UNIVERSITE DE LIEGE
Faculté des Sciences Appliquées

LES AUTOMATES PROGRAMMABLES

Tome I

Caractéristiques et méthodologie de programmation

Dr. Ir. H. LECOCQ

Professeur

– Dernières mises à jour 2005 –

Table des matières

Chapitre 1

INTRODUCTION

1.1. CONTEXTE INDUSTRIEL

Du point de vue de la gestion et de l'automatisation, on classe généralement les entreprises industrielles en deux grandes catégories : les entreprises de procédés continus (process industries) et les entreprises manufacturières (manufacturing industries).

Dans les premières, la production est décrite en termes de débits de matières. C'est typiquement le cas des usines physico-chimiques et pétrochimiques. Le processus de production y est généralement caractérisé par une séquence de réactions physico-chimiques se déroulant de manière continue ou quasi-continue. Il est clair que, dans ce type d'entreprise, la production est strictement figée, tant du point de vue de la nature des produits que du point de vue de l'outil de production.

Dans les secondes, qualifiées de discontinues ou de discrètes, on fabrique des "objets" dénombrables qui peuvent évidemment être de complexité très diverse. Les industries mécaniques, électriques et électroniques appartiennent à cette catégorie. Le processus de production se présente en général ici comme une succession d'opérations de mise en forme et d'assemblage réalisées manuellement ou à l'aide de machines.

La suite de l'exposé sera principalement consacrée à cette seconde catégorie d'entreprises. Bien entendu, certaines des notions qui seront présentées ci-après sont également applicables à la première catégorie.

C'est l'évolution du marché qui explique les problèmes rencontrés actuellement par les entreprises manufacturières, surtout par celles qui s'adressent au grand public. Il y a peu de temps encore, le marché se caractérisait par le fait que le producteur était roi. Il y avait peu de concurrence et peu de produits. Le consommateur n'était pas difficile et achetait ce qui était disponible.

Qu'on se rappelle la Ford T du début du siècle qui fut produite à un million d'exemplaires par an pendant seize ans ! C'est pour ce genre de production que Taylor avait développé sa philosophie : spécialisation des équipements et spécialisation du personnel à qui on ne demandait que des travaux élémentaires et répétitifs.

Actuellement, le marché se caractérise plutôt par le fait que le client est devenu roi. La concurrence s'est considérablement accrue et mondialisée, rendant le consommateur plus difficile et beaucoup plus critique, notamment au niveau de la qualité des produits. Le cycle de vie des produits s'est également considérablement raccourci : trois à quatre ans pour une automobile, parfois beaucoup moins pour un ordinateur. En termes de production, cela signifie une grande variété de produits à cycle de vie très court et en petites séries. Cette situation peut être résumée par le diagramme de la figure 1.1.

Des 30 % représentés par la production manufacturière dans l'activité industrielle globale, 40 % concernent une production par lot et seulement 15% une production de masse. Des 40 % de production par lot, 75 % concernent des lots de moins de 50 pièces !

Pour survivre, les entreprises doivent donc arriver à produire vite, bien et bon marché, tout en étant capables de s'adapter rapidement à l'évolution des produits.

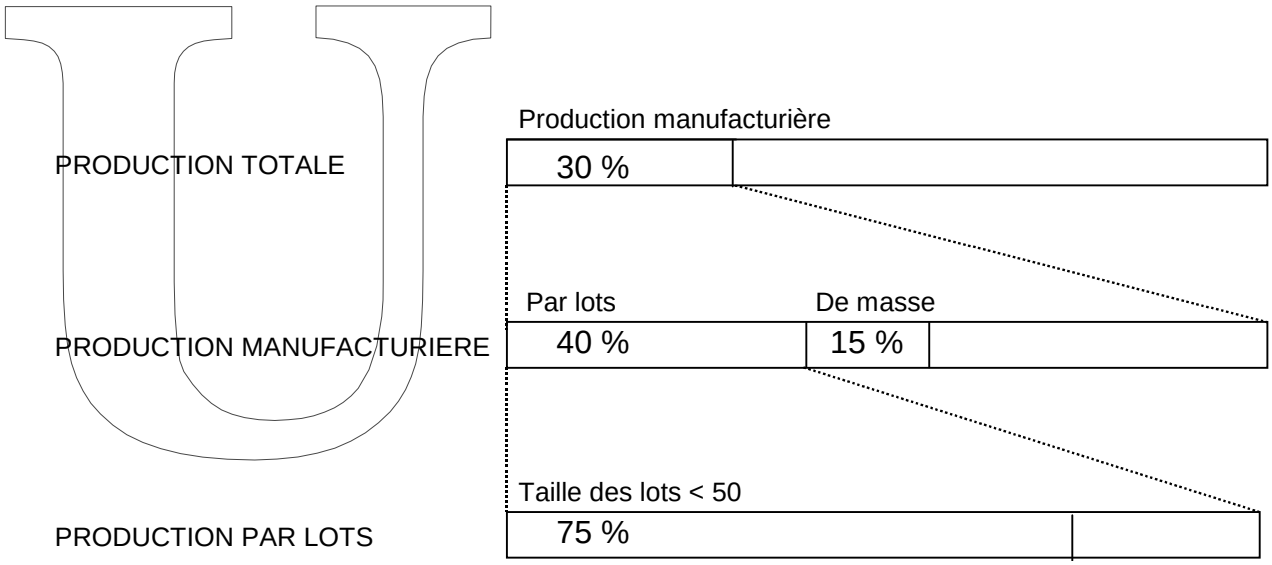


Figure 1.1. Position et structure de la production manufacturière

1.2. LES NOUVELLES REGLES DE PRODUCTION

Les nouvelles règles de production qui répondent à la question peuvent être résumées, de manière imagée, par cinq zéros : zéro défaut, zéro panne, zéro stock, zéro délai et zéro papier. La signification des quatre premiers zéros est claire; le cinquième indique la volonté de supprimer la paperasse qui alourdit trop souvent le travail du personnel et est cause de nombreuses erreurs. Idéalement, on devrait d'ailleurs encore y ajouter deux zéros : zéro accident et zéro problème social.

Plus techniquement, ces nouvelles règles de production relèvent d'une philosophie appelée "Juste-à-Temps" (Just-in-Time ou JIT en anglais) aussi connue sous le nom de "production à flux tendus" [Béranger, 1987].
Il s'agit d'un principe d'organisation industrielle, apparu au début des années 80, qui préconise d'acheter ou de produire seulement ce dont on a besoin, quand on en a besoin.
Ceci devant être respecté aussi bien au niveau des produits finis (ne fabriquer que ce qui est commandé) qu'au niveau des pièces constitutives de ces produits.

Le premier résultat en est évidemment une réduction drastique des stocks, et partant, une diminution sensible des charges financières de l'entreprise. Il ne s'agit cependant pas là du but principal recherché. En réalité, la réduction des stocks n'est que l'amorce d'une réaction en chaîne qui conduit à des bouleversements en profondeur du fonctionnement de l'entreprise.

En effet, pour produire sans stock tout en garantissant des délais de livraison normaux, il est nécessaire d'avoir des temps de fabrication très courts, une grande flexibilité pour pouvoir suivre la demande (en variété et en quantité) et une haute fiabilité de production afin d'éviter les aléas.

Au niveau de l'organisation de la production, cela implique :

- la division de l'usine en cellules permettant l'enchaînement rapide des opérations concernant une même pièce ou un même produit de manière à éviter les stockages intermédiaires;
- la limitation des déplacements accélérant le travail et facilitant le suivi de production;
- la flexibilité des cellules en question : changement rapide d'outils et de programmes de fabrication;
- le contrôle et la maîtrise de la qualité à chaque étape de la fabrication afin de ne pas enrayer le processus;
- la fiabilité des machines, pour les mêmes raisons que ci-dessus, ce qui suppose la mise en place d'une politique rigoureuse de maintenance préventive;
- la polyvalence et l'augmentation de la qualification des opérateurs qui deviennent responsables de la quantité et de la qualité des pièces ou produits fabriqués, voire même du bon fonctionnement des machines;
- des relations nouvelles avec les fournisseurs afin qu'ils entrent aussi dans le processus, tant au niveau des délais que de la qualité des produits fournis.

Remarquons que, jusqu'à présent, il n'a encore été question ni d'automatisation ni d'informatisation. C'est qu'en effet la philosophie du Juste-à-Temps concerne avant tout l'organisation de la production.

Il faut en effet considérer la production comme une chaîne dont les maillons doivent tous être de même résistance : il ne sert en effet à rien, globalement, de renforcer à l'extrême certains maillons, s'il en est d'autres qui demeurent fragiles.

Une saine démarche consistera donc à analyser les flux de matières et d'informations associés au processus de production, à les rationaliser et à les simplifier au maximum dans l'optique du Juste-à-Temps. Ce n'est qu'alors, et alors seulement, que l'opportunité d'automatiser ou d'informatiser telle ou telle partie du processus apparaîtra clairement.

En l'occurrence, l'automatisation permettra d'accélérer la fabrication et/ou de garantir la constance de la qualité. Pour les raisons qui ont été exposées ci-dessus, l'automatisation devra être flexible. Cette flexibilité doit se traduire au niveau de la structure des machines qui seront aussi polyvalentes et adaptatives que possible, avec une gestion d'outils et une alimentation en pièces complètement automatisées. A cet égard, le robot apparaît évidemment comma la machine flexible par excellence.

Cette flexibilité doit aussi se retrouver au niveau du système de commande des machines dont les modes de fonctionnement devront pouvoir être facilement modifiés. Ce dernier point ne pose plus actuellement de réel problème dans la mesure où pratiquement toutes les nouvelles machines de production sont commandées par des dispositifs à base de microprocesseurs, avec programme enregistré. De plus, des portes de communication existent presque toujours qui permettent de télécharger et de modifier les programmes à partir d'autres ordinateurs.

L'informatisation, quant à elle, a pour but d'améliorer la manipulation des informations relatives au processus de production. Ces informations concernent non seulement la fabrication proprement dite mais aussi la conception des produits, la gestion technique, financière et administrative de l'usine, le management, le marketing, ...

Ces différentes facettes de la production ont déjà fait, de longue date, l'objet de développements informatiques spécifiques. Cependant, dans la plupart des cas, ceux-ci ont été menés indépendamment les uns des autres. Il en résulte que d'importants flux d'informations continuent de circuler par la voie manuelle (papiers, plans, réencodage, etc.) tandis que des informations similaires se retrouvent dans des bases de données différentes, avec tous les risques d'incohérence que cela comporte.

Les nouvelles règles de production évoquées dans ce paragraphe (et en particulier le "zéro papier") conduisent tout naturellement à préconiser l'intégration des moyens informatiques d'une entreprise. Le terme intégration recouvre ici non seulement l'interconnexion physique des ordinateurs par des réseaux de communication mais aussi, et surtout, leur interconnexion logique.

On entend par là que le système informatique distribué initial apparaît à l'utilisateur comme un système informatique centralisé et homogène; les effets recherchés étant essentiellement l'unicité et la disponibilité des informations. En d'autres termes, l'intégration offre à chacun l'accès direct à l'information voulue, au moment voulu et à l'endroit voulu.

Dans ce contexte d'automatisation et d'informatisation de la production on trouve, dans la littérature, un nombre impressionnant de nouvelles expressions et de nouveaux sigles. Nous tenterons ici d'expliciter les plus courants d'entre eux. Bien entendu, ceci sera fait sous toutes réserves étant donné qu'aucune normalisation n'est encore disponible.

N.B. : Plusieurs de ces définitions sont tirées de [Béranger, 1987]

AGV (Automated Guided Vehicle)
Voir VGA

ATELIER FLEXIBLE
Atelier de production automatisé capable de réaliser des produits différents. Le volume de production pour chaque type de produit est totalement variable (flexibilité) et est modulé en fonction de la demande. Gérés par un ordinateur central, les ateliers flexibles sont composés de machines à commande numérique, de robots, de convoyeurs et véhicules automatisés, de magasins de stockage automatisés.

CAD (Computer Aided Design)
Voir CAO

CAM (Computer Aided Manufacturing)
Voir FAO

CAO (Conception Assistée par Ordinateur)
Voir CAD
Programme d'aide à la conception de pièces ou d'assemblages comprenant des fonctions de calculs, de manipulations géométriques (en 2 ou 3 dimensions) et de dessin.

CELLULE FLEXIBLE

Voir FMS

Unité de production automatisée constituée d'une ou plusieurs machines à commande numérique, de magasins de stockage et d'alimentation et d'un robot qui prend en charge le déplacement et le positionnement des pièces traitées.
Pilotées par ordinateur, les cellules flexibles ont pour vocation d'usiner ou d'assembler successivement des pièces différentes (notion de flexibilité) sans intervention humaine ou reréglage. Elles constituent l'unité de base des ateliers flexibles.

CENTRE D'USINAGE

Voir MOCN

CIM (Computer Integrated Manufacturing)

Voir PRODUCTIQUE

Production intégralement informatisée. Concept clé de l'usine du futur où toutes les tâches – de la commande des produits à leur expédition, en incluant la conception, la fabrication, le stockage, la manutention, ainsi que le contrôle qualité et la maintenance des machines – sont gérées et contrôlées par un système informatique intégré composé d'ordinateurs interconnectés par réseau de communication.

CNC (Computer Numerical Control)

Voir MOCN

DAO (Dessin Assisté par Ordinateur)

Programme de dessin remplaçant la planche à dessin traditionnelle par une planche à dessin électronique constitué d'un écran d'ordinateur.

FAO (Fabrication Assistée par Ordinateur)

Voir CAM

Programme qui assure la conversion directe des plans (créés par la CAO ou la DAO) en tâches de fabrication pour machines-outils dirigées par ordinateur. La FAO permet en général aussi de simuler le travail des machines-outils.

FMS (Flexible Manufacturing System)

Voir CELLULE FLEXIBLE

GPAO (Gestion de Production Assistée par Ordinateur)

Voir PPS

Programme assurant la planification de la production et des approvisionnements d'une usine à partir des données commerciales (prévisions de vente, commandes fermes), des niveaux de stocks et d'informations concernant la structure des produits et leur méthode de fabrication.

MAGASIN AUTOMATISE

Dispositif de stockage automatisé piloté par ordinateur. L'ordinateur gère le contenu du magasin, les emplacements physiques, les déplacements des chariots automatisés qui effectuent les entrées, les sorties et les mouvements des conteneurs ou palettes dans le magasin.

MOCN (Machine-Outil à Commande Numérique)

Voir CNC

Machine-outil dont l'enchaînement des tâches, y compris les changements d'outils, est programmé et s'effectue sous le contrôle d'un ordinateur. La MOCN peut donc

fonctionner sans l'intervention permanente d'un opérateur, dont le rôle se limite au chargement et au déchargement de la machine. Les MOCN sont le plus fréquemment des tours, des fraiseuses, des perceuses, des scies et autres machines de découpe.

PPS (Introduction Planning System)
Voir GPAO

PRODUCTIQUE
Voir CIM.
Le terme "PRODUCTIQUE" est pratiquement l'équivalent français du terme CIM.
Tout au plus pourrait-on trouver au CIM une connotation informatique dominante tandis que la PRODUCTIQUE serait plutôt centrée sur le processus de production proprement dit.

VGA (Véhicule Guidé Automatisé)
Voir AGV
Véhicule sur roues, sans conducteur, commandé à distance. Le guidage se fait généralement par un réseau de fils implantés dans le sol. Les VGA sont utilisés pour transporter des pièces ou des composants dans une usine.

1.3. ARCHITECTURE INFORMATIQUE INTEGREE

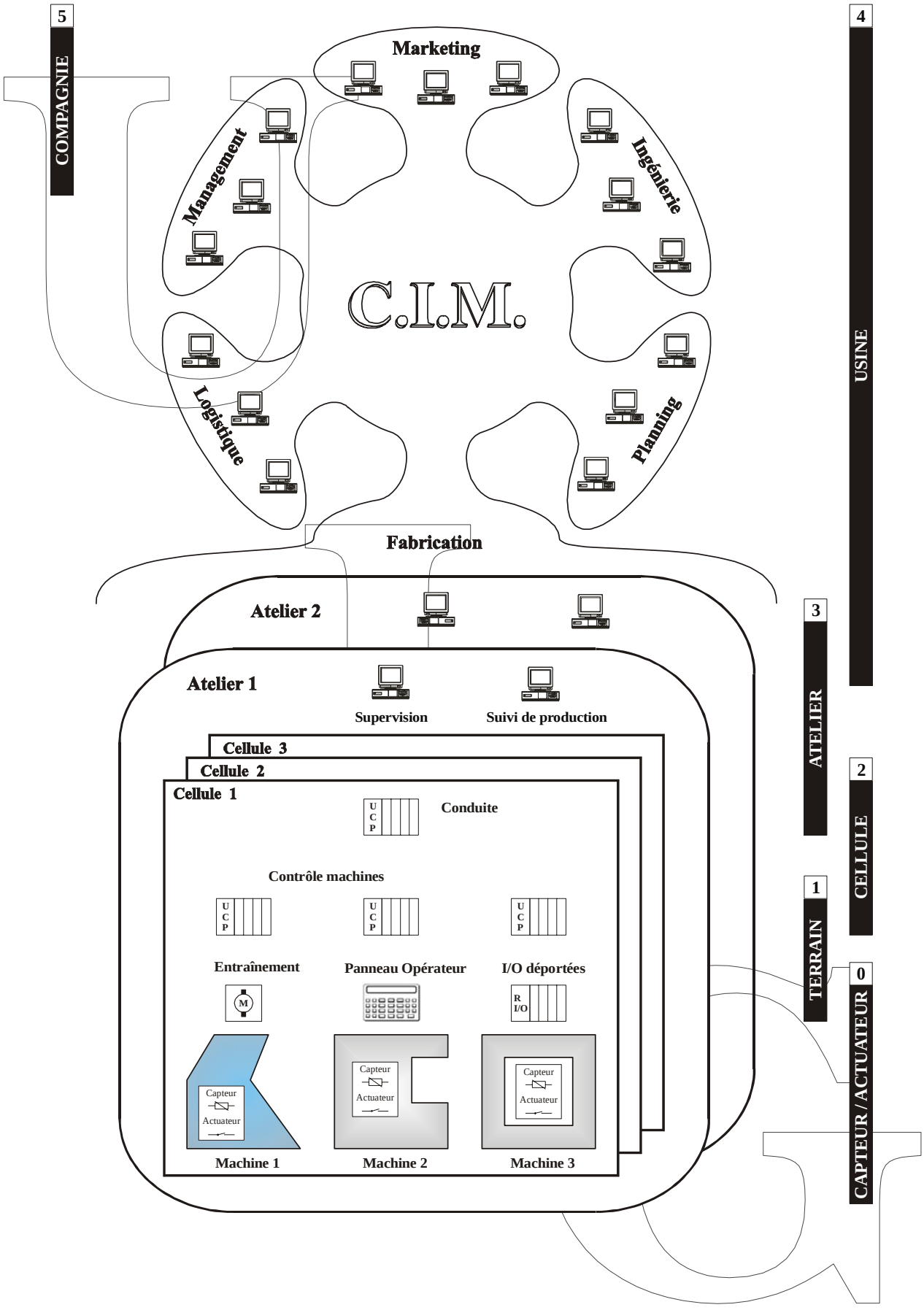
La figure 1.2 montre une architecture informatique intégrée conforme aux idées actuellement en vigueur.

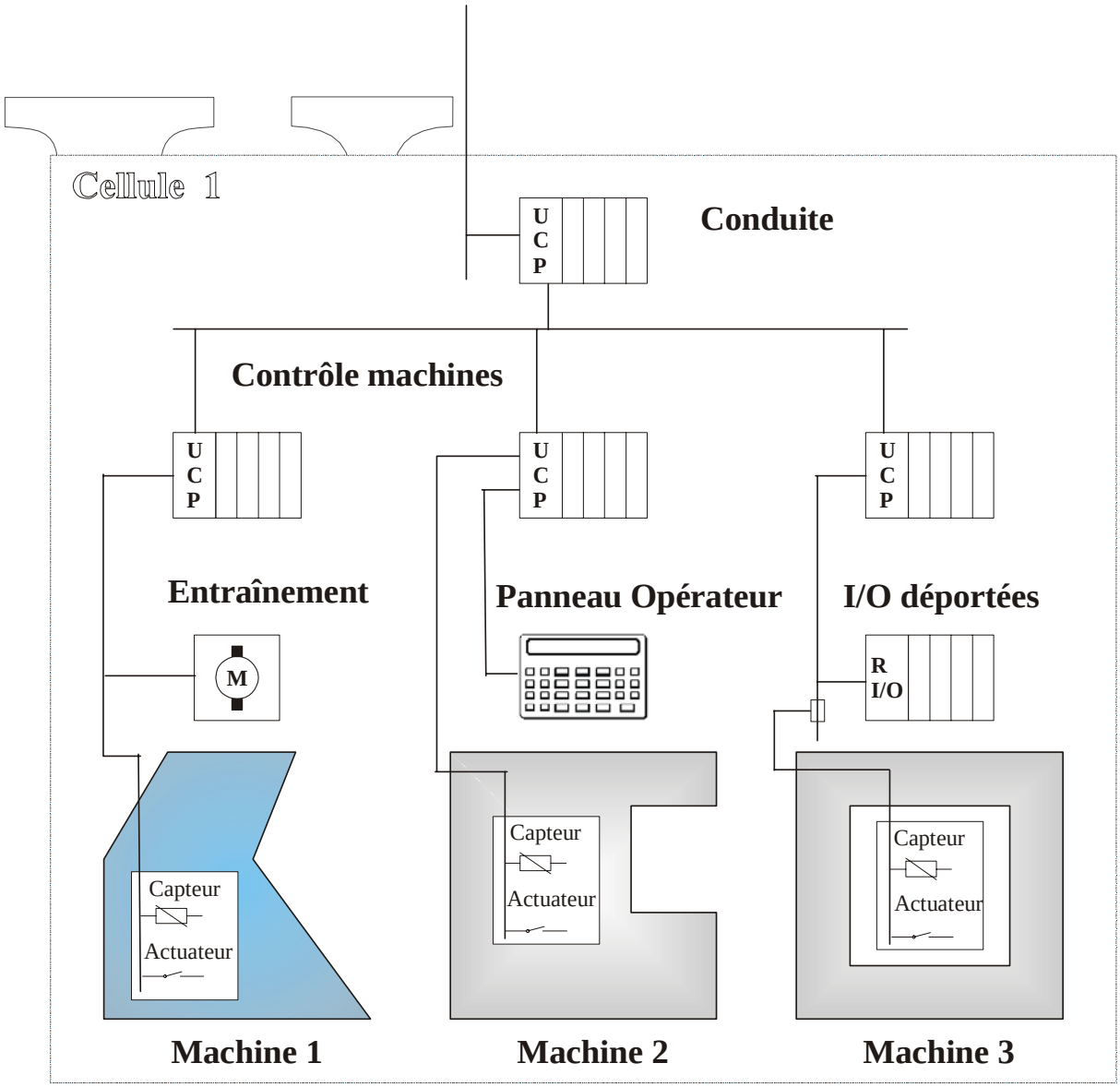
Une usine est généralement divisée en ateliers responsables de la fabrication d'un produit ou d'une gamme de produits de même famille. L'atelier, à son tour, est composé de cellules de production qui regroupent des machines fortement interactives.

Les machines de production modernes (CNC, robots, etc) sont pratiquement toutes commandées par des dispositifs à microprocesseurs notés génériquement UCP (unité de commande programmable) sur la figure 1.2. Ces unités commandent les machines par l'intermédiaire d'actuateurs (contacteurs, vannes, etc.) sur base d'informations fournies par des capteurs (détecteurs de fin de course, codeurs de position, thermocouples, etc). A noter que les unités en question sont souvent incluses dans les machines, parfois à l'insu des utilisateurs d'ailleurs.

Les unités de commande peuvent échanger entre elles des informations par leurs entrées/sorties. Celles-ci sont cependant destinées, avant tout, à être reliées à des capteurs et à des actuateurs. Les communications que l'on peut réaliser par ce moyen sont donc assez limitées; on les réserve à des fonctions vitales de protection et de synchronisation.

La supervision et la gestion de la cellule nécessite des échanges d'informations bien plus élaborés qui ne peuvent se concevoir que par l'intermédiaire de réseaux de communication. Ceci ne pose en principe aucun problème technique puisque les unités de commande sont à base de microprocesseurs et que les informations à échanger sont donc de nature informatique.


5
COMPAGNIE
4
USINE
Marketing
Management
Ingénierie
C.I.M.
Logistique
Planning
Fabrication
Atelier 2
3
ATELIER
Atelier 1
Supervision
Suivi de production
Cellule 3
Cellule 2
Cellule 1
2
CELLULE
U C P
Conduite
Contrôle machines
1
TERRAIN
Entraînement
Panneau Opérateur
I/O déportées
M
R I/O
0
CAPTEUR / ACTUATEUR
Capteur
Actuateur
Machine 1
Machine 2
Machine 3



UCP : unité de commande programmable

Figure 1.2. Le concept CIM d'automatisation et d'informatisation de la production

C'est également par réseau que des communications pourront être établies avec les ordinateurs de gestion d'atelier et, à un niveau fonctionnel plus élevé, avec le bureau d'études et le département de gestion de production.

Plutôt que d'utiliser un seul réseau pour toutes ces communications, la tendance est actuellement de hiérarchiser les réseaux comme cela est montré à la figure 1.2. Les raisons sont d'ordre technique et économique. Il est bien clair, en effet, que, selon le niveau où l'on se situe, les performances demandées au système de communication seront sensiblement différentes, tant en ce qui concerne le débit d'information qu'en ce qui concerne l'immunité aux parasites industriels.

C'est ainsi que l'on distingue en gros cinq niveaux :

– **le niveau processus** : c'est un niveau, apparu assez récemment, où l'on se propose de remplacer le câblage fil à fil, traditionnellement utilisé pour le raccordement des capteurs et actuateurs aux entrées et sorties des contrôleurs, par une liaison de type série (paire téléphonique, fibre optique, ...). Le but est essentiellement, ici, de réduire le volume du câblage qui représente toujours un poste main d'oeuvre important, à l'installation comme à la maintenance.
Ce type de réseau, appelé FIELDBUS ou BUS de terrain, fait actuellement l'objet de travaux de standardisation.

– **le niveau cellule** : à ce niveau, on utilise des systèmes de communication simples et robustes, parfois limités à des liaisons point à point (cfr cellule de droite de la figure 1.2). Par ailleurs, dans le cadre restreint d'une cellule, les unités programmables sont souvent de même type et l'on peut alors utiliser les réseaux proposés par les constructeurs des unités eux-mêmes.
Si ces solutions ne conviennent pas, il faudra alors recourir aux réseaux "ouverts" normalement prévus pour le niveau atelier.

– **le niveau atelier** : le maître mot, à ce niveau, est l'ouverture. Dans la plupart des cas, en effet, on aura à faire communiquer entre eux des systèmes informatiques de nature et d'origine diverses.
Qui dit ouverture dit standardisation, tant au niveau matériel qu'au niveau logiciel. C'est en l'occurrence le protocole MAP (Manufacturing Automation Protocol) qui semble, à l'heure actuelle, rallier tous les suffrages. Il faut cependant préciser que ce protocole, inspiré par la firme General Motors, est inutilement complexe, et donc coûteux, pour de très nombreuses applications de petite ou moyenne envergure. L'étude d'un standard mini-MAP moins ambitieux a donc également été entreprise. Ces deux standards, très prometteurs, sont malheureusement encore loin d'être complètement finalisés.
C'est pourquoi, il ne faut pas perdre de vue l'existence du standard de fait ETHERNET, adopté par de nombreux constructeurs en dépit de certaines particularités défavorables sur le plan industriel; ainsi la nature probabiliste des temps de transfert des messages est assez peu compatible avec des exigences temps réel strictes.

– **le niveau usine** : tout comme le précédent, ce niveau va devoir mettre en communication des systèmes informatiques hétérogènes. On se trouve cependant ici dans un environnement bureautique beaucoup moins sévère que celui d'un atelier, si bien que des réseaux du type ETHERNET s'y trouveront tout à fait à leur place.
Des efforts sont également menés, à ce niveau, pour arriver, au départ d'ETHERNET, à un standard international : il s'agit du protocole TOP (Technical Office Protocol).

– **le niveau corporation** : ce niveau concerne les échanges de données entre divers sièges d'une même société.
En Belgique, ce genre de communication doit nécessairement transiter par le réseau public placé sous le monopole de la R.T.T. Pendant longtemps, il a fallu se contenter, en cette matière, des lignes téléphoniques destinées à la parole dont la bande passante de 300 à 3.400 Hz limitait considérablement les performances : au mieux 9.600 bits/s.
Actuellement, un véritable réseau de transmission de données est disponible (DCS) sur lequel les vitesses de transmission peuvent atteindre 48 kbits/s maximum. D'autre part, des lignes à haut débit (câbles ou satellites) peuvent être mises à la disposition des usagers avec des vitesses pouvant atteindre 140 Mbits/s !

1.4. LES AUTOMATES PROGRAMMABLES

Comme annoncé au paragraphe précédent, les machines de productions modernes sont pratiquement toutes commandées par des systèmes programmables (les UCP de la figure 1.2). Cela signifie que le fonctionnement de ce type de machine est complètement régi par un programme constitué d'une suite d'instructions stockées dans une mémoire. Ces instructions sont exécutées séquentiellement par un processeur central unique qui a accès à tous les capteurs et actuateurs de la machine.

L'intérêt des systèmes programmables est double :

– la simplicité d'abord car, avec un même matériel de commande, il devient possible de traiter une variété d'applications qui, autrement, auraient chaque fois requis des matériels différents;

– la flexibilité ensuite car le changement du mode de fonctionnement de la machine commandée s'obtient par simple modification du programme enregistré en mémoire.

Bien que toutes les UCP soient actuellement basées sur des microprocesseurs, on peut cependant les distinguer selon le degré de généralité des instructions qui sont mises à la disposition du concepteur d'application.

– Micro-ordinateurs :
C'est évidemment le cas le plus général qui permet d'effectuer n'importe quel traitement . Toutes les fonctionnalités du microprocesseur sont accessibles au programme au travers de langages informatiques universels du type ASSEMBLEUR, C, FORTRAN, PASCAL, BASIC, ...

– *Automates programmables* :
Les instructions disponibles ici conservent encore un caractère universel mais dans le cadre restreint de traitements logiques, c'est-à-dire de traitements qui portent sur des variables binaires (tout ou rien).

– *Contrôleurs dédicacés* :
Les instructions disponibles dans ce dernier cas se limitent à la description de tâches bien précises.
Dans le contexte de la production manufacturière qui nous intéresse ici, on en a deux exemples typiques avec les contrôleurs de machines-outils et les contrôleurs de robot: les instructions servent essentiellement à définir des séquences de mouvements.

Les régulateurs PID programmables constituent un autre exemple d'UCP abondamment utilisées dans l'industrie mais, cette fois, dans le domaine des processus continus.

Les restrictions progressives de généralité expliquées ci-dessus sont obtenues en interposant des couches logicielles de plus en plus "épaisses" entre le microprocesseur de base et le programmeur d'application comme cela est symbolisé à la figure 1.3.

L'effet recherché est évidemment une réduction concomitante de la difficulté de programmation et, partant, du niveau de qualification requis pour le développement, le test et la maintenance des programmes. Ainsi, la programmation d'un micro-ordinateur fera en principe appel à des spécialistes en informatique industrielle. Par contre, ce sont normalement les automaticiens d'usine qui programmeront les automates programmables.

Quant aux contrôleurs dédicacés, ils sont souvent destinés à être programmés par les opérateurs eux-mêmes.

Dans cet ordre d'idée, les automates programmables associent avec un remarquable bonheur des langages de programmation tout à fait conformes aux modes de travail des praticiens concernés et un champ d'application industriel extrêmement vaste. On estime en effet qu'en production manufacturière, plus de 80 % des besoins en matière de contrôle peuvent être couverts par des automates programmables !

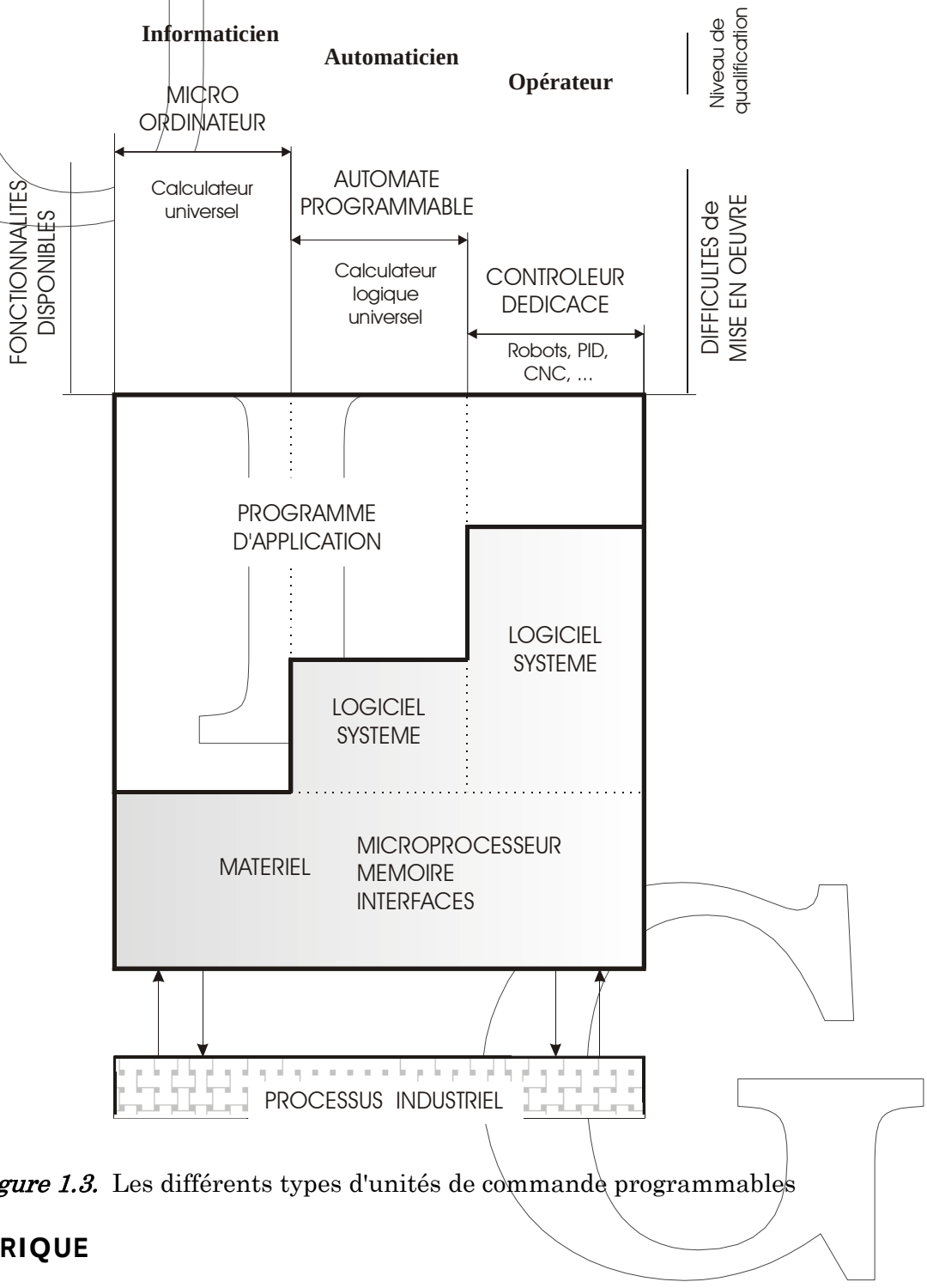


Figure 1.3. Les différents types d'unités de commande programmables

1.4.1. HISTORIQUE

La présentation qui précède est en fait le reflet de la situation actuelle. En réalité, les automates programmables sont apparus en 1969, avant même que n'existent les microprocesseurs. Les premiers processeurs d'automates furent donc construits à l'aide des circuits intégrés disponibles à l'époque.

Ce qu'il est important de noter c'est que les automates furent au départ, et restent encore maintenant, des machines conçues par des automaticiens pour des automaticiens, indépendamment donc des constructeurs d'ordinateurs. Leur parfaite adéquation aux besoins industriels en est la conséquence la plus marquante.

Cette adéquation concerne non seulement les aspects logiciels, comme expliqué ci-dessus, mais aussi les aspects matériels. Les automates sont en effet parfaitement adaptés 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, dispositifs de sécurité en cas de panne ou de chute de tension, etc.

Toutes ces considérations expliquent la foudroyante percée des automates programmables dans l'industrie et l'extraordinaire vigueur d'un marché qui, depuis des années, affiche une progression annuelle de 20 à 30 % ! Cette vigueur s'accompagne d'ailleurs d'une dynamique innovatrice tout à fait remarquable.

Ainsi les microprocesseurs ont-ils été utilisés, dès leur apparition, pour simplifier la conception des processeurs d'automates. Les constructeurs ont ainsi pu développer des familles homogènes d'automates capables de résoudre de manière efficace et rentable toute une gamme de problèmes de contrôle, des plus simples (quelques entrées/sorties) aux plus compliqués (quelques centaines d'entrées/sorties).

D'autre part, sous la poussée enthousiaste des utilisateurs, les automates sont rapidement sortis de leur champ d'application initial purement logique pour intégrer des traitements arithmétiques, des régulations PID, des commandes d'axe, des manipulations de chaînes de caractères, etc.

Il faut cependant noter que, même avec des possibilités qui approchent celles des ordinateurs, la programmation des automates garde toute sa spécificité, à savoir son adéquation aux modes de raisonnement des automaticiens.

