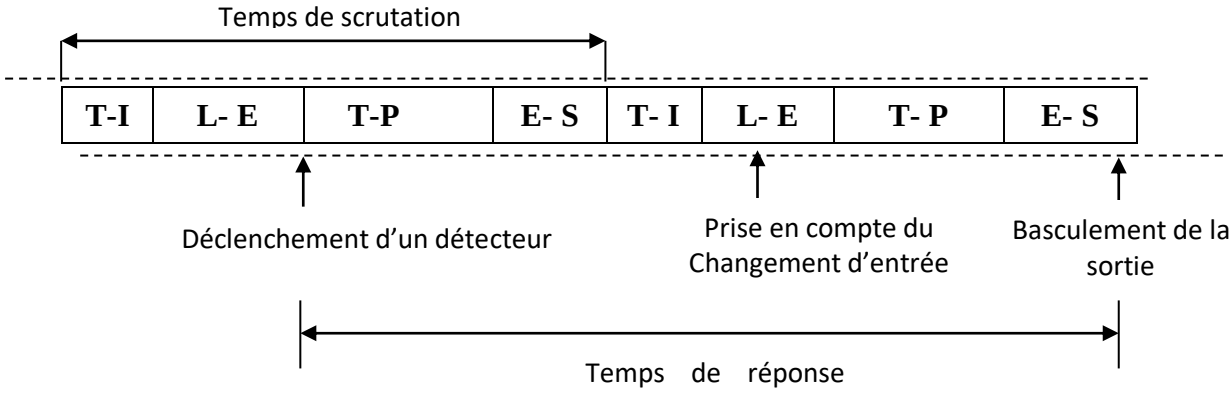


Figure V.2. Temps de scrutation et de réponse total.

Le temps de réponse total est le temps qui s'écoule entre le changement d'état d'une entrée et le changement d'état de la sortie correspondante : Le temps de réponse total est au plus égal à deux fois le temps de scrutation (sans traitement particulier). Ce temps est de l'ordre d'une vingtaine de millisecondes et est contrôlé par une temporisation appelée chien de garde.

V.3. Affectation et Adressage

Avant de commencer à programmer, il est très important de réaliser l'affectation des entrées et des sorties de l'automate, et connaitre son adressage.

L'affectation des entrées est en fonction des capteurs et boutons de commandes utilisées. Les sorties sont affectées avec les pré-actionneurs, voyants ...etc. L'affectation consiste également à identifier ces variables destinées à mémoriser les états et valeurs intermédiaires durant l'exécution du programme.

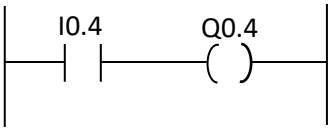
L'adressage consiste à identifier les variables d'E/S et les variables internes par des adresses. La notion d'adressage d'un automate permet de connaître le type d'objet, le format des données que l'on va pouvoir former et son emplacement (ou numéro). Le tableau I.1 montre la signification des différents symboles que l'on peut trouver dans une adresse.

Préfixe	Signification
I (ou E)	Lecture de l'état d'une entrée.
Q (ou A)	Lecture/ Ecriture de l'état d'une sortie.
M et V	Lecture/ Ecriture de l'état d'une variable interne (mémento)
SM	Lecture/ Ecriture d'un bit Système
S	Lecture/Ecriture d'un bit Relai séquentiel
C ou (Z)	Compteurs
SC	Compteurs rapides
T	Temporisateurs
A ou (P)	Analogique
%	Norme IEC 1131-3
B	Taille d'un byte ou octet
W	Taille d'un word : mot de 16 bits
D	Taille d'un double word : mot double de 32 bits

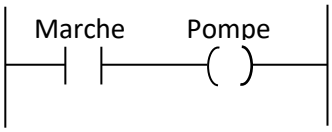
Tableau V.1. Signification des différents symboles d'une adresse.

Exemples:
I0.4 : bit d'entrée 4 module 0.
IW125 : mot d'entrée 125.
Q0.4: bit de sortie 4 module 1.
QB17 : Octet de sortie 17.
M12.15 : bit mémoire interne 12.15.
MD48 : mot double mémoire interne 48.
AIW 288 : Entrée Analogique 288.
AQW304 : Sortie Analogique 304.

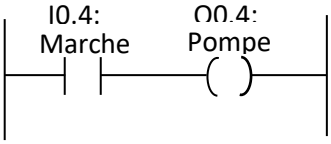
Remarque 1 : l'adressage peut être absolu ou symbolique. L'adressage absolu revient à utiliser les adresses physiques de l'automate. Pour donner de la clarté au programme, on utilise l'adressage symbolique. Cela revient à substituer chaque adresse physique par une adresse symbolique qui représente au mieux la fonction. La table mnémonique d'affectation est « l'outil » utilisée pour créer un adressage symbolique.



Adressage absolu.



Adressage symbolique.



Adressage absolu et symbolique.

Remarque 2 : le mode d'adressage dépend du constructeur de l'API. Il faut donc consulter la notice de cet automate.

V.4. Langages de programmation pour API

Les langages de programmation des API sont de natures diverses étant donné la diversité, des utilisateurs pouvant les utiliser. Ils peuvent être classés en deux familles :

V.4.1. Langages graphiques

Ils permettent une transcription graphique aussi directe que possible des modèles afin de faciliter les tâches de programmation et de réduire les sources d'incertitudes. On distingue les trois langages suivants : Schéma à contact, Schéma par blocs et Diagramme fonctionnel de séquence.

- **Schéma à contacts**: Le **LD** est le plus utilisé et très facile à prendre à main et idéale pour visualiser et diagnostiquer des programmes pendant les opérations de maintenance. Il est adapté au traitement combinatoire.

➢ **Schéma par blocs**: Le **FBD** se présente sous forme un diagramme pour réaliser des variables, des opérations ou des fonctions, simples ou très sophistiquées. Les blocs sont programmés (bibliothèque) ou programmables.

➢ **Diagramme fonctionnel de séquence**: Le **SFC** permet de représenter graphiquement et de façon structurée le fonctionnement d'un automatisme séquentiel. Il est dérivé du grafcet.

V.4.2. Langages textuels

Il s'agit de décrire le fonctionnement du processus par un programme sous forme un texte. Deux langage existent : Liste d'instruction et littéral structuré.

➢ **Liste d'instructions** : Le **IL** est un langage "machine" qui permet l'écriture de traitements logiques et numériques. Il peut être comparé au langage assembleur. Très peu utilisé par les automaticiens.

➢ **Littéral structuré** : Le **ST** est un langage de type "informatique" permettant l'écriture structurée (algorithmes) de traitements logiques et numériques. Il est de même nature que le Pascal. Peu utilisé par les automaticiens.

V.5. Programmation en langage à contact

Le langage à relais (relais) est basé sur un symbolisme très proche de celui utilisé pour les schémas électriques. Il est constitué de plusieurs de plusieurs réseaux. Chaque réseau contient des lignes horizontales contenant des contacts, des blocs fonctionnels et de bobines entre deux barres d'alimentation. Les contacts permettent de lire la valeur d'une variable booléenne. Les blocs fonctionnels sont des blocs préprogrammés qui permettent de réaliser des fonctions avancées : temporisation, comptage, communication, ...etc. Les bobines permettent d'écrire la valeur d'une variable booléenne. L'évaluation de chaque réseau se fait de la gauche vers la droite. L'évaluation de l'ensemble des réseaux se fait du haut vers le bas. Chaque réseau est repéré par une étiquette (LABEL).

Dans ce suit, on présente les jeux d'opérations, les plus utilisées, SIMATIC (norme de Siemens), pour les automates Siemens S200. La programmation est s'effectuée à l'aide du logiciel Step7-Micro/Win version 4.

Remarque 3 : La représentation des jeux d'opérations dépend de l'automate, de la norme et de la version du logiciel utilisé en programmation.

V. 5.1. Opérations combinatoires sur bits

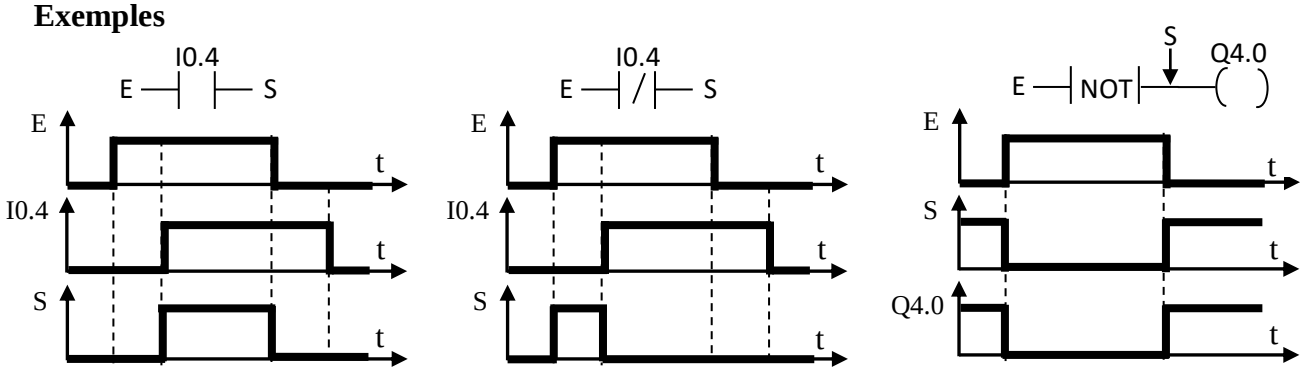
Le tableau ci-dessous montre les principales opérations combinatoires sur bits.

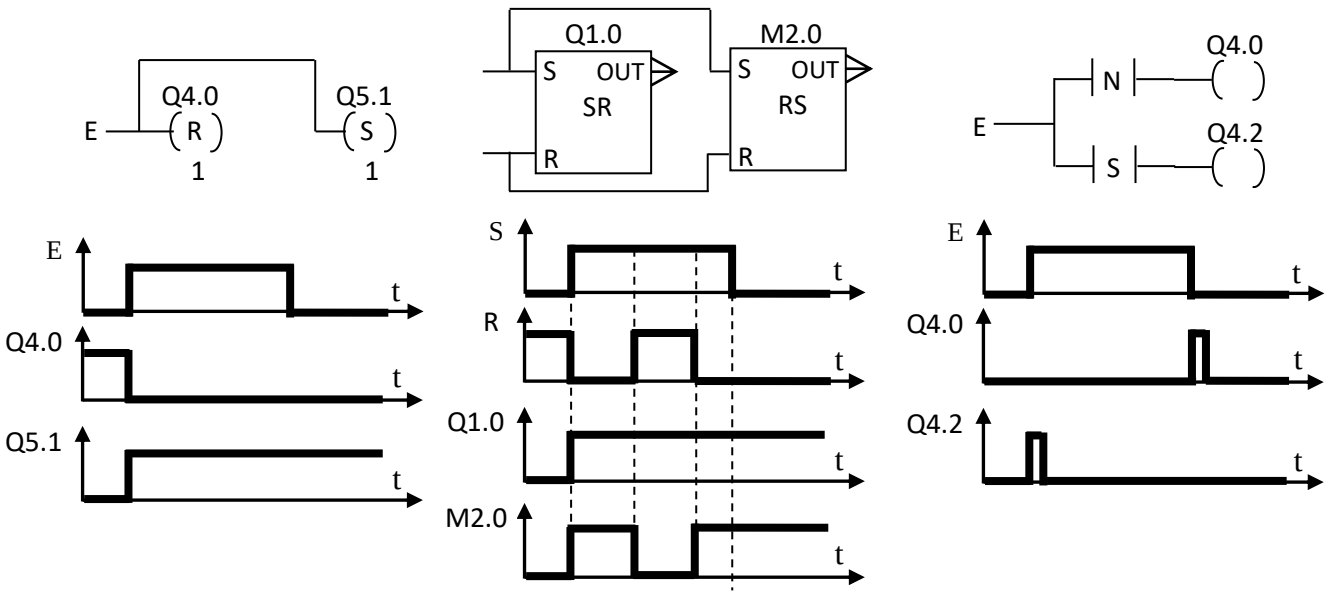
Opération	Graphe	Commentaire
Contact à fermeture	<opérande> ⊣ ⊢	Le contact est fermé si la valeur du bit interrogé sauvegardée en <opérande> égale 1. En revanche, si le bit <opérande> est 0, le contact est ouvert. Le courant traverse le contact et l'opération fournit un résultat logique (RLG) égal à 1 **si et seulement si** le contact est fermé et le RLG à son entée égal 1.
Contact à ouverture	<opérande> ⊣/⊢	Le contact est fermé si la valeur du bit interrogé sauvegardée en <opérande> est 0. En revanche, si l'état de signal en <opérande> est 1, le contact est ouvert. Le courant traverse le contact et l'opération fournit un résultat logique (RLG) égal à 1 **si et seulement si** le contact est fermé et le RLG à son entée égal 1.

<table>
<tr><td>Inverser RLG</td><td>—| NOT |—</td><td>Cette opération inverse le bit de résultat logique (RLG).</td></tr>
<tr><td>Bobine de sortie</td><td>< opérande >
—( )—</td><td>Cette opération fonctionne comme une bobine dans un schéma à relais. Si l'énergie atteint la bobine (RLG = 1), le bit en <opérande> est mis à 1. Si l'énergie n'atteint pas la bobine (RLG = 0), le bit en <opérande> est mis à 0.</td></tr>
<tr><td>Mettre à 0</td><td>< opérande >
—(R)—
N</td><td>Cette opération ne s'exécute que si le RLG des opérations précédentes a la valeur 1. Si l'énergie atteint la bobine (RLG égale 1), l'opération met les N bits internes à 0, en commençant de l'adresse de l'opérande indiqué.</td></tr>
<tr><td>Mettre à 1</td><td>< opérande >
—(S)—
N</td><td>Cette opération ne s'exécute que si le RLG des opérations précédentes a la valeur 1. Dans ce cas, les N bits sont mis à 1, en commençant de l'adresse de l'opérande précisé.</td></tr>
<tr><td>Bascule RS (mise à 1, mise à 0)</td><td>< opérande >
S OUT
RS</td><td>Cette opération s'exécute comme suit :
<table>
<tr><td>S</td><td>R</td><td>Opérande (out)</td></tr>
<tr><td>0</td><td>0</td><td>L'opérande inchangé</td></tr>
<tr><td>0</td><td>1</td><td>mise à zéro du bit de l'opérande</td></tr>
<tr><td>1</td><td>0</td><td>Mise à 1 du bit de l'opérande</td></tr>
<tr><td>1</td><td>1</td><td>Mise à 1 puis mise à 0. L'opérande reste donc à 0</td></tr>
</table></td></tr>
<tr><td>Bascule SR (mise à 0, mise à 1)</td><td>< opérande >
S OUT
SR</td><td>Cette opération s'exécute comme suit :
<table>
<tr><td>S</td><td>R</td><td>Opérande (out)</td></tr>
<tr><td>0</td><td>0</td><td>L'opérande inchangé</td></tr>
<tr><td>0</td><td>1</td><td>mise à zéro du bit de l'opérande</td></tr>
<tr><td>1</td><td>0</td><td>Mise à 1 du bit de l'opérande</td></tr>
<tr><td>1</td><td>1</td><td>Mise à 0 puis mise à 1. L'opérande reste donc à 1</td></tr>
</table></td></tr>
<tr><td>Détecter de front descendant</td><td>—| N |—</td><td>Cette opération détecte le passage de 1 à 0 de l'état de signal du RLG avant l'opération et montre cette transition avec un RLG égal à 1 sous forme une impulsion. Après l'impulsion, le résultat logique est 0.</td></tr>
<tr><td>Détecter front montant</td><td>—| P |—</td><td>Cette opération détecte la transition de 0 à 1 de l'état de signal du RLG, avant l'opération, et montre cette transition avec un RLG égal à 1 sous forme une impulsion. Après l'impulsion, le résultat logique est 0.</td></tr>
</table>

Tableau V.2. Opérations combinatoires sur bits.