Ce sont également les automates programmables qui ont ouvert la voie au contrôle décentralisé. Les premiers automates, en effet, avec leurs processeurs câblés, étaient des machines relativement chères que l'on s'efforçait de charger au maximum.
Depuis l'apparition des automates à microprocesseurs, et la baisse subséquente de leur coût, la tendance est plutôt de répartir les traitements dans des automates de plus faible capacité interconnectés par un réseau de communication. On y gagne en fiabilité intrinsèque (la panne d'un automate n'a que des conséquences limitées), en vitesse (parallélisme des traitements) et en câblage (à condition de pouvoir placer chaque automate au voisinage direct de la partie du processus qu'il commande).

C'est ainsi que pratiquement tous les constructeurs d'automates ont créé très tôt leur propre réseau de communication. Sur la base de cette expérience, ce sont eux aussi qui sont parmi les plus actifs dans le projet MAP (Manufacturing Automation Protocol) de création d'un standard international de communication.

1.4.2. IMPACT DES PC

Le monde des automates programmables n'a pas été long à tirer profit des possibilités nouvelles offertes par l'avènement des micro-ordinateurs de type PC.

D'un côté, en effet, les PC, par l'intermédiaire des réseaux de communications évoqués ci-dessus, permettent de réaliser des systèmes de supervision d'automates programmables très sophistiqués (synoptiques couleur, animations, historiques, etc) à des prix tout à fait abordables. Certains constructeurs d'automates ont développé leurs propres logiciels de supervision. D'autres se sont ralliés à des logiciels "ouverts" conçus par des tiers pour différentes marques d'automates.

D'un autre côté, les PC et leurs formidables capacités de traitement sont de plus en plus utilisés, actuellement comme support de programmation, en remplacement des anciennes consoles spécialisées. Graphismes, documentation, archivage, tests dynamiques, etc, autant de possibilités nouvelles qui accroissent significativement le confort du programmeur.

La mixité des langages est un corollaire particulièrement spectaculaire de cette évolution : sur certains automates, il est maintenant possible d'utiliser conjointement, dans un même programme, différents langages de programmation (relais, littéral, fonctionnel) avec traduction automatique des uns vers les autres; de quoi satisfaire les desiderata de chacun. La figure 1.4. en montre un exemple concret.

On a même vu apparaître des "ateliers de génie logiciel" universels, offrant toutes les facilités énumérées ci-dessus mais sans référence à un type particulier d'automate.
Ce n'est que lorsque le développement de l'application est terminé que, par le jeu d'un postprocesseur approprié, le programme est traduit dans le langage de l'automate cible.

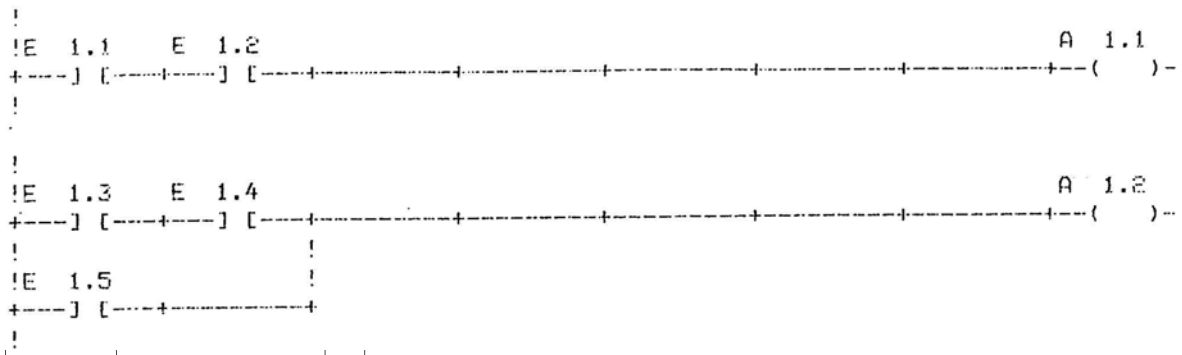
Une traduction en sens inverse existe également pour la mise au point et les tests. Avec de tels ateliers logiciels, il devient donc en principe possible d'utiliser une variété d'automates de marques différentes au travers d'une méthodologie unique de programmation.

1.4.3. NORMALISATION

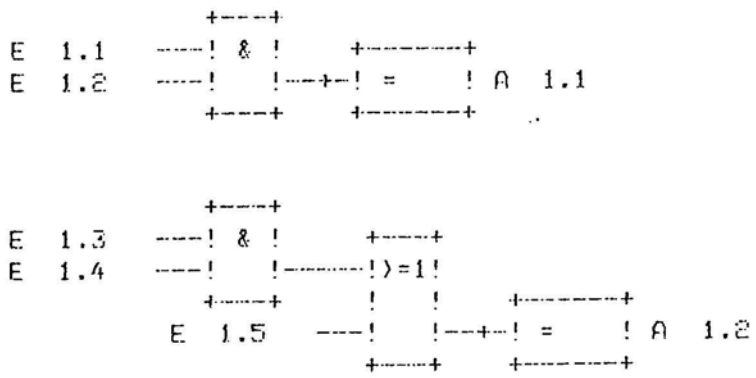
Au niveau de l'exploitation des automates, on se trouve encore actuellement dans une situation comparable à celle qui prévalait en informatique il y a une vingtaine d'années : chaque constructeur propose ses langages de programmation propres, son environnement de programmation propre, ses réseaux de communications propres, ...

Il en résulte une "fidélisation" forcée des utilisateurs car le passage d'une marque d'automates à une autre entraîne un énorme effort de formation pour tout le personnel impliqué.

Les bureaux d'études en automatisme, quant à eux, dont les clients imposent en général le type d'automate à utiliser, dépensent une énergie considérable à garder une maîtrise suffisante de ces systèmes hétéroclites et ont beaucoup de mal à réimplanter des applications similaires d'un automate dans un autre.



a. Relais



b. Blocs fonctionnels

```
0000      :U   E 1.1                    TEST STATE OF INPUT 1
0001      :U   E 1.2                    DO AN AND WITH INPUT 2
0002      :=   A 1.1                    AFFECT RESULT TO INPUT 1
0003      :***

0004      :U   E 1.3                    TEST STATE OF INPUT 3
0005      :U   E 1.4                    DO AN AND WITH INPUT 4
0006      :O   E 1.5                    ORED RESULT WITH INPUT 5
0007      :=   A 1.2                    AFFECT FINAL RESULT TO OUTPUT
0008      :***
```

c. Liste d'instructions

Figure 1.4. Exemple de mixité des langages de programmation

Face à cette situation, des travaux de normalisation ont été entrepris. Ils ont d'abord concerné les systèmes de communication de manière à assurer, si pas la compatibilité, du moins l'interopérabilité des différents automates. A côté du projet MAP déjà évoqué ci-dessus, qui a démarré en 1984, on peut mentionner les propositions relatives aux réseaux de terrain PROFIBUS, FIP, CAN, ASI, ...

A l'heure actuelle, la situation n'est pas encore stabilisée et l'interopérabilité est malheureusement loin d'être chose acquise. Le point sera fait sur la question dans l'ouvrage consacré aux communications et réseaux.

Plus récemment (1992), la CEI a entrepris l'élaboration de normes spécifiques aux automates programmables sous le sigle CEI 1131.

Ces normes visent les aspects suivants :

– CEI 1131-1 (1992) : informations générales
– CEI 1131-2 (1992) : spécifications et essais des équipements
– CEI 1131-3 (1993) : langages de programmation
– CEI 1131-4 (à l'étude) : recommandations à l'utilisateur
– CEI 1131-5 (à l'étude) : spécification service de messagerie

On peut déjà noter que, pour leurs nouveaux modèles, les constructeurs commencent à se référer aux normes. Reste évidemment à contrôler la conformité réelle de ces nouveaux produits. Dans la suite du texte, nous nous efforcerons de suivre la norme dans ses grandes lignes.

1.4.4. TENDANCES DU MARCHE

Au début des années 80, on dénombrait sur le marché belge pas moins de 130 modèles d'automates proposés par une vingtaine de constructeurs [LECOCQ et al., 1983]

Le marché s'est depuis lors fortement décanté et il ne reste plus guère actuellement qu'une demi-douzaine de constructeurs en présence. C'est qu'en effet l'utilisateur, dans le choix d'un automate, privilégie de plus en plus aujourd'hui la "solidité" du constructeur et la qualité de sa représentation locale (service après-vente notamment).

A côté de ces constructeurs d'automates patentés, la normalisation mentionnée au paragraphe précédent ouvre le marché du contrôle industriel à de nouveaux acteurs.

En premier lieu, elle conforte la position des développeurs d'ateliers logiciels ouverts, qui disposent maintenant d'une base de travail solide universellement reconnue. La différence est essentielle car, auparavant, ces développeurs devaient inventer eux-mêmes une méthodologie de programmation et essayer de la faire accepter par leurs clients potentiels, avec tous les risques que cela supposait pour ces derniers en cas de disparition des concepteurs de l'atelier.

Par ailleurs, si la cible principale de ces ateliers logiciels reste bien entendu les automates classiques, rien n'empêche cependant ces ateliers de générer du code pour des calculateurs universels, transformant ceux-ci en véritables automates.

S'il s'agit de calculateurs industriels, ils disposeront d'emblée de la périphérie industrielle nécessaire à la liaison avec le processus.

Dans le cas contraire, on pourra s'appuyer sur la normalisation en matière de communication et piloter des blocs d'entrées/sorties déportées d'automates par le truchement de réseaux ouverts (FIP ou PROFIBUS par exemple).

L'intérêt de cette solution est que les calculateurs universels sont généralement pourvus d'une périphérie standard (disques, écrans, etc.) et d'exécutifs temps réel multitâches. Ceci permet d'associer harmonieusement des fonctions d'automatismes classiques et des fonctions plus spécifiques programmées en langages informatiques comme cela est nécessaire dans beaucoup d'applications pratiques.

Chapitre 2

STRUCTURE DES AUTOMATES PROGRAMMABLES

2.1. ROLE D'UN AUTOMATE

La figure 2.1. schématise le contrôle logique d'un processus.

Les entrées de l'automate (qualifié de "partie commande") sont des signaux provenant du processus (qualifié de "partie opérative) par l'intermédiaire de capteurs (fin de course, détecteurs de présence, etc.). Les sorties de l'automate agissent sur le processus par l'intermédiaire d'actionneurs (contacteurs, commandes de vannes, etc.).

Le rôle de l'automate est de réagir aux changements d'état de ses entrées en modifiant l'état de ses sorties selon une loi de contrôle déterminée a priori par le concepteur du système. Cette loi est dite combinatoire si, à chaque instant, l'état des sorties peut être directement déduit de l'état des entrées. Elle est de type séquentiel, s'il faut en plus tenir compte de l'évolution antérieure du système. Cette dernière peut en général être complètement décrite par l'état d'un nombre fini de variables logiques mémorisées au sein de l'automate.
La possibilité d'intervention d'un opérateur humain a également été symbolisée à la figure 2.1.

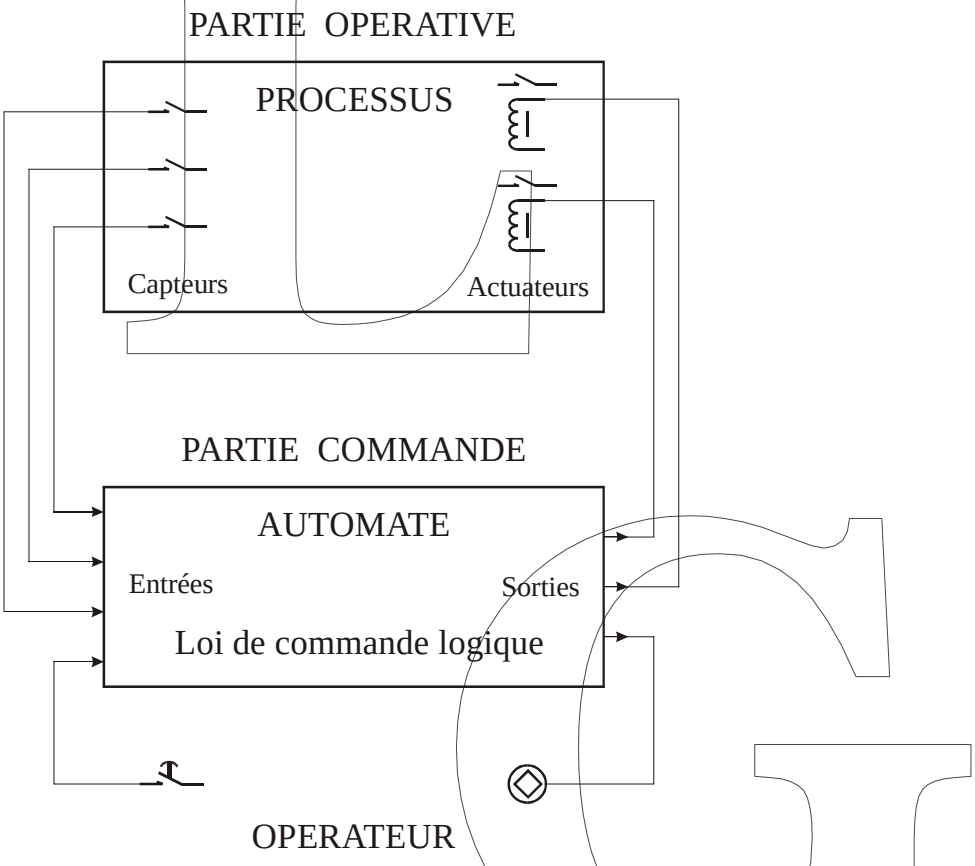


Figure 2.1. Le contrôle logique d'un processus

2.2. PRINCIPE DE LA LOGIQUE PROGRAMMEE

2.2.1. LOGIQUE CABLEE

En logique câblée, la loi de contrôle évoquée ci-dessus est réalisée en interconnectant judicieusement des opérateurs matériels réalisant des fonctions logiques de base. Suivant la technologie adoptée, il peut s'agir d'opérateurs fluidiques (interconnectés par tuyauteries), de relais électromagnétiques ou de relais statiques (interconnectés par fil).

Comme on sait, le nombre de types d'opérateurs nécessaires pour réaliser l'ensemble des fonctions logiques possibles peut être très réduit. Par exemple, les familles d'opérateurs suivantes : [AND, OR, NOT], [porte NAND], [porte NOR], [relais normalement ouvert, relais normalement fermé] permettent, chacune, de réaliser n'importe quelle fonction logique.

Bien entendu, la disposition d'opérateurs supplémentaires (bistables, compteurs, etc.), si elle n'est pas théoriquement nécessaire, est de nature à simplifier considérablement la réalisation d'une fonction logique donnée.

Pour fixer les idées, on considère, à la figure 2.2., l'exemple d'une fonction logique combinatoire supposée réalisée de manière câblée à l'aide d'opérateurs AND, OR, NOT et de relais électromagnétiques.

En extrapolant cet exemple, on peut aisément comprendre les problèmes posés par la logique câblée :

- le volume de matériel est directement proportionnel à la complexité de la fonction réalisée;
- la fonction en question est physiquement inscrite dans le câblage et donc particulièrement difficile à modifier, que ce soit en phase de mise au point ou lors d'extensions ultérieures du processus.

Par contre, la logique câblée présente un certain nombre d'avantages par rapport à la logique programmée définie ci-après.
En particulier, la vitesse de traitement ne dépend pas de la complexité du problème puisque tous les éléments logiques travaillent en parallèle. Pour les relais statiques, cette vitesse peut d'ailleurs être très élevée.
D'autre part, les relais électromagnétiques permettent d'attaquer les étages de puissances (contacteurs par exemple) sans changer de technologie.

2.2.2. LOGIQUE PROGRAMMEE

L'idée de la logique programmée est de n'utiliser qu'un seul jeu d'opérateurs de base (qui portera le nom d'unité logique). Pour réaliser une fonction logique donnée, telle celle présentée à la figure 2.2., on emploiera ce jeu unique pour calculer successivement les différents termes de la fonction et les combiner de manière à arriver ainsi, de proche en proche, au résultat cherché.

On travaille donc en quelque sorte ici par "balayage". Il est clair que si ce balayage est répété à une cadence suffisamment rapide par rapport à la dynamique des signaux (et c'est évidemment une condition sine qua non), on aura l'impression d'un fonctionnement parallèle semblable à celui de la logique câblée.

En pratique, on essaye généralement d'avoir des cadences de répétition du même ordre de grandeur que les temps de basculement des relais (de quelques ms à quelques dizaines de ms).

La manière dont le balayage en question doit être effectué est décrite par une suite d'instructions stockées dans une mémoire et constituant ce que l'on appelle un programme.

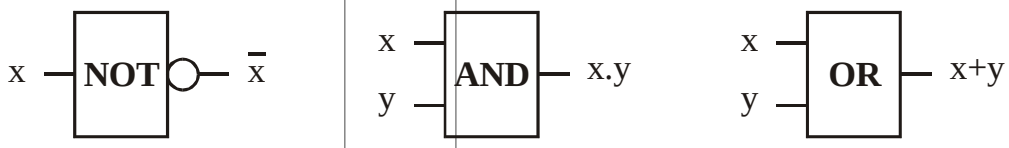
Exemple de fonction logique combinatoire

$$\mathbf{O1} = (\overline{\mathbf{I1}} \,.\, \mathbf{I2} + \mathbf{I3}) \,.\, (\mathbf{I4} + \mathbf{I5})$$

O1 := ((NOT **I1** AND **I2)** OR **I3**) AND (**I4** OR **I5**)

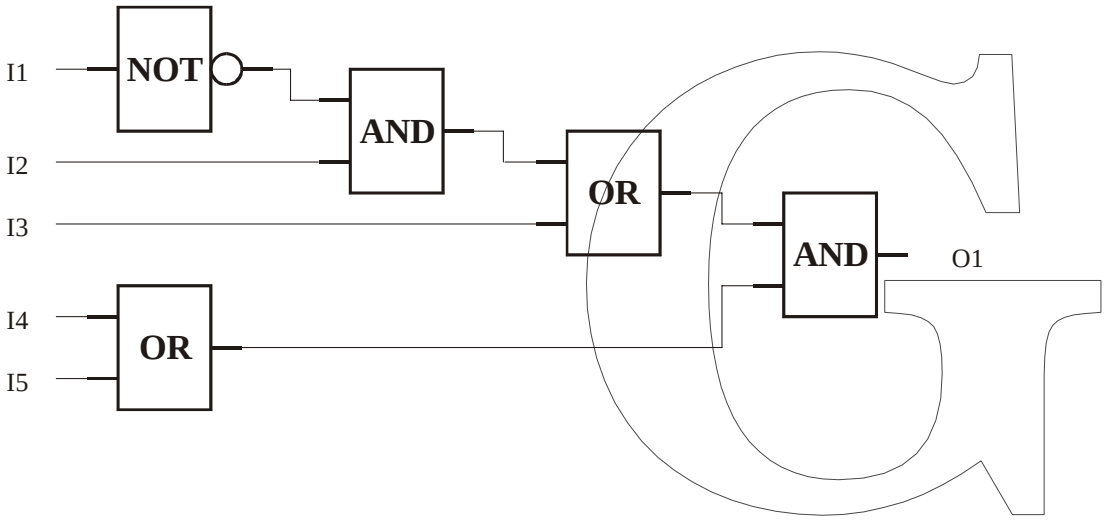
Solution électronique:

Opérateurs disponibles



Inversion logique | ET logique | OU logique

Réalisation de la fonction



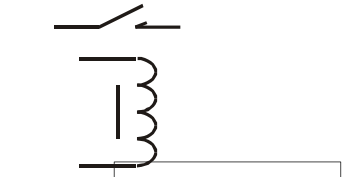
Exemple de fonction logique combinatoire

O1 = (**/I1** . **I2** + **I3**) . (**I4** + **I5**)

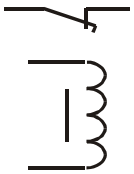
O1 := ((NOT **I1** AND **I2)** OR **I3**) AND (**I4** OR **I5**)

Solution à relais

Opérateurs disponibles

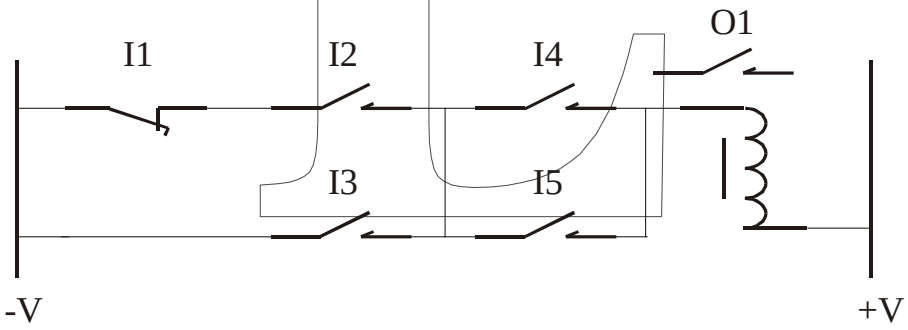


Relais normalement ouvert



Relais normalement fermé

Réalisation de la fonction



Représentation symbolique

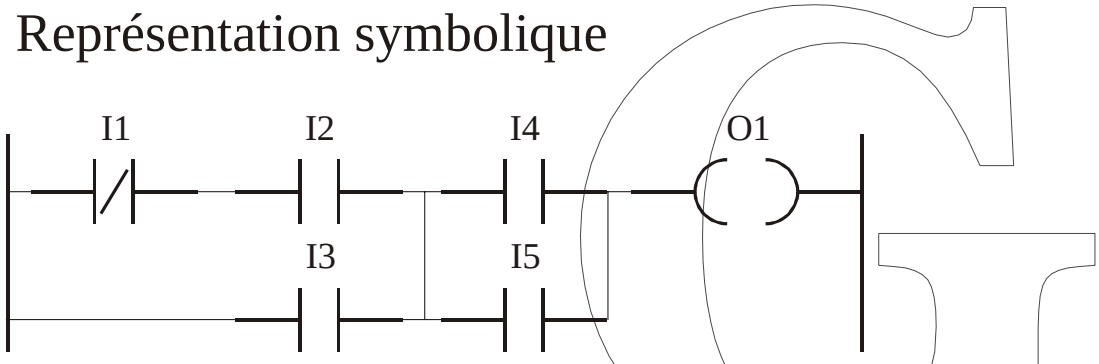


Figure 2.2. Logique câblée

2.3. PRINCIPE DE L'AUTOMATE PROGRAMMABLE

La figure 2.3. montre comment les choses peuvent être organisées.

2.3.1. UNITE LOGIQUE

L'unité logique comporte les 3 opérateurs de base décrits plus haut. Il lui est associé une mémoire de données, constituée d'une batterie de bistables, appelés bits, qui sera utilisée pour la réalisation de fonctions logiques séquentielles ainsi que pour la mémorisation de résultats intermédiaires dont nous verrons la nécessité dans l'exemple ci-après.

Une série de sélecteurs symbolise la possibilité de mettre en communication les opérateurs de l'unité logique avec les entrées, les sorties et la mémoire de données. Bien entendu, dans la réalité, tout se passe de manière électronique.

2.3.2. ACCUMULATEUR LOGIQUE

On remarque aussi la présence d'un accumulateur (ACCU) qui est une cellule de mémoire particulière intervenant de manière privilégiée dans les opérations de l'unité logique. En effet, d'une part l'accumulateur constitue une opérande obligée pour les opérateurs AND et OR et, d'autre part, le résultat des opérations est systématiquement obtenu dans cet accumulateur.

Comme nous le comprendrons mieux ci-après, cette manière de faire conduit à un codage très économique (en place mémoire) des instructions puisque, des trois opérandes intervenant normalement dans une opération (deux entrées et une sortie), une seule doit être explicitement indiquée.

2.3.3. UNITE DE COMMANDE

La figure 2.3. montre encore comment le fonctionnement de l'ensemble est géré par l'unité de commande de l'automate sous le contrôle des instructions stockées dans la mémoire de programme.

Les instructions sont amenées l'une après l'autre dans le registre d'instruction. Elles sont décodées et, en fonction des opérations qu'elles indiquent, l'unité de commande engendre la séquence d'ordres nécessaires à leur exécution.

Dans le symbolisme de la figure 2.3., ces ordres consistent en fait à positionner les différents sélecteurs. L'exécution séquentielle et cyclique des instructions du programme est figurée par la présence d'un contacteur rotatif. Dès que l'exécution d'une instruction est terminée, l'unité de commande donne l'ordre au contacteur rotatif d'avancer d'un pas, afin de sélectionner l'instruction suivante et ainsi de suite, indéfiniment.

On peut déjà, à ce stade, pressentir tout l'intérêt de la logique programmée par rapport à la logique câblée :

– le matériel est complètement banalisé; la spécialisation à chaque application particulière, se fait au niveau logiciel c'est-à-dire au niveau du programme enregistré dans la mémoire. Il en résulte une extrême souplesse pour la mise en oeuvre et les modifications du système, et une simplification appréciable de sa maintenance.

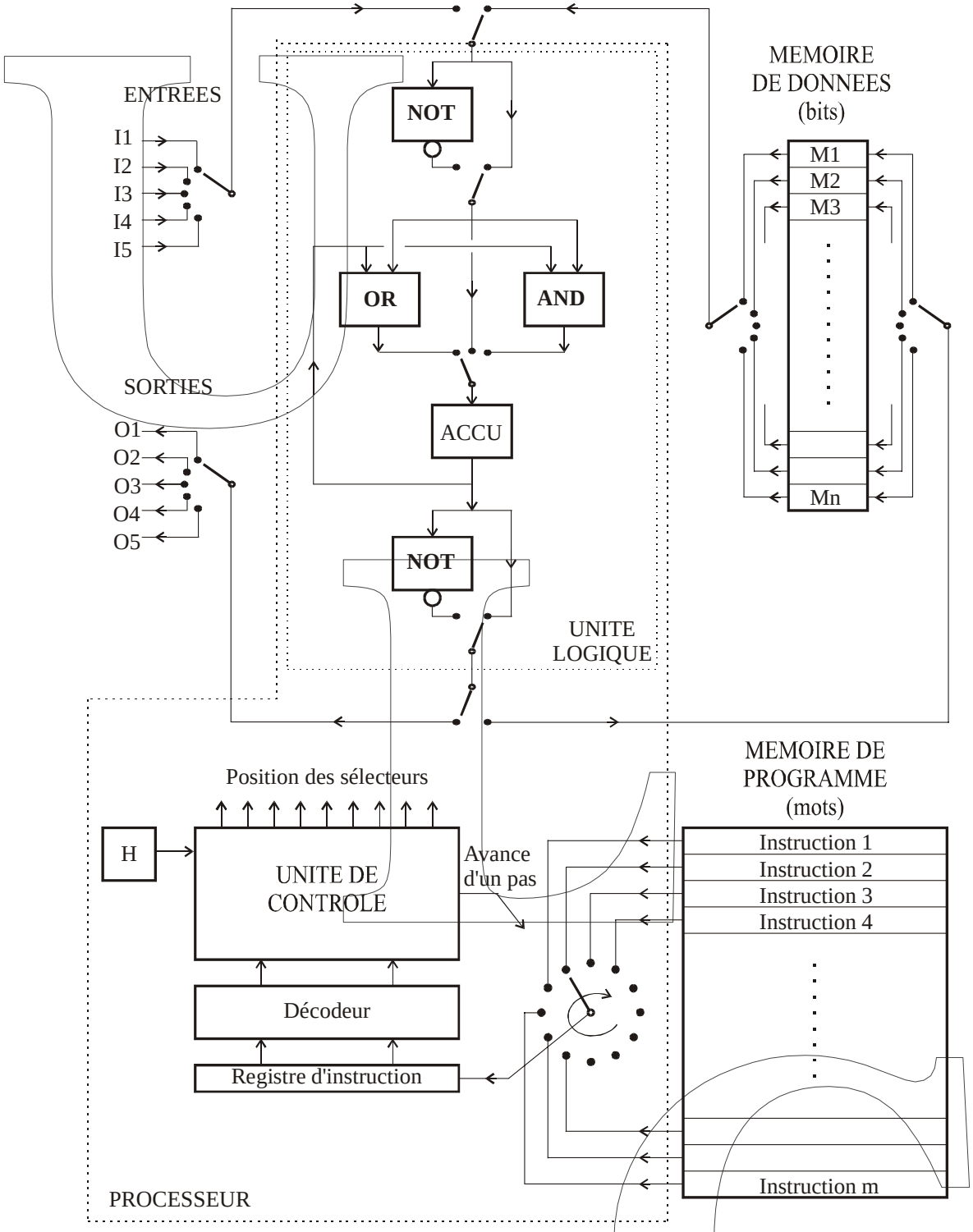


Figure 2.3. Principe de l'automate programmable

– la liaison entre le volume de matériel et la complexité des fonctions réalisées est infiniment moindre que dans le cas câblé, puisqu'elle n'affecte plus ici que les interfaces d'entrées/sorties et la taille de la mémoire.

2.3.4. FORMAT DES INSTRUCTIONS

D'une manière générale, une instruction doit fournir à l'unité de commande toutes les indications nécessaires sur la nature de l'opération à effectuer et sur l'opérande explicite qui y intervient.

Il est clair qu'il s'agit là d'un contenu informatif dépassant largement le simple tout ou rien. Pour le représenter dans la mémoire, il sera dès lors nécessaire d'utiliser un ensemble de bits formant un mot. La mémoire de programme sera donc organisée en mot comme montré à la figure 2.3.

La figure 2.4. montre l'exemple d'un format d'instruction utilisant des mots de 16 bits. On y trouve un champ de 5 bits, destiné à préciser l'opération à effectuer (code opératoire), l'autre de 11 bits, destiné à contenir l'adresse de l'opérande.

Avec ce format, on peut donc coder un maximum de $2^5 = 32$ opérations et on peut adresser un maximum de $2^{11} = 2048$ (2 K) opérandes.
En pratique, on réserve souvent un bit du code opératoire pour indiquer si l'opérande doit être inversée ou non avant traitement. De même, on considère souvent un sous-champ dans l'adresse de l'opérande où l'on code le type de l'opérande considérée. On a prévu, ici, deux bits qui permettent de distinguer :

% I : les entrées
% Q : les sorties
% M : les mémoires

On peut ainsi adresser $2^9 = 512$ opérandes de chaque type. Remarquons que, si l'on n'avait pas adopté une structure avec accumulateur pour le processeur de l'automate, il aurait fallu, en principe, multiplier par 3 le champ d'adresse !

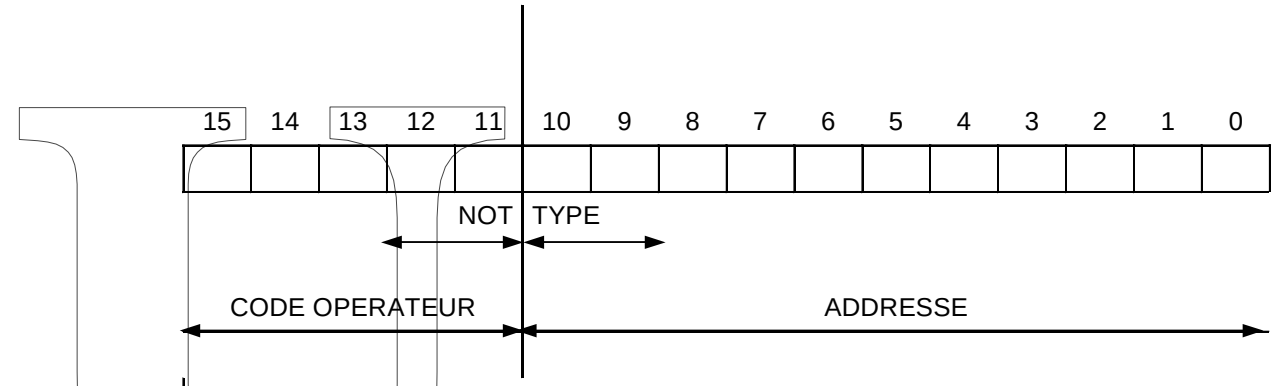
Sur la figure 2.4., on a également indiqué le codage d'un jeu d'instructions plausibles pour notre automate.

A côté des instructions relatives aux opérations logiques proprement dites (AND, OR), on a dû aussi prévoir une instruction pour charger une opérande (entrée ou mémoire) dans l'accumulateur (LD) et, réciproquement, une instruction (ST) pour ranger le contenu de l'accumulateur à une adresse donnée (sortie ou mémoire).

Enfin, l'instruction RET, à placer obligatoirement en bout de programme, indique à l'unité de commande du processeur que la séquence d'instructions du programme a été complètement balayée et qu'il y a lieu de relancer le cycle à la première instruction.

La figure 2.5. montre le programme correspondant à la fonction logique de la figure 2.2.

Les commentaires associés aux instructions expliquent la démarche suivie et mettent clairement en évidence la notion de balayage évoquée ci-avant.
Remarquons qu'après l'instruction 3, on a été obligé de sauver temporairement la valeur de l'accumulateur de manière à pouvoir utiliser celui-ci pour le calcul du OR entre % 14 et % 15.



SYMBOLES								
Dir/Inv								
N					1			Inversion logique
–					0			Utilisation directe
Type								
I						0	1	Entrée
O						1	0	Sortie
M						1	1	Mémoire
Fonction								
AND	0	0	0	1				AND avec l'accumulateur
OR	0	0	1	0				OR avec l'accumulateur
LD	0	0	1	1				Charger l'opérande dans l'ACCU
ST	0	1	0	0				Placer l'ACCU dans l'opérande
RET	0	0	0	0				Boucler à la première instruction

Figure 2.4. Exemple de format d'instruction

2.3.5. ORGANISATION DU CYCLE D'UN AUTOMATE

La dynamique d'un processus étant ce qu'elle est, rien n'empêche certaines entrées de changer d'état en plein milieu du cycle de calcul de l'automate. Il pourrait donc arriver que les premières instructions du programme aient travaillé avec des entrées dans un certain état et que les dernières instructions travaillent avec des entrées dans un autre état. On comprend que cela pose un problème de cohérence qui, dans certains cas, pourrait fausser les résultats et donc se révéler dangereux pour le matériel et le personnel.

Réciproquement, on notera qu'au cours d'un cycle de l'automate et selon la manière dont les calculs sont organisés, les variables, et les sorties en particulier, peuvent prendre des états intermédiaires différents des états finals désirés. Il serait évidemment risqué de laisser passer ces transitoires vers le processus.

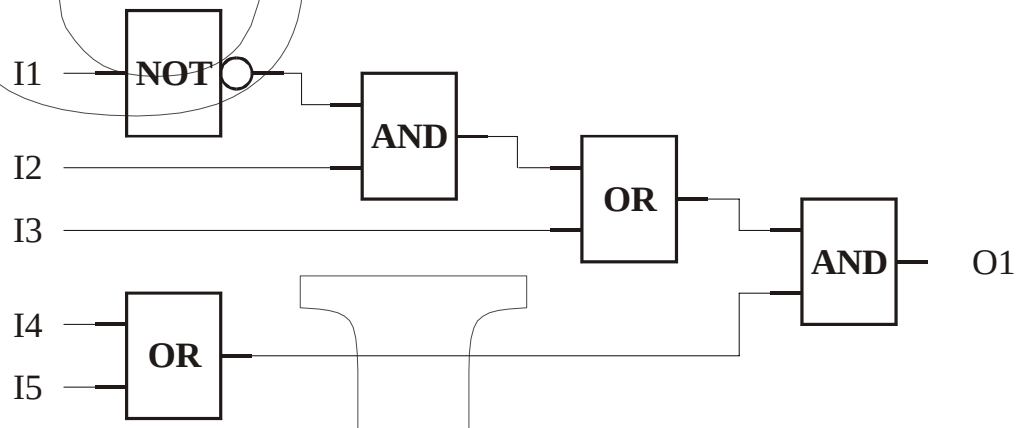
Sur base des considérations qui précèdent, on organise généralement le cycle de l'automate comme montré à la figure 2.6. Les cartes d'entrées sont scrutées en tout début de cycle et leur état est placé dans une table image des entrées. Toute référence à une entrée dans le programme est alors transformée automatiquement (c'est-à-dire de manière

transparente pour l'utilisateur) en une référence à la table image des entrées, celle-ci demeurant évidemment inchangée durant tout le cycle.

De même, les sorties calculées par le programme sont envoyées dans une table image des sorties. Ce n'est qu'en fin de cycle, lorsque tous les calculs sont terminés, que cette table est transférée dans les cartes de sorties.

Exemple de fonction logique combinatoire

O1 := ((NOT **I1** AND **I2)** OR **I3**) AND (**I4** OR **I5**)



Solution programmée

1	LDN	I1	mettre I1 inversé dans l'ACCU
2	AND	I2	ET entre ACCU et I2 (résultat dans ACCU)
3	OR	I3	OU entre ACCU et I3 (résultat dans ACCU)
4	ST	M1	sauver ACCU dans mémoire M1
5	LD	I4	mettre I4 dans ACCU
6	OR	I5	OU entre ACCU et I5 (résultat dans ACCU)
7	AND	M1	ET entre ACCU et M1(résultat dans ACCU)
8	ST	O1	mettre la valeur de l'ACCU dans la sortie O1
9	RET		boucler sur l'instruction n° 1

Figure 2.5. Exemple de programme

2.3.6. LANGAGE ET CONSOLE DE PROGRAMMATION

Les configurations de 1 et 0 présentées à la figure 2.4. constituent ce que l'on appelle le langage machine car c'est le seul que comprenne le processeur d'un automate.

Il ne saurait évidemment être question d'obliger un automaticien à programmer dans un tel langage du fait de la longueur des instructions et de leur aspect hermétique.

Il est donc indispensable de mettre à la disposition de l'utilisateur un langage de programmation beaucoup plus fonctionnel pour décrire à l'automate la loi de contrôle qu'il veut voir exécuter.

Dans l'exemple de programme de la figure 2.5., on a en fait déjà utilisé un langage de programmation, le langage à liste d'instructions : les opérations réalisées et les adresses des opérandes y apparaissent en clair. Ce langage reste cependant encore fort voisin du langage machine et nous verrons, au chapitre 4, des langages plus conformes aux modes de raisonnement des automaticiens. Ceci implique évidemment la nécessité d'une traduction du langage de programmation vers le langage machine préalablement à toute exécution.