Exemples

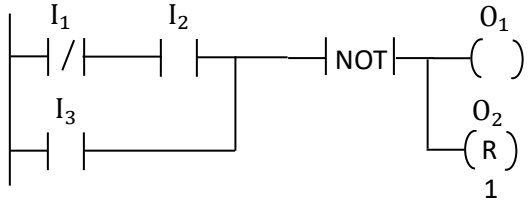




V.5.2. Association de contacts et de bobines

- **Contacts en série** : L'association de contacts en série permet de réaliser des « ET » logiques.
- **Contacts en parallèle** : L'association de contacts en parallèle permet de réaliser des « OU » logiques.
- **Bobines en série** : L'association de bobines en série n'est pas possible.
- **Bobines en parallèle** : L'association de bobines en parallèle permet de commander plusieurs bobines par la même équation logique.

Exemple : Réaliser la fonction logique suivante : $\overline{O}_1 = \overline{I}_1 . I_2 + I_3$, et mettre la sortie O_2 à 0 si la fonction O_1 est nulle.

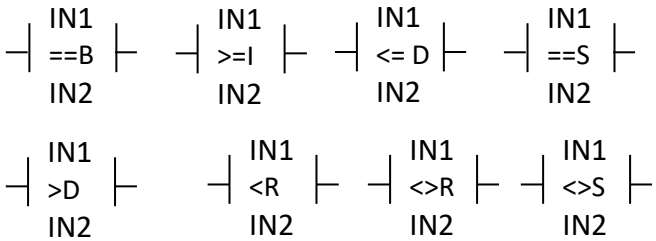


La sortie O_1 est égale 1 et la sortie O_2 est remise à 0,

V.5.3. Opérations de comparaison

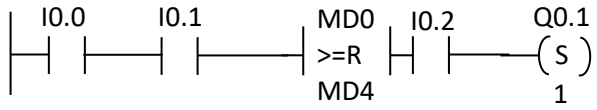
Les opérations de comparaison comparent les entrées **IN1** et **IN2** selon les types de comparaison suivants :

- == : IN1 égal à IN2.
- <> : IN1 différent de IN2.
- > : IN1 supérieur à IN2.
- < :IN1 inférieur à IN2.
- >= : IN1 supérieur ou égal à IN2.
- <= : IN1 inférieur ou égal à IN2.



- Les symboles B, I, D, R et S signifient que le format de données des entrées IN1 et IN2 est octet, entier, double entier, réel et ASCII, respectivement.
- Les opérations de comparaison d'octets ne sont pas signées. Les autres sont signées.
- Ces opérations ne s'exécutent que si le RLG des opérations précédentes a la valeur 1.
- Si la comparaison est vraie, le résultat logique (RLG) de sortie est 1 (le contact est fermé).

Exemple

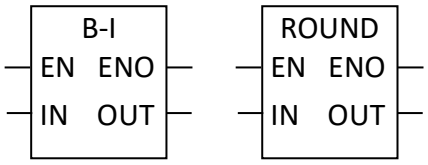


La sortie Q0.1 est mise à 1 si l'état de signal est 1 aux entrées I0.0 ET I0.1, ET si MD0 >= MD4 ET si l'état de signal I0.2 est 1 à l'entrée.

V.5.4. Opérations de conversion

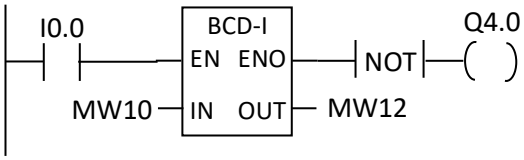
Les opérations de conversion lisent le contenu du paramètre d'entrée IN puis le convertissent. Le résultat est rangé dans le paramètre de sortie OUT. Ces opérations ne s'exécutent que si l'entrée EN a la valeur 1. La sortie ENO prend la valeur 1 si l'opération est réalisée.

Parmi ces opérations, Vous disposez les suivantes :



- B_I : Convertir octet en entier de 16 bits,
- I_B : Convertir entier de 16 bits en octet,
- BCD_I : Convertir nombre DCB en entier de 16 bits,
- I_BCD : Convertir entier de 16 bits en nombre DCB,
- I_DI : Convertir entier de 16 bits en entier de 32 bits
- ROUND : Arrondir : convertit une valeur réelle IN en nombre entier de 32 bits
- TRUNC : Tronquer : convertit un nombre réel IN en nombre entier de 32 bits et place la partie entière du résultat dans la variable indiquée par OUT.

Exemple

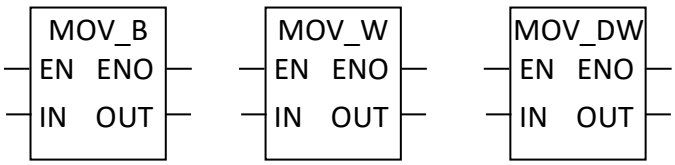


Si l'état de signal est 1 à l'entrée I0.0, le contenu du mot de mémento MW10 est lu comme nombre DCB à trois chiffres et converti en nombre entier de 16 bits. Le résultat est rangé dans le mot de mémento MW12. La sortie Q4.0 est mise à 1 si la conversion n'est pas exécutée (ENO = 0).

V 5.5. Opérations de transfert

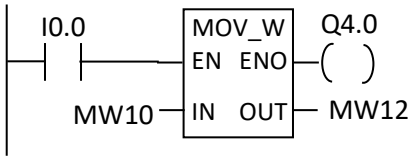
Cette opération est activée par l'entrée de validation EN. La valeur indiquée dans l'entrée IN est copiée à l'adresse précisée dans la sortie OUT. L'état de signal de ENO est identique à celui de EN.

Vous disposez des opérations de transfert suivantes :



- MOV-B: Transfert d'un octet,
- MOV-W : Transfert d'un mot,
- MOV-D : Transfert d'un double mot,
- MOV-R : Transfert d'un réel.

Exemple



L'opération est exécutée si I0.0 est à 1. Le contenu de MW10 est alors copié dans le mot de données 12 du bloc de données en cours. La sortie A 4.0 est mise à 1 si l'opération est exécutée.

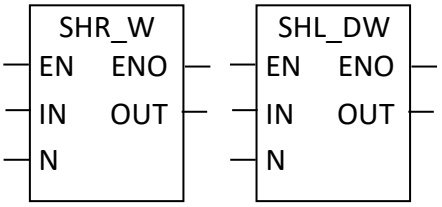
V 5.6. Opérations de décalage et de rotation

Les opérations de décalage permettent de décaler bit par bit le contenu de l'entrée IN vers la gauche ou vers la droite. Le nombre de bits de décalage est donné par l'entrée N.

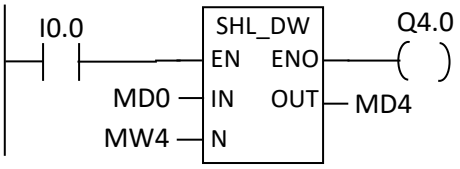
Vous disposez des opérations de décalage suivantes :

- SHR_B : Décalage vers la droite d'un octet.
- SHR_W : Décalage vers la droite d'un mot.

- SHR_DW : Décalage vers la droite d'un double mot.
- SHL_B : Décalage vers la gauche d'un octet.
- SHL_W : Décalage vers la gauche d'un mot.
- SHL_DW : Décalage vers la gauche d'un double mot.



Exemple

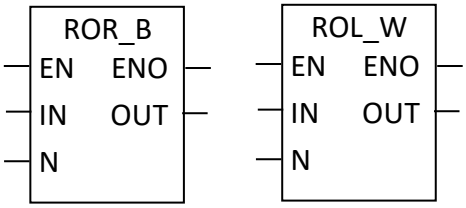


L'opération SHL_DW est exécutée si l'état de signal est 1 à l'entrée I0.0. Le double mot de mémento MD0 est chargé et décalé vers la gauche du nombre de bits précisé dans MW4. Le résultat (double mot) est rangé dans MD4.

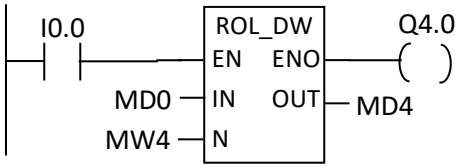
Les opérations de rotation permettent d'effectuer la rotation bit par bit vers la droite ou vers la gauche du contenu de l'entrée IN. Le nombre de bits de rotation est précisé dans le paramètre d'entrée N.

Vous disposez des opérations de rotation suivantes :

- ROR_B : Rotation vers la droite d'un octet.
- ROR_W : Rotation Décalage vers la droite d'un mot.
- ROR_DW : Rotation vers la droite d'un double mot
- ROL_B : Rotation vers la gauche d'un octet.
- ROL_W : Rotation vers la gauche d'un mot.
- ROL_DW : Rotation vers la gauche d'un double mot.



Exemple

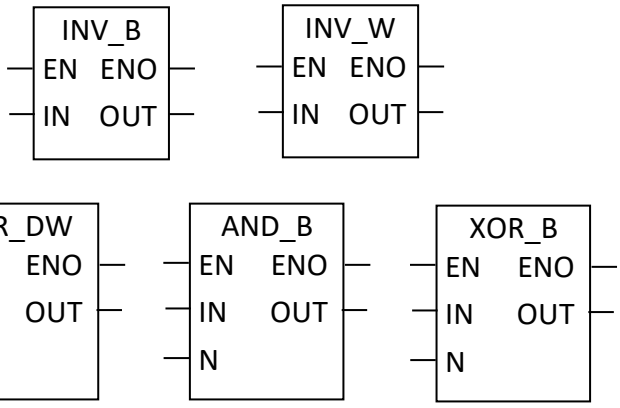


L'opération ROL_DW est exécutée si l'état de signal est 1 à l'entrée I0.0. Le double mot de mémento MD0 est chargé et fait l'objet d'une rotation vers la gauche du nombre de bits précisé dans MW4. Le résultat (double mot) est rangé dans MD4. La sortie Q4.0 est mise à 1.

V. 5.7. Opérations logiques

Les opérations logiques permettent sur nombres permettent d'exécuter les fonctions logiques suivantes sur nombre et deux nombres :

- INV_B : Inverser octet, c.-à-d., former le complément à un.
- INV_W : Inverser mot.
- INV_D : Inverser double mot.
- AND_B: ET octet
- AND_W : ET mot
- AND_DW : ET double mot
- OR_B : OU octet
- OR_W : OU mot
- OR_D : OU double mot
- XOR_B : OU exclusif octet
- XOR_W : OU exclusif mot
- XOR_D : OU exclusif double mot

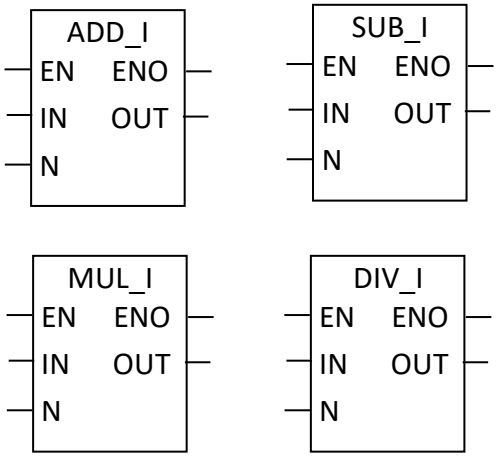


V. 5.8. Opérations arithmétiques

Les opérations arithmétiques sur nombres permettent d'exécuter les fonctions arithmétiques suivantes sur deux nombres:

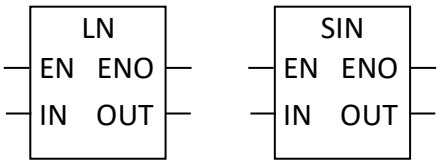
❖ Opérations Additionner, Soustraire, Multiplier et Diviser

- ADD_I Additionner entiers de 16 bits
- SUB_I Soustraire entiers de 16 bits
- MUL_I Multiplier entiers de 16 bits
- DIV_I Diviser entiers de 16 bits
- ADD_DI Additionner entiers de 32 bits
- SUB_DI Soustraire entiers de 32 bits
- MUL_DI Multiplier entiers de 32 bits
- DIV_DI Diviser entiers de 32 bits.
- ADD_R Additionner réels de 32 bits.
- SUB_R Soustraire réels de 32 bits.
- MUL_R Multiplier réels de 32 bits.
- DIV_R Diviser réels de 32 bits.



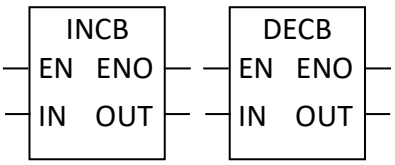
❖ Opérations numériques

- SQRT Racine carrée d'un nombre réel.
- LN Logarithme naturel d'un nombre réel.
- EXP Valeur exponentielle sur la base d'un nombre réel.
- Sinus (SIN), Cosinus (COS) et Tangente (TAN).
- PID

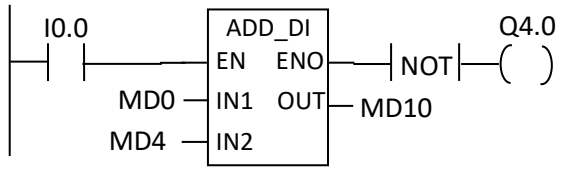


❖ Opérations d'incrémentation (+1) et de décrémentation (-1)

- INC_B et DEC_B : Incrémenter octet et Décrémenter octet.
- INC_W et DEC_W : Incrémenter mot et Décrémenter mot.
- INC_DW et DEC_DW : Incrémenter octet et Décrémenter double mot.



Exemple

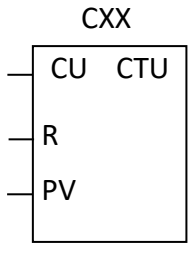


L'opération ADD_DI est exécutée si l'état de signal est 1 à l'entrée I0.0. Le résultat de l'addition MD0 + MD4 est rangé dans le double mot de mémento MD10. Si le résultat est hors de la plage autorisée pour un nombre entier de 32 bits ou si l'état de signal est 0 à l'entréeI0.0, la sortie Q4.0 est mise à 1.

V. 5.9. Opérations de comptage

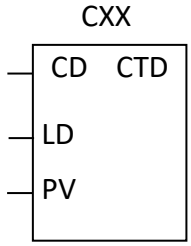
➢ Compteur incrémental (CTU)

L'opération Compteur incrémental incrémente en partant de la valeur en cours à chaque front montant de l'entrée d'incrémentation CU. Lorsque la valeur en cours "Cxx" est supérieure ou égale à la valeur prédéfinie PV, le bit de compteur Cxx est activé. Le compteur est remis à zéro lorsque l'entrée de remise à zéro R est activée. Le compteur incrémental arrête le comptage lorsqu'il atteint la valeur maximale 32 767. La valeur courante de CXX est de type entier de 16bits.