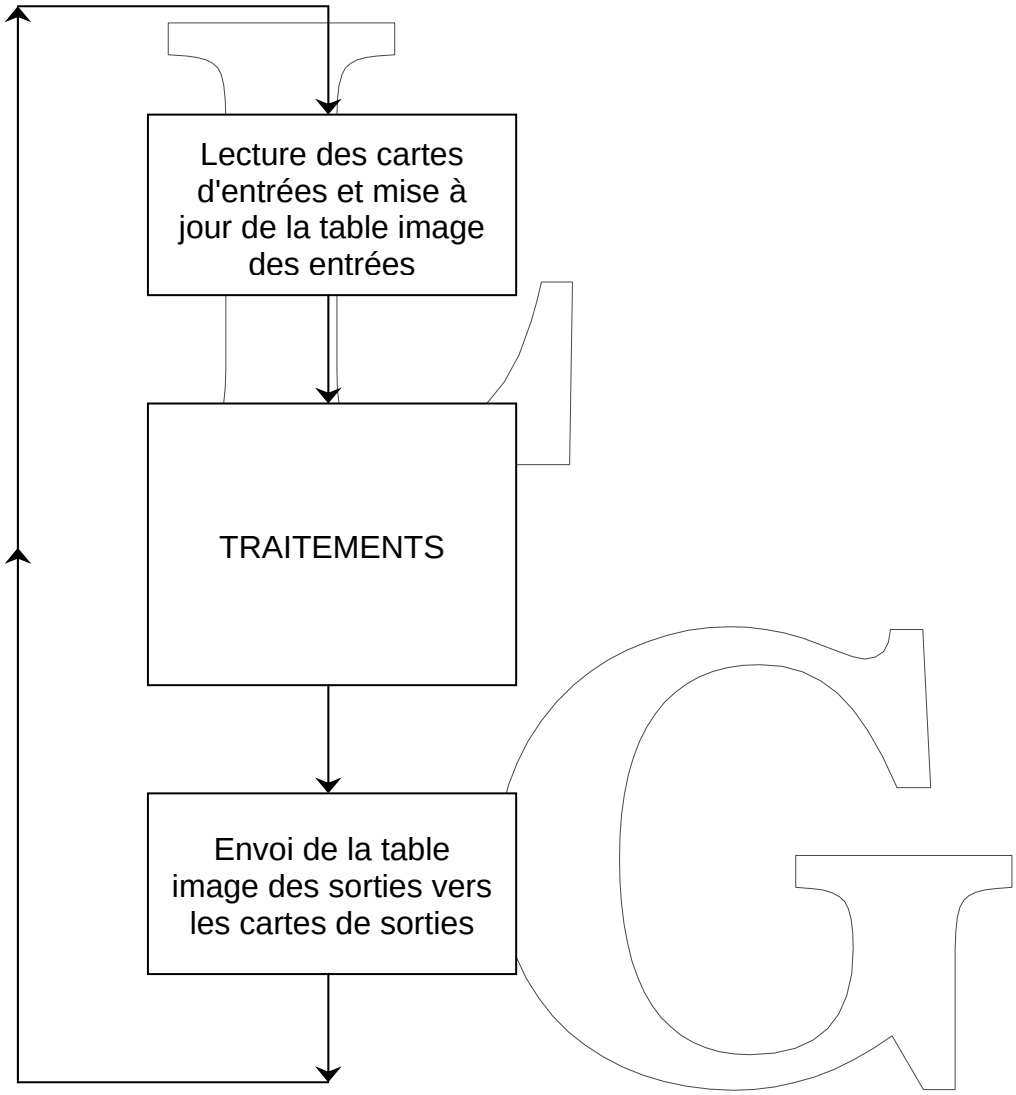


Figure 2.6. Organisation du cycle d'un automate

Dans le contexte des automates, cette traduction est en général réalisée par un dispositif extérieur appelé console de programmation. Il est clair que cette traduction sera plus ou moins directe (et donc la console plus ou moins simple) selon le degré d'éloignement du langage par rapport au langage machine.

Au départ, la console constituait un équipement spécialisé, souvent fort coûteux, propre à chaque type d'automate programmable. Actuellement, tous les constructeurs utilisent des PC comme console de programmation, d'autant que les versions portables de ces ordinateurs permettent de travailler sans problème sur site industriel.

2.4. TECHNOLOGIE DE REALISATION

2.4.1. BUS D'ECHANGE

Dans l'exemple du paragraphe 2.3.4, on a pu constater que les échanges d'informations entre les différents éléments de l'automate (entrées, sorties, mémoires) transitent toujours par le processeur. De plus, du fait du fonctionnement séquentiel du système, il n'y a jamais qu'un élément à la fois qui est en communication avec le processeur.

Il est dès lors tout à fait possible d'utiliser un chemin commun et une procédure commune pour ces échanges. Le chemin en question, appelé BUS, est constitué de lignes d'adresses, de lignes de données et de lignes de contrôle. Ces lignes véhiculent des signaux binaires.

L'automate programmable se présente donc finalement comme montré à la figure 2.7., sous la forme d'une collection de modules fonctionnels greffés sur le BUS. On reconnaîtra là, la structure typique d'un ordinateur.

Nous passerons maintenant rapidement en revue les principaux modules que l'on peut rencontrer en pratique.

2.4.2. PROCESSEUR

Le schéma fonctionnel présenté au paragraphe 2.3. constitue en quelque sorte un cahier des charges et nous examinerons maintenant comment il peut être réalisé pratiquement.

Comme on l'a signalé dans l'introduction, la technologie câblée utilisée dans les premiers automates a été complètement abandonnée au profit de l'utilisation de microprocesseurs.

Etant donné ses capacités universelles de traitement, le microprocesseur peut évidemment assurer les fonctionnalités demandées à un automate.

Il est en effet possible de faire correspondre au jeu d'instructions de l'automate, un jeu de sous-routines écrites dans le langage du microprocesseur et qui donnent lieu aux traitements désirés. Ce jeu de sous-routines constitue ce que l'on appelle le logiciel système de l'automate; il est stocké dans une mémoire dite "système".

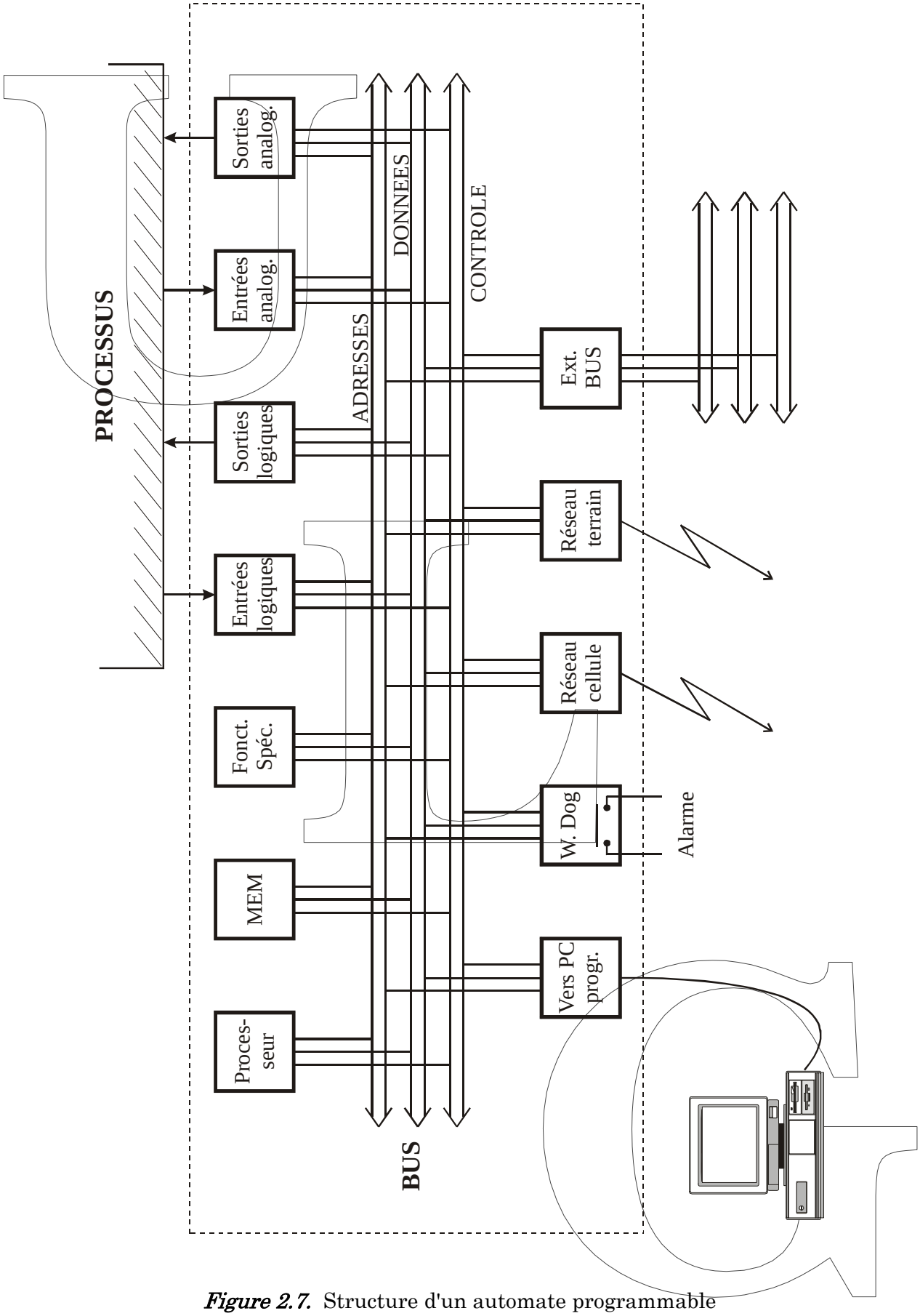


Figure 2.7. Structure d'un automate programmable

Pour exécuter un programme de logique, le microprocesseur analysera successivement chaque instruction logique et, selon le code opératoire trouvé, lancera la sous-routine qui convient.

On comprend qu'il s'agit d'une procédure assez lourde, encore aggravée par le fait qu'un microprocesseur est prévu pour travailler de préférence sur des mots. L'accès au bit, bien que toujours possible, est plus ou moins malaisé selon le type de microprocesseur, ce qui allonge les sous-routines systèmes.

Si malgré ce sombre tableau, les microprocesseurs sont abondamment utilisés par les constructeurs d'automates, c'est pour les trois raisons principales suivantes :

– Les vitesses de traitement atteintes par les microprocesseurs modernes sont telles que le problème des performances ne se pose pas vraiment en pratique courante (ce n'était pas le cas avec les microprocesseurs de la première génération).
Ceci est d'ailleurs d'autant plus vrai que la tendance est actuellement à la décentralisation des systèmes de contrôle, mettant en oeuvre des réseaux de petites unités de traitement de préférence à de grosses unités centralisées.

– Le développement prodigieux des microprocesseurs et des circuits associés rend les solutions à microprocesseur de très loin les plus faciles à réaliser et, partant, les plus économiques.

– L'emploi de microprocesseurs ouvre la porte à toute espèce d'extension du jeu d'instructions des automates programmables : créer une nouvelle instruction revient simplement à ajouter la sous-routine correspondante au logiciel système.

Il n'empêche que, dans les automates de haut de gamme, on recourt systématiquement à des solutions multiprocesseurs comme cela est symbolisé à la figure 2.8.

On y retrouve en fait un processeur "câblé" ultrarapide sous forme d'un circuit ASIC (Application Specific Integrated Circuit) pour les traitements purement logiques. Un microprocesseur plus standard est utilisé pour les fonctionnalités additionnelles courantes (arithmétiques notamment). Le cas échéant, un troisième processeur prendra en charge les communications vers la console et/ou des réseaux informatiques.

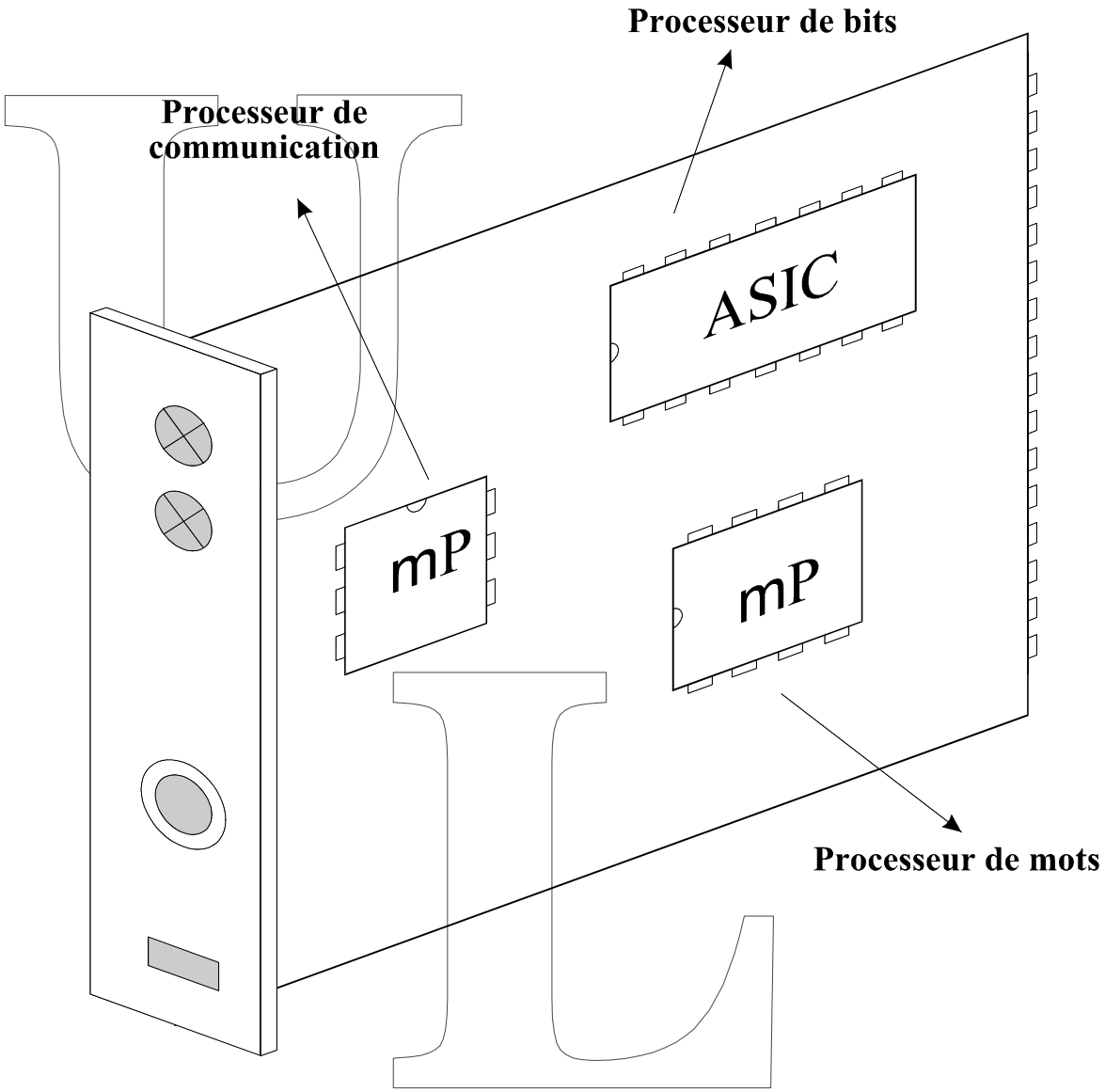


Figure 2.8. Exemple d'une carte processeur d'un automate haut de gamme

2.4.3. MEMOIRES

Les automates actuels utilisent essentiellement des mémoires à semi-conducteurs. Comme on sait, ce type de mémoire pose le problème de la perte d'information en cas de coupure d'alimentation. Cela a conduit au développement de deux grandes familles de mémoires : les mémoires vives (RAM), permettant la lecture et l'écriture mais "volatiles", et les mémoires mortes (ROM), n'autorisant que la lecture mais non volatiles.

Dans les automates, on a des exigences différentes pour la mémoire de données et la mémoire de programme.

- **Mémoire de données**

  C'est clairement la place ici, de mémoires de type RAM.
  Même dans le cas de mémoires RAM, il est possible de se prémunir contre des coupures accidentelles d'alimentation en utilisant une technologie spéciale (CMOS) à très faible consommation électrique du moins, à l'état de repos (5000 fois moins qu'une RAM classique).

  On peut en effet dans ce cas prévoir, au niveau des cartes de mémoire, des micro-batteries, des micro-piles voire des condensateurs, susceptibles d'alimenter les circuits en cas de coupure de l'alimentation principale. Les temps de sauvegarde garantis vont de quelques jours à quelques mois.

- **Mémoire de programme**

  La mémoire de programme de l'automate doit pouvoir subir sans dommage des coupures plus ou moins longues d'alimentation (entretiens, congés, etc.). C'est la technologie EPROM (erasable PROM) que l'on rencontre le plus fréquemment ici. Elle présente l'inconvénient que les circuits de mémoire doivent être programmés et effacés (par exposition aux UV dans un dispositif extérieur à l'automate). Il en résulte des manipulations particulièrement gênantes en phase de mise au point de l'équipement.

  La technologie EEPROM (electrically erasable PROM), de développement plus récent, commence à apparaître chez certains constructeurs. L'effacement se fait ici par voie électrique et peut donc être réalisée in situ.

  Enfin, signalons qu'une certaine confiance s'est établie, à l'usage, dans les mémoires RAM gardées et que beaucoup de constructeurs proposent une option de ce type pour la mémoire programme de leurs automates.

- **Mémoire système**

  Cette mémoire, présente dans le cas d'automates à microprocesseurs, est programmée en usine par le constructeur; elle peut donc sans problème être réalisée en technologie EPROM voire PROM (c'est-à-dire programmable une seule fois, sans possibilité d'effacement).

2.4.4. MODULES D'ENTREES/SORTIES INDUSTRIELLES

Il s'agit de modules réalisant l'interface entre les signaux de processus (en provenance des capteurs ou à destination des actionneurs) et les signaux du BUS interne de l'automate.

L'interface réalise trois fonctions principales :

– le découplage mécanique (bornier) entre le câblage processus et le câblage interne de l'automate;

– le découplage électrique entre les signaux de processus et l'électronique de l'automate, avec toutes les adaptations, conversions et protections nécessaires;

– la synchronisation des transferts conformément aux procédures d'échange du BUS de l'automate.

Des entrées/sorties logiques, c'est-à-dire par tout ou rien, sont évidemment toujours présentes par définition.

Pour les automates qui le permettent, on peut aussi trouver des modules d'entrées/sorties analogiques.

Les modules d'entrées/sorties industrielles sont des éléments particulièrement bien soignés sur tous les automates. Ils seront étudiés en détail au chapitre 3.

2.4.5. FONCTIONS SPECIALES

Dès lors que les processeurs des automates comportent des microprocesseurs, ces derniers sont en principe capables de prendre en charge toute espèce de fonction spéciale. L'utilisation de modules séparés vise alors à soulager le processeur principal de traitements requérant des temps de calcul importants et qui pourraient poser des problèmes d'exécution en temps réel. C'est ainsi qu'on trouve des modules de régulation PID, de commande d'axe, de comptage rapide, ...

Ceci ne fait que renforcer le caractère multiprocesseur des automates déjà évoqué au paragraphe 2.4.2. On trouve même des modules à greffer sur le bus des automates qui constituent de véritables PC tournant sous Windows ! Notons que, réciproquement, on trouve des cartes automates pouvant être intégrées dans des PC.

2.4.6. MODULE DE COUPLAGE

On classera dans cette rubrique les modules qui permettent de connecter l'automate à d'autres systèmes de traitement et qui travaillent donc dans un contexte de signaux d'entrées/sorties informatiques plutôt qu'industriels.

– **Console de programmation et de test**

Il s'agit de coupleurs qui servent pour la liaison à l'automate des consoles de programmation et de test.

– **Extension du BUS**

Les coupleurs sont utilisés pour étendre le BUS de l'automate dans le cas de configuration multi-châssis. Notons que les extensions possibles sont toujours très limitées en longueur (quelques mètres maximum).

– **Communications**

Il s'agit de coupleurs permettant d'établir des communications à distance par lignes séries : paires téléphoniques, coaxes, fibres optiques, ...
Ils sont étudiés en détail dans l'ouvrage "Réseaux locaux industriels".

– **Décentralisation des entrées/sorties**

Les coupleurs dont question ici servent à décentraliser des châssis d'entrées/sorties industrielles. Les liaisons sont de type série, comme dans le cas précédent, ce qui permet des décentralisations relativement importantes (de l'ordre du kilomètre sans

amplification). Cette possibilité de décentralisation permet, dans de nombreux cas, de réduire substantiellement le volume de câblage entre le processus et l'automate.

2.4.7. MODULES DE SURVEILLANCE ET DE CONTROLE

Il s'agit de modules chargés de surveiller et de contrôler le bon fonctionnement du matériel aussi bien que du logiciel. La fonction la plus communément rencontrée est celle dite du "chien de garde" (watch dog) qui contrôle le cycle de l'automate.

Le principe consiste à obliger l'automate à envoyer à chaque cycle une impulsion au système de surveillance. Celui-ci vérifie simplement que le temps entre deux impulsions ne dépasse pas une limite fixée à priori. Seront ainsi détectés, soit, une panne de processeur, soit un bouclage intempestif au sein du programme.

Le "watch dog" ainsi que d'autres fonctions de surveillance classique seront examinés au chapitre 3.

2.5. LES AUTOMATES PROGRAMMABLES VIRTUELS (Soft PLC)

Le développement foudroyant de la puissance des PC et l'apparition de systèmes d'exploitation multitâches (Windows NT et suivant) ont permis d'introduire la notion de "Soft PLC".

Il s'agit en fait, d'une tâche s'exécutant sous Windows et qui simule très exactement le fonctionnement d'un automate programmable. On affecte une priorité élevée à la tâche en question de manière à obtenir un temps de cycle aussi constant que possible pour cet automate virtuel. Ce temps est de l'ordre de quelques dizaines de ms.

La mise en œuvre des programmes (langages, tests, etc.) est également en tout point semblable à celle des automates classiques.

Bien entendu, il n'existe pas, pour les PC, de cartes d'entrées/sorties industrielles du type décrit au chapitre 3. Le problème peut cependant être contourné en faisant usage de réseaux de terrain (voir tome 2) qui proposent pratiquement tous des coupleurs pour PC. Toute la périphérie industrielle décentralisée devient ainsi disponible (voir § 3.2.2.).

Un dernier point faible subsiste cependant : les dispositifs de sécurité tels que décrits au chapitre 3 (surveillance du cycle, réaction aux coupures d'alimentation, ...) ne sont pas encore courants sur les PC.

Il n'empêche que, pour des besoins didactiques ou même pour des applications industrielles non critiques, le Soft PLC constitue une alternative très intéressante sur le plan économique.

Chapitre 3

INTERFACES INDUSTRIELLES ET DISPOSITIFS DE SECURITE

3.1. INTRODUCTION

Ce chapitre étudie les problèmes de l'interfaçage entre un automate programmable et le processus qu'il contrôle. On notera que des problèmes d'interfaçage se posent pour tout système informatique appelé à être directement raccordé à un processus physique. Les solutions présentées ci-dessous sont donc tout à fait générales.

3.1.1. TYPES DE SIGNAUX

Au niveau d'un processus industriel, les signaux de mesure et de commande peuvent être rangés en deux grandes catégories : les signaux logiques, encore appelés digitaux ou "tout ou rien", et les signaux analogiques.

– **les signaux logiques** : il s'agit bien entendu des signaux les plus fréquemment rencontrés dans le domaine des automates programmables. Ils représentent l'état de contacteurs, de boutons poussoirs, de détecteurs de présence, de voyants lumineux, etc.
Ces signaux se présentent typiquement sous les formes suivantes :
  . tensions continues : 12, 24, 48 ou 60 V (avec une nette prédominance du 24V)
  . tensions alternatives : 110, 220 V
  . contacts libres de potentiel

  Dans ce dernier cas, il ne s'agit pas à proprement parler d'un signal; l'information binaire y est en fait représentée par l'état ouvert ou fermé d'un contact dont les deux bornes sont accessibles à l'utilisateur.

– **les signaux analogiques** : ils représentent électriquement des grandeurs physiques qui varient d'une manière continue : vitesses, températures, pressions, etc.

  Ces signaux se présentent typiquement sous les formes suivantes :
  . tensions haut niveau, par exemple - 10 à + 10 V
  . tensions bas niveau, par exemple - 50 à + 50 mV
  . courants, par exemple 0 - 20 mA, 4 - 20 mA

  Dans ce dernier cas, on repasse généralement en tension à l'entrée de l'automate à l'aide d'une résistance de précision. Les transmissions en courant sont favorables du point de vue de l'immunité aux parasites. De plus, la limitation à 20 mA empêche la formation d'arc électrique en cas de court circuit et prévient ainsi les risques d'explosion en environnement dangereux.
  Enfin, la gamme 4 - 20 mA a encore l'avantage de permettre la détection aisée d'une coupure des conducteurs. En effet, dans ce cas, le courant est nul et sort donc de la gamme normale.

Pour terminer, on signalera encore que la plupart des automates programmables possèdent des entrées analogiques spécialement adaptées aux capteurs usuels de température : thermocouples standards et sondes Pt 100.

3.1.2. ROLE DES INTERFACES INDUSTRIELLES

Pour les entrées, l'interface doit réaliser les fonctions suivantes :

- adaptation mécanique aux standards industriels de câblage (borniers à vis par exemple);
- adaptation électrique aux standards industriels (signaux en courant, en tension, signaux bas niveaux, hauts niveaux, etc);
- protection contre les tensions de mode commun (voir § 3.1.3);
- atténuation des signaux parasites (bruit capté par les lignes de transmission, rebondissement des contacts, etc).

Pour les sorties, c'est-à-dire pour les commandes envoyées au processus, on trouve bien entendu des fonctions analogues. En plus, il faut prendre garde qu'en cas de panne du calculateur, des commandes aberrantes ne puissent être émises, qui présenteraient un danger pour la vie du matériel et/ou du personnel.

Ceci suppose la présence de dispositifs permettant au minimum

- de détecter de manière précoce les anomalies de fonctionnement du calculateur;
- de placer les commandes dans un état de sécurité défini à l'avance.

Ceux-ci seront présentés au paragraphe 3.5.

Il importe encore de signaler, dans cette introduction, que les interfaces et dispositifs de sécurité qui viennent d'être évoqués se rencontrent de manière tout à fait courante dans le contexte des mini-ordinateurs de processus et des automates programmables. Comme on l'a dit déjà dit plus haut, ce n'est pas encore le cas des PC.

3.1.3. TENSIONS DE MODE COMMUN

On présentera brièvement dans ce paragraphe quelques considérations sur les tensions de mode commun, dans la mesure où tous les lecteurs ne sont pas familiers avec le problème.

Origine

Les tensions de mode commun trouvent leur origine dans les différences de potentiels qui peuvent exister entre les références de tension des capteurs et les références de tension des systèmes de traitement auxquels ils sont raccordés.

La figure 3.1.a présente l'exemple du raccordement d'un capteur par un seul fil, avec retour par la masse.

Si une différence de potentiel existe entre les masses (v sur la figure 3.1.a), le signal reçu sera évidemment biaisé :

$$s = e = c + \Delta v$$

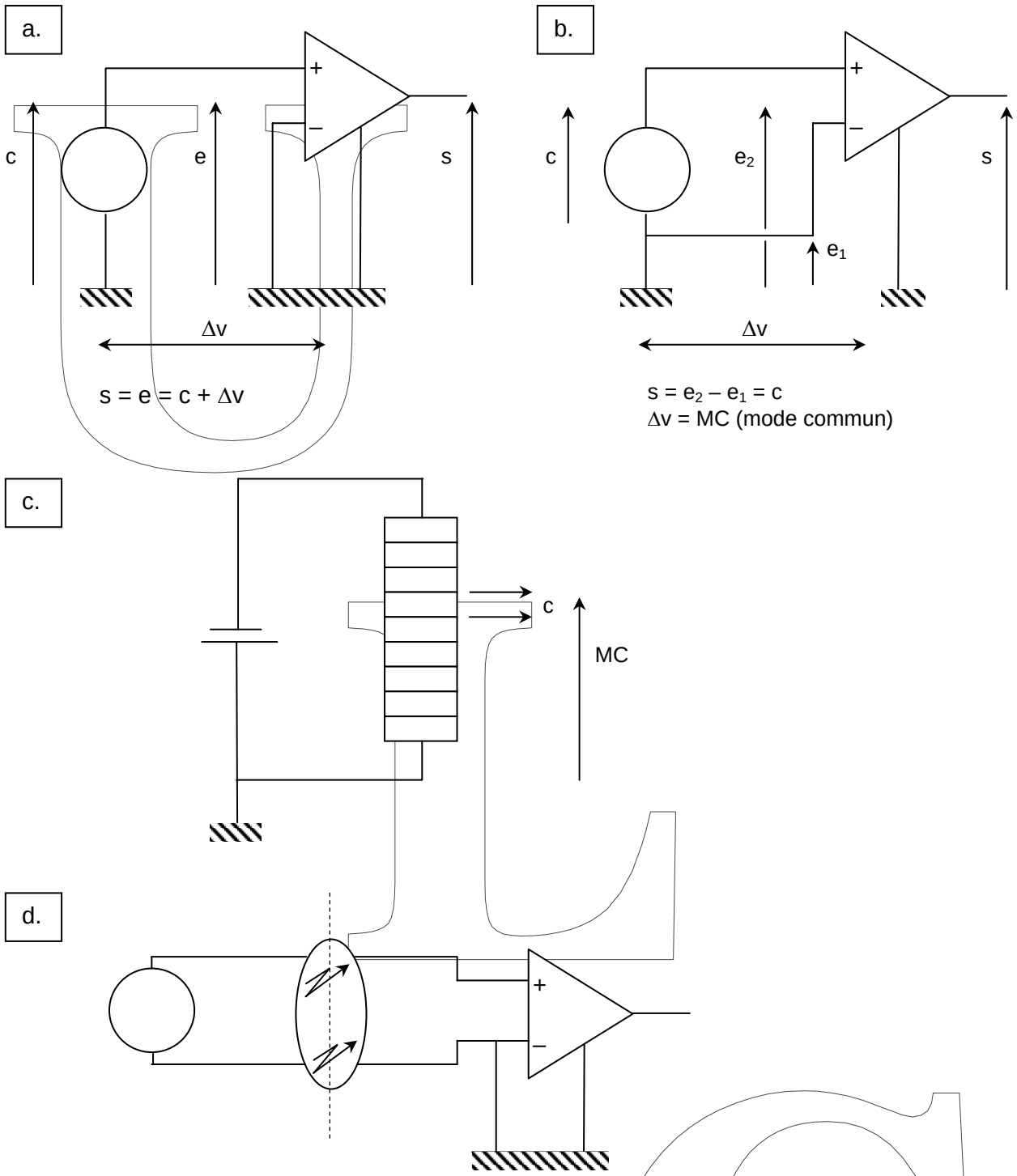


Figure 3.1. Tension de mode commun

a. Raccordement unifilaire
b. Raccordement différentiel
c. Exemple de capteur différentiel sur cellule d'électrolyse
d. Principe de l'isolation galvanique

Pour éviter ce risque, on travaille généralement en différentiel comme montré à la figure 3.1.b.

Lorsque la tension est nulle aux bornes du capteur, les entrées positive et négative de l'amplificateur vont donc se trouver à une tension v par rapport à la référence de cet amplificateur. Cette tension commune aux deux entrées est précisément ce que l'on appelle le **mode commun**.

Les tensions de mode commun peuvent avoir différentes causes :

- existence de courants de circulation dans le sol qui, en fonction de la distance, peuvent donner lieu à des chutes ohmiques non négligeables, quelques volts typiquement. Ces courants de circulation résultent généralement de l'emploi, volontaire ou involontaire, de la terre comme circuit de retour.

- mode commun induit par des parasites électromagnétiques de forte intensité, par exemple, ceux provenant de contacteurs, thyristors, postes de soudure, etc.
  Les tensions de mode commun induites peuvent atteindre quelques centaines de volts. Dans cette même catégorie, on peut ranger les mises sous tension accidentelles des capteurs ou des conducteurs.

- capteurs différentiels : c'est un cas où, par sa disposition même, le capteur se trouve à un certain potentiel par rapport à la terre. La figure 3.1.c présente l'exemple typique de la mesure de tension aux bornes d'une cellule d'électrolyse.

Problèmes

Les tensions de mode commun sont sources potentielles de deux problèmes : erreur de mesure et claquage des circuits.

- erreur de mesure : le travail en différentiel élimine bien entendu la propagation directe du mode commun à la sortie de l'amplificateur. L'impédance des entrées vis-à-vis de la masse de l'amplificateur, si grande soit-elle, n'est cependant jamais infinie. Un courant de fuite existera donc toujours et, par suite, une chute ohmique proportionnelle au mode commun. La valeur de cet effet résiduel est une caractéristique importante des amplificateurs, elle s'exprime en terme de taux de réjection du mode commun. Une valeur typique sera, par exemple, 120 dB de 0 à 60 Hz. [Source : Computer Products, inc.]

- claquage de circuits : pour des raisons purement électroniques, les amplificateurs sont détruits lorsque (signal + mode commun) dépassent des valeurs de l'ordre de 15 V.

Solutions

S'il n'y a aucun risque de claquage, la solution consiste simplement à choisir un amplificateur ayant un taux de réjection adéquat.

Si des risques de claquage existent, il est indispensable de procéder à une isolation galvanique entre le capteur et l'amplificateur. Ceci implique l'interposition, dans la chaîne, d'un couplage non-électrique. Nous verrons ci-après des exemples de couplages optiques, magnétiques ou par relais.

3.2. PRINCIPES D'ORGANISATION

L'accès à l'automate d'un signal de mesure en provenance du processus (ou inversement, l'envoi d'une commande vers le processus) met en jeu une cascade de trois éléments :

– le bornier qui assure un découplage mécanique entre le câblage venant du processus et l'automate;

– les circuits de conditionnement de signaux qui réalisent les adaptations, isolations, filtrages, conversions requis pour arriver finalement à des signaux compatibles avec l'électronique de l'automate;

– l'interface entre ces signaux et le BUS de l'automate.

Dans la pratique, la distinction n'est pas toujours aussi nette et l'on trouve, en gros, trois types d'organisations présentées à la figure 3.2.

3.2.1. SOLUTION INTEGREE

Dans ce premier cas (figure 3.2.a), les trois éléments définis ci-dessus sont regroupés sur une même carte de circuit imprimé, directement greffée sur le BUS de l'automate.

Il s'agit d'une solution que l'on rencontre plutôt dans les petits systèmes. Elle présente l'inconvénient d'amener les signaux bruts du processus au voisinage du BUS et de nécessiter un câblage point-à-point entre l'automate et les capteurs/actuateurs.

3.2.2. PERIPHERIES DEPORTEES

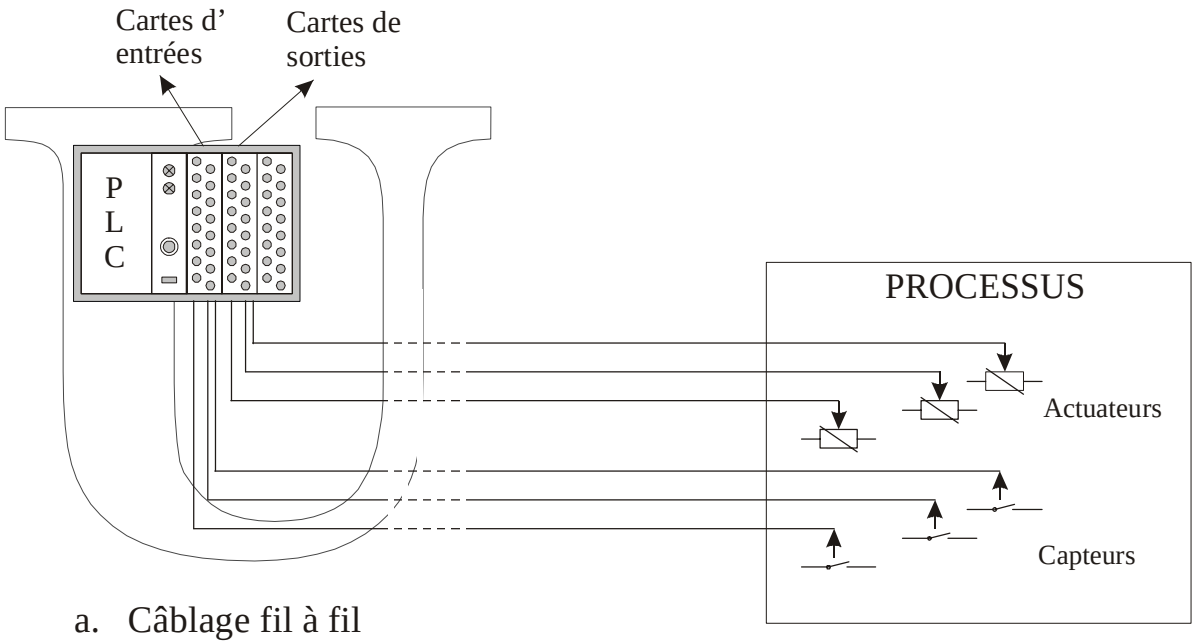
Dans cette deuxième optique (figure 3.2.b), l'utilisation de réseaux de terrain (voir tome 2) permet de décentraliser les cartes d'entrées/sorties dans des modules comportant leur propre BUS et leur propre alimentation.

On peut placer ces modules au voisinage des processus ou parties de processus contrôlés, ce qui conduit à une réduction substantielle du câblage.