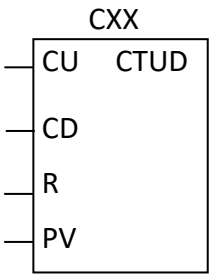
➢ Compteur décrémental (CTD)

L'opération Compteur décrémental décrémente en partant de la valeur en cours à chaque front montant de l'entrée de décrémentation CD. Lorsque la valeur en cours Cxx est égale à zéro, le bit de compteur Cxx est activé. Le compteur remet le



bit de compteur Cxx à 0 et charge la valeur prédéfinie PV dans la valeur en cours lorsque l'entrée de chargement LD est activée. Le compteur s'arrête lorsqu'il atteint zéro et le bit de compteur Cxx est alors mis à 1. La valeur courante de CXX est de type entier.

- **Compteur incrémental/décrémental (CTUD)**

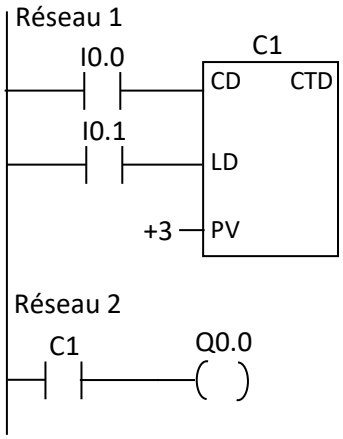


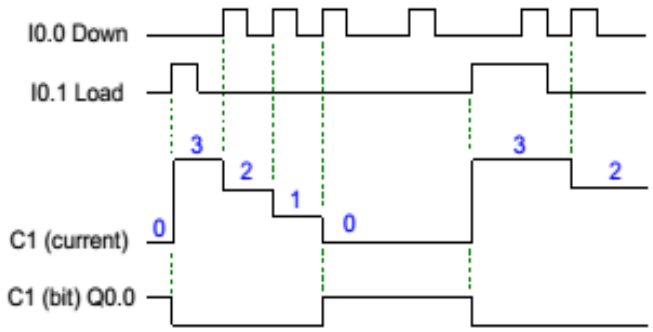
L'opération Compteur incrémental/décrémental incrémente en partant de la valeur en cours à chaque front montant de l'entrée d'incrémentation CU et décrémente à chaque front montant de l'entrée de décrémentation CD. La valeur en cours Cxx du compteur contient le décompte en cours. La valeur prédéfinie PV est comparée à la valeur en cours à chaque exécution de l'opération de comptage.

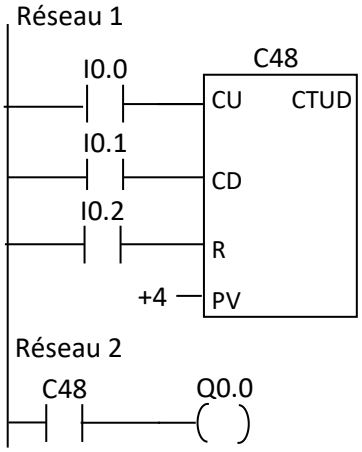
Lorsqu'il atteint la valeur maximale de 32 767, le front montant suivant à l'entrée d'incrémentation fait prendre à la valeur en cours la valeur minimale de -32 768. Lorsque la valeur minimale -32 768 est atteinte, le front montant suivant à l'entrée de décrémentation fait prendre à la valeur en cours la valeur maximale de 32 767.

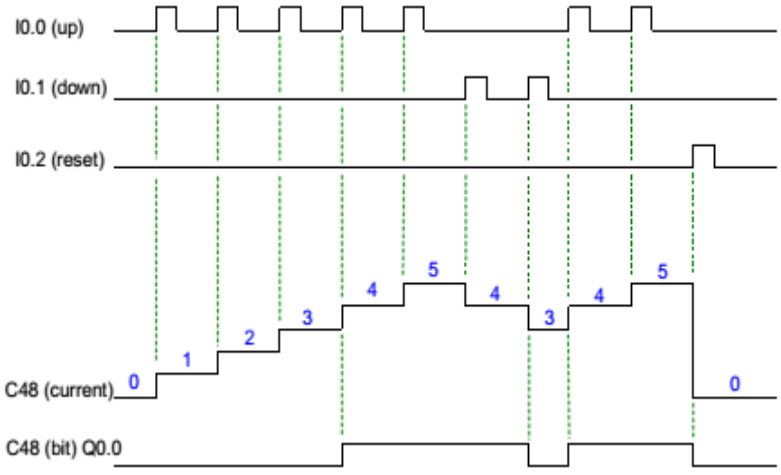
Lorsque la valeur en cours Cxx est supérieure ou égale à la valeur prédéfinie PV, le bit de compteur Cxx est activé. Sinon, le bit de compteur est désactivé. Le compteur est remis à zéro lorsque l'entrée de remise à zéro R est activée ou que l'opération "Mettre à 0" est exécutée. La valeur courante de CXX est de type entier.

Exemples



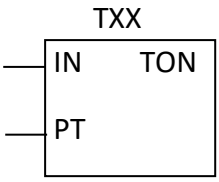






V. 5.10. Opérations de temporisation

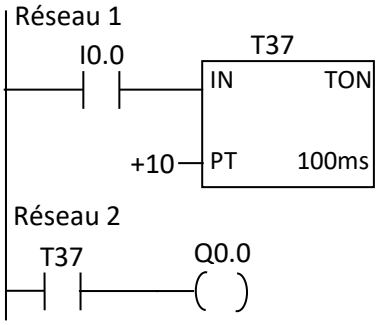
➢ Démarrer temporisation sous forme de retard à la montée (TON)

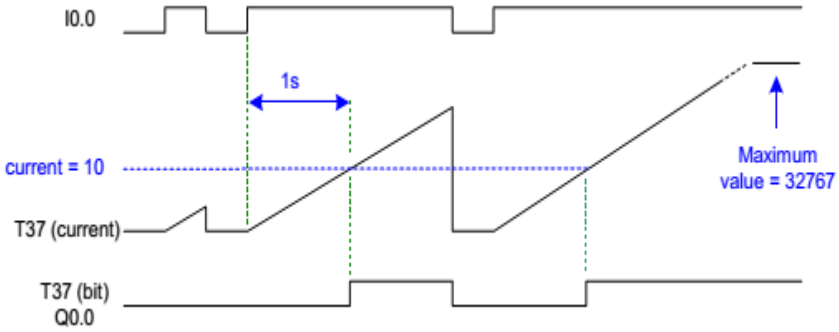


Cette opération sert à retarder l'activation d'une sortie (le bit du temporisateur) pour un intervalle de temps donné après que l'entrée (IN) a été activée.

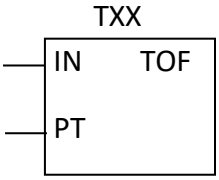
- La temporisation démarre en cas de front montant à l'entrée de démarrage IN.
- La temporisation s'initialise si l'état de signal à l'entrée IN passe de 1 à 0 alors que la temporisation s'exécute.
- La temporisation TON poursuit le comptage, une fois la valeur prédéfinie atteinte, Jusqu' à la valeur maximale (32 767).
- L'état de signal à la sortie égale 1 lorsque si la valeur en cours ≥ la valeur prédéfinie. Dans les autres cas, la sortie prend la valeur 0.
- L'intervalle de temps est déterminé par le produit de la résolution de la temporisation par la valeur de l'entrée PT (type entier).
- Le numéro de la temporisation (Txx) détermine la résolution de la temporisation : 1ms (T32,T96), 10ms (T33 à T36, T97 à T100) et 100ms (T37 à T63, T101 à T255).