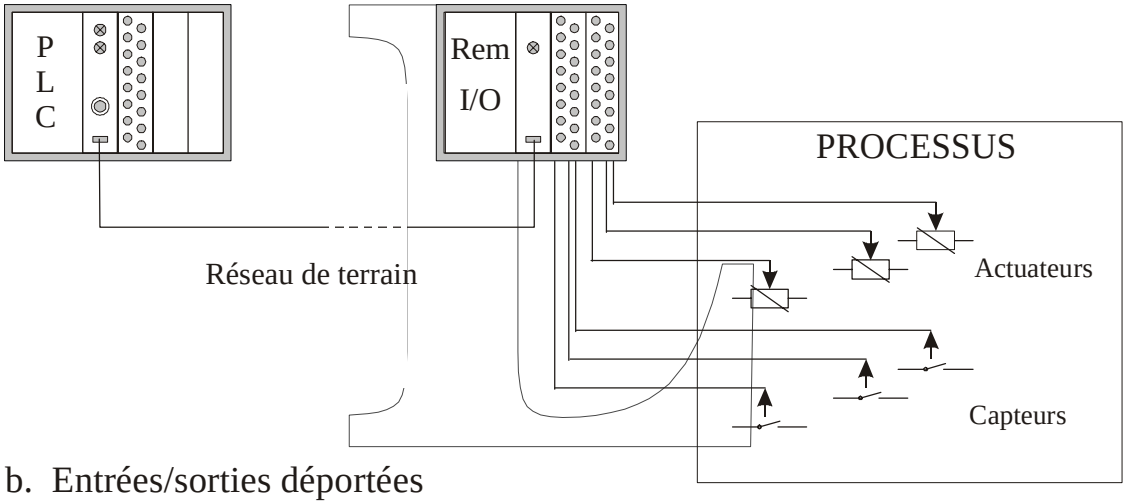
3.2.3. RESEAUX DE CAPTEURS ET ACTUATEURS

Dans cette troisième optique (figure 3.2.c), les capteurs et actuateurs sont directement greffés sur un réseau de capteurs/actuateurs relié à l'automate. Le conditionnement des signaux se fait au sein même des capteurs et actuateurs.

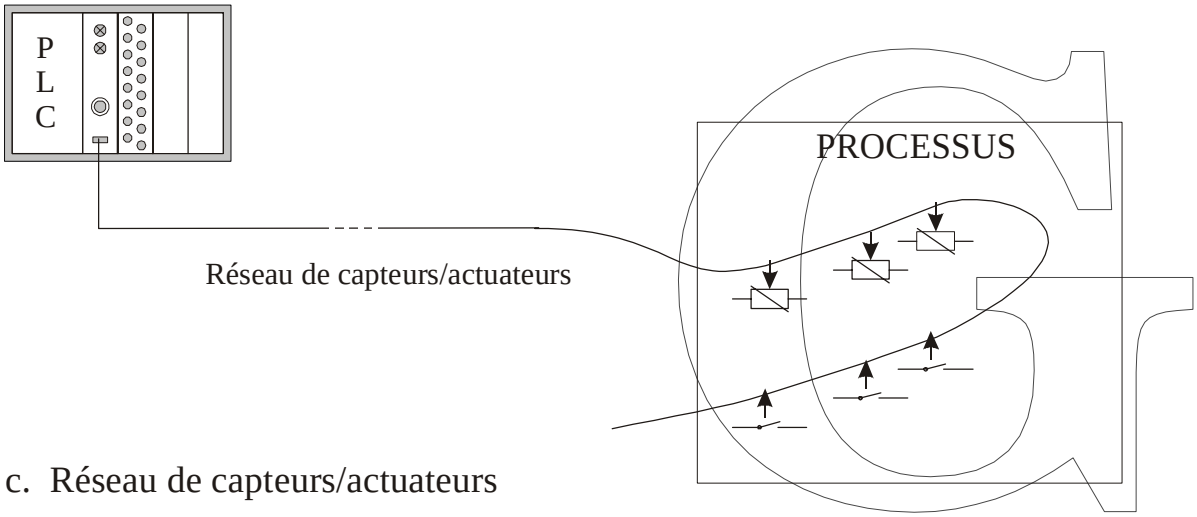
La réduction du câblage est évidemment maximale dans ce dernier cas. On ne peut cependant utiliser que des capteurs et actuateurs spécialement conçus pour le réseau considéré ce qui rend la solution moins universelle que les précédentes.



a. Câblage fil à fil



b. Entrées/sorties déportées



c. Réseau de capteurs/actuateurs

Figure 3.2. Organisation d'entrées/sorties

3.3. CONDITIONNEMENT DES ENTREES/SORTIES LOGIQUES

3.3.1. ENTREES LOGIQUES

La figure 3.3. montre les principaux éléments d'un conditionneur d'entrée logique typique :

- adaptation de niveau en fonction des standards cités au § 3.1.1;
- protection contre les surtensions;
- redressement éventuel dans le cas de signaux alternatifs;
- isolation galvanique entre processus et calculateur;
- filtrage destiné à éliminer les parasites hautes fréquences;
- seuils de discrimination des états logiques 1 et 0;
- visualisation de l'état, indispensable pour les tests et la maintenance.

La figure 3.3. donne aussi quelques exemples de solutions technologiques :

(a) - solution toute électrique avec isolation galvanique réalisée par opto-coupleur
(b) - isolation galvanique par relais
(c) - isolation galvanique par transformateur dans le cas de signaux alternatifs.

Remarque :

On rencontre encore des entrées logiques dites d'interruption. En fait leur conditionnement est analogue à celui décrit ci-dessus, la seule différence est qu'elles aboutissent aux lignes d'interruption du BUS plutôt qu'aux lignes de données.

3.3.2. SORTIES LOGIQUES

La figure 3.4. montre les principaux éléments d'un conditionneur de sortie logique typique :

- visualisation d'état
- isolation galvanique
- adaptation de niveau et de puissance en fonction des standards définis au paragraphe 3.1.1. En général, cette adaptation se fait à partir d'une alimentation extérieure.
- protection contre les surcharges (courts-circuits), les surtensions (charges inductives); absorption de l'énergie de coupure des relais, etc.

La figure 3.4. donne aussi quelques exemples de solutions technologiques:

(a) - sortie à relais (contact libre de potentiel) assurant à la fois l'isolation et l'adaptation de puissance
(b) - sortie à transistor avec isolation par opto-coupleur
(c) - sortie à triac pour les signaux alternatifs.

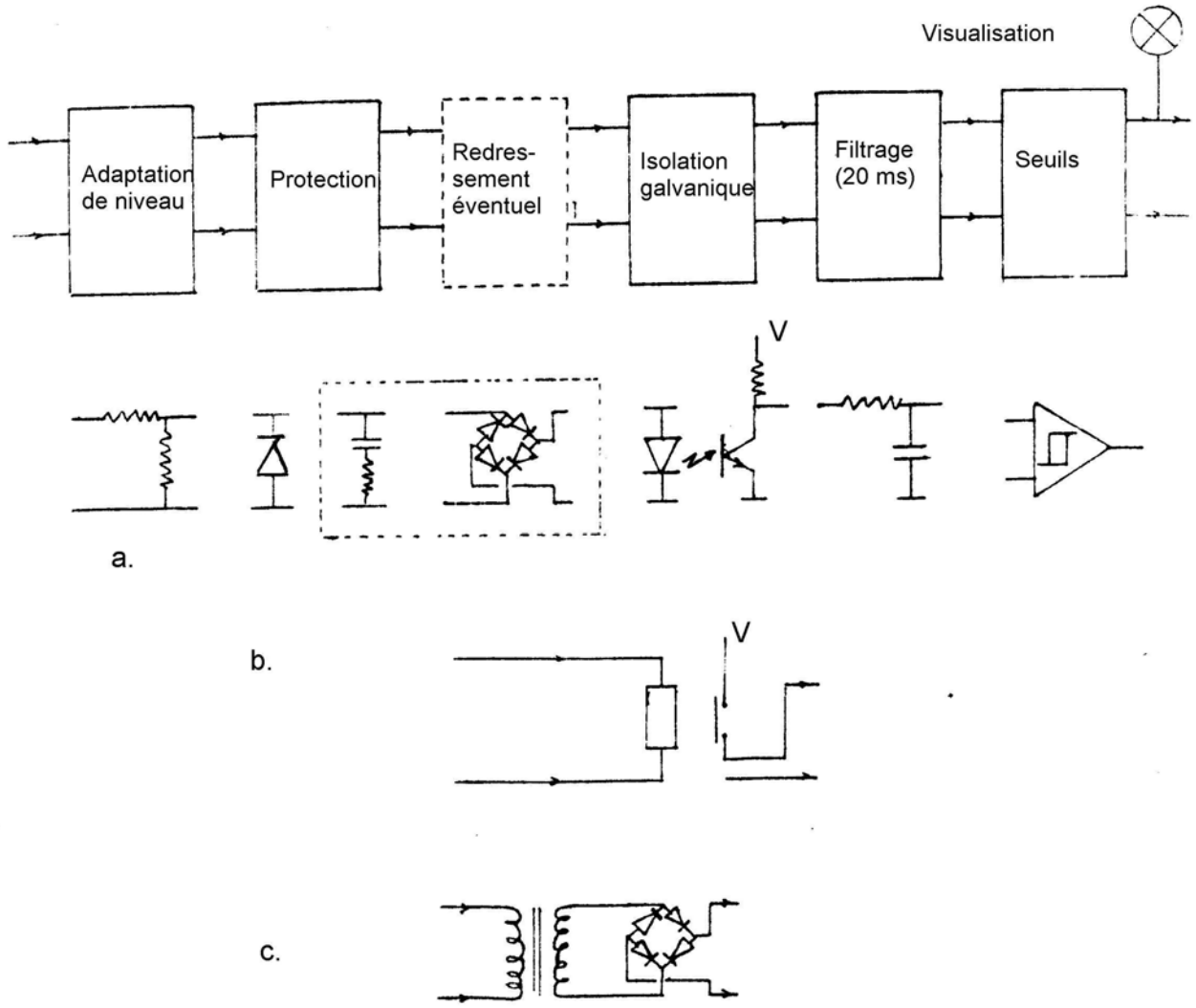


Figure 3.3. Eléments caractéristiques d'un conditionneur d'entrées logiques

a. solution toute électrique
b. isolation galvanique par relais
c. isolation galvanique par transformateur

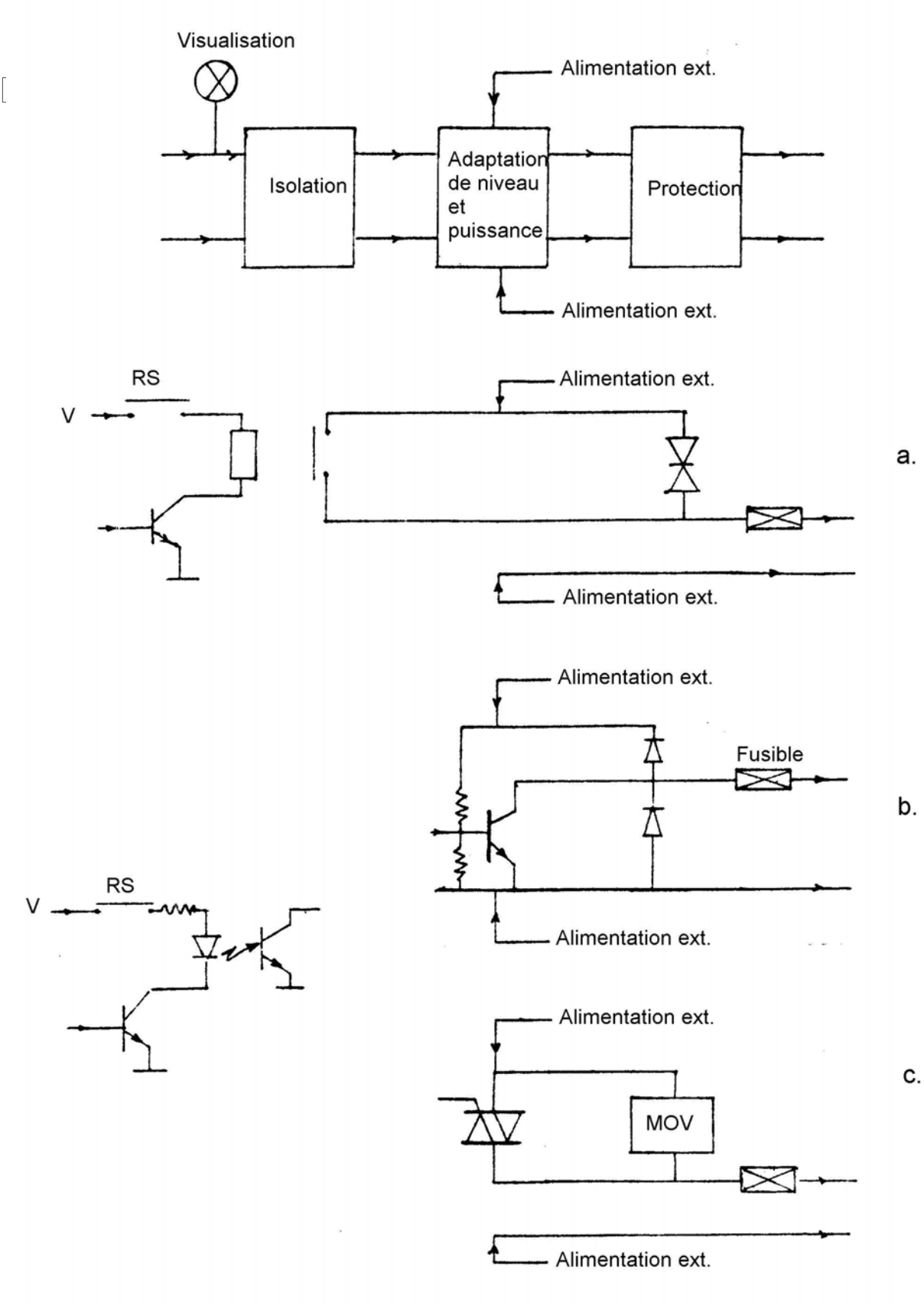


Figure 3.4. Conditionnement d'une sortie logique
a. Sortie à relais
b. Sortie à transistor
c. Sortie à triac

3.3.3. MESURES DE SECURITE

Dans le cas des sorties à relais, on peut imaginer une procédure de sauvegarde très simple qui consiste, en cas d'anomalie, à couper l'alimentation des relais par un relais de sécurité (RS sur la fig. 3.4). Les commandes logiques qui en résultent pour le processus dépendent alors uniquement du type de relais qui a été choisi : normalement ouvert, normalement fermé ou bistable (si l'on souhaite maintenir l'état précédent).

Lors d'un retour à la normale (de même d'ailleurs que lors d'une mise sous tension du système), il y a lieu de ne refermer le relais RS qu'après avoir procédé à une initialisation cohérente des circuits d'interface.

On a donc ainsi le moyen de définir, avec une certaine souplesse, une position de repli statique pour les sorties logiques. Si cela n'est pas suffisant (par exemple, si l'on désire pouvoir effectuer une commande manuelle de processus), il faudra alors élaborer des solutions spécifiques à chaque cas particulier.

Pour les autres types de sorties logiques, on peut aussi envisager de piloter l'alimentation de l'opto-coupleur de sortie par un relais de sécurité. La position de repli se limite cependant ici à une retombée générale des commandes à zéro.

3.4. CONDITIONNEMENT DES ENTREES/SORTIES ANALOGIQUES

Avant de pouvoir être manipulés par un système informatique, les signaux analogiques doivent être convertis en signaux digitaux, c'est-à-dire transformés en nombres contenus dans des registres. L'opération inverse étant évidemment nécessaire pour les sorties analogiques.

Dans ce paragraphe, nous rappellerons d'abord brièvement le principe des convertisseurs, avant de passer au conditionnement proprement dit des entrées/sorties analogiques.

3.4.1. PRINCIPE DES CONVERTISSEURS

Convertisseurs numériques/analogiques

La figure 3.5. montre le principe d'un convertisseur numérique/analogique.

La sortie analogique s'obtient très directement par sommation pondérée de courants dans un amplificateur opérationnel. Ces courants pondérés sont élaborés dans un réseau de résistances (intégrées) de précision. Leur sélection se fait par une batterie d'inverseurs statiques sous le contrôle direct des bits du mot à convertir.

Pour obtenir une sortie bipolaire, on déplace le zéro de l'amplificateur d'une quantité équivalente au bit de plus fort poids (1/2 de la gamme) mais de signe opposé. Notons que l'on s'écarte ainsi de la représentation des nombres par la méthode du complément à 2.

Caractéristiques typiques : résolution 8 à 12 bits; temps de conversion de l'ordre de 1 µs.

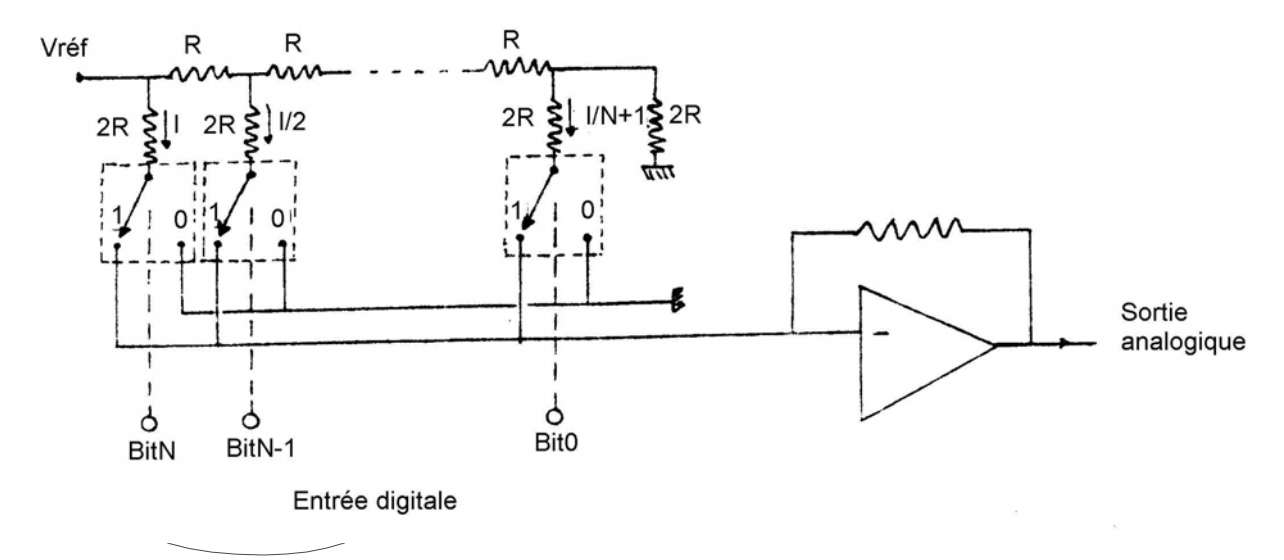


Figure 3.5. Schéma fonctionnel d'un convertisseur numérique analogique

Convertisseurs analogiques/numériques à approximations successives

Le principe de ce type de convertisseurs est montré à la figure 3.6.

La tension à convertir est comparée à une tension fournie par un générateur de nombres suivi d'un convertisseur numérique/analogique tel que précédemment décrit. Le générateur de nombres enclenche successivement les bits du mot, en commençant par celui de plus fort poids, et ne retient que ceux qui ne provoquent pas de basculement du comparateur. La conversion est terminée lorsque le résultat de la comparaison est inférieur à la moitié du bit de plus faible poids.

La figure 3.6.b. montre l'exemple d'une séquence d'approximations dans le cas d'un convertisseur hypothétique de 4 bits.

Caractéristiques typiques : résolution de 8 à 12 bits; temps de conversion quelques dizaines de µs.

Convertisseurs analogiques/numériques à double pente

Dans ce type de convertisseur, la tension à convertir est intégrée pendant un temps déterminé T_i (cfr. la figure 3.7). La sortie de l'intégrateur est alors ramenée à 0 par intégration d'une tension de référence de sens opposé. Le temps nécessaire pour ce retour à 0 (T_c) est une mesure de la tension d'entrée. Il est comptabilisé dans un compteur dont l'état final peut donc être pris comme résultat de la conversion.

Cette manière de faire permet d'atteindre des précisions élevées (au détriment évidemment du temps de conversion). De plus, l'intégration opérée sur l'entrée constitue un filtrage très efficace. En particulier, si l'on adopte des temps d'intégration proportionnels à la période du réseau (20 ms ou des multiples), on obtient une excellente réjection de ce dernier.

Caractéristiques typiques : résolution 8 à 14 bits; temps de conversion quelques dizaines à quelques centaines de ms.

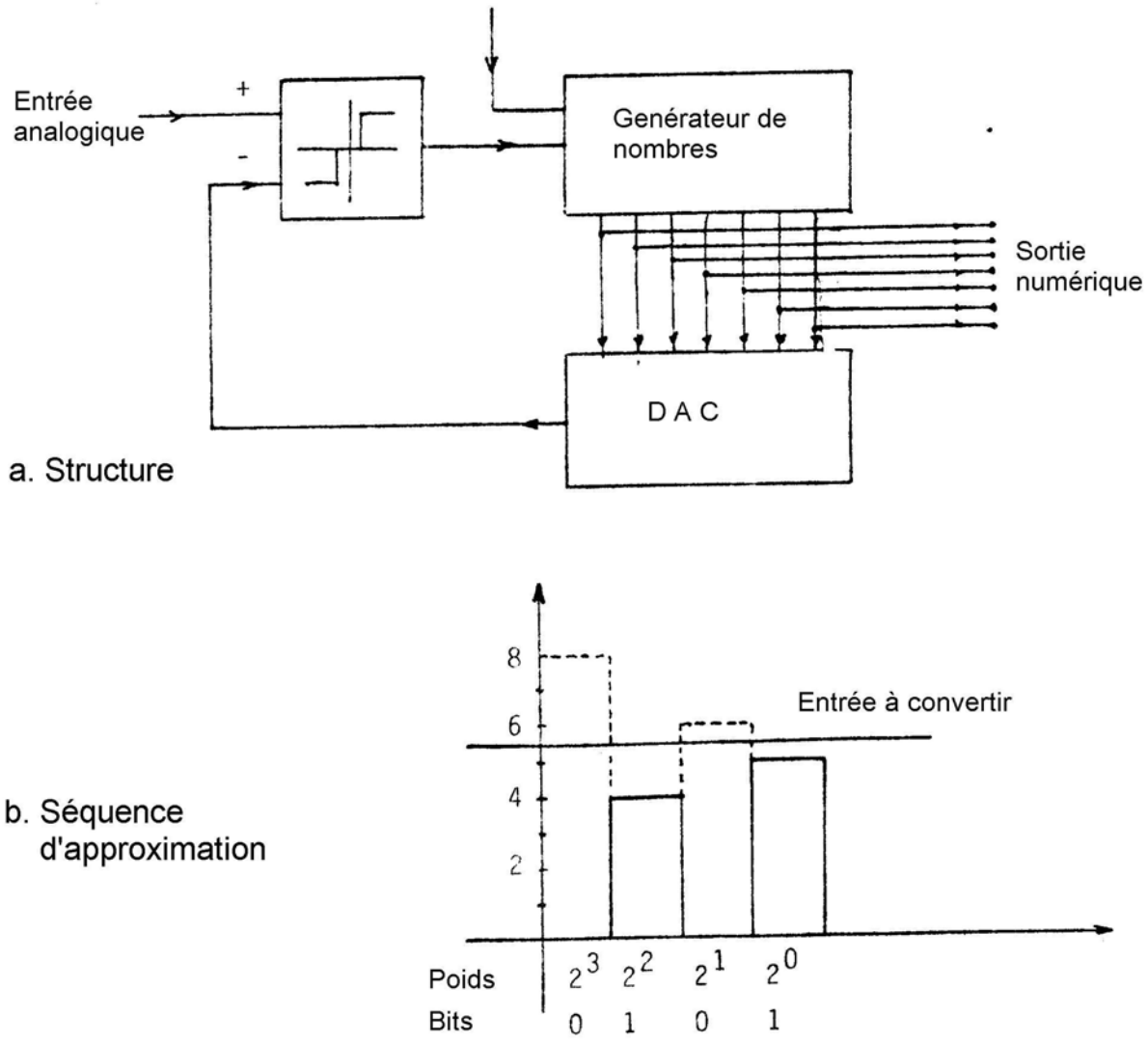


Figure 3.6. Schéma fonctionnel d'un convertisseur analogique/numérique à approximations successives

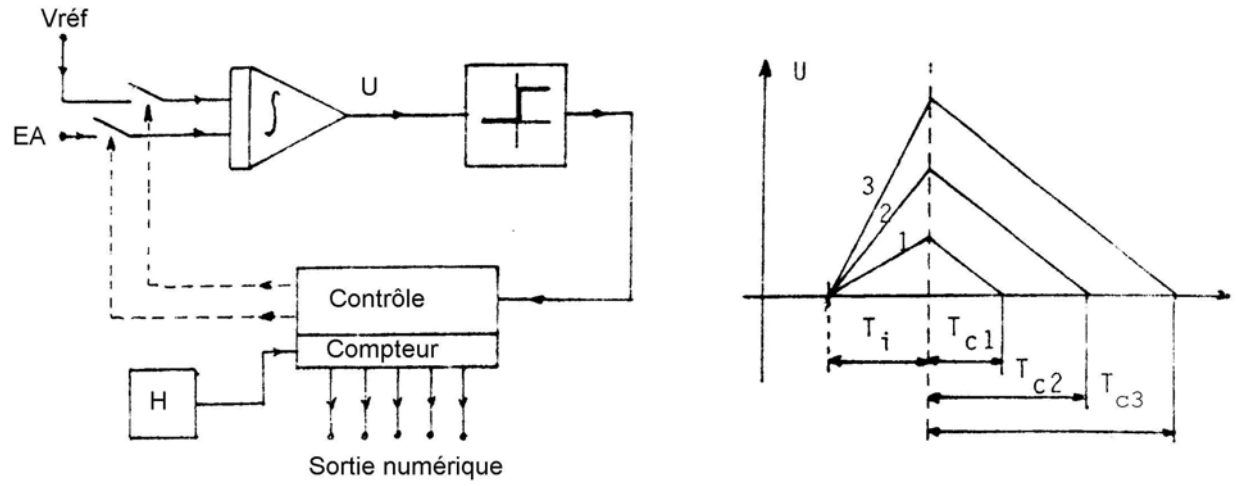


Figure 3.7. Principe du convertisseur analogique/numérique à double pente

Convertisseurs analogiques/numériques à quadruple pente

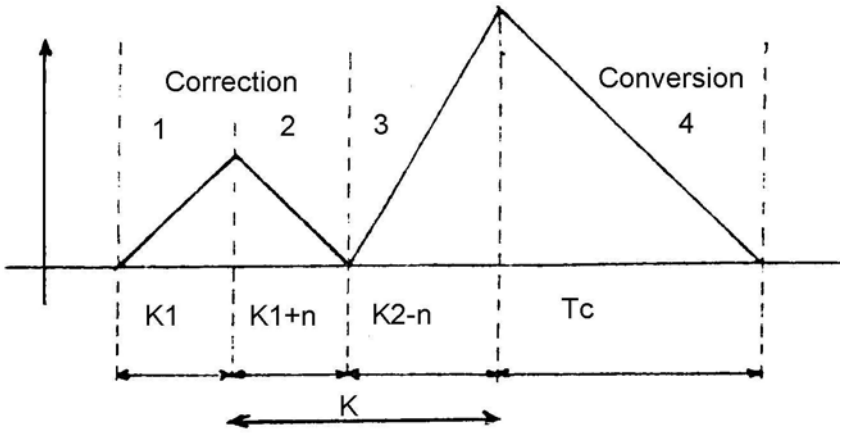
Il s'agit ici d'un perfectionnement du système précédent destiné à compenser les dérives de zéro et de gain (dues à des différences de T°, au vieillissement, etc.).

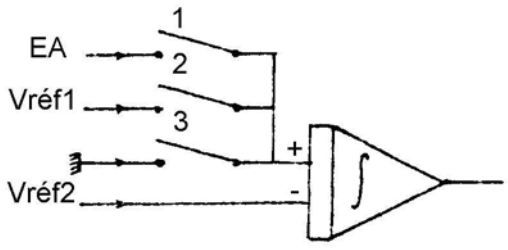
On procède en fait à deux conversions successives par la méthode de la double pente: la première avec une entrée nulle, la seconde avec la tension à convertir effectivement. Le résultat du premier cycle (qui est l'erreur de dérive) est soustrait du résultat du deuxième cycle. On arrive ainsi à des dérives résiduelles de 1 pp m/°C soit cinq fois meilleures que dans la technique à double pente.

La figure 3.8. illustre cette manière de faire.

En pratique, on introduit une deuxième tension de référence V réf.2 de manière à avoir, même pendant le cycle de correction, des variations de la sortie de l'intégrateur largement supérieures aux seuils de commutation des circuits.

La correction quant à elle s'introduit très directement au niveau des temps d'intégration en maintenant K constant.





Vréf1 = 2 x Vréf2

Phase	Interrupteur fermé	Tension intégrée	Durée
1	3	0 – Vréf2	K1
2	2	Vréf1 – Vréf2	K1 + n
3	1	EA – Vréf2	K2 – n
4	2	Vréf1 – Vréf2	Tc

Figure 3.8. Principe du convertisseur analogique/numérique à quadruple pente

3.4.2. ENTREES ANALOGIQUES

Les convertisseurs analogiques/numériques restent des éléments relativement coûteux et encombrants. On ne peut en général envisager d'en affecter un à chaque entrée analogique, ce qui impose donc d'avoir recours à des multiplexeurs.

Le conditionnement des entrées analogiques se présente dès lors comme schématisé à la figure 3.9.

On y distingue, outre la conversion déjà décrite, les fonctions suivantes:

Adaptation d'entrée

Il s'agit à ce niveau d'uniformiser les signaux autant que faire se peut avant d'aborder la partie commune du conditionnement. Par exemple, atténuer les entrées haut niveau hors standards, amplifier les entrées bas niveau, effectuer la transformation courant/tension, compléter un pont de Wheatstone pour les capteurs à variations de résistance, ...

Isolation galvanique

Le problème est de nouveau ici 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.

Filtrage

Elimination des parasites industriels de fréquence supérieure à celles du signal utile.

La figure 3.9.b. montre un exemple de solution simultanée aux problèmes d'adaptation, isolation, filtrage. Elle est basée sur l'utilisation d'un amplificateur d'isolation travaillant par modulation et démodulation de phase d'une porteuse haute fréquence (50 K Hz). L'isolation y est obtenue par transformateur (1000 V ou plus).

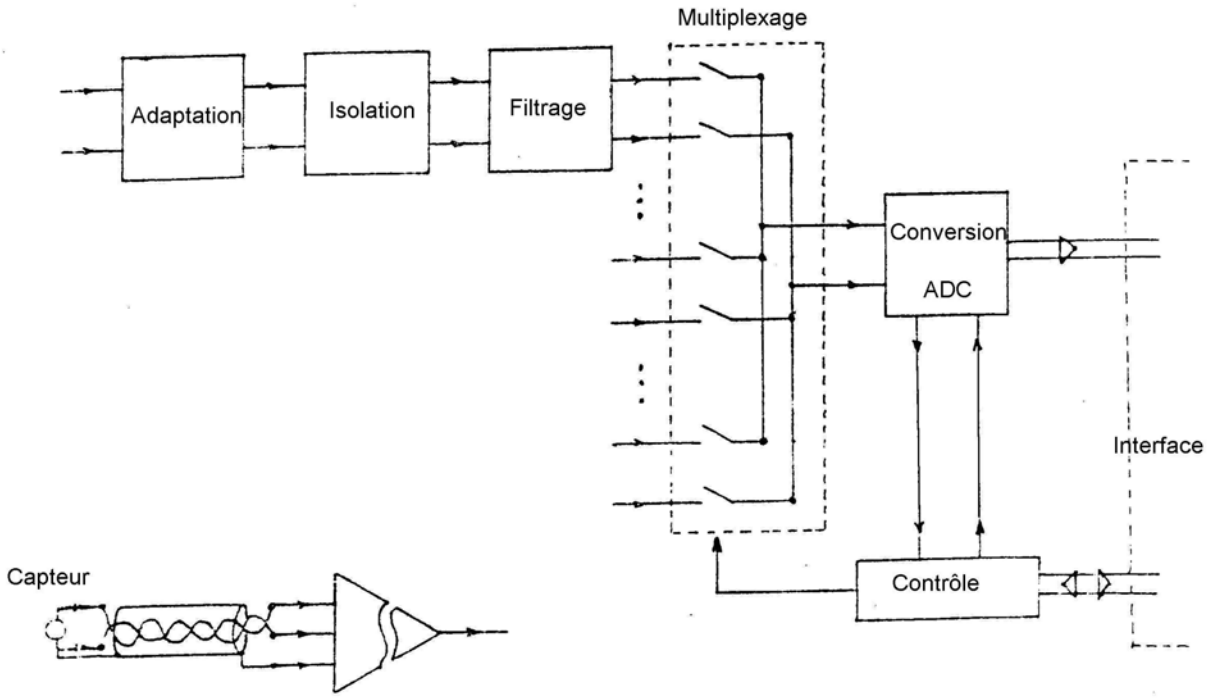
Ajoutons encore que, comme on peut s'en douter, les amplificateurs d'isolation sont des éléments relativement coûteux.

Multiplexage

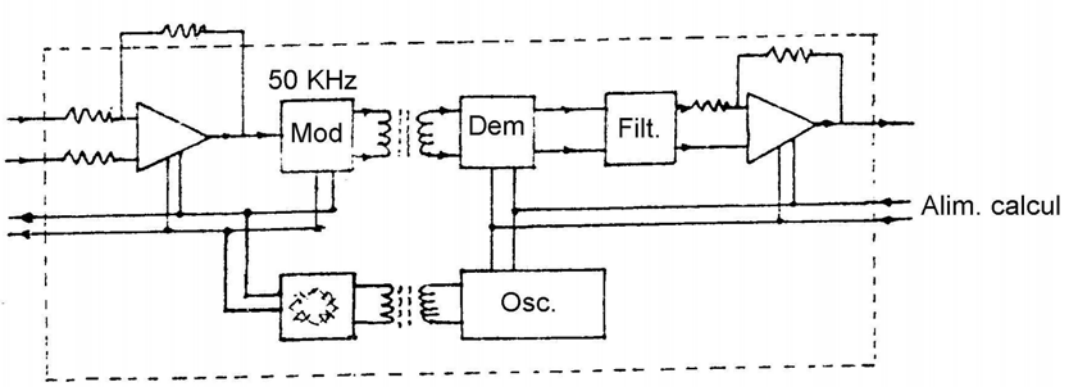
On utilise essentiellement deux types de multiplexeurs :

– ***multiplexeurs statiques*** (transistors à effet de champ)

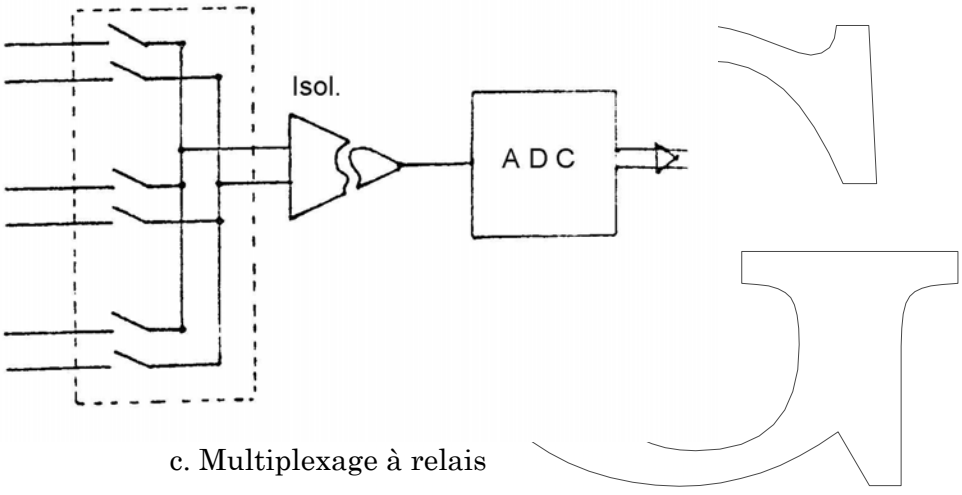
Ils sont très rapides (temps de commutation de l'ordre de la µs) mais ne supportent pas de tensions de mode commun élevées (10 V max.). De plus, les tensions résiduelles des transistors à effet de champ empêchent de les utiliser directement pour des entrées de bas niveau. En général, les multiplexeurs statiques devront donc être précédés d'isolateurs et d'amplificateurs (cfr. figure 3.9.b.).



a./Principe

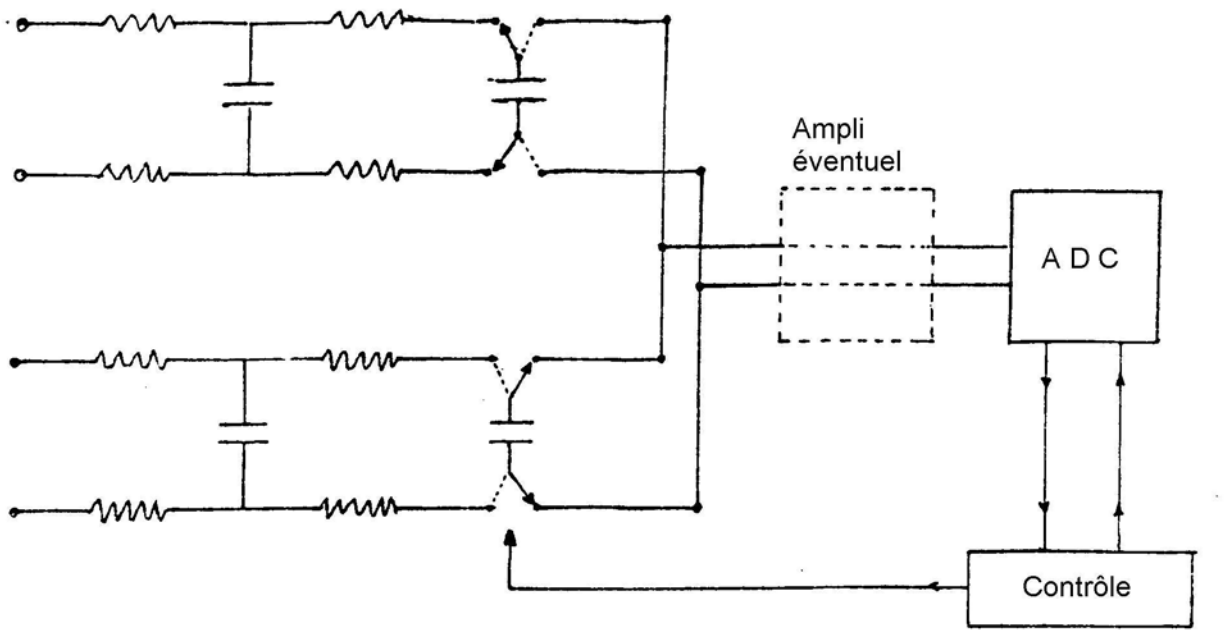


b. Utilisation d'un ampli d'isolation

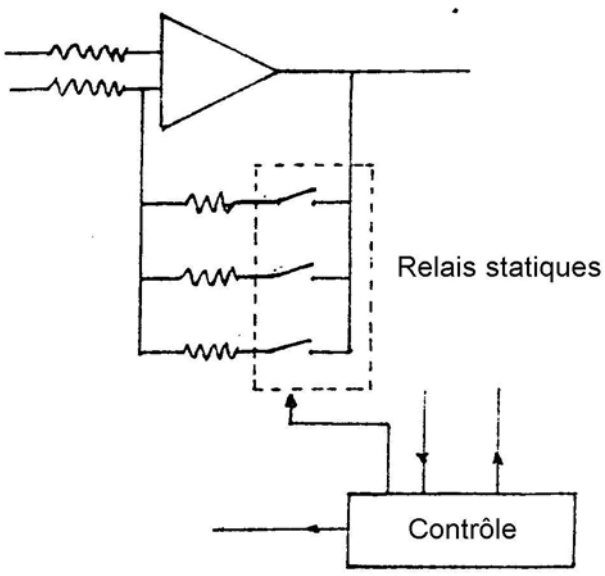


c. Multiplexage à relais

Figure 3.9. Conditionnement des entrées analogiques



d. Principe de la capacité basculante



e. Amplificateur à gain programmable

Figure 3.9. Conditionnement des entrées analogiques (suite)

– ***multiplexeurs à relais électromagnétiques***

Ils sont évidemment plus lents (temps de commutation de quelques dizaines de ms) mais assurent une isolation galvanique parfaite des voies d'entrées entre elles.

Pour isoler le calculateur lui-même, on peut par exemple placer un amplificateur d'isolation (unique) juste après le multiplexeur (figure 3.9.c.).

Une autre solution, très souvent utilisée en pratique, est basée sur le principe de la capacité basculante. Ce principe est illustré à la figure 3.9.d. : dans une position des contacts, la capacité se charge à la valeur de la tension d'entrée à convertir; dans l'autre position, la capacité est isolée de l'entrée et connectée au convertisseur.

Comme le montre aussi la figure 3.9.d., le système à capacité basculante peut être agencé de manière à réaliser à la fois l'isolation, le filtrage et le multiplexage. Pour éviter dans ce cas d'éventuels amplificateurs d'adaptation, on préférera souvent faire précéder le convertisseur d'un amplificateur unique à gain programmable (figure 3.9.e.).

3.4.3. SORTIES ANALOGIQUES

Etant donné le coût peu élevé et le faible encombrement des convertisseurs numériques/analogiques, on en utilise en général un par sortie. Hormis cela, les sorties analogiques posent en principe les mêmes problèmes d'isolation et d'adaptation que les entrées analogiques.

Il existe cependant assez peu de solutions standards au niveau des interfaces proposées par les constructeurs d'ordinateurs dans la mesure où les problèmes en question sont souvent pris en compte dans le contexte plus général de la sécurité des commandes analogiques.

3.4.4. MESURES DE SECURITE

Le problème de la sécurité des commandes analogiques en cas de panne du calculateur se pose de manière beaucoup plus aiguë que dans le cas des commandes logiques. En effet, il est en général difficile de définir ici un état de sécurité statique: une possibilité de commande manuelle du processus est presque toujours exigée et, souvent même, une régulation ne fut-ce qu'élémentaire doit être maintenue.

Il en résulte que les sorties analogiques émises par un automate ne seront envoyées au processus que par l'intermédiaire de stations d'interface comprenant le système de commutation nécessaire et une mémoire analogique tampon.

La figure 3.10. montre le principe d'une telle station.

Notons que la commande manuelle impose un minimum de dispositifs de commande, bien sûr, mais aussi de visualisation de l'état du processus (au moins de la mesure). Ceux-ci sont souvent inclus dans la station elle-même.

D'autre part, pour que la commutation des modes de travail puisse s'effectuer sans choc, il est nécessaire de prévoir une initialisation correcte des systèmes en "stand by".

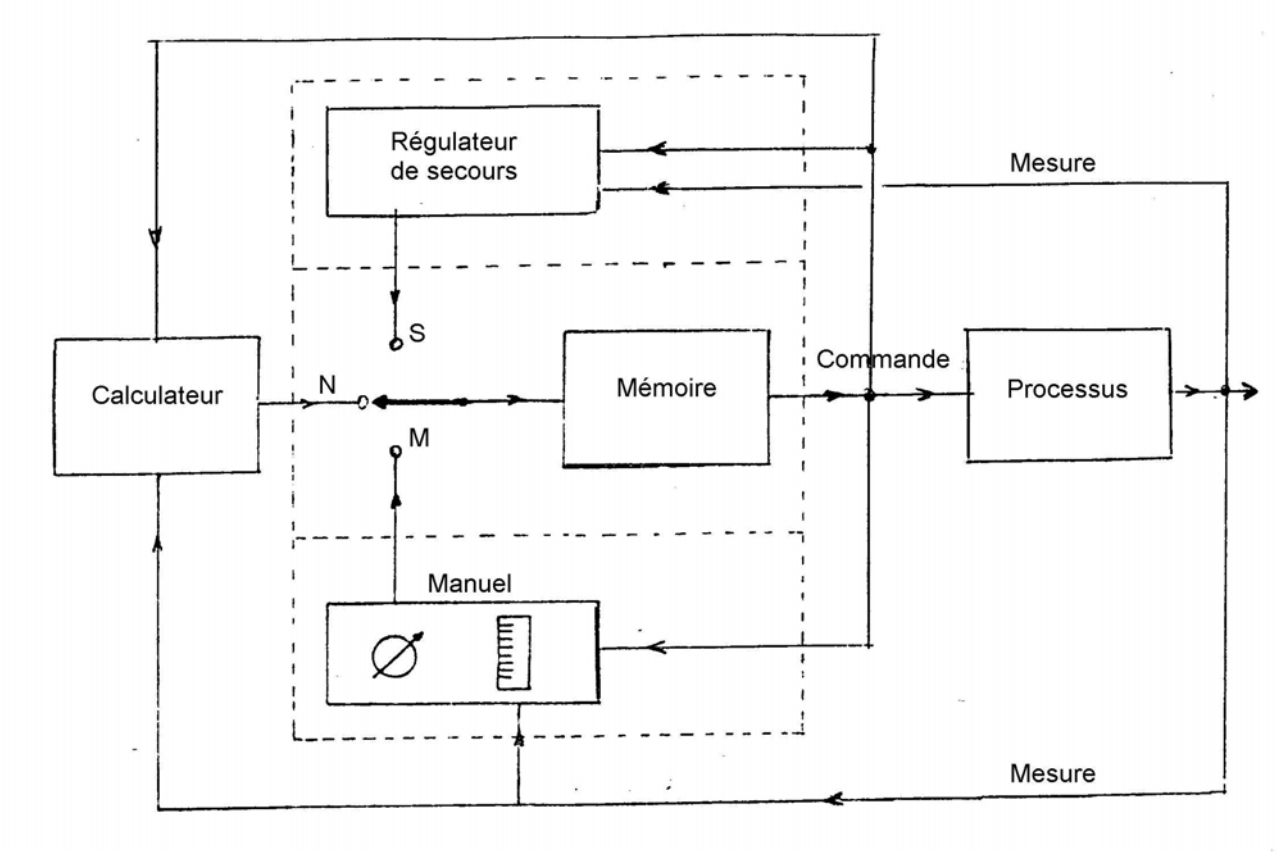


Figure 3.10. Principe d'une station d'interface

A cet égard, l'utilisation d'une mémoire de type intégrateur (analogique ou moteur pas-à-pas) facilite grandement les choses. En effet, il suffit, dans ce cas, de commuter à tension de sortie nulle.

3.5. DISPOSITIFS DE SECURITE

Toutes les mesures de sécurité envisagées aux paragraphes précédents n'ont de sens que s'il existe par ailleurs des moyens de détecter les pannes ou anomalies de fonctionnement d'un automate.

Des solutions plus ou moins sophistiquées existent dans ce domaine; nous nous bornerons ici à examiner deux procédures élémentaires mais indispensables dans toutes applications industrielles : la surveillance de cycle et la surveillance d'alimentation.

3.5.1. SURVEILLANCE DE CYCLE ("WATCH-DOG")

Dans cette procédure, on oblige l'automate à envoyer périodiquement une impulsion au système de surveillance. Celui-ci vérifie simplement que le temps entre deux impulsions ne dépasse pas une limite fixée. Cette limite doit évidemment être choisie petite vis-à-vis de la dynamique du processus (par exemple : égale à la période d'échantillonnage des algorithmes de régulation).

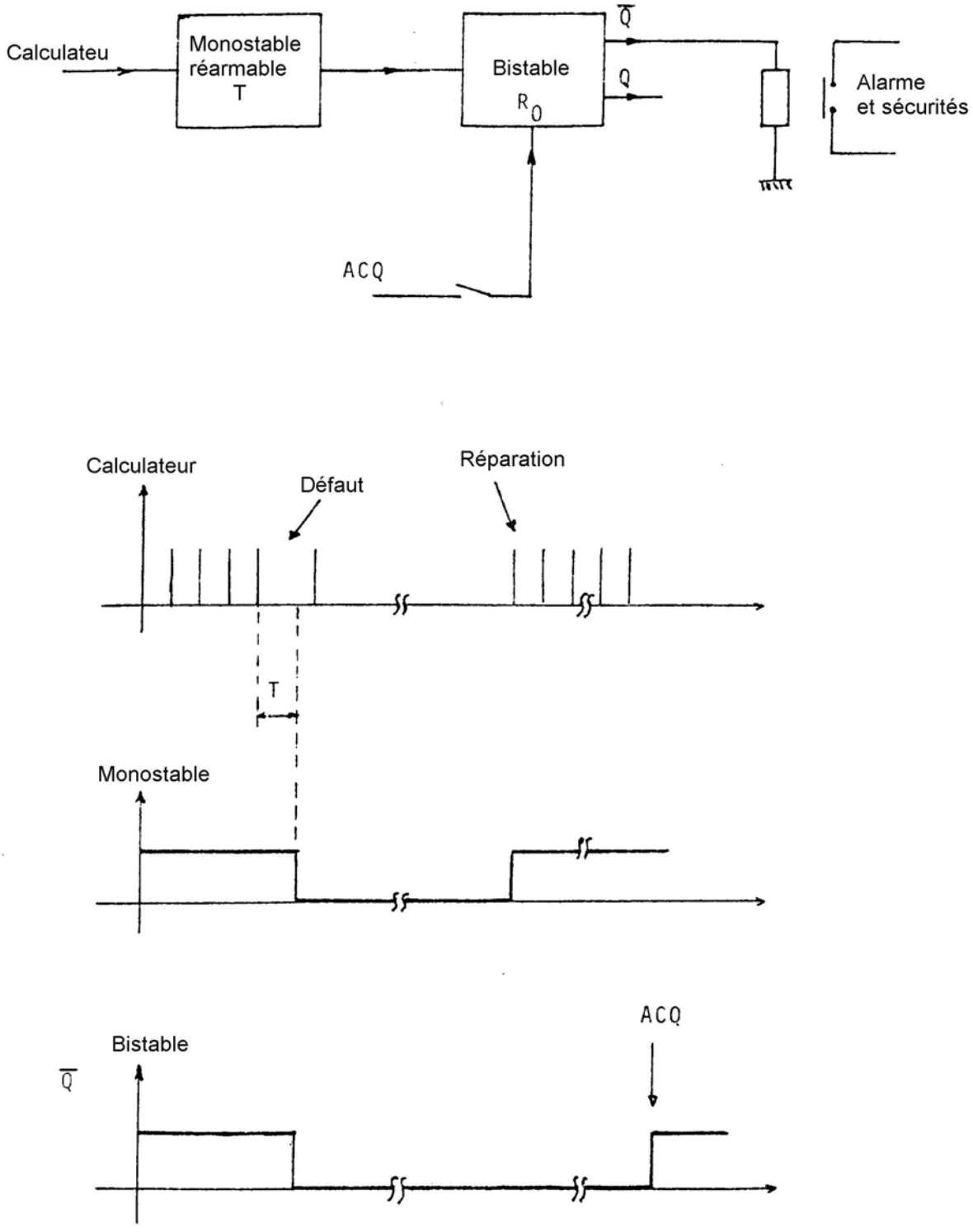


Figure 3.11. Principe d'un watch-dog élémentaire

Dans le cas où la limite est dépassée, le système de surveillance émet un signal d'alarme (en général ouverture d'un relais) qui peut être utilisé, par exemple, pour provoquer les commutations prévues au paragraphe 3.3.3.

La figure 3.11. montre un exemple de réalisation.

Il utilise un monostable réarmable dont la période correspond à la limite de temps imposée. Ce monostable est suivi d'un bistable qui mémorise le déclenchement du premier et impose un "acquittement" manuel du défaut.

3.5.2. SURVEILLANCE D'ALIMENTATION

Le but du système de surveillance d'alimentation est d'assurer un comportement bien déterminé du calculateur en cas de coupure (ou microcoupure) du réseau et de remise sous tension.

Coupure de tension ("power fail")

Le comportement d'un système informatique devient aléatoire dès que l'une de ses alimentations stabilisées s'écarte de quelques % de sa valeur nominale. Heureusement ces alimentations comportent des tampons d'énergie importants, grâce au filtrage, et cela peut être mis à profit pour prendre un minimum de mesures de sauvegarde.

La figure 3.12. montre comment les choses se passent en pratique.

Le système de surveillance teste en permanence la tension du réseau (donc avant redressement et stabilisation). Dès qu'une baisse est constatée (10 % par exemple), le signal PWR VALID est mis à zéro provoquant une interruption de priorité maximum. Le processeur dispose alors d'un temps T1 (quelques ms, dépendant du tampon d'énergie) pendant lequel il peut effectuer un certain nombre de sauvetages (registres internes, registres tampons des interfaces, etc.) pour autant bien entendu, qu'il existe une mémoire RAM non volatile. Cela étant fait, le processeur s'arrête volontairement, évitant ainsi tout comportement aberrant au moment de la chute des alimentations régulées.

Remise sous tension ("power restart")

Le système de surveillance d'alimentation détecte le rétablissement des alimentations continues et, après un délai de sécurité T2 (cfr. figure 3.12.), le signal PWR VALID est réactivé, provoquant une nouvelle interruption. La sous-routine d'interruption correspondante restaure le contexte du processeur au moment de la coupure d'alimentation et reprend les traitements interrompus.

Bien entendu, tout autre scénario peut être envisagé sur la base des deux interruptions engendrées par le système de surveillance d'alimentation.

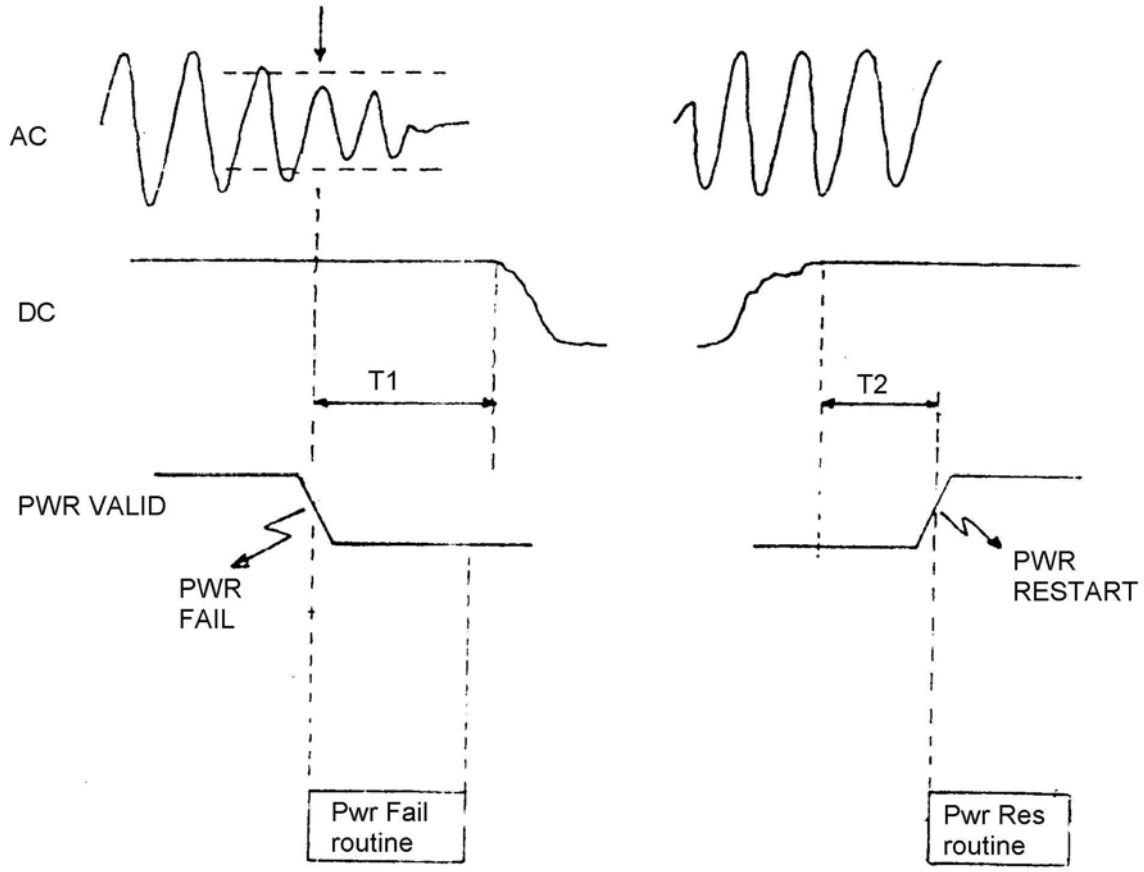


Figure 3.12. Exemple de séquences de coupure et de rétablissement du réseau

Chapitre 4

LANGAGES DE PROGRAMMATION

4.1. INTRODUCTION

La mise en oeuvre d'un automate programmable comporte trois phases principales :

– la conception du système de contrôle, qui se fait à l'aide d'outils méthodologiques et de modes de représentation propres à l'automaticien;

– la programmation de l'automate, qui consiste à transposer le système de contrôle obtenu dans le langage de programmation propre à l'automate;

– l'exécution du programme, enfin, prise en charge par un logiciel interne à l'automate, l'exécutif, qui, comme nous le verrons ci-après, peut être plus ou moins élaboré.

Comme nous l'avons dit au chapitre 2, tout automate peut résoudre n'importe quel problème de logique combinatoire ou séquentiel. Il est bien certain cependant que la facilité d'utilisation d'un automate et ses performances seront directement fonction de la cohérence qui existera entre les trois phases mentionnées ci-dessus.

Dans le présent chapitre, nous essayerons de faire le point de la situation en insistant sur la programmation séquentielle qui a donné lieu à de nombreuses initiatives de la part des constructeurs, principalement des constructeurs européens.

4.2. PROBLEMES DE NATURE COMBINATOIRE

Il s'agit de problèmes où tous les signaux provenant du processus (ou en tout cas une majorité d'entre eux) sont susceptibles de provoquer, à n'importe quel moment, une réaction de l'automate. Le cas typique étant celui d'une surveillance de conditions d'alarme (dépassement de niveau, surchauffe de moteurs, ...).

4.2.1. MODE DE REPRESENTATION

Le formalisme à relais, à logigrammes ou à équation booléennes, convient tout à fait bien à la représentation de ce genre d'automatisme. Il suppose en effet que l'on a affaire à des modules fonctionnels travaillant en parallèle et de manière instantanée (cfr. figure 4.1.)

4.2.2. PROGRAMMATION

Les langages de programmation à relais, logigrammes, ou équations booléennes sont évidemment les prolongements naturels des modes de représentation qui précèdent. Programmer dans ces langages consiste pratiquement à reproduire sur la console de programmation les schémas ou les équations établis sur concepteur (cfr. fig. 3.2.). C'est, dans ce cas, la console qui se charge d'établir la séquence d'instructions machines correspondante.

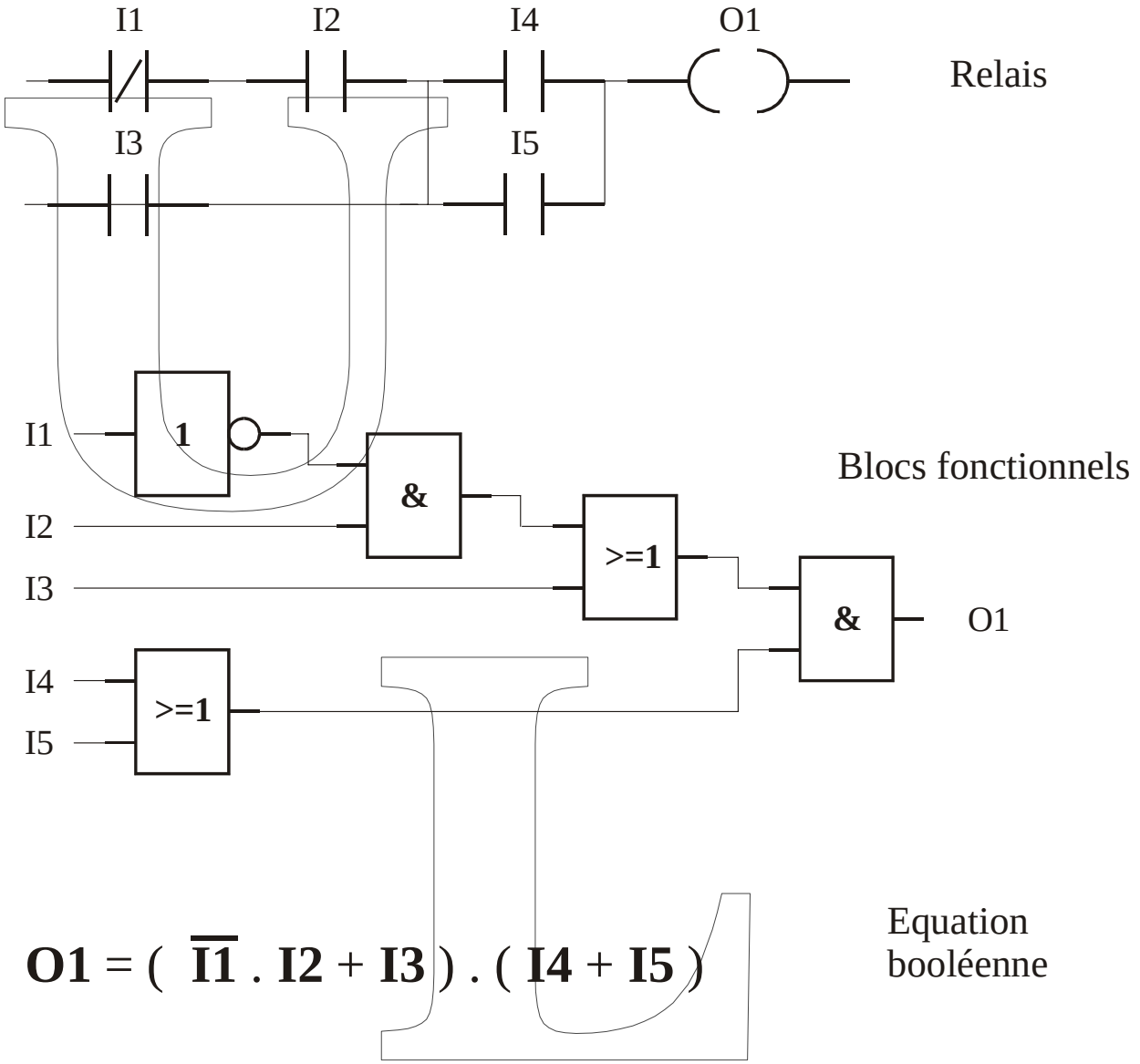


Figure 4.1. Modes de représentation parallèles

La figure 4.2. montre un exemple de programmation en langage relais.

Dans ce contexte, le langage de programmation par liste d'instructions décrit au § 2.3.5. apparaît comme beaucoup moins cohérent avec la méthode de conception "parallèle" envisagée ici. En effet, il faudra que le programmeur "sérialise" lui-même la fonction logique à réaliser en tenant compte de la manière séquentielle selon laquelle travaille effectivement le processeur de l'automate (présence d'un accumulateur notamment). Le listing d'un programme n'aura dès lors plus l'aspect fonctionnel du cas précédent. En tout état de cause, il ne sera plus compréhensible qu'accompagné du schéma dont il est issu, ce qui complique les problèmes de documentation et d'archivage.

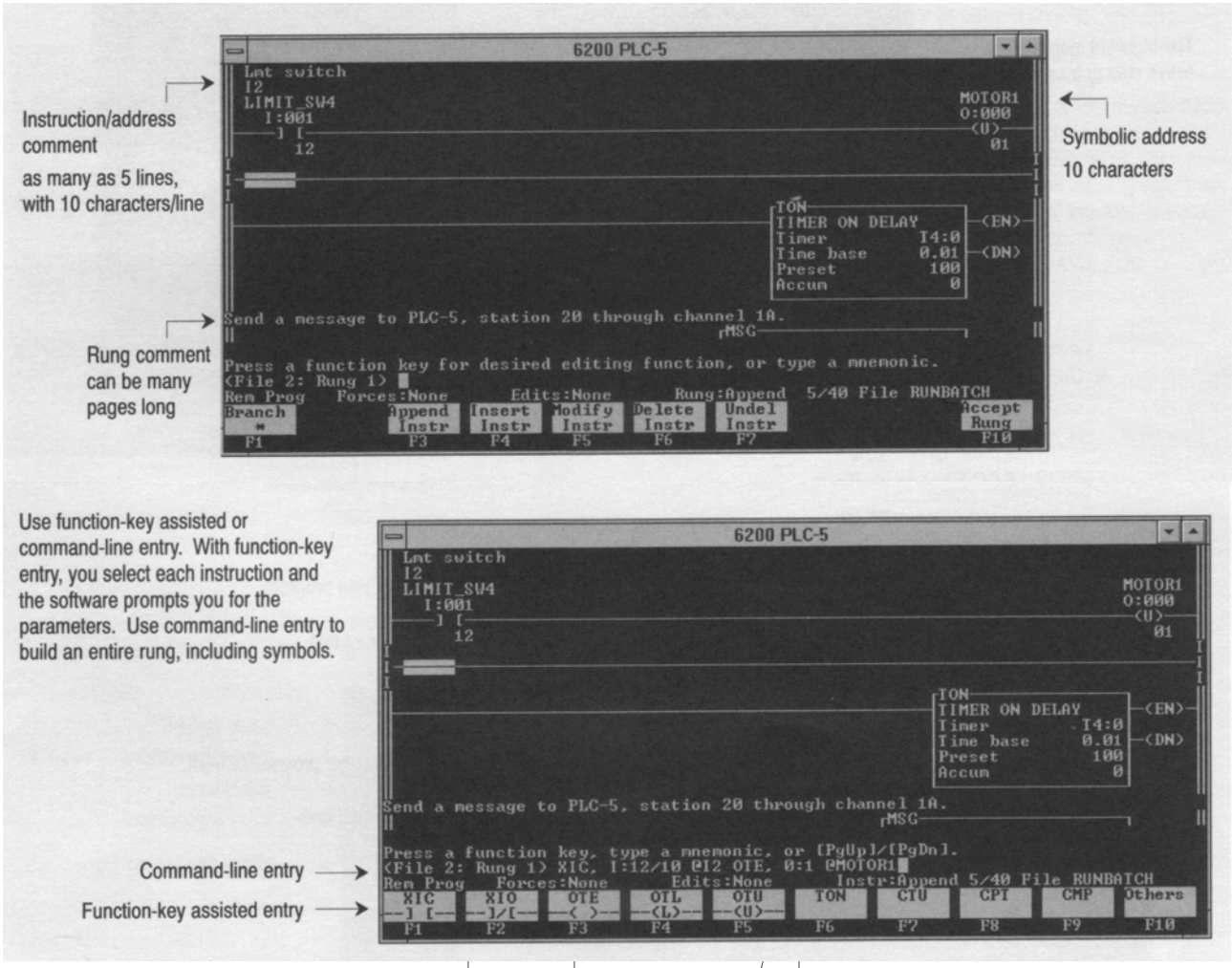
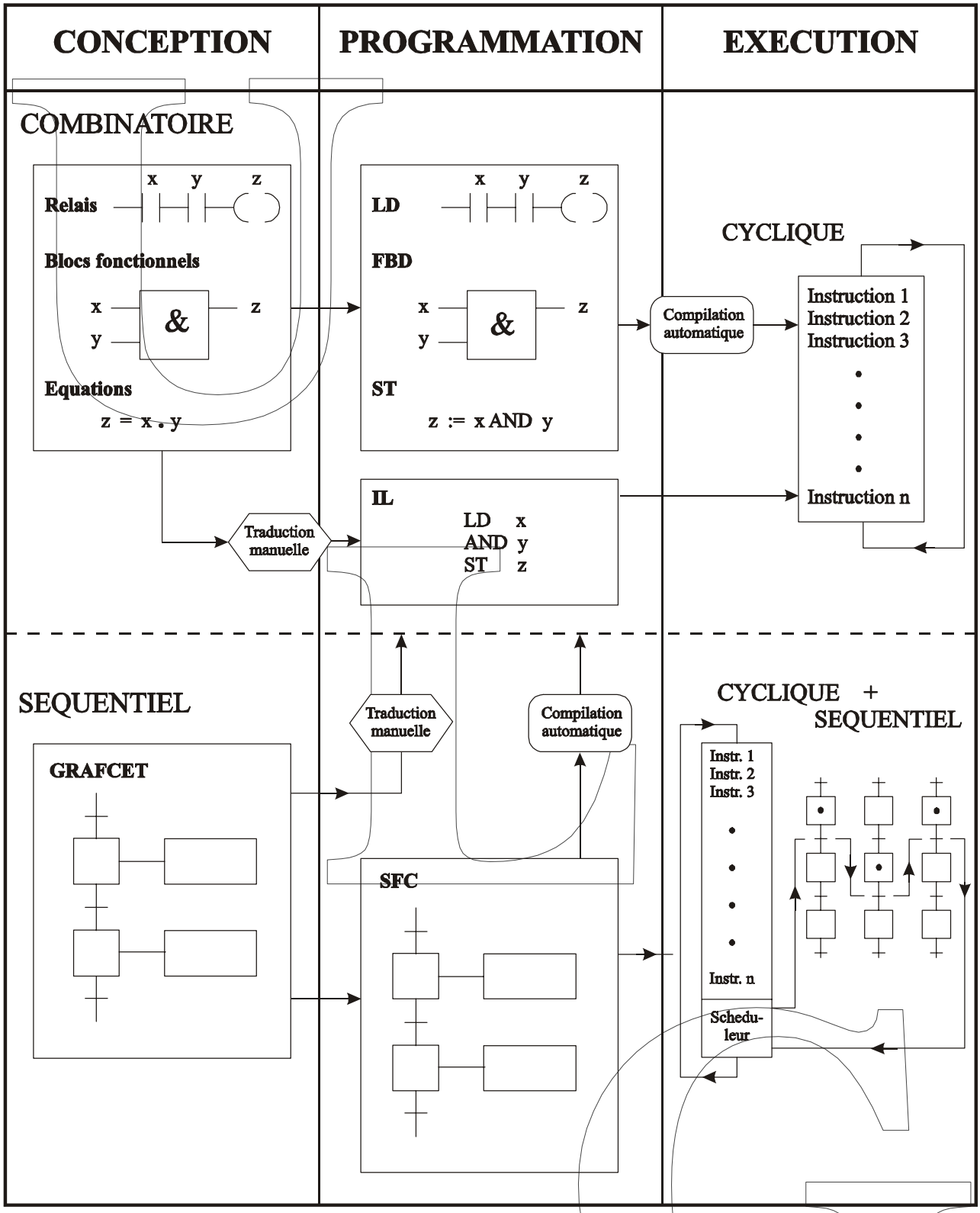


Figure 4.2. Exemple de programmation en langage relais
Source : ALLEN-BRADLEY

4.2.3. EXECUTION

Le mode d'exécution qui s'impose dans le contexte parallèle considéré ici est le mode d'exécution linéaire et cyclique puisqu'il s'agit en fait de simuler au mieux le fonctionnement d'un ensemble de modules fonctionnels travaillant en parallèle. Toutes les entrées de l'automate seront ainsi scrutées à chaque cycle et toutes ses sorties seront recalculées à chaque cycle.

La figure 4.3. symbolise les trois phases qui viennent d'être décrites.



LD : Ladder Diagram **FBD** : Function Block Diagram **ST** : Structured Text

IL : Instruction List **SFC** : Sequential Function Chart

Figure 4.3 Cohérence entre les méthodes de conception de programmation et d'exécution d'un automatisme

4.3. PROBLEMES DE NATURE SEQUENTIELLE

Il arrive souvent, dans l'industrie, que les processus à contrôler aient une nature séquentielle marquée. C'est le cas notamment de tous les processus comportant des déplacements d'outils, des transports de matière, etc. Du point de vue du système de contrôle, ces processus se caractérisent par le fait qu'à un moment donné, seuls un nombre limité de signaux provenant du processus doivent être pris en considération et que seules un nombre limité de commandes sont susceptibles d'être appliquées au processus.

Pour fixer les idées, nous adopterons l'exemple très simple présenté à la figure 4.4.

Un chariot initialement dans la position A (fin de course IN 2 ouvert) est mis en mouvement (contact OUT 1) vers la position B par action sur le bouton poussoir IN 1. Arrivé en B (fin de course IN 3 ouvert) le chariot est renvoyé vers A (contact OUT 2).

On peut distinguer trois étapes dans le déroulement de ce processus :

1ère étape : chariot en position de départ, moteur arrêté (OUT 1 = OUT 2 = 0).
On voit bien que, dans cette étape, seule l'entrée IN 1 peut faire évoluer la situation et seule la sortie OUT1 peut être activée.

2ème étape : consécutive à la pression sur IN 1. Chariot en mouvement vers B (OUT 1 = 1), seule l'entrée IN3 peut modifier la situation.

3ème étape : consécutive à l'arrivée du chariot en B (IN3 = 0). Retour du chariot vers A (OUT 1 = 0, OUT 2 = 1) seule IN 2 peut modifier la situation.

4.3.1. CONCEPTION

Formalisme parallèle

Dans le cas extrêmement simple de l'exemple, la conception du système de contrôle pourrait encore s'envisager par la méthode parallèle du § 4.2.1.

La figure 4.4. montre la solution obtenue dans le formalisme à relais.

On peut faire les remarques suivantes :

- la <u>séquence</u> caractéristique du processus n'apparaît absolument pas dans ces modes de représentation. Il s'ensuit des problèmes de conception, de mise au point et simplement de documentation qui ne font évidemment que croître avec la complexité du processus.

- la <u>scrutation cyclique systématique</u> des entrées à laquelle conduit la méthode parallèle, est fondamentalement inutile, puisque dans les différentes étapes décrites ci-dessus, il n'y a jamais qu'une entrée à la fois susceptible de faire évoluer la situation.
De même le calcul cyclique systématique des sorties est en principe superflu puisque ces sorties ne peuvent en fait varier que lorsqu'on passe d'une étape à la suivante.