Exemple





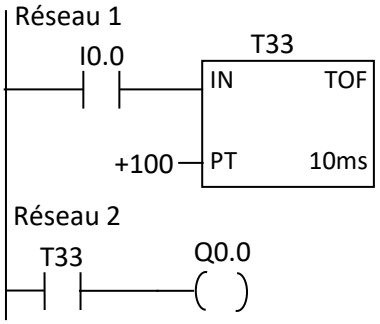
➢ Démarrer temporisation sous forme de retard à la retombée (TOF)

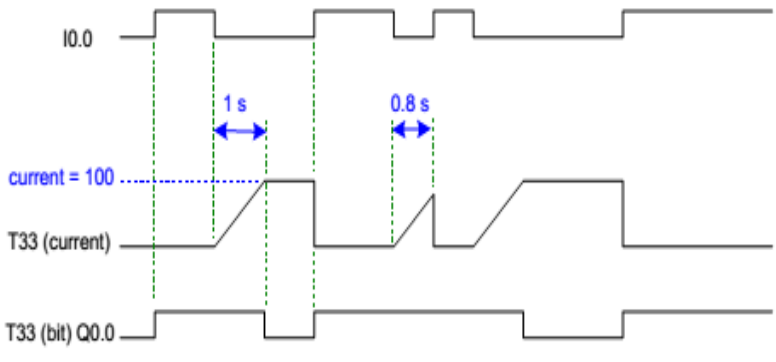


L'opération Démarrer temporisation sous forme de retard à la retombée sert à retarder la désactivation d'une sortie (le bit du temporisateur) pour un intervalle de temps donné après que l'entrée (IN) a été désactivée.

- La temporisation démarre en cas de front descendent à l'entrée de démarrage IN.
- La temporisation s'initialise si l'état de signal à l'entrée IN passe de 0 à 1 alors que la temporisation s'exécute.
- La temporisation continue à s'écouler jusqu'à ce que le temps écoulé atteigne le temps prédéfini.
- Si valeur en cours = valeur prédéfinie, le comptage est s'arrêté.
- L'état de signal à la sortie égale 0 lorsque si valeur en cours = valeur prédéfinie ou si Si valeur en cours =0 à condition que l'entrée IN reste égal 0. Dans les autres cas, l'état de signal à la sortie est 1.

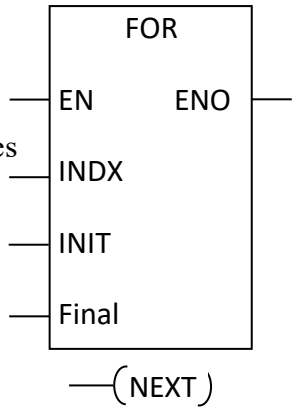
Exemple





V.5.11. Opérations de gestion du programme

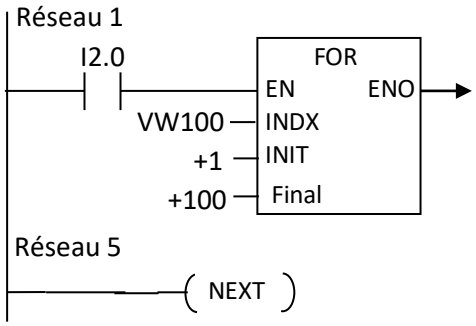
➢ **Opération de boucle FOR/NEXT**



Les opérations FOR et NEXT permettent de définir une boucle qui est exécutée le nombre de fois précisé.

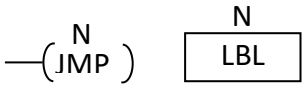
- Vous pouvez imbriquer jusqu'à huit boucles FOR/NEXT les unes dans les autres.
- L'opération FOR exécute les instructions figurant entre les mots--clés FOR et NEXT.
- Vous précisez le nombre de boucles en cours INDX, la valeur initiale INIT et la valeur finale FINAL.
- L'opération NEXT signale la fin de la boucle déclenchée par FOR.

Exemple



Lorsque I2.0 est activé, les instructions figurant entre l'opération FOR et l'opération NEXT sont exécutées 100 fois, la valeur INDX étant incrémentée: 1, 2, 3, ...100. Lorsque la boucle est réactivée, elle copie la valeur initiale dans le nombre de boucles en cours (INDX).

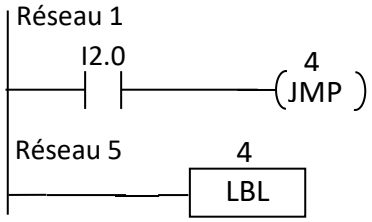
➢ **Opérations de saut JMP/LBL**



L'opération Sauter au repère (JMP) effectue un saut à l'intérieur du programme au repère N indiqué si le résultat logique précédent égal 1.

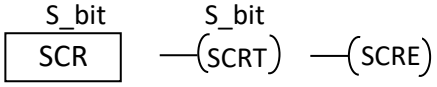
- L'opération Définir repère (LBL) précise la destination N d'un saut. Le repère de saut N est une constante entre 0 et 255.
- Vous pouvez utiliser l'opération de saut dans le programme principal, dans des sous-programmes et dans des programmes d'interruption.
- Vous ne pouvez pas sauter du programme principal à un repère se trouvant dans un sous-programme ou un programme d'interruption. De même, vous ne pouvez pas sauter d'un sous--programme ou d'un programme d'interruption à un repère se trouvant hors de ce sous--programme ou de ce programme d'interruption.

Exemple



Si I2.0 est activé, sauter au repère LBL4

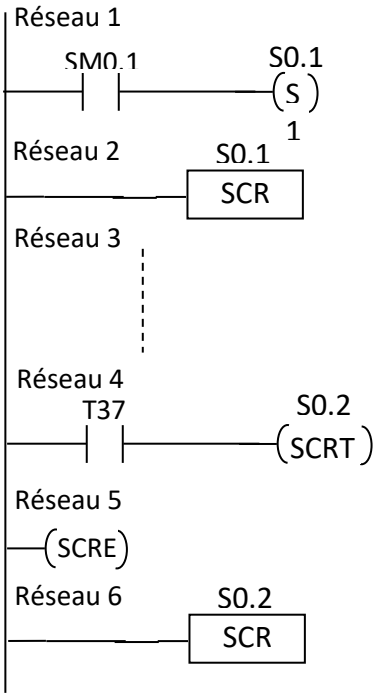
➢ **Opérations SCR**



Si votre application est constituée d'une séquence d'opérations devant être répétée, l'appellation des opérations SCR (relais séquentiels) permet de structurer votre programme afin qu'il correspond de manière directe à votre application. Vous pouvez ainsi programmer et tester votre application plus facilement et plus rapidement.

- L'opération **SCR** signale le début d'un segment SCR si le bit du segment égal à 1.
- L'opération de changement de relais séquentiel **SCRT** permet de passer la main d'un segment SCR actif à un autre segment SCR, si le résultat logique précédent égal 1. L'exécution de l'opération SCRT en présence d'un flux de signal remet à 0 le bit S du segment actuellement actif et met à 1 le bit S du segment référencé.
- L'opération Fin de relais séquentiel (SCRE) signale la fin d'un segment SCR si le résultat logique précédent égal 1.

Exemple



Réseau 1 : Activer Segment 1 lors du premier cycle.

Réseau 2 : Début du Segment 1.

Réseau 3 : Instructions du Segment 1.

Réseau 4 : Passage au Segment 2 après l'activation du bit de temporisation.

Réseau 5 : Fin de la zone SCR pour Segment 1.

Réseau 6 : Début du Segment 2.

➢ **Opérations de fin de traitement**

- L'opération Fin de traitement conditionnelle (**END**) met fin au cycle en cours, si le résultat logique précédent égal 1. Vous pouvez vous servir de l'opération Fin de traitement conditionnelle dans le programme principal, mais pas dans les sous--programmes ni dans les programmes d'interruption.
- L'opération Fin de traitement conditionnelle (**RET**) met fin l'exécution du sous programme ou du programme d'interruption, si le résultat logique précédent égal 1.
- L'opération **STOP** met immédiatement fin à l'exécution de votre programme, si le résultat logique précédent égal 1, en faisant passer la CPU S7-200 de l'état de fonctionnement "Marche" (RUN) à l'état "Arrêt" (STOP).