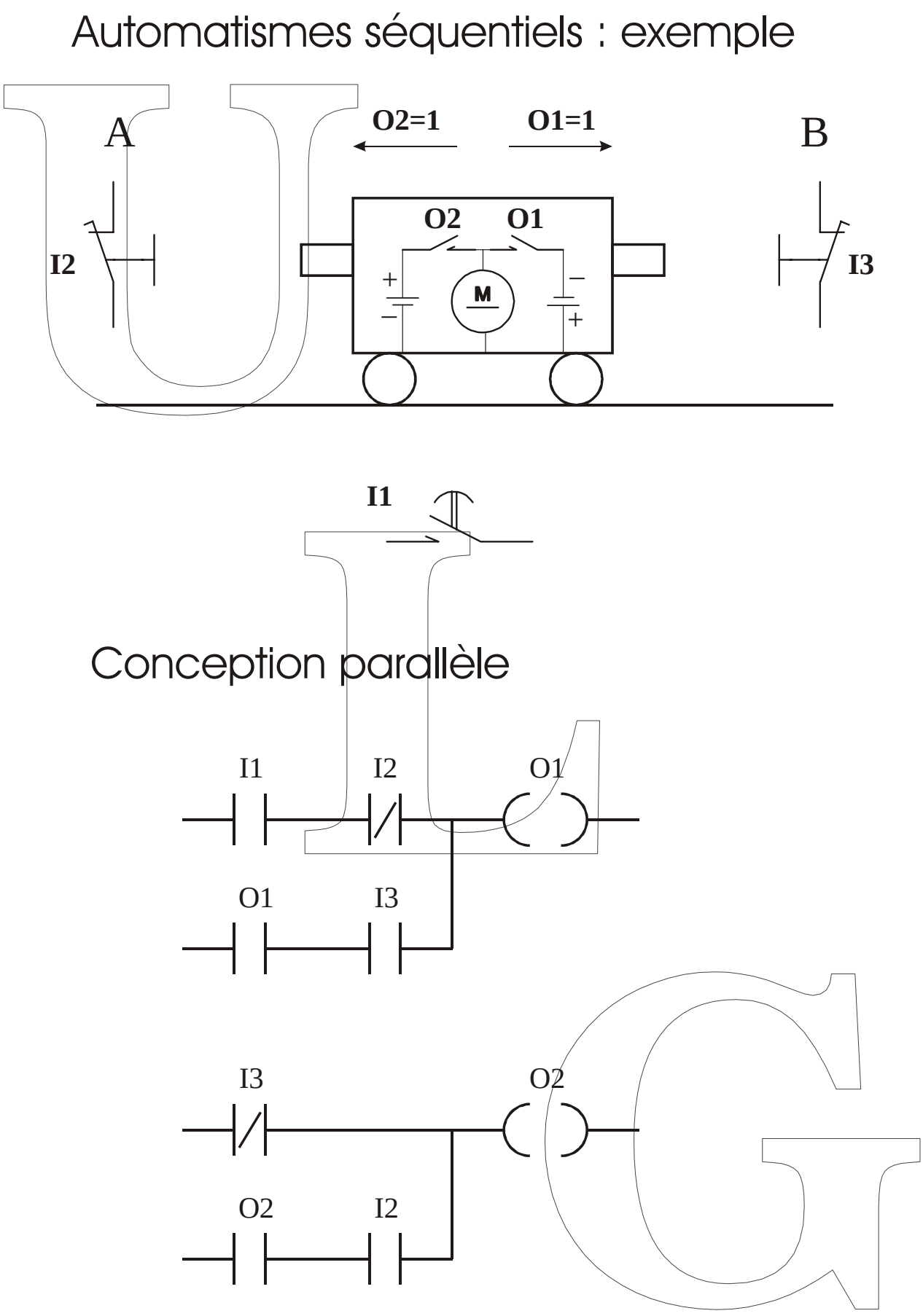


Figure 4.4. Utilisation du formalisme combinatoire dans le cas d'un automatisme séquentiel

GRAFCET

La conception par GRAFCET est manifestement mieux adaptée au problème comme le montre la figure 4.5.

Le GRAFCET, sous le sigle SFC (Sequential Function Chart), a fait l'objet d'une normalisation (cfr. chapitre 8). C'est pourquoi, dans la suite du texte, nous l'adopterons comme référence.

Les carrés représentent les étapes du fonctionnement et les rectangles associés, les actions à prendre lorsque l'étape est active. Le passage d'une étape à la suivante est verrouillé par une condition logique appelée réceptivité.

Lorsqu'une étape est active et que la réceptivité associée est vraie, on passe automatiquement à l'étape suivante, tandis que la première est désactivée.

Une présentation détaillée du GRAFCET sera faite au chapitre 6.

4.3.2. PROGRAMMATION

Il est parfaitement possible de programmer un automate à partir d'un GRAFCET dans n'importe lequel des langages combinatoires présentés ci-dessus.

Considérons par exemple le cas d'un langage à relais élémentaire : à chaque étape de l'évolution du processus on va associer une variable interne qui vaudra 1 si l'étape est active et 0 dans le cas contraire.

La figure 4.6. montre le résultat obtenu avec un langage à relais.

Il peut sembler à première vue assez lourd mais il a l'avantage de :

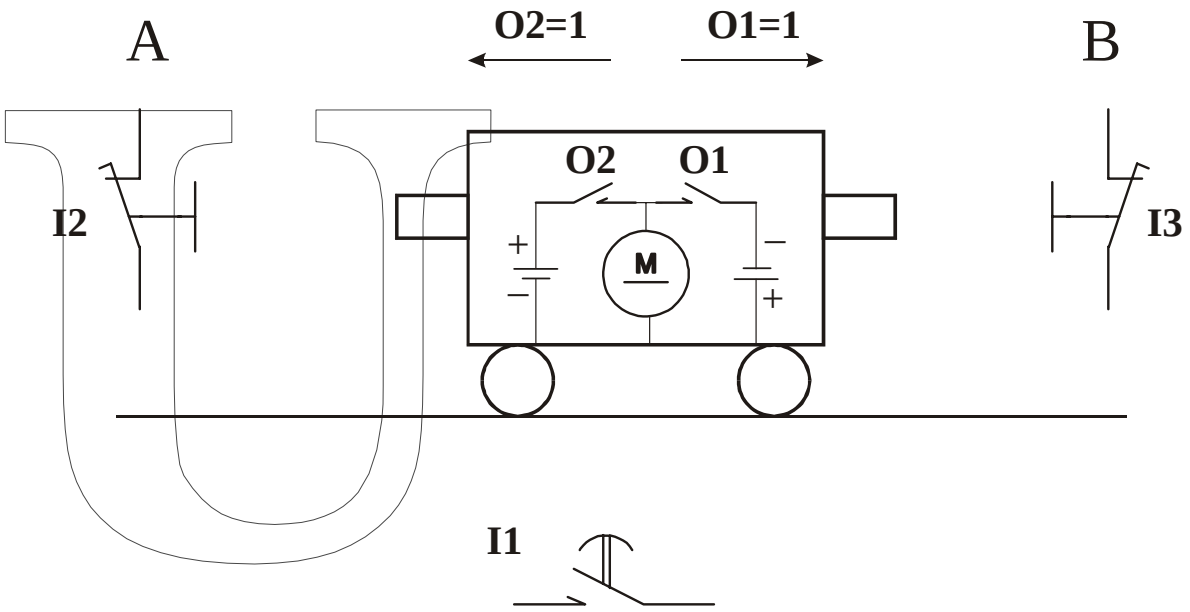
– mettre clairement en évidence la séquence réalisée;
– systématiser la conception du système : les conditions d'activation et de désactivation des variables d'étape se construisent en effet toutes de la même façon;

ce qui facilite les tests de mise au point.

Bien entendu, cette manière de faire ne trouve vraiment tout son intérêt que dans des cas plus complexes que celui de l'exemple.

Le caractère détourné de la démarche qui vient d'être suivie apparaît clairement sur la figure 4.3. Le mode de représentation séquentiel de départ a dû être transposé en un mode de représentation parallèle pour pouvoir être programmé et donner lieu finalement à une exécution fondamentalement séquentielle !

Notons également que cette exécution séquentielle peut entraîner des conditions de courses assez pernicieuses dans le calcul des variables, si bien que la transposition d'un GRAFCET ne sera en général pas aussi simple que ce qui est présenté ici. Nous y reviendrons au paragraphe 6.6. lorsque les règles d'évolution du GRAFCET auront été expliquées en détail.



GRAFCET

Graphe de Commande Etape-Transition

Actions

Etape 1 : Arrêt en A

1

O1 = 0 ; O2 = 0

I1 — **Réceptivité** : bouton poussoir

Etape 2 : Mouvement de A vers B

2

O1 = 1 ; O2 = 0

/I3 — **Réceptivité** : arrivée en B

Etape 3 : Mouvement de B vers A

3

O1 = 0 ; O2 = 1

/I2 — **Réceptivité** : retour en A

Figure 4.5. Mode de représentation séquentiel

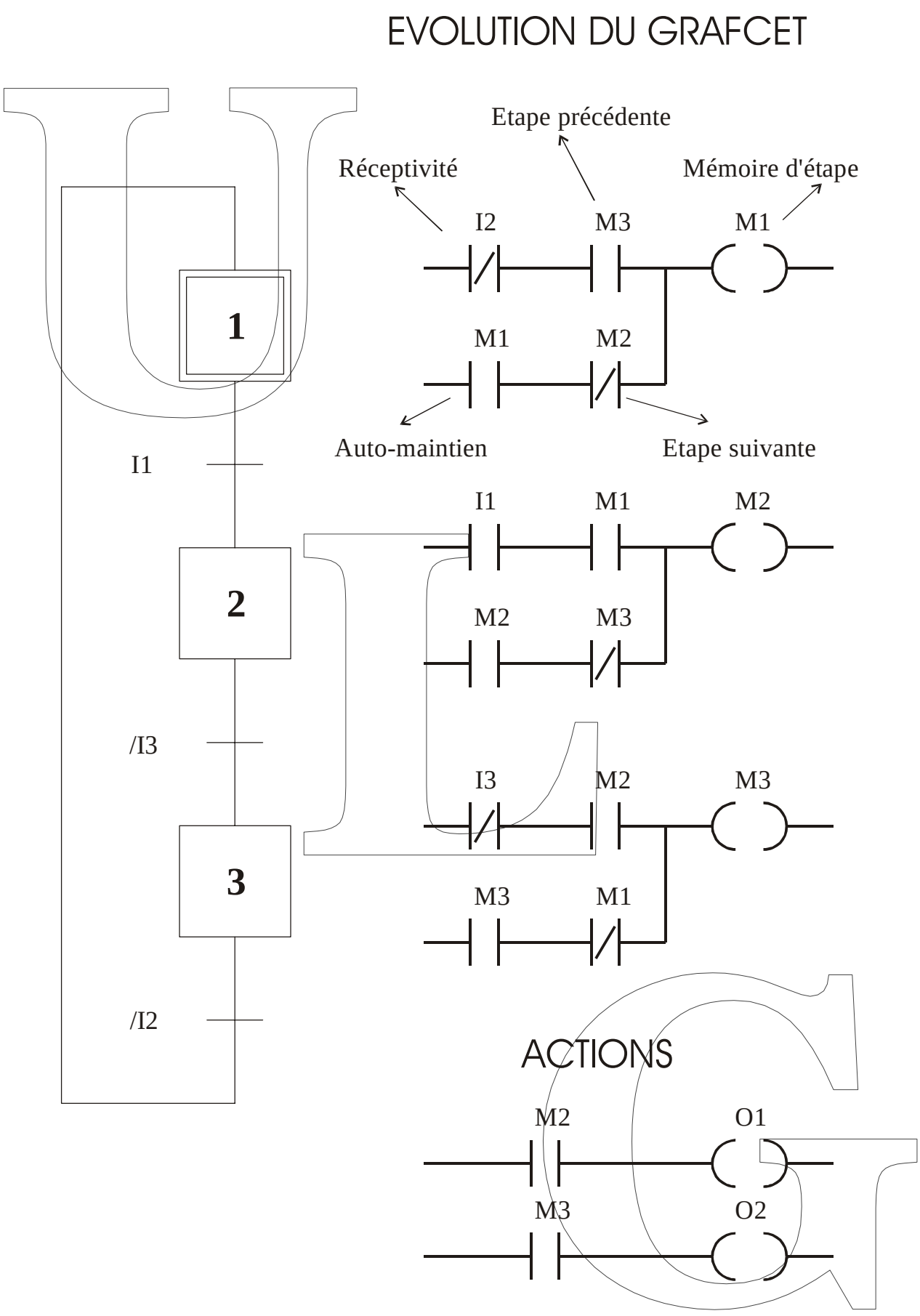


Figure 4.6. Principe de la transposition du GRAFCET en langage combinatoire

4.4. LANGAGES SEQUENTIELS

Etant donné l'importance pratique des problèmes séquentiels, les constructeurs d'automates ont prévu un certain nombre d'outils de programmation séquentielle destinés à faciliter la tâche de l'utilisateur.

4.4.1. EXTENSION DU LANGAGE COMBINATOIRE

Relais à accrochage (LATCH) : ce sont des relais qui reproduisent le fonctionnement de bistables type SR (SET/RESET). Le bistable est enclenché lorsque son entrée SET vaut 1. Il reste dans cet état même si cette entrée repasse à 0. Pour le déclencher, il faut que l'entrée RESET soit mise à 1.

L'emploi de relais à accrochage dans la transposition du GRAFCET évite de devoir utiliser des auto-verrouillages (cfr. figure 4.6.).

4.4.2. LANGAGE GRAFCET

S'appuyant sur la puissance de traitement des micro-ordinateurs PC, de plus en plus de constructeurs offrent, actuellement, des langages de programmation graphiques qui consistent à reproduire, sur un écran, le dessin même d'un GRAFCET.

La figure 4.7. en donne un exemple concret.

A l'exécution, les étapes actives apparaîtront en surbrillance par exemple, permettant de suivre aisément l'évolution dynamique du système.

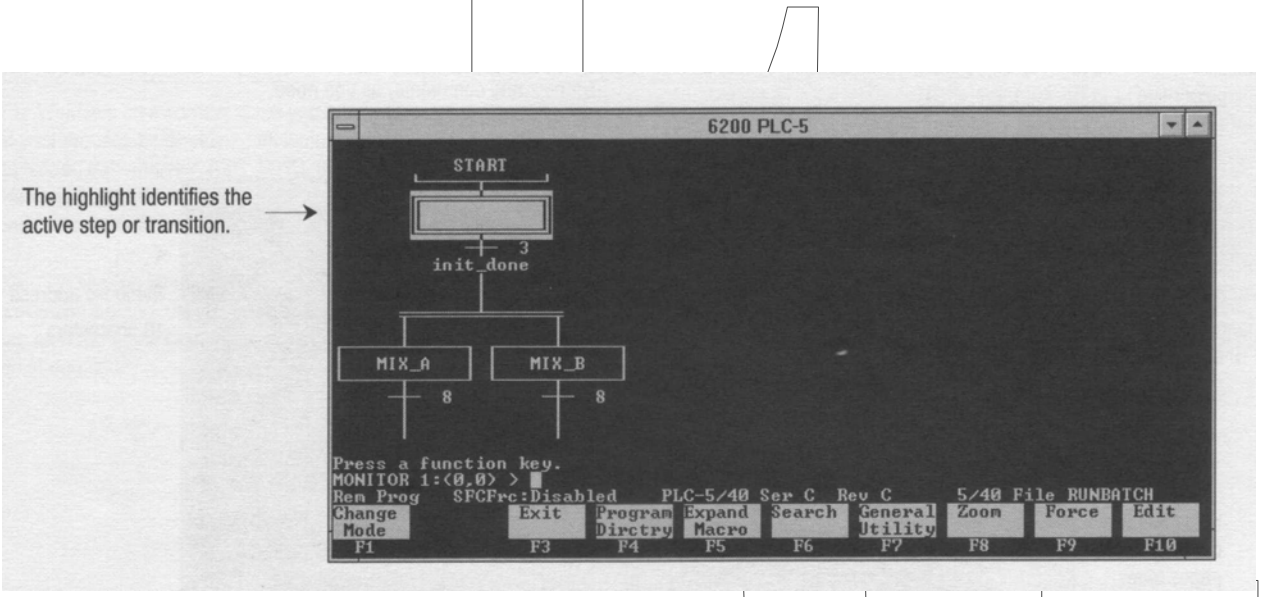


Figure 4.7. Exemple de langage de programmation en GRAFCET
Source : ALLEN-BRADLEY

4.5. EXECUTION SEQUENTIELLE

A partir du langage GRAFCET évoqué ci-dessus, deux prolongements sont possibles au niveau de l'exécution.

Le premier consiste à opérer la traduction du GRAFCET dans un langage "parallèle" traditionnel automatisant donc, en quelque sorte, la procédure décrite au § 4.3.2. Cette manière de faire débouche sur une exécution cyclique "aveugle" avec les inconvénients dénoncés au paragraphe 4.3.1. C'est celle que l'on trouve généralement dans les ateliers logiciels multi-automates (cf. chapitre 9). Notons que la traduction est, dans ce cas, bidirectionnelle c'est-à-dire que l'utilisateur travaille exclusivement au niveau du GRAFCET que ce soit pour l'édition ou pour le test des programmes.

Le deuxième consiste à adapter l'exécution à la nature séquentielle du problème. Au lieu de balayer cycliquement et systématiquement tout le programme, c'est-à-dire de scruter toutes les entrées et de recalculer toutes les sorties, le processeur ne va plus faire ici que de vérifier les conditions logiques qui intéressent l'évolution du système, c'est-à-dire les réceptivités, et, le cas échéant, déterminer les actions correspondant à une nouvelle étape active.

Cette dernière optique, que l'on trouve dans les langages GRAFCET proposés par les constructeurs d'automates, a des conséquences importantes sur les performances en temps de réponse des automates. Il est en effet possible d'avoir de très grands programmes en mémoire tout en conservant des temps de réaction entrées/sorties extrêmement courts.

En pratique, on trouvera généralement, dans un automatisme, plusieurs séquences se déroulant en parallèle et, en plus, on aura un certain nombre de fonctions combinatoires à réaliser, ne fut-ce que pour le traitement des alarmes.

On en arrive ainsi au mode d'exécution schématisé à la fig. 4.8.

L'automatisme a été décomposé en sa partie combinatoire et en ses différentes séquences parallèles.

La partie combinatoire est placée dans une première branche qui sera exécutée linéairement et cycliquement comme dans un automate classique.

Les séquences seront placées chacune dans une série d'autres branches réservées à cet effet (le nombre total de branches possibles est une caractéristique de l'automate).

La branche combinatoire cyclique comporte également un programme appelé schéduleur chargé de gérer l'évolution des différentes branches séquentielles. Il s'agit en l'occurrence de tester dans chaque branche la condition logique verrouillant l'étape en cours et, le cas échéant, d'exécuter l'étape suivante. Le schéduleur doit donc, entre autres, mémoriser l'état d'avancement des différentes séquences.

Il est clair, comme le montre la figure 4.3, qu'avec le mode d'exécution séquentielle qui vient d'être décrit, on atteint une parfaite cohérence entre la conception, la programmation et l'exécution d'un automatisme.

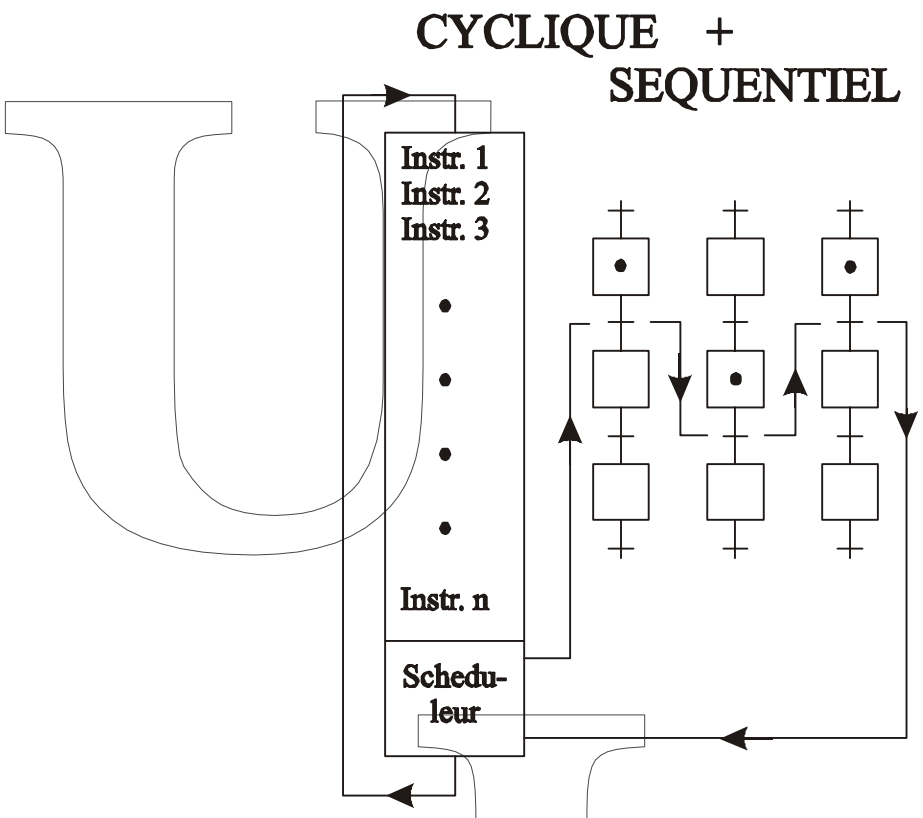


Figure 4.8. Principe de l'exécution séquentielle

Chapitre 5

FONCTIONS SPECIALES

5.1. INTRODUCTION

Dans ce chapitre, nous essayerons de donner un aperçu des principales fonctions spéciales que l'on peut trouver sur les automates programmables.

Notons que ces fonctions sont, soit réalisées par voie logicielle, c'est-à-dire par le processeur principal de l'automate programmable, soit par voie matérielle, c'est-à-dire par un module spécial greffé sur le BUS de l'automate. Nous le spécifierons dans chaque cas particulier.

5.2. EXTENSION DE LA LOGIQUE DE BASE

Il s'agit d'extensions mettant à la disposition de l'utilisateur des fonctions couramment rencontrées dans les automatismes, ce qui lui évite de devoir chaque fois les reprogrammer à partir des instructions logiques élémentaires.

5.2.1. TEMPORISATION

La fonction de temporisation permet de décaler un signal logique par rapport à un autre d'un délai fixé. Ce décalage peut être réalisé à l'enclenchement (figure 5.1.a) ou au déclenchement (figure 5.1.b). D'autres variantes se rencontrent encore chez certains constructeurs que nous ne pourrions toutes détailler ici.

Sur les automates modernes, la temporisation est une opération purement logicielle réalisée à partir d'une horloge interne à 100 ms ou, plus rarement, à 10 ms.

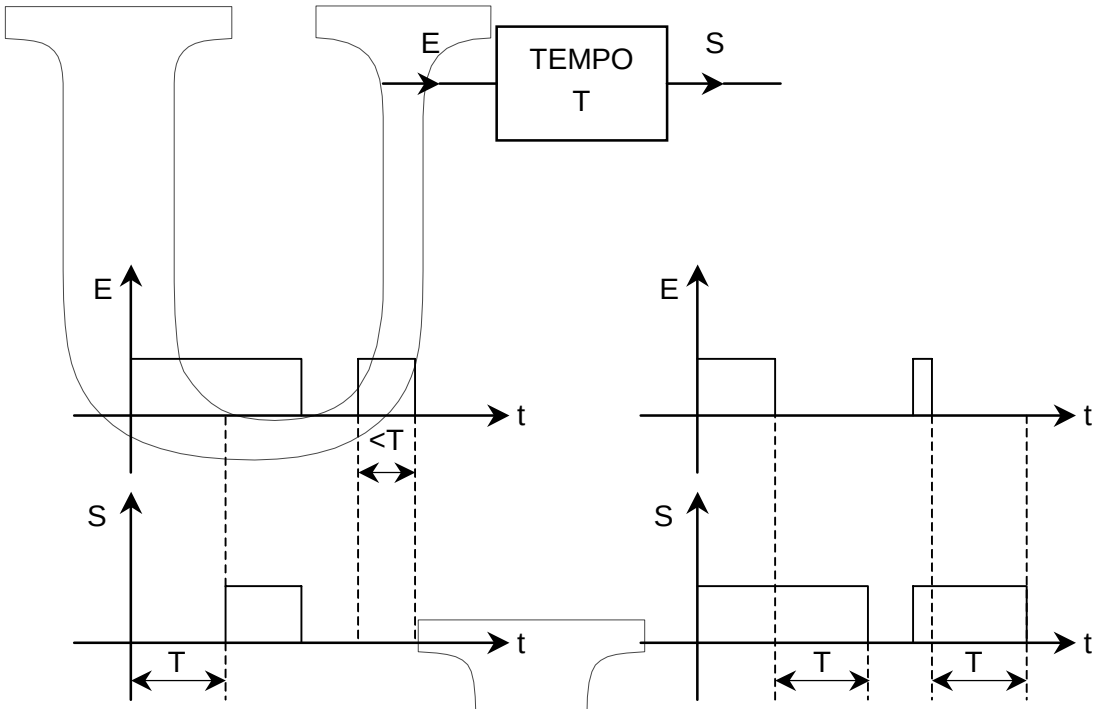
5.2.2. COMPTAGE

Le comptage et le décomptage sont également des fonctions spéciales abondamment utilisées en pratique : comptage de pièces, mesure de vitesse, positionnement, etc.

La figure 5.2. montre le cas d'un compteur avec présélection: les impulsions d'entrées sont comptées et lorsque le résultat N atteint une valeur de présélection P définie à la programmation, un signal logique de sortie, S, est émis.

Des compteurs logiciels existent dans tous les automates, avec de multiples variantes, mais ils ne peuvent évidemment compter que des impulsions espacées dans le temps d'une durée significativement supérieure au temps de cycle de l'automate (10 à 100 ms).

Les comptages plus rapides font appel à des cartes spéciales qui acceptent alors des fréquences d'entrées de plusieurs dizaines de KHz typiquement.



a. Retard à l'enclenchement b. Retard au déclenchement

Figure 5.1 Fonction de temporisation

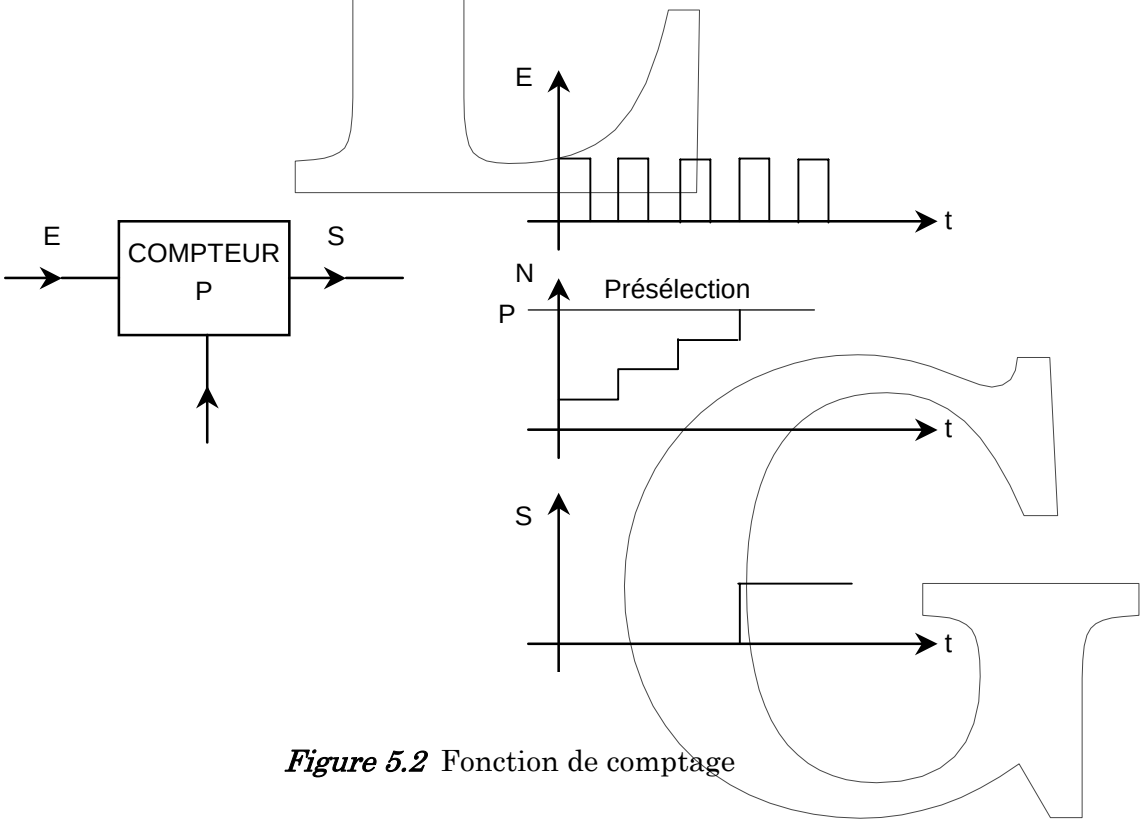


Figure 5.2 Fonction de comptage

5.2.3. DIFFERENTIATEURS

Ce sont des fonctions souvent fort utiles, qui délivrent des impulsions d'une durée égale à un cycle de l'automate lors du changement d'état de variables logiques. La figure 5.3. en illustre le principe ainsi que le symbolisme en langage relais.

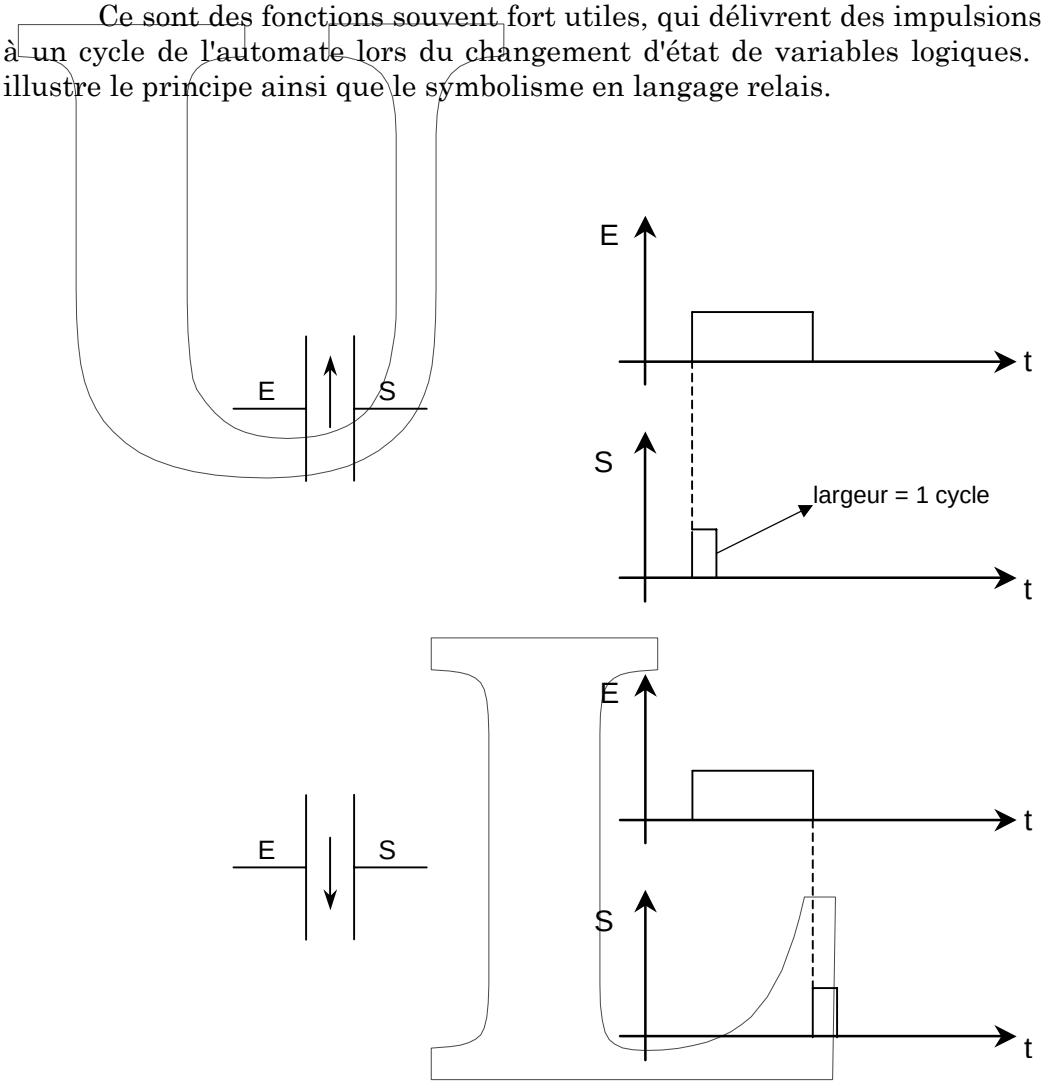


Figure 5.3. Principe des différentiateurs

5.3. FONCTIONS DE SEQUENCEMENT

Ces fonctions sont mentionnées ici pour mémoire. Elles ont été décrites en détail au paragraphe 4.4.1 :

– relais à accrochage.

5.4. FONCTIONS D'ORGANISATION DU CYCLE

Il s'agit de fonctions qui permettent à l'utilisateur d'intervenir sur le déroulement du cycle de l'automate.

5.4.1. BRANCHEMENTS ET REPETITIONS

– **Fonctions de branchement** :

elles permettent de sauter, conditionnellement ou inconditionnellement, une série d'instructions lors du déroulement du programme.

On distingue encore entre sauts avant, sauts arrières, sauts absolus (vers une adresse donnée) ou relatifs (sauts d'un nombre d'instructions données).

– **Boucles de répétition** :

elles permettent de répéter un groupe d'instructions un nombre donné de fois.

La figure 5.4. schématise l'effet de ces différentes instructions sur le déroulement d'un programme.

5.4.2. SOUS-ROUTINES

L'instruction de saut vers une sous-routine a pour effet de faire abandonner par le processeur l'exécution de la séquence d'instructions en cours au profit d'une séquence (la sous-routine) située ailleurs dans la mémoire. Cette deuxième séquence se termine par une instruction spéciale (RETURN) qui ramène le processeur à l'endroit exact où il avait quitté la séquence initiale.

Les sous-routines peuvent être utilisées dans deux optiques différentes :

– **pour économiser de la place mémoire** :

Si la même séquence d'instructions se reproduit plusieurs fois dans un programme, on a intérêt à en faire une sous-routine qui ne devra dès lors plus être présente qu'une seule fois dans la mémoire (cfr. figure 5.4.e).

– **pour structurer les programmes** :

Lorsque les problèmes d'automatisme deviennent trop complexes, on est naturellement amené à les décomposer en sous-ensembles plus simples, correspondant par exemple à des entités technologiques (machines ou parties cohérentes de machine), que l'on n'a plus alors qu'à coordonner.

Il est intéressant de pouvoir reproduire cette structuration hiérarchique dans l'écriture du programme.

On trouvera ainsi un module d'organisation qui définira les interactions entre les modules du niveau hiérarchique inférieur. Chacun de ces derniers pouvant à son tour être décomposé et ainsi de suite.

Chaque module aura donc une mission fonctionnelle bien définie. En organisant judicieusement la décomposition, on peut même arriver à ce que les programmes correspondant aux différents modules aient une taille qui permette d'en saisir d'un seul coup d'oeil tout le contenu informatif (une page de listing par exemple).

La manière dont les sous-routines permettent de structurer un programme est schématisé à la figure 5.4.f.

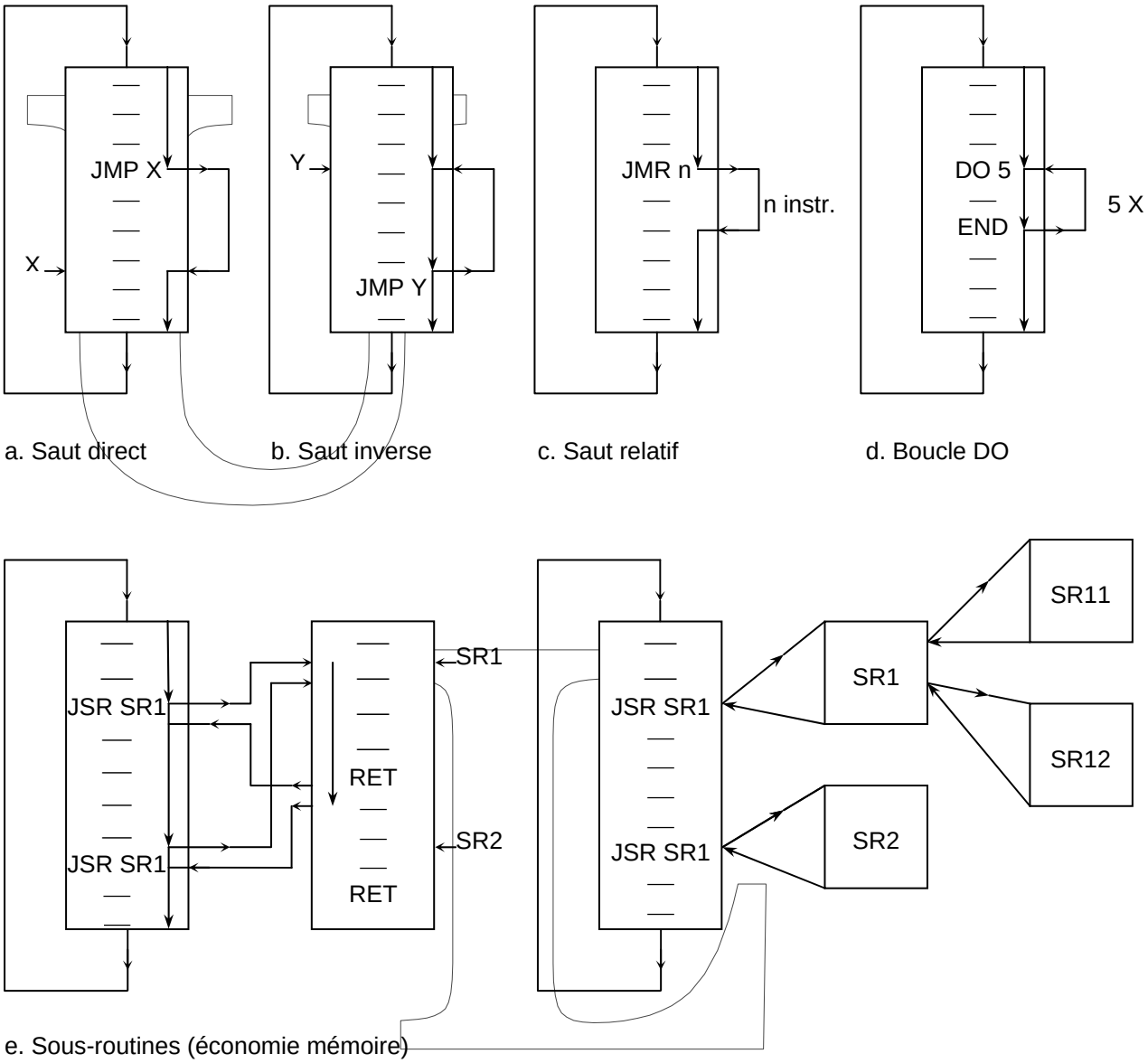


Figure 5.4. Instructions pour le contrôle du cycle

5.4.3. INTERRUPTIONS

Dans ce qui a été vu jusqu'ici, l'automate ne pouvait prendre en considération les événements survenant dans le monde extérieur (changement d'état d'une entrée par exemple) que dans le cadre d'une scrutation cyclique de toutes les sources possibles d'événements (c'est-à-dire des entrées). Il était donc nécessaire d'adopter un temps de cycle inférieur au temps de réaction minimal exigé de l'automate pour l'événement le plus urgent. Si le nombre de ces entrées urgentes est faible vis-à-vis du nombre total des entrées, cela conduit inévitablement à devoir utiliser des automates surpuissants.