V.6. Programmation en langage par blocs

La programmation en langage FBD consiste à relier graphiquement entre eux des blocs symbolisant les fonctions souhaitées. Graphiquement, la forme générale de ces blocs est identique à la représentation utilisée par le langage ladder. Il y a quand même quelques différences aux niveaux les opérations sur bits, opération de comparaison et opérations de gestion du programme. En plus, les contacts et les bobines n'existent pas dans le langage FBD mais ils sont remplacés par des blocs comme le montre le tableau V.3.

Remarque 4 : L'inversion logique de signaux booléens d'entrée ou de sortie peut être indiquée par un petit cercle.

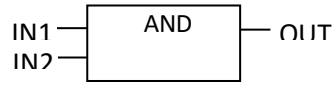
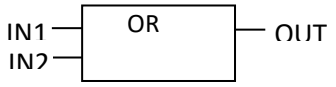
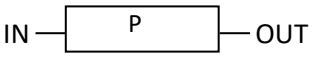
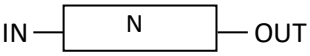
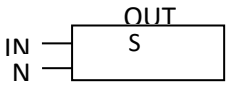
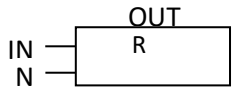
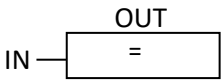
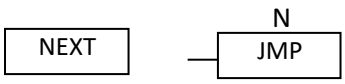
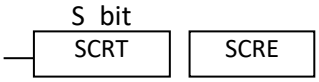
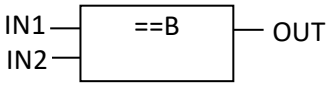
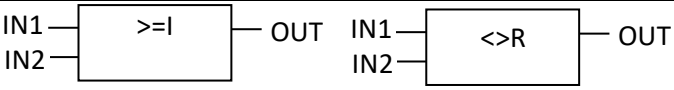
Opération	Graphe	Opération	Graphe
ET logique		**OU logique**	
Détecter de front montant		**Détecter de front descendant**	
Mettre à 1		**Mettre à 0**	
Sortie		**NEXT/ JMP**	
SCRT/SCRE		**END/RET/STOP**	
Opérations de comparaison			

Tableau V.3. Blocs du langage FBD.

V.7. Programmation en langage liste d'instructions

Le langage à liste d'instructions est composé d'une suite d'instruction, chaque instruction doit débuter une nouvelle ligne de programme et doit contenir un opérateur suivi d'une ou plusieurs opérandes. Les opérateurs standard du langage IL normalisés sont repris au tableau ci-dessous.

Opération	Commentaire
LD	Charger la valeur de l'opérande
LDN	Charger la valeur inverse de l'opérande
A	ET logique entre le résultat précédent et l'état de l'opérande
AN	ET logique entre le résultat précédent et l'état inverse de l'opérande
O	OU logique entre le résultat précédent et l'état de l'opérande
ON	OU logique entre le résultat précédent et l'état inverse de l'opérande
NOT	Inverser le résultat
EU	Détecter front montant
ED	Détecter front descendant
=	Ecrire la sortie
S	positionner l'opérande à un
R	Remettre l'opérande à un
LDB=	Comparaison de type égalité
AB=	Comparaison de type égalité précédé d'un contact en série
OB=	contact en parallèle avec la Comparaison de type égalité
BTI, ITB	Conversion d'un octet en entier, Conversion d'un entier en octet.
+I ; -I ;*I ; /I	Addition, Soustraction, Multiplication et Division de deux nombres entiers
INCB ; DECB	Incrémentation et décrémentation d'un octet
INVB ; ANDB ; ORB	Inversion d'un octet, et logique de deux octets, ou logique de deux octets
MOVB	Transférer un octet
SLB ; SRW	Décalage vers la gauche d'un octet, Décalage vers la droite d'un mot
RLB ; RRD	Rotation vers la gauche d'un octet, Décalage vers la droite d'un double mot

Tableau V.4. Opérations de base du langage Liste d'instructions.

V.8. Exercices

Exercice 01

Programmer, en langage Ladder, les fonctions suivantes :

$y_1 = \overline{I_1 . I_2 + I_3 . I_4}$

$y_2 = (I_1 + I_2).(I_3 + I_4)$

$y_3 = (\overline{I_1} . I_2).(I_3 + \overline{I_4})$

Exercice 02

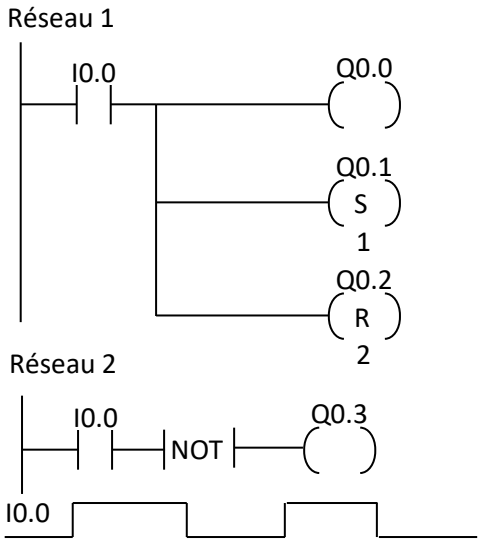
Donner le programme correspond, en langage Ladder, pour chacune des fonctions suivantes :

1- La sortie y est égale à 1 si l'entrée $e = 0$ sinon elle prend la valeur zéro.
2- La sortie y_1 est mise à 1 si les entrées e_1 et e_2 prennent les valeurs 1 et 0, respectivement.
3- La sortie y_2 est remise à 0 si les entrées $e_1 = 0$ et $e_2 = 1$.
4- La sortie y_3 est mise à 1 si les entrées $e_1 = 1$ ou $e_2 = 0$ sinon elle est remise à zéro.

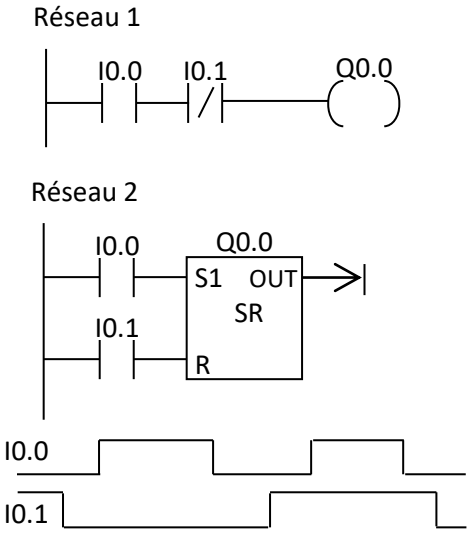
Exercice 03

Tracer les chronogrammes des sorties Q0.0, Q0.1 et Q0.2 pour les exemples suivants :

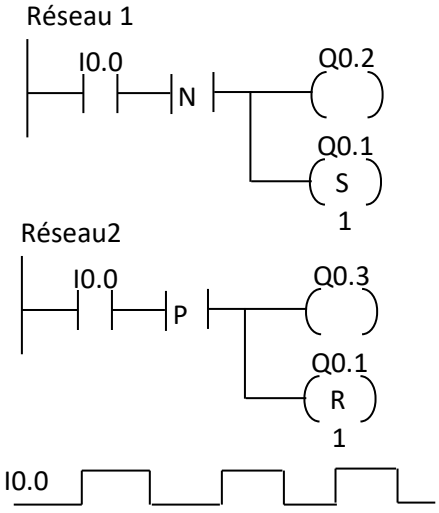
Exemple 1



Exemple 2

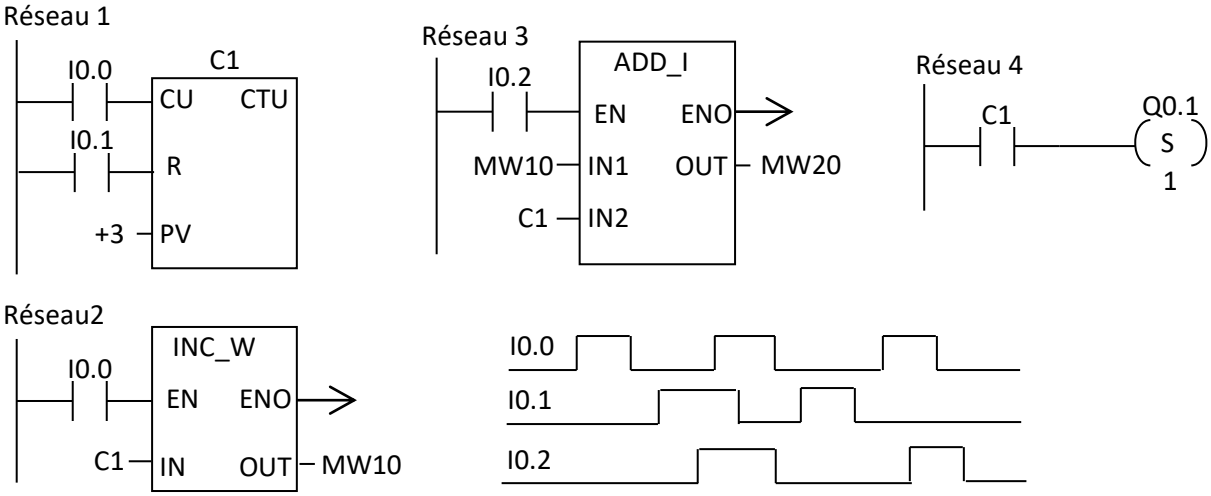


Exemple 3



Exercice 04

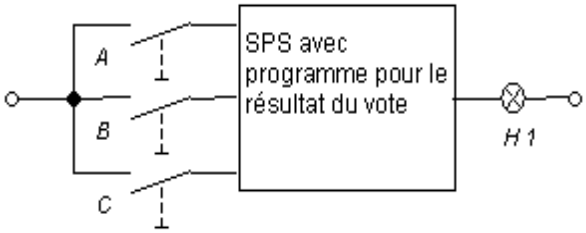
Soit l'exemple de programmation, en langage ladder, suivant:



- Donner les différentes valeurs des variables C1, MW10 et MW20.
- Tracer le chronogramme de la sortie Q0.1.

Exercice 05

Un système de vote est doté de trois boutons A, B, C et d'une lampe H1. Ces trois boutons sont utilisés par une commission qui se compose de trois membres ayant le droit de vote, les votes doivent être, entre autres, évaluées à l'aide d'un API. Pour cela, chaque utilisateur dispose du droit de vote (BP à fermeture A, B, C). Si la majorité des participants génère un "OUI" (décision affirmative), alors la lampe (H1) s'éclaire. Si la majorité des participants génère un "NON" (décision négative), alors la lampe (H1) ne s'allume pas, elle reste éteinte.



- Etablir la table de vérité de cet affichage puis trouver l'équation booléenne qui représente la variable de sortie H1.
- Programmer l'équation obtenue en langage Ladder.
- Reprogrammer le fonctionnement du système, en langage Ladder, par l'utilisation des opérations suivantes : >=I, ADD_I et MOVW.

Exercice 06

Pour commander une lampe à l'aide d'un bouton poussoir unique, on se propose de réaliser un circuit à une entrée notée B (le bouton poussoir), et une sortie notée L (la lampe) tel que :

- la lampe s'allume en appuyant sur le bouton si elle était éteinte et reste allumée lorsqu'on lâche le bouton ;
- la lampe s'éteint en appuyant sur le bouton si elle était allumée et elle reste éteinte lorsqu'on lâche le bouton.

Donner le programme, en langage ladder, correspond à ce fonctionnement.

Exercice 07

Etablir un programme, en langage ladder, qui décrit la commande d'une pompe (P) à l'aide de deux boutons poussoirs (Marche (M)-Arrêt(A)):

- En appuyant sur M (marche),
  - si la pompe (P) est arrêtée, elle démarre et continue à tourner lorsqu'on lâche le bouton M;
  - si la pompe fonctionne, elle continue à fonctionner.
- En appuyant sur A,
  - si la pompe fonctionne, elle s'arrête et reste arrêtée lorsqu'on lâche le bouton A ;
  - si la pompe est arrêtée, elle demeure arrêtée.

Exercice 08

Programmer l'exercice n : 04 du TD 03 (parking automatique), en langage Ladder sans passer par le grafcet.

Exercice 09

Programmer les grafcets de l'exercice 07 (TD 02) en langage Ladder.

.

Références

1- Automates programmables Industriels, Wiluam Bolton, livre, 2015.
2- Automates Programmables Industriels, L. Bergougnoux, cours polycopié, Ecole polytechnique de Marseille, 2005.
3- Automate Programmable S7-200, Manuel d'utilisation, Siemens, 2008.
4- Automating with SIMATIC, Sixth edition, Hans Berger, livre, 2016.
5- Automatisme et Automatique, Jean-Yves Fabert, livre, 2005
6- Automatismes industriels, Jean-Michel Bleux et J-P Herve, livre, 1996.
7- Automatismes, Salim Ben Saoud, Cours, http://rel.uvt.rnu.tn.
8- Automatismes Industriels & GRAFCET, Dubois. http://www.geea.org.
9- Automates et Informatique industrielle, Mohamad Khalil. cours polycopié, Centre Universitaire de Technologie franco-libanais.
10- Cahier de Technologie, Badra Sahbi, http://www.sahbitechnologie.sitew.com.
11- Ce qu'il faut savoir sur les automatismes, Philippe Grare et Imed Kacem, livre, 2008.
12- Comprendre, maitriser et appliquer le grafcet, M. Blanchard, livre, 2005.
13- Cours de réseau de Petri, Yann Morère, cours, http://www.morere.eu, 2002.
14- Du Grafcet aux réseaux de Petri, Hassane Alla et René David, livre, 1992.
15- IEC 61131-3: Programming Industrial Automation Systems, Second Edition, Karl-Heinz John · Michael Tiegelkamp, livre, 2010.
16- Introduction aux automatismes industriels, Lecourtier, Saint jean, livre 1985.
17- Le GRAFCET et sa mise en ouvre, Patrick Trau.
18- Le Grafcet, Adepa et Afcet, livre, 1995.
19- Les Automates Programmables Industriels, Alain GONZAGA, Cours, http://www.geea.org.
20- Les Automates Programmables, Ir. H. Lecocq, cours polycopié, université de Liege, 2005.
21- Les réseaux de Petri, Robert Valette, cours polycopié, LAAS-CNRS Toulouse, 2000.
22- Les réseaux de Petri : un outil de modélisation, Annie Choquet-Geniet, livre, 2006.
23- Logique programmée et Gracet, Jean-Paul Vabre, Alain jacques et Jean Claude Lafont, livre, 1987.
24- Programmable logic controllers, Fifth edition, Frank D. Petruzella, livre, 2017.
25- Programmable logic controllers, Dag h. hanssen, livre, 2015.
26- Réseaux de Petri, G. Scorletti et G. Binet, 2006.
27- Réseaux de Petri : Elaboration des systèmes de production, Marc Bourcerie, livre, 2011.
28- Systèmes logiques et GRAFCET, Belkacem Ould Bouamama, cours polycopié, Ecole polytechnique de Lille.

.