Les interruptions constituent, à ce égard, une solution beaucoup plus efficace. Les sources d'interruptions possibles sont connectées individuellement ou par l'intermédiaire d'un circuit OU à des lignes spéciales du BUS, les lignes d'interruption.

Le processeur scrute automatiquement (c.-à-d. que l'utilisateur ne doit rien prévoir dans son programme) ces lignes à la fin de chaque instruction (donc toutes les quelques

microsecondes !). S'il détecte une demande d'interruption, il interrompt l'exécution du programme et, toujours automatiquement, sauve, dans une partie de la mémoire prévue à cet effet, toutes les informations nécessaires à la reprise ultérieure du programme (par exemple : adresse de l'instruction, valeur de l'accumulateur, etc.). Il entreprend alors l'exécution d'un programme particulier, prévu par l'utilisateur, et qui réalise le traitement d'urgence désiré. Celui-ci terminé, le processeur reprend l'exécution du programme interrompu.

Le besoin des interruptions s'est surtout fait sentir pour les automates à microprocesseurs dont le temps de cycle est quand même relativement lent.

5.4.4. REMARQUE

Il est important de noter qu'avec la plupart des fonctions examinées ci-dessus, on rompt la cohérence entre modes de représentation des automatismes et modes d'exécution des programmes (cfr. chapitre 4).

D'autre part, si ces fonctions peuvent encore s'intégrer de manière plus ou moins plausible (du moins sur le plan formel) dans un langage de type liste d'instructions, elles apparaissent par contre comme particulièrement inadaptées aux langages graphiques de type parallèle.

Il n'empêche que les fonctions d'organisation du cycle, maniées par des programmeurs avertis, permettent souvent de réaliser de substantiels gains en place mémoire et/ou en temps de cycle.

5.5. OPERATIONS SUR MOTS

Avec les fonctions présentées ci-après, on quitte le domaine de la logique pure pour aborder des traitements couramment rencontrés dans les ordinateurs. Ils portent non plus sur des bits mais sur des mots dans lesquels on peut coder des nombres, des caractères alphanumériques (code ASCII), etc.

On trouve ainsi des opérations :

- arithmétiques : +, –, x, :
- logiques (sur mots) : AND, OR, XOR, ...
- de test : =, >, <
- de conversion de code : BCD → BIN, BIN → BCD
- etc...

Ces opérations s'intègrent plus ou moins harmonieusement dans les différents langages de programmation décrits au chapitre 4.

La figure 5.5. donne quelques exemples.

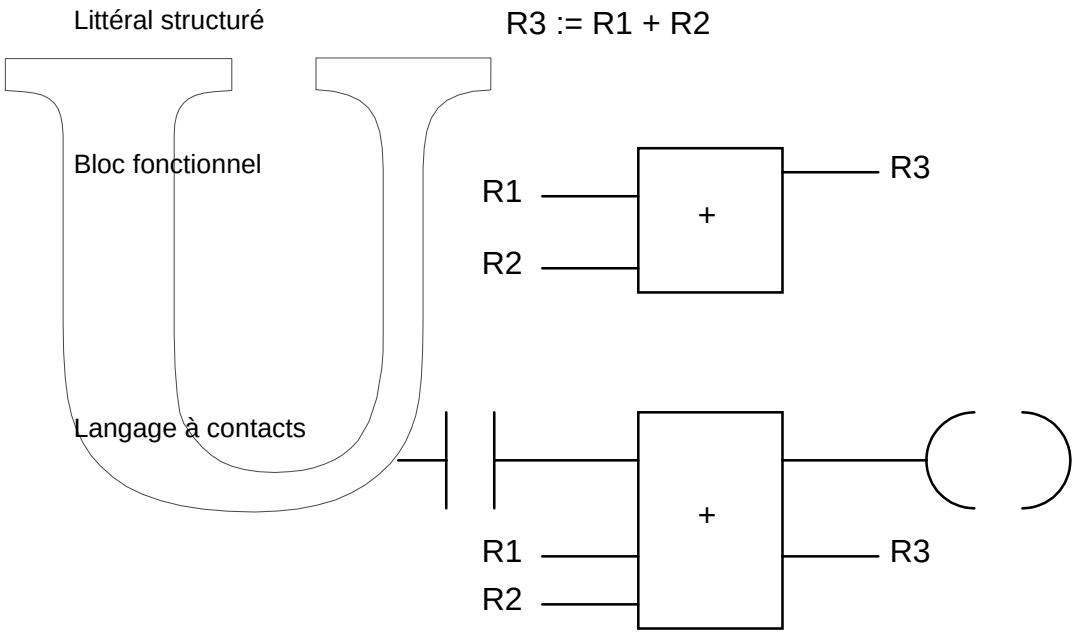


Figure 5.5. Exemple de programmation de fonctions extra logiques

5.6. MANIPULATION DE DONNEES

Il s'agit ici d'opération portant également sur des mots ou des groupes de mots qui relèvent de la gestion de données, plutôt que du calcul. Nous donnerons ci-dessous quelques exemples typiques.

5.6.1. GESTION DE TABLES DE VALEUR

Des registres de données consécutifs peuvent être organisés en table. L'accès aux données (en entrées et en sorties) se fait sur l'intermédiaire d'un pointeur qui réalise en fait un adressage indexé par rapport au début de la table.

La figure 5.6.a. donne l'exemple de la définition d'une table de données et du pointeur associé. On peut par exemple s'en servir pour réaliser un générateur de fonction : la valeur du pointeur constituant l'abscisse et la valeur pointée l'ordonnée.

5.6.2. OPERATIONS MATRICIELLES

Il s'agit d'instructions permettant de travailler sur des blocs de variables binaires contigües (entrées, mémoires) :

- transferts de bloc à bloc
- comparaison de deux blocs
- AND, OR, XOR de bloc à bloc
- etc.

Des programmes très compacts peuvent ainsi être réalisés pour, par exemple :

- surveiller les changements d'état d'un groupe d'entrées;

– comparer, à différents stades de l'évolution d'un automatisme, les états d'un groupe d'entrée avec des valeurs de références et détecter ainsi d'éventuelles anomalies;
– etc.

5.6.3. EDITION DE TEXTE

Cette opération consiste à sortir sur une imprimante (ou sur un écran) un texte, c.-à-d. une suite de caractères alphanumériques (un caractère est codé sur un byte), stocké dans une table de données.

Notons que cette édition comporte la gestion d'une porte de communication série que nous étudierons plus en détail dans l'ouvrage "Réseaux locaux industriels". Elle demande un certain temps qui dépend de la longueur du message et de la vitesse d'impression de l'imprimante.

C'est pourquoi la fonction édition comporte un signal de retour (cfr figure 5.6.b.) indiquant si le message a été complètement traité. On ne peut envoyer d'autres messages tant que ce signal n'est pas présent.

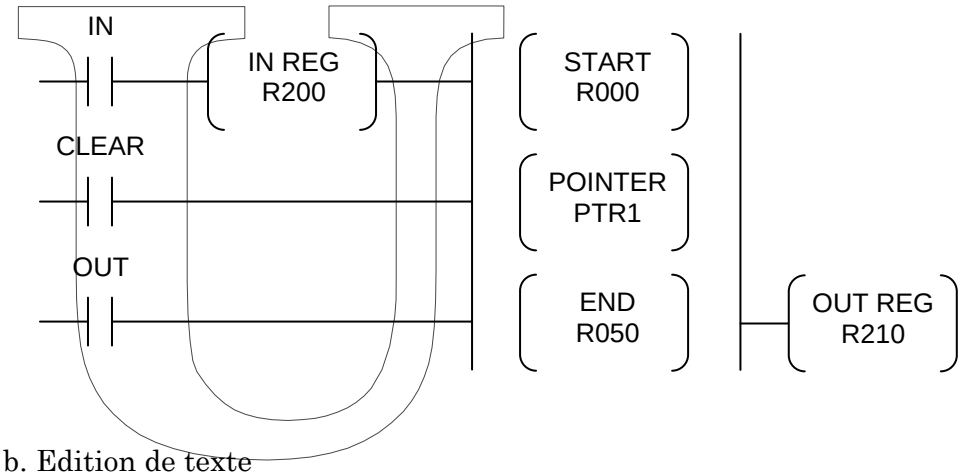
5.6.4. Pile FIFO (First In First Out)

Il s'agit d'une table gérée de manière particulière : les données qui y sont introduites vont automatiquement s'empiler dans le "fond" de la table. En sortie, c'est toujours le dernier élément de la table qui est prélevé tandis que les autres descendent tous d'un cran dans la table (cfr figure 5.6.c).

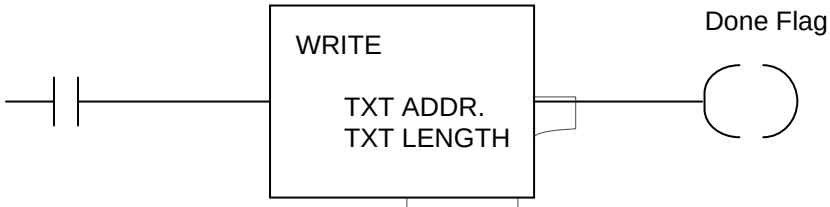
Le FIFO permet de réaliser un tampon entre un processus de "production" d'informations et un processus de "consommation" d'informations qui ne sont pas synchrones. Un cas d'utilisation typique est celui de l'édition de messages d'alarme. Des rafales d'alarmes peuvent en effet se produire dans un temps très court de manière telle que l'imprimante sera en général incapable de "suivre".

Pour éviter de perdre des messages, on les empilera dans un FIFO dont ils seront extraits au rythme propre de l'imprimante. Il faut évidemment doter le FIFO d'une capacité suffisante pour pouvoir absorber les pointes de charge sans saturer.

a. Table de données



b. Edition de texte



c. Registre FIFO

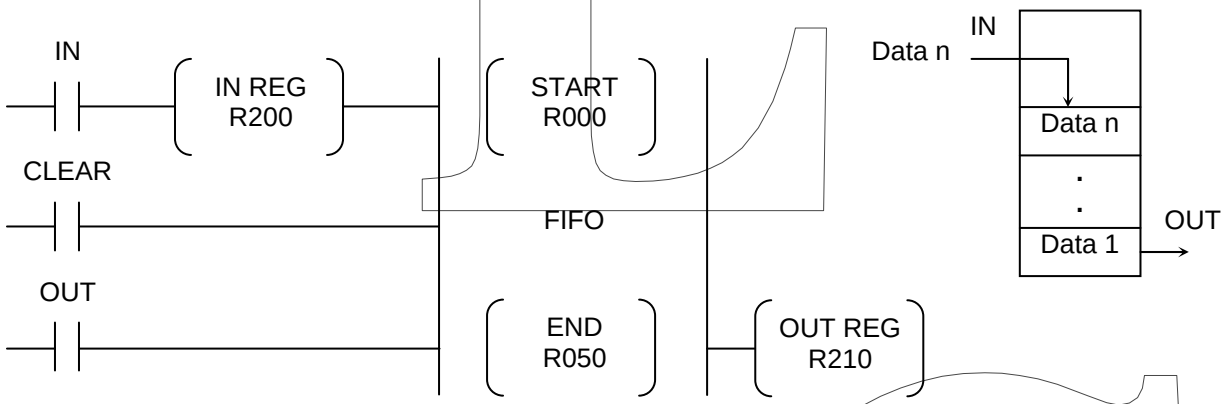


Figure 5.6. Exemples d'opérations de manipulation de données

5.7. REGULATION

Dans ce paragraphe, nous traiterons le cas du régulateur PID étant donné son importance particulière : c'est en effet lui qui ouvre aux automates l'accès à l'autre grand domaine du contrôle de processus, la régulation.

5.7.1. RAPPEL DE L'ALGORITHME PID

Le problème général de la régulation se présente comme il est montré à la figure 5.7.

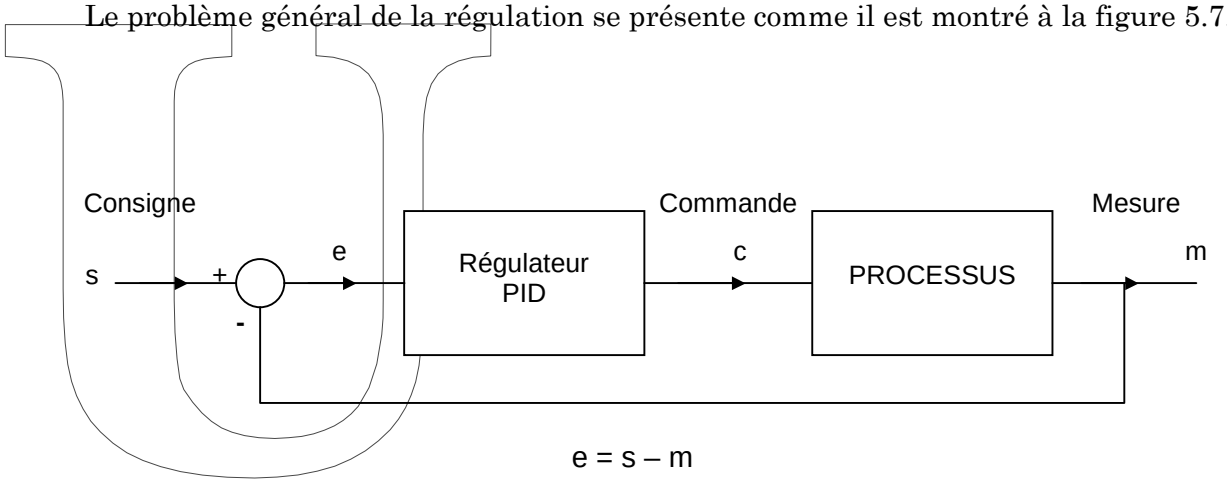


Figure 5.7. Principe d'une boucle de régulation

Le régulateur a pour mission de garder aussi petit que possible, dynamiquement et statiquement, l'écart entre la mesure d'une grandeur (vitesse, température, ...) et une valeur de consigne fixée par l'opérateur. Pour cela, il élabore une commande appropriée qui est appliquée au processus (courant d'un moteur, position d'une vanne, ...)

L'algorithme PID permet de répondre à la question dans la plupart des cas courants. Le signal de commande (c) est obtenu à partir de l'écart (e) entre mesure et consigne par la formule suivante :

$$c = K\left(e + \frac{1}{T_i}\int e\,dt + T_d\frac{de}{dt}\right) \qquad (1)$$

On y distingue trois termes :

- un terme proportionnel à l'erreur
- un terme intégral qui a pour but d'annuler l'erreur en régime
- un terme dérivé, c.-à-d. proportionnel aux variations de l'erreur, qui réalise une certaine anticipation.

Dans ce contexte, la mise en oeuvre du régulateur consiste à fixer la valeur des paramètres K, T_i, T_d.

Notons que la détermination de ces paramètres n'est pas évidente, des livres entiers lui sont consacrés ! Un des principaux problèmes qui se pose étant de garantir la stabilité du système bouclé régulateur-processus.

L'algorithme de type continu présenté ci-dessus n'est pas directement réalisable dans des systèmes digitaux tels que les ordinateurs ou les automates, car ceux-ci ne peuvent travailler que de manière discontinue.

On utilisera dès lors une approximation discontinue de l'algorithme de la forme suivante :

$$c_n = K\left[e_n + \frac{T}{T_i}\sum_{j=0}^{n} e_j + \frac{T_d}{T}(e_n - e_{n-1})\right] \qquad (2)$$

La commande c ne sera plus calculée qu'à des instants bien précis, $t_n = n \times T$, multiples d'une période d'échantillonnage T, qui devient un nouveau paramètre à choisir. Entre ces instants, la commande est maintenue constante.

Au lieu d'un signal c continu, on obtiendra donc un signal en escalier. Il est clair qu'en choisissant une période d'échantillonnage T suffisamment petite par rapport à la dynamique du processus, l'approximation sera tout à fait satisfaisante.

Remarquons d'ailleurs que dans le domaine continu, les dynamiques sont en général beaucoup plus lentes que dans le domaine logique que l'on a considéré jusqu'ici. C'est ainsi que les périodes d'échantillonnage nécessaires pour les régulateurs sont rarement inférieures à la seconde (contre ± 20 ms pour les automates logiques).

5.7.2. PROGRAMMATION DE L'ALGORITHME

Avec les automates qui possèdent les opérations sur mots (cfr. § 5.5), il est en principe possible de programmer l'algorithme PID en langage automate.
Il faut cependant noter que sur l'équation de base (2) se greffent un nombre plus ou moins important de fonctions de "service" destinées à :

- permettre un passage automatique/manuel;
- contrôler la valeur du terme intégral en mode manuel pour éviter les chocs lors du passage en automatique;
- mettre les régulateurs en cascade;
- contrôler la valeur de la mesure et de l'écart mesure - consigne par rapport à des seuils d'alarme et, le cas échéant, émettre les signaux d'alarme;
- ...

Il résulte de ceci que la programmation d'un PID n'est pas une chose aisée et qu'elle ne sera en général pas à la portée de l'utilisateur. Certains constructeurs ont réalisé eux-mêmes cette programmation et fournissent le listing correspondant à leurs clients qui n'ont, dès lors, plus qu'à le recopier. Il est bien clair cependant que cette solution est de portée très limitée et ne devrait s'utiliser que dans des cas tout à fait occasionnels de régulation.

Pour des problèmes de régulation plus importants, il est nécessaire de disposer de régulateurs PID préprogrammés en langage informatique plutôt qu'en langage automate. De plus, pour éviter de charger exagérément le processeur logique principal, il est préférable de recourir à des solutions multiprocesseurs avec un ou plusieurs processeurs spécialement réservés à la régulation.

5.7.3. MISE EN OEUVRE

La mise en oeuvre d'un régulateur PID préprogrammé comprend deux phases : la configuration du régulateur et le choix des paramètres de régulation. Ce dernier aspect

n'entre pas dans le cadre de cet ouvrage et nous renvoyons donc le lecteur à la littérature spécialisée.

En ce qui concerne la configuration du régulateur, elle se fait généralement à l'aide la console de programmation et d'un logiciel de dialogue spécifique.

Il s'agit en fait de choisir, parmi toutes les options prévues par le constructeur, celles qui sont nécessaires à l'application considérée.

Pour fixer les idées, la figure 5.8. montre la structure fonctionnelle de principe du régulateur PID proposé par SIEMENS et la figure 5.9, sa structure fonctionnelle détaillée.

L'utilisateur raisonne exclusivement sur cette représentation fonctionnelle tant à la configuration qu'à l'exploitation. Ainsi, les différentes options prévues par la constructeur seront-elles choisies en positionnant les "sélecteurs" S1 à S20 de la figure 5.9. (en réalité, il s'agit du positionnement de bits internes exploités par l'algorithme de calcul PID).

Sans essayer d'entrer dans tous les détails, on remarquera que ces sélecteurs permettent :

– de choisir le mode d'action du régulateur : continu, tout ou rien, pas-à-pas ("incrémental");
– les traitements effectués sur la mesure : contrôle de plausibilité, moyenne, linéarisation ("polygone");
– les traitements effectués sur la consigne : générateur de rampe, filtrage ("lissage").

Les "relais" RL1 à RL5, quant à eux, sont utilisés lorsque le régulateur est en service pour des commutations de mode tel le passage automatique/manuel (RL4). Ces "relais" peuvent être actionnés, soit à partir de la console, soit à partir du programme principal (logique) de l'automate.

Enfin, on remarquera encore des cercles comprenant l'inscription MP1 à MP12. Il s'agit de "points de mesure" situés aux endroits importants de la structure et qui peuvent être exploités en phase de test voire même en service normal. Ils peuvent en effet être "branchés" sur les détecteurs de seuils existant dans la structure (S17 et S18) ou sur des canaux analogiques de sortie (S19 et S20).

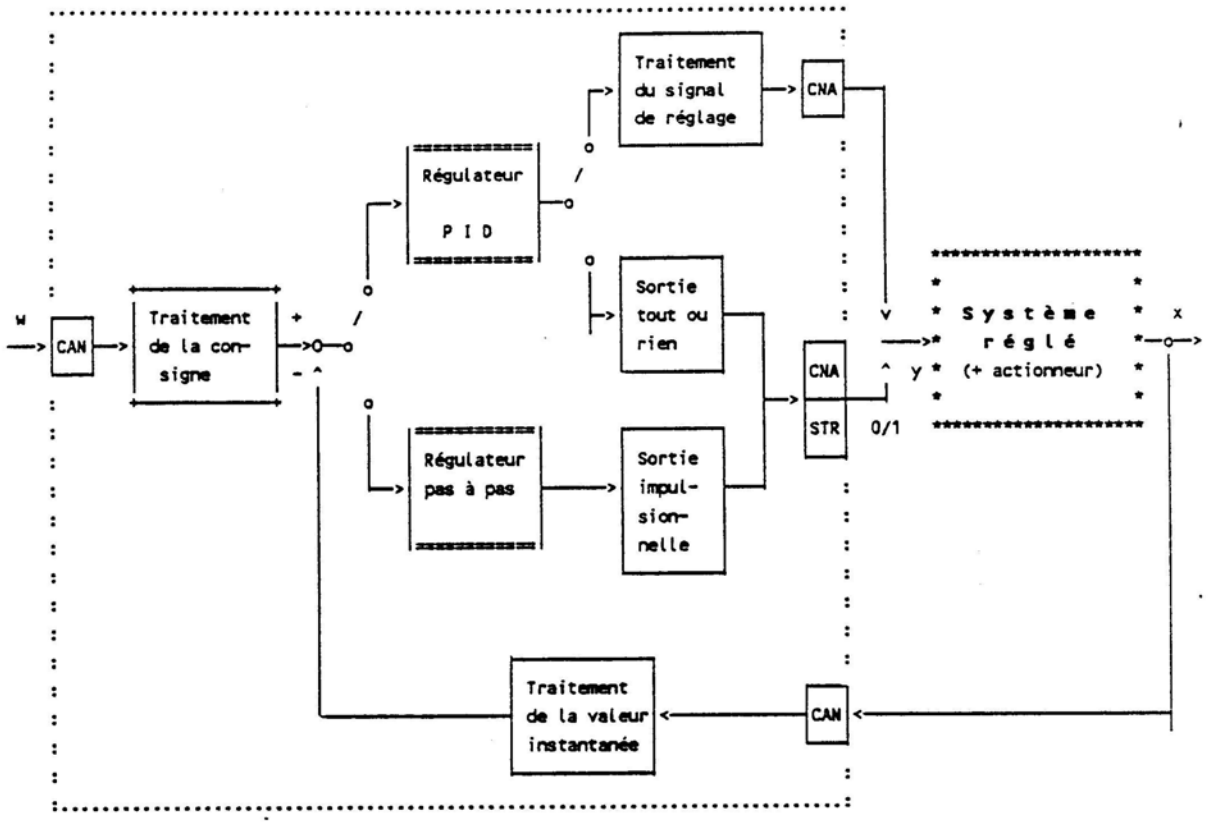


Figure 5.8. Structure de principe du régulateur PID
(Source SIEMENS)

5.8. COMMANDE D'AXE

Le positionnement d'un mobile le long d'un axe est une fonction très souvent rencontrée en automatisation de la production : machines de découpe, d'assemblage, d'emballage, lignes de transfert, systèmes de manutention, ...

La position est généralement mesurée par un codeur incrémental qui engendre des trains d'impulsions trop rapides pour pouvoir être directement saisis par un automate. Le recours à une carte de comptage rapide est donc normalement nécessaire.

Cependant, la commande d'axe est une fonction assez facile à standardiser si bien que la plupart des constructeurs d'automates proposent des processeurs de commande d'axe reprenant non seulement l'acquisition des impulsions du codeur, mais aussi le contrôle et l'enchaînement de mouvements usuels.

Pour fixer les idées, on considérera l'exemple du produit TELEMECANIQUE.

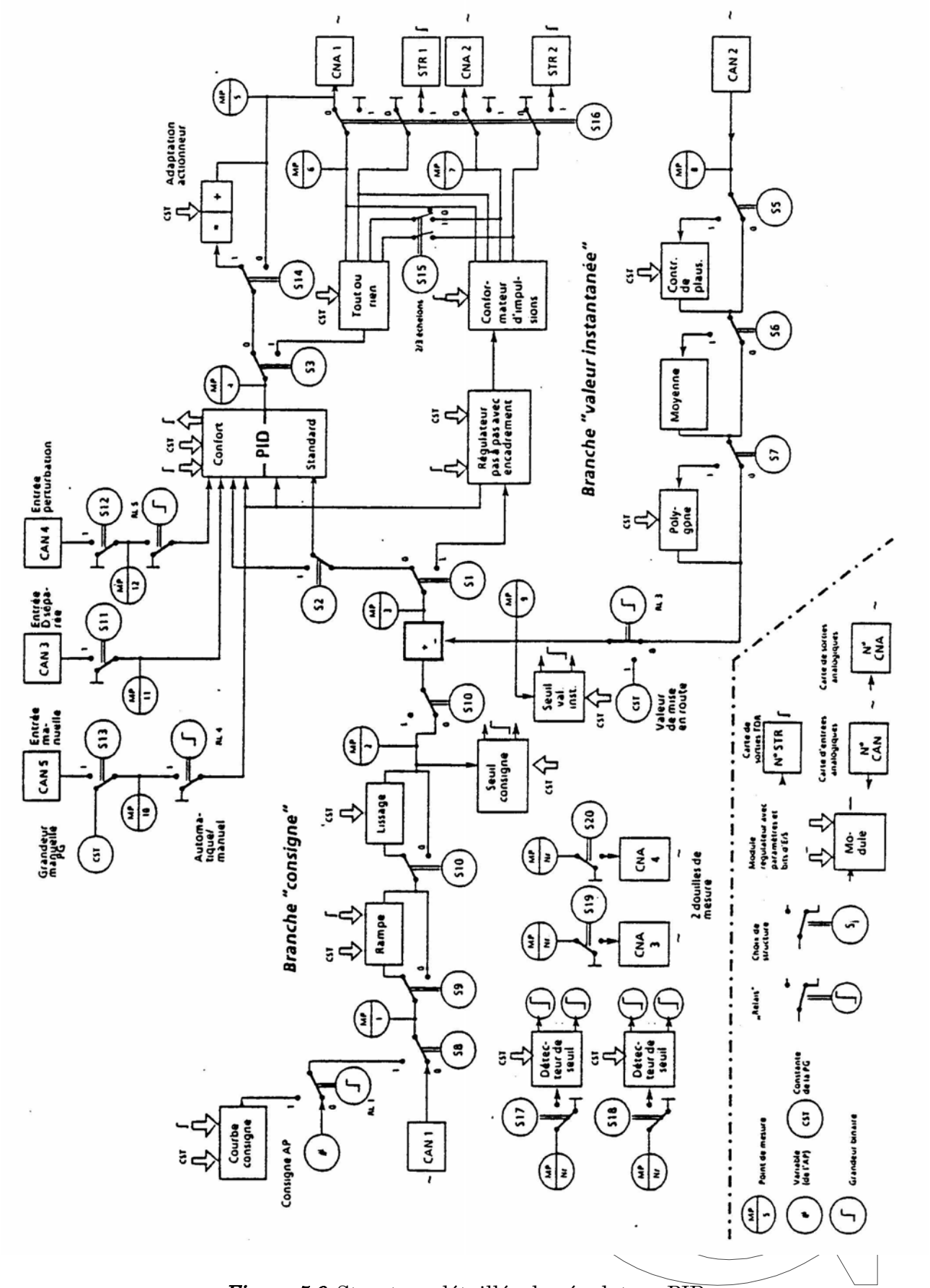


Figure 5.9. Structure détaillée du régulateur PID
(Source SIEMENS)

5.8.1. PRINCIPE DE FONCTIONNEMENT

La figure 5.10. schématise le fonctionnement du module de commande d'axe. On remarque que ce module ne prend en charge que l'asservissement de position tandis que l'asservissement de vitesse est assuré par un dispositif extérieur, le "variateur", qui réalise aussi les amplifications de puissance nécessaires.

Les caractéristiques de la boucle d'asservissement sont déterminées par le gain de position KP et le coefficient d'anticipation de vitesse KV. Ces coefficients doivent être ajustés par l'utilisateur dans chaque cas particulier (figure 5.10.b).

Des limitations d'accélération et de décélération peuvent être introduites dans la boucle. Elles sont modifiables par programme (figure 5.10.c).

Le module de commande d'axe est, d'autre part, doté d'un jeu d'instructions simple et complet, orienté application, qui permet de décrire des trajectoires, de les enchaîner et d'assurer la coordination avec le processeur principal de l'automate.

5.8.2. PROGRAMMATION

Les instructions évoquées ci-dessus peuvent être rangées en instructions de déplacement, instructions d'organisation des mouvements et instructions de gestion des paramètres.

- déplacement (figure 5.11.a.)
  - déplacement jusqu'à une position avec arrêt (code GP9) ou sans arrêt (code GP1). Dans ce dernier cas, le programme passe à l'instruction suivante lorsque la position est atteinte. Les positions peuvent être absolues ou indexées.
    Dans chacun de ces cas, les valeurs peuvent être :
    - immédiates (contenues dans l'instruction)
    - obtenues par apprentissage (cfr. § 5.8.3.)
    - communiquées en temps réel par le programme principal de l'automate.
  - déplacement jusqu'à l'arrivée d'un événement suivi ou non d'un arrêt.

- organisation des mouvements (figure 5.11.b.)
  - appel à des sous-programmes
  - saut conditionnel ou inconditionnel
  - attente d'événement
  - temporisation

- gestion des paramètres
  Elle permet des opérations de calcul et l'affectation de nouvelles valeurs aux positions, aux compteurs internes et aux paramètres caractéristiques des contrôles.

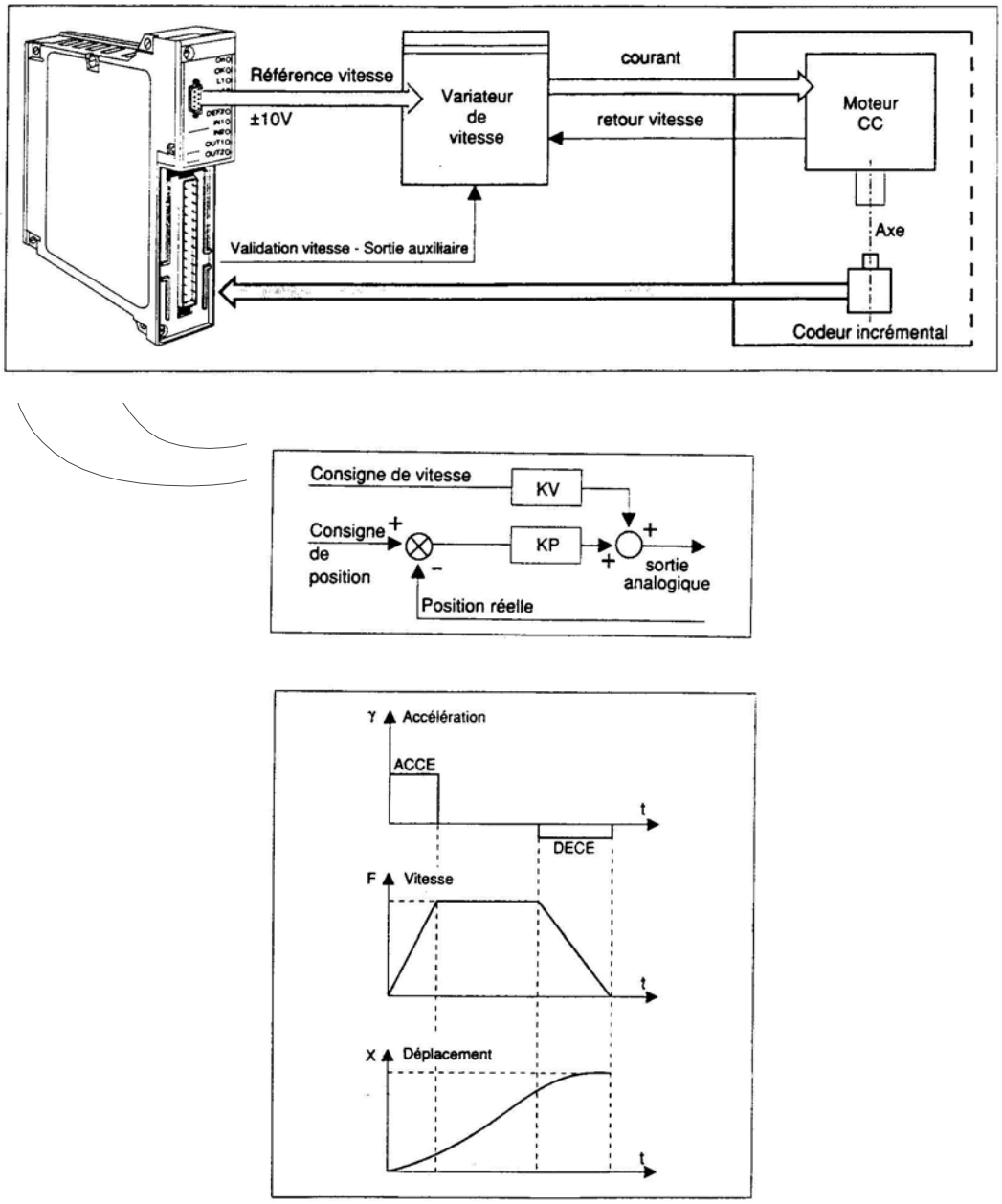


Figure 5.10. Principe d'une commande d'axe
a. Structure
b. Asservissement de position
c. Limitation d'accélération et de décélération

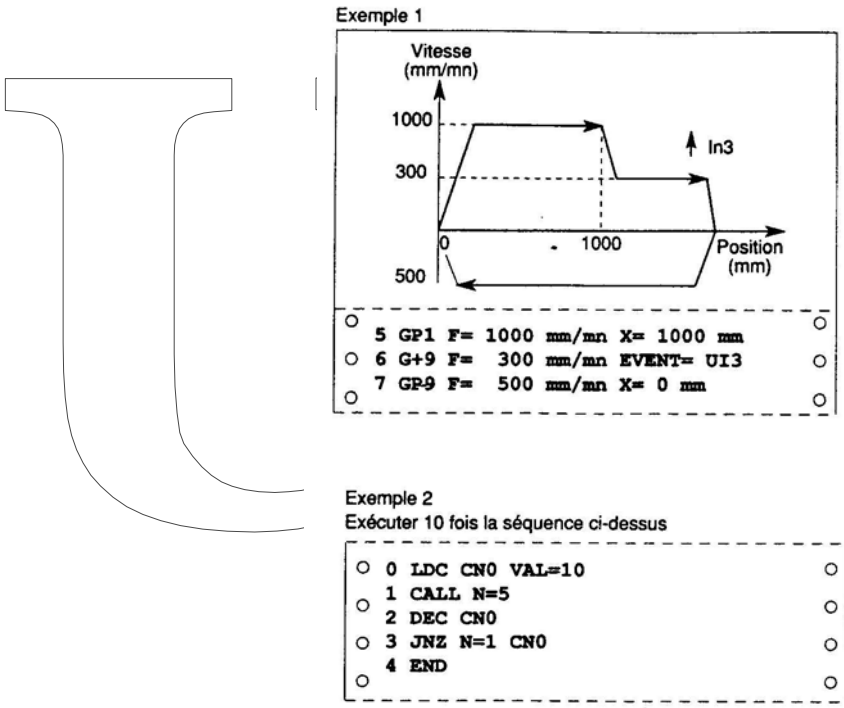


Figure 5.11. Programmation d'une commande d'axe
a. Déplacements
b. Organisation des mouvements

5.8.3. TERMINAL D'EXPLOITATION

Des terminaux d'exploitation spécifiques sont souvent adjoints aux cartes de commande d'axe (cfr. figure 5.12).

Ils ont pour objet :

– la sélection des modes de marche

– en mode de mise au point :
  - la modification des paramètres
  - l'apprentissage de positions et la prise d'origine
  - le test des programmes (instruction par instruction, ...)

– en fonctionnement automatique :
  - la visualisation des paramètres et du programme
  - l'observation des déplacements : position et vitesse

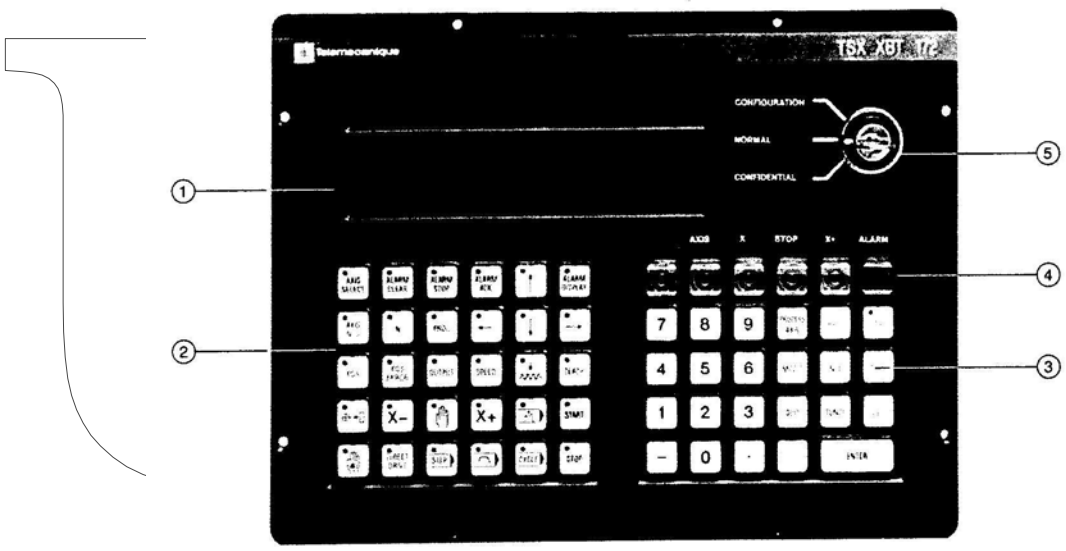


Figure 5.12. Commande d'axe. Exemple de terminal d'exploitation

Chapitre 6

GRAFCET ou SFC

GRAFCET : Graphe de Commande Etape-Transition
SFC : Sequential Function Chart

6.1. PRINCIPES GENERAUX

6.1.1. HISTORIQUE

Au départ, le GRAFCET a été conçu comme un outil méthodologique de description du cahier des charges de la partie commande d'un système automatisé.

Il résulte de travaux entrepris dès 1975 sous l'égide de l'AFCET (Association Française pour la Cybernétique Economique et Technique) par un groupe d'industriels et de chercheurs.
L'ADEPA (Agence nationale pour le DEveloppement de la Production Automatisée) s'est ensuite attachée à sa promotion tout en lui conférant une forme susceptible d'aboutir à des normes nationales et internationales.

De fait, le GRAFCET a été normalisé en France en 1982, mais il a fallu attendre 1988 pour le voir reconnaître sur le plan international par la norme 848 édictée par la CEI (Commission Electrotechnique Internationale). Pour l'occasion, il a d'ailleurs été rebaptisé SFC (Sequential Function Chart)

Entre temps, l'outil méthodologique de description, le GRAFCET ou SFC est devenu un véritable langage de programmation. Comme tel, il est repris dans la récente norme CEI 1131-3 (1993) relative aux langages de programmation pour automates programmables.

On peut dire que le GRAFCET, en tant qu'outil de description fonctionnelle, a été unanimement adopté par les industriels, du moins en Europe occidentale.
Par contre, curieusement, son utilisation comme langage de programmation n'a rencontré, jusqu'ici, qu'un succès mitigé. Il y a sans doute plusieurs raisons à ces réticences :

– la normalisation encore trop récente

– le manque de convivialité des premiers éditeurs de GRAFCET.
Ce problème devrait s'atténuer avec le remplacement généralisé des consoles de programmation par des PC sous WINDOWS. Il n'empêche que la taille relativement réduite des écrans standards actuels (14") reste un obstacle sérieux à une programmation graphique confortable.

– l'absence de normalisation au niveau de la sémantique du GRAFCET, c'est-à-dire au niveau de son implantation effective dans les automates.
Chaque constructeur développe dès lors sa propre philosophie, laquelle n'est en général pas connue de l'utilisateur. Il peut en résulter des ambiguïtés conduisant à des réactions

imprévisibles et potentiellement dangereuses du système dans des situations limites (parallélisme et synchronisation notamment).

Le résultat est que l'énorme majorité des programmeurs recourt à la traduction manuelle du GRAFCET en langage automate (cf. figure 4.3.).

Nous examinerons au paragraphe 6.6. les différentes méthodes couramment utilisées.

Remarque

L'essentiel des notions présentées ci-après sont tirées de la brochure de l'ADEPA consacrée au GRAFCET, de la norme CEI 1131-3 et de [BOSSY - 1979].

6.1.2. APPROCHE PROGRESSIVE DU CAHIER DES CHARGES DE LA PARTIE COMMANDE

L'automaticien chargé de la conception et de la réalisation de la partie commande doit rechercher dans le cahier des charges une description claire, précise, sans ambiguïtés ni omission, du rôle et des performances de l 'équipement à réaliser.

Pour y parvenir, il est souhaitable de diviser la description en deux niveaux successifs et complémentaires :

– le premier niveau décrit le comportement de la partie commande vis-à-vis de la partie opérative : c'est le rôle des spécifications fonctionnelles permettant au concepteur de comprendre ce que l'automatisme doit faire, face aux différentes situations pouvant se présenter.

– le deuxième niveau ajoute aux exigences fonctionnelles les précisions indispensables aux conditions de fonctionnement des matériels, grâce aux spécifications technologiques et opérationnelles.

En sériant les problèmes, fonctionnels d'un côté, technologiques de l'autre, cette approche évite au lecteur de se sentir submergé d'emblée sous une foule de détails plus nuisibles qu'utiles.

Niveau 1 - Spécifications fonctionnelles

Les spécifications fonctionnelles caractérisent les réactions de l'automatisme face aux informations issues de la partie opérative, dans le but de faire comprendre au concepteur quel devra être le rôle de la partie commande à construire. Elles doivent donc définir de façon claire et précise les différentes fonctions, informations et commandes impliquées dans l'automatisation de la partie opérative, sans préjuger en aucune façon des technologies.

En conséquence, ni la nature ni les caractéristiques des différents capteurs ou actionneurs utilisés n'ont leur place dans ces spécifications. Peu importe, à ce niveau, que l'on effectue un déplacement à l'aide d'un vérin hydraulique ou pneumatique, ou encore d'un moteur électrique. Ce qu'il faut savoir c'est dans quelles circonstances ce déplacement doit s'effectuer.

Par contre, il importe que les sécurités de fonctionnement prévues soient incorporées dans les spécifications fonctionnelles, dans la mesure où elles ne dépendent pas directement de la technologie de ces capteurs ou actionneurs.

Niveau 2 - Spécifications technologiques

Les spécifications technologiques précisent la façon dont l'automatisme devra physiquement s'insérer dans l'ensemble que constitue le système automatisé et son environnement. Ce sont les précisions à apporter en complément des spécifications fonctionnelles pour que l'on puisse concevoir un automatisme pilotant réellement la partie opérative.

C'est à ce niveau seulement que doivent intervenir les renseignements sur la nature exacte des capteurs et actionneurs employés, leurs caractéristiques et les contraintes qui peuvent en découler. A ces spécifications d'interface peuvent également s'ajouter des spécifications d'environnement de l'automatisme: température, humidité, poussières, anti-déflagrance, tensions d'alimentation, etc.

Spécifications opérationnelles

Les spécifications opérationnelles ont trait au suivi de fonctionnement de l'automatisme au cours de son existence. Il s'agit là des considérations concernant l'équipement une fois réalisé et mis en exploitation : fiabilité, absence de pannes dangereuses, disponibilité, possibilités de modification de l'équipement en fonction des transformations de la partie opérative, facilité de maintenance, dialogue homme - machine,

Ces considérations, primordiales pour l'exploitant du processus à automatiser en raison de leurs répercussions sur le plan économique, sont souvent sous-estimées dans les cahiers des charges. Parfois difficiles à exprimer de façon quantitative, elles n'en ont pas moins d'incidence sur la manière de réaliser l'équipement.

6.1.3. EXEMPLE INTRODUCTIF

Le GRAFCET est un outil de description du cahier des charges de la partie commande du système automatisé utilisable tant au niveau 1 qu'au niveau 2.

Le fonctionnement de l'automatisme peut être représenté graphiquement par un ensemble :

– d'**ETAPES** auxquelles sont associées des **ACTIONS**

– de **TRANSITIONS** auxquelles sont associées des **RECEPTIVITES**

– de **LIAISONS ORIENTEES** reliant les étapes aux transitions
et les transitions aux étapes

Avant d'introduire ces concepts et leur représentation nous allons montrer leur importance ainsi que leur signification pratique à partir d'un exemple simplifié.

On se propose d'étudier l'automatisation d'une presse destinée à la fabrication de pièces à partir de poudres comprimées.

Découpage Partie Opérative - Partie Commande

La partie opérative représentée schématiquement à la figure 6.1. se compose

- d'un poinçon inférieur fixe C
- d'un poinçon supérieur A et d'une matrice B mobiles
- d'un sous-ensemble de mise en place de la matière
- d'un sous-ensemble d'évacuation de la pièce comprimée

Fonctionnement général du système

Le cycle de travail est le suivant :

– la matrice étant en haut de sa course, le poinçon inférieur qui y demeure engagé, délimite au-dessus de lui un espace suffisant pour recevoir la matière à comprimer. Le poinçon supérieur est alors dans sa position la plus haute ce qui dégage la partie supérieure de la matrice et permet l'introduction de la matière.

– quand la matière pulvérulente est en place, le poinçon supérieur descend, comprime la matière en pénétrant dans la matrice puis remonte en position haute.

– la matrice descend alors jusqu'à ce que le poinçon inférieur affleure, ce qui libère la pièce qui vient d'être comprimée. Cette pièce peut ensuite être évacuée.

– enfin la matrice reprend sa place et un nouveau cycle peut alors commencer.

Ces actions ne pourront être obtenues que si la partie commande émet les ordres convenables au moment voulu.

Les moments voulus seront déterminés d'après les compte rendus ou informations provenant de la partie opérative.

Etude de la partie commande - GRAFCET de niveau 1

Considérons la presse arrêtée dans l'attente d'une nouvelle charge de matière. La matrice et le poinçon sont immobiles et la descente de ce dernier ne sera commandée par l'automate qu'après la réception de l'information "matière en place". Cependant, cette même information, si elle est renouvelée par erreur pendant la remontée du poinçon, n'aura aucun effet sur le comportement de la partie commande. Nous dirons que l'automate était "réceptif" dans le premier cas pour l'information "matière en place" et qu'il ne l'était plus dans le second.

Nous dirons que la partie commande demeure dans une "étape" tant que son comportement est constant. Elle reste dans cette "étape" jusqu'à ce que les informations pour lesquelles elle est "réceptive" provoquent le franchissement d'une "transition" conduisant à une nouvelle étape où la partie commande adoptera alors un nouveau comportement.

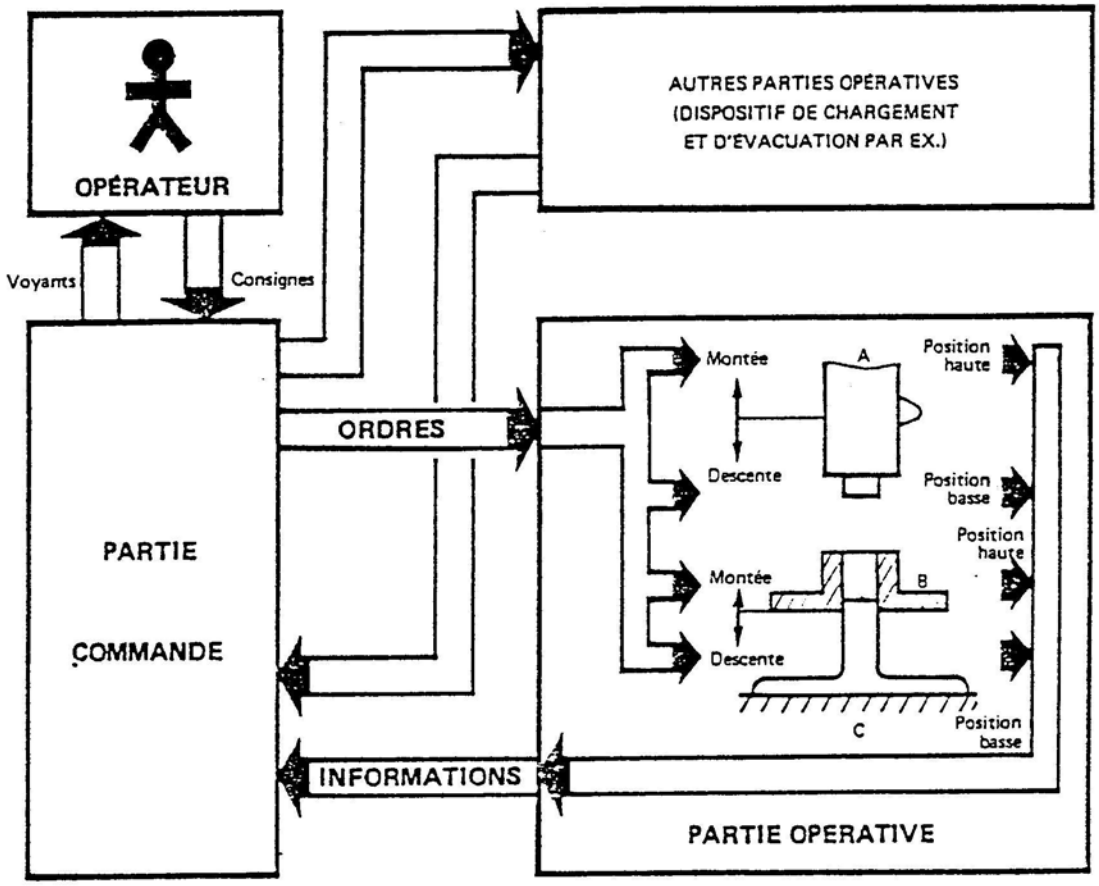


Figure 6.1. Exemple d'une presse de compression de poudres

Nous pouvons maintenant représenter le fonctionnement d'une partie commande comme une **succession alternée** d'étapes et de transitions.

En conséquence nous associerons :

– à chaque étape, les actions à effectuer

– à chaque transition, les informations permettant leur franchissement, sous forme d'une condition logique appelée réceptivité.

De cette manière le fonctionnement de la partie commande nécessaire à la presse sera décrit ainsi:

Etape 1	;	action	:	mise en place de la matière
Transition 1-2	;	réceptivité	:	matière en place et départ cycle
Etape 2	;	action	:	descente du poinçon
Transition 2-3	;	réceptivité	:	fin de compression
Etape 3	;	action	:	remontée du poinçon
Transition 3-4	;	réceptivité	:	poinçon en haut
Etape 4	;	action	:	descente matrice
Transition 4-5	;	réceptivité	:	matrice en bas
Etape 5	;	action	:	évacuation de la pièce comprimée
Transition 5-6	;	réceptivité	:	pièce évacuée
Etape 6	;	action	:	remontée matrice
Transition 6-1	;	réceptivité	:	matrice en haut

Il s'avère plus commode de représenter ce fonctionnement sous forme graphique d'où le GRAFCET de la figure 6.2., illustré à des fins de compréhension des états correspondants de la partie opérative.

Aux actions ci-dessus, strictement nécessaires au fonctionnement de la presse, pourraient s'ajouter d'autres actions vers le monde extérieur telles l'allumage de voyants, etc.. (par exemple "appel de l'opérateur pour évacuer la pièce" dans l'étape 5).

Enfin notons que nous avons attribué un rôle particulier à l'une des étapes: l'étape initiale. Le choix de cette étape est imposé par des considérations fonctionnelles liées à la partie opérative.

GRAFCET de niveau 2

Le GRAFCET que nous venons d'établir est un GRAFCET de niveau 1 car il ne prend en compte que l'aspect fonctionnel sans aucune implication technologique : par exemple nous ne savons pas comment physiquement donner l'ordre de descente au poinçon, ni comment on s'assure que la pièce est évacuée.

Il convient maintenant de préciser les choix technologiques des actionneurs et des capteurs (figure 6.3.a.) :

– cet exemple étant simplifié la procédure d'arrêt d'urgence, les modes de marche ainsi que les sécurités ne sont pas traités.

– la mise en place de la matière est assurée manuellement par l'opérateur. Un voyant V est allumé pendant toute la durée de la mise en place. Celle-ci terminée l'opérateur autorise la poursuite des opérations en appuyant sur un bouton-poussoir d.

– les mouvements du poinçon supérieur et de la matrice sont effectués au moyen de vérins hydrauliques double-effet. Les positions haute et basse du poinçon et de la matrice sont contrôlées à l'aide de capteurs de fin de course (respectivement: a0 et a1, b1 et b0).

– l'évacuation de la pièce est obtenue au moyen d'un jet d'air maintenu pendant une seconde. Ce jet d'air est commandé par une électrovanne E.

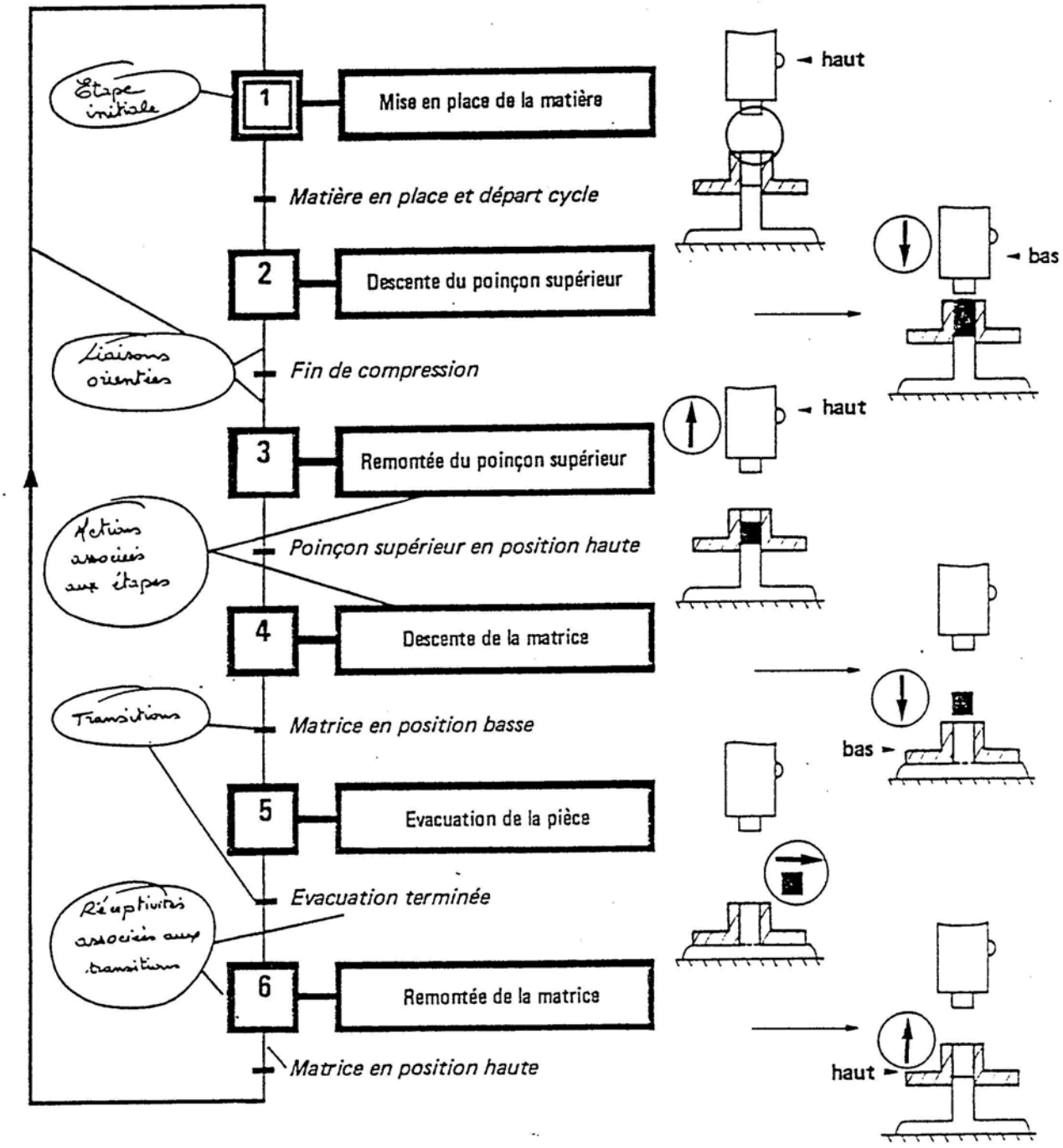


Figure 6.2. GRAFCET de niveau 1 de la presse

La liste ci-après rappelle les variables introduites ainsi que leur signification respective. Il est commode en pratique de les présenter sous forme d'un tableau des informations et des actions de la partie commande (figure 6.3.b.).

ORDRES

vers le milieu extérieur et l'opérateur

V	:	voyant "Prêt"

commande des actionneurs

a +	:	descente poinçon
a –	:	remontée poinçon
b –	:	descente matrice
b +	:	remontée matrice
E	:	évacuation

lancement de temporisations

LT1	:	lancement temporisation d'évacuation

INFORMATIONS

déroulement du cycle

d	:	autorisation de départ cycle

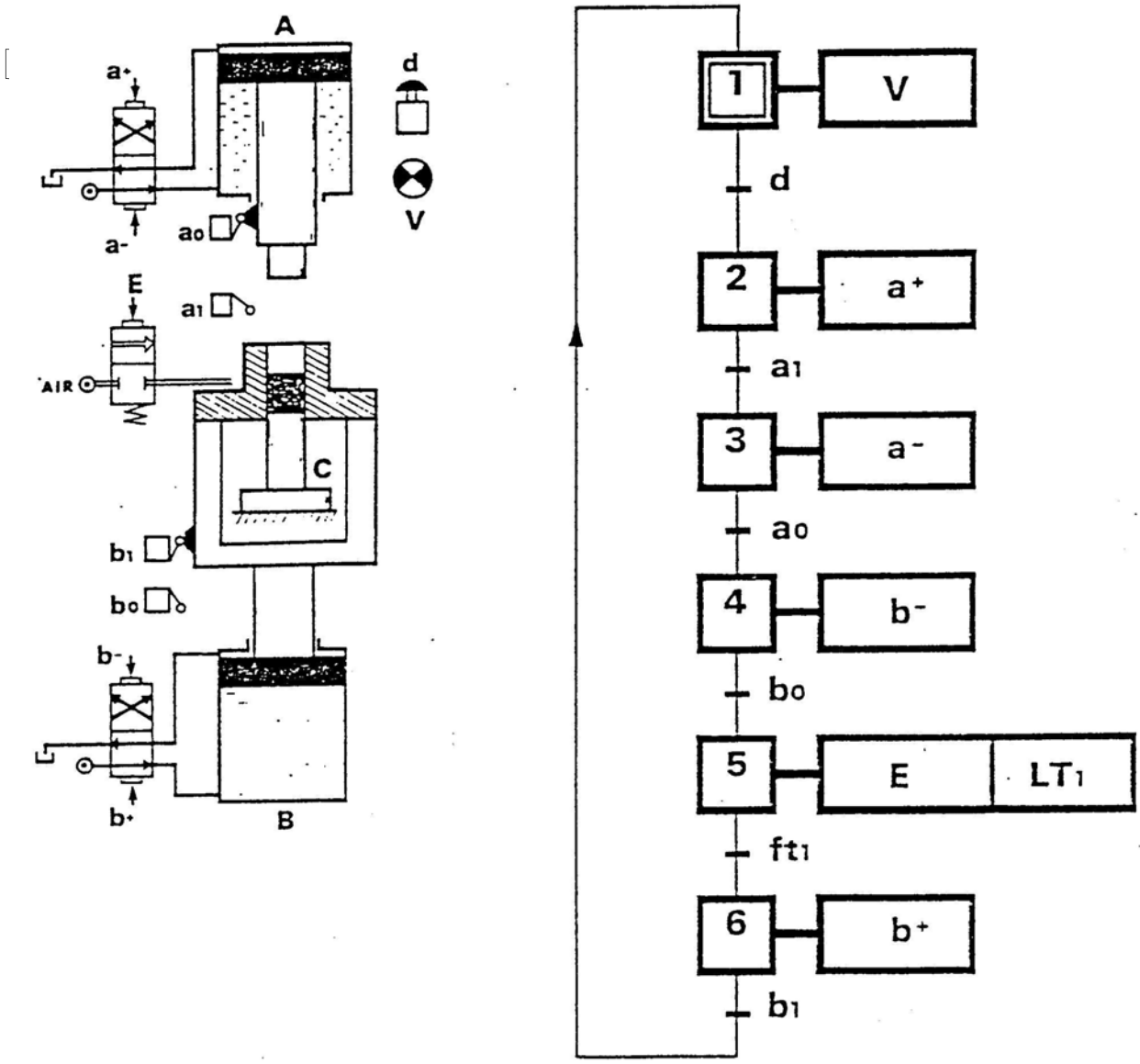
fin de course des actionneurs

a1	:	position basse du poinçon
a0	:	position haute du poinçon
b0	:	position basse de la matrice
b1	:	position haute de la matrice

fin de temporisation

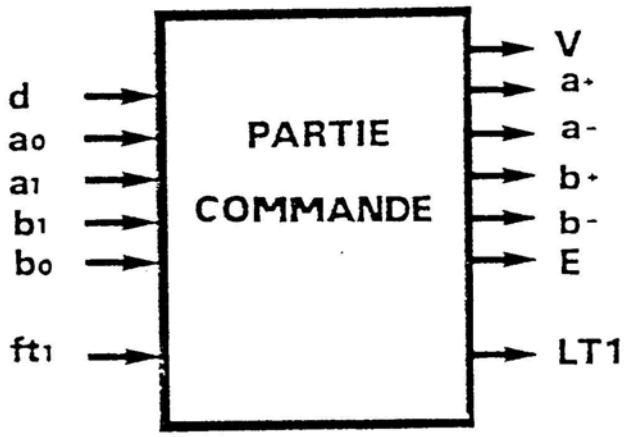
ft1	:	fin de temporisation d'évacuation

On peut alors établir, pour la partie commande, le GRAFCET de niveau 2 montré à la figure 6.3.c.



a. Choix technologiques

c. GRAFCET niveau 2



b. Mise en évidence des variables

Figure 6.3. GRAFCET de niveau 2 de la presse

6.2. ELEMENTS DE BASE DU GRAFCET

L'exemple simplifié précédent a permis de présenter de manière intuitive les trois concepts fondamentaux du GRAFCET : étape - transition - liaisons orientées. Nous allons maintenant en donner des définitions plus précises.

6.2.1. ETAPE

Une ETAPE correspond à une situation dans laquelle le comportement de tout ou partie du système par rapport à ses entrées et ses sorties est invariant.

L'ETAPE se représente par un carré ou un rectangle repéré numériquement, le repère étant placé à la partie supérieure (figure 6.4.a.)

En addition à ce repère, un nom symbolique peut être adjoint, représentatif de la fonction principale de l'étape (ex: ATTENTE, FIN, SYNCHRONISATION, etc.) (figure 6.4.b.)

Une étape est soit active soit inactive et à un instant donné la situation du système automatisé est entièrement définie par l'ensemble des étapes actives. On précise pour chaque étape les actions à effectuer caractéristiques de cette situation. Ces actions ne sont effectives que lorsque l'étape est active.

Il est commode de montrer les étapes actives à un instant bien précis en plaçant un point ou un repère quelconque dans la partie inférieure des symboles correspondants (figure 6.4.c.).

Au niveau des spécifications fonctionnelles, on ne définit pas les actionneurs ni les capteurs, mais uniquement les actions à effectuer et leur enchaînement.

Les actions à effectuer lorsque l'étape est active sont décrites de façon littérale ou symbolique à l'intérieur d'un ou plusieurs rectangles de dimensions quelconques reliés à la partie droite de l'étape (figure 6.4.d.)

De plus l'exécution de ces actions peut être soumise à d'autres conditions logiques, fonction de variables d'entrée, de variables auxiliaires ou de l'état actif ou inactif d'autres étapes.

Dans l'exemple de la figure 6.4.e., lorsque l'étape 27 est active, il faut :

- allumer L1 si DEF est présent
- allumer L4 si PP est absent
- fermer la trappe n° 2 si l'étape 15 est active (X 15 = 1)
- lancer une temporisation de 10 secondes
- etc.

Au niveau des spécifications technologiques (niveau 2), on devra préciser la façon dont les actions sont réalisées compte tenu du matériel défini pour les capteurs et les actionneurs. Pour ce faire, on utilisera l'un ou l'autre des langages de programmation fourni par le constructeur (relais, liste d'instructions, ...).

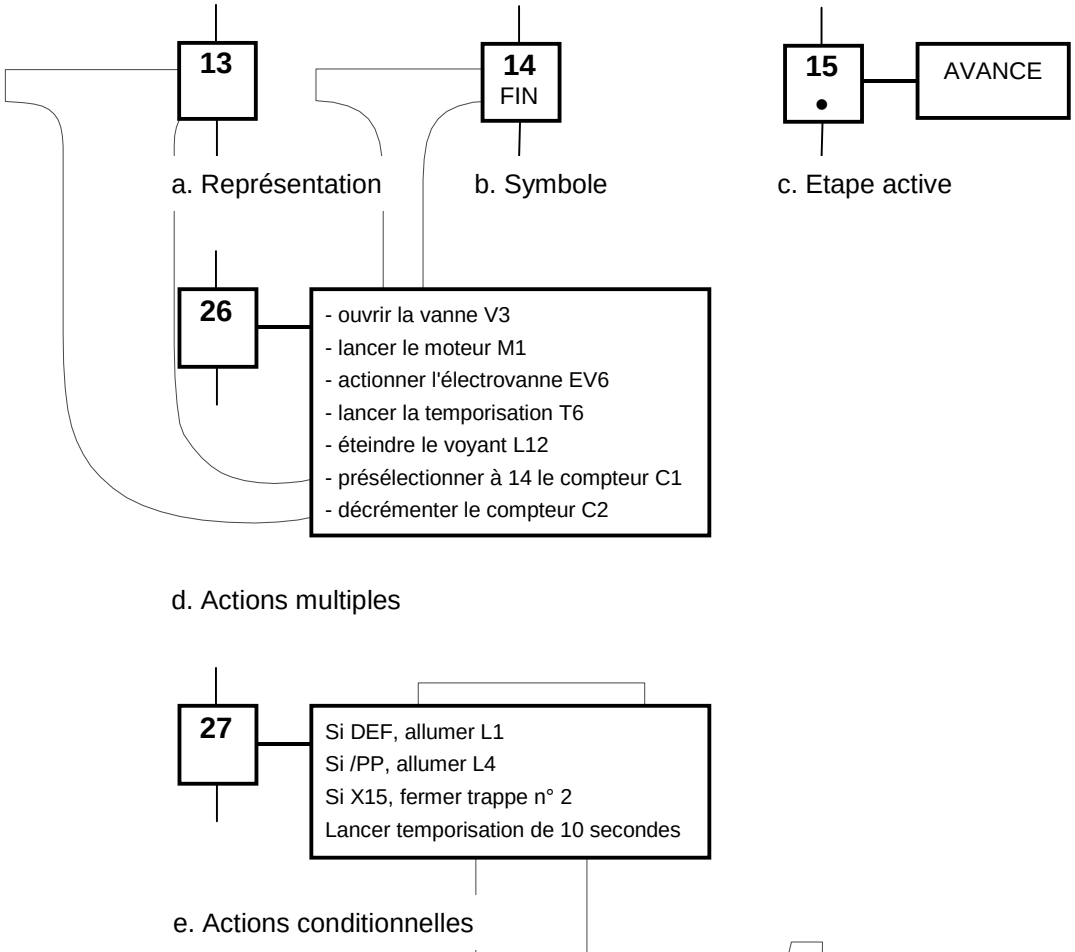


Figure 6.4. Etapes du GRAFCET

6.2.2. TRANSITIONS

Les TRANSITIONS indiquent les possibilités d'évolution entre étapes. On associe à chaque transition une condition logique appelée RECEPTIVITE qui permet de distinguer, parmi toutes les informations disponibles, uniquement celles qui sont susceptibles à un instant donné de faire évoluer la partie commande.

La RECEPTIVITE, écrite sous forme de proposition logique, est une fonction combinatoire d'informations extérieures (directives de l'opérateur, états de capteurs, de compteurs, de temporisateurs, etc.), de variables auxiliaires ou de l'état actif ou inactif d'autres étapes (figure 6.5.a.)

Ces réceptivités peuvent s'exprimer sous des formes diverses :

Ex :
- position droite d'un mobile
- fin de course de trappe ouverte actionné
- température > 300°C
- valeur du compteur C 10 > valeur de consigne
- on a appuyé 3 fois sur le bouton M, etc.

Les réceptivités peuvent aussi faire intervenir des changements d'état de variables.

La notation a↑ représente le front "montant" de la variable a (passage de l'état logique "0" à l'état logique "1") et la notation y↓ représente le front "descendant" de la variable y (passage de l'état logique "1" à l'état logique "0") (figure 6.5.b.)

Pour faire intervenir le temps dans une réceptivité, il suffit d'indiquer après le repère t son origine et sa durée, l'origine sera l'instant du début de la dernière activation d'une étape antérieure.

Ex : la notation t/8/l0s signifie 10 secondes écoulées depuis la dernière activation de l'étape 8.

Nota : Une réceptivité toujours vraie est écrite = 1.

6.2.3. LIAISONS ORIENTEES

Les liaisons indiquent les voies d'évolution de l'état du GRAFCET. Les liaisons sont horizontales ou verticales sauf dans des cas isolés où des traits obliques apporteraient de la clarté au diagramme. L'essentiel est d'adopter une représentation qui contribue au mieux à la clarté du fonctionnement (Cf. norme NFZ 67-010).

Le sens général de parcours est du haut vers le bas. L'arrivée et le départ sur une étape sont représentés verticalement, l'arrivée étant à la partie supérieure. Si dans des cas très particuliers, l'arrivée devait être faite à la partie inférieure une flèche serait obligatoire.

Des flèches doivent être utilisées chaque fois qu'une meilleure compréhension pourra en résulter et chaque fois que l'orientation fixée n'est pas respectée. Pour éviter toute ambiguïté, il est préférable d'éviter les croisements continus des lignes de liaison.

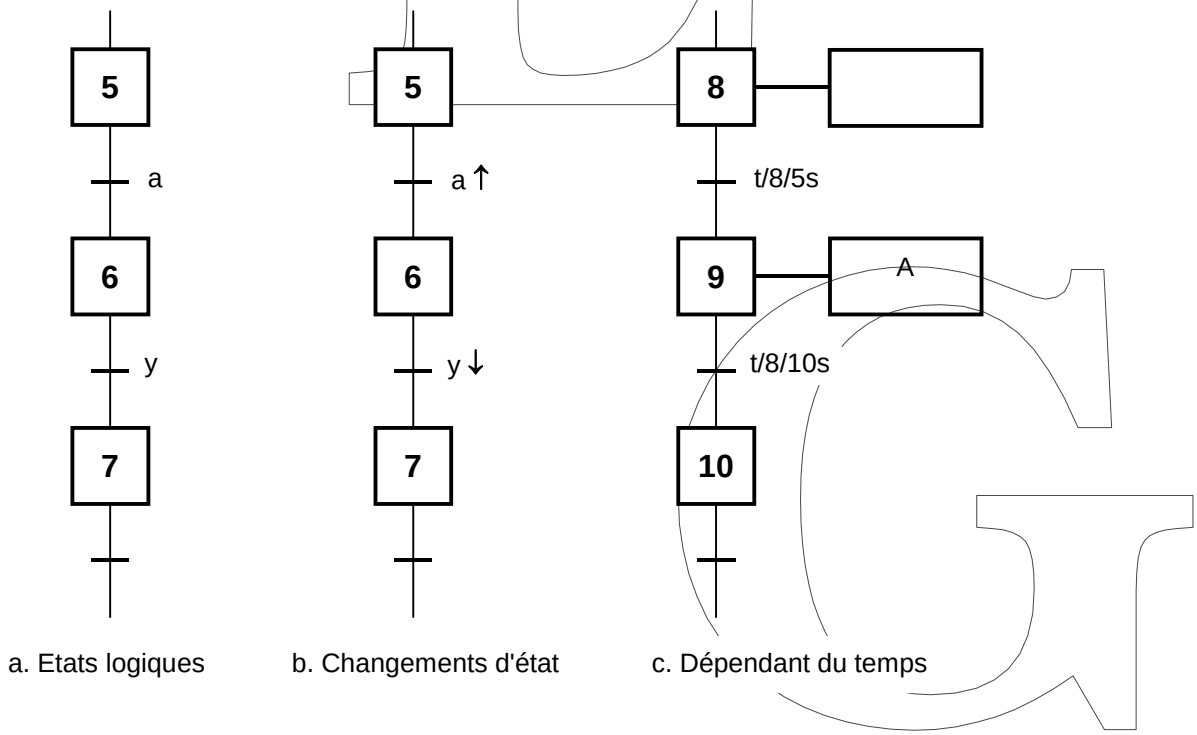


Figure 6.5. Transitions et réceptivités

6.2.4. REGLES D'EVOLUTION

Le caractère actif ou inactif de chacune des étapes devant évoluer, les trois concepts précédents ne peuvent suffire à définir un GRAFCET. Il est en plus nécessaire de fixer un ensemble de règles d'évolution.

Règle 1

L'INITIALISATION précise les étapes actives au début du fonctionnement. Elles sont activées inconditionnellement et repérées sur le GRAFCET en doublant les côtés des symboles correspondants (figure 6.6.a.)

Règle 2

Une TRANSITION est soit validée, soit non validée Elle est validée lorsque TOUTES les étapes immédiatement précédentes sont actives. Elle ne peut être franchie que :

- lorsqu'elle est validée,
- ET que la réceptivité associée à la transition est vraie.

La TRANSITION est alors obligatoirement franchie (figure 6.6.b.)

Règle 3

Le franchissement d'une TRANSITION entraîne l'activation de TOUTES les étapes immédiatement suivantes et la désactivation de TOUTES les étapes immédiatement précédentes.

Exemple : Cas de transition entre plusieurs étapes (figure 6.6.c.)

Lorsque plusieurs étapes sont reliées à une même transition on convient, pour des raisons pratiques, de représenter le regroupement de liaisons par deux traits parallèles (cf. normes NFZ 67.010 - ISO 1028).

Règle 4

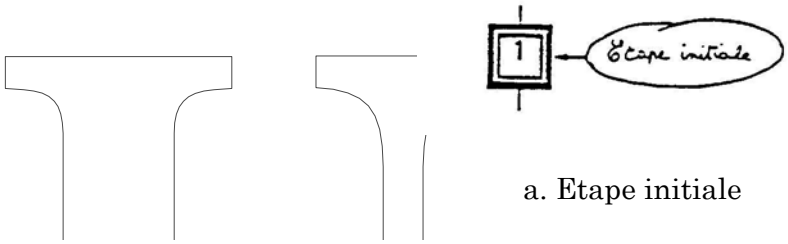
Plusieurs transitions simultanément franchissables sont simultanément franchies.

Règle 5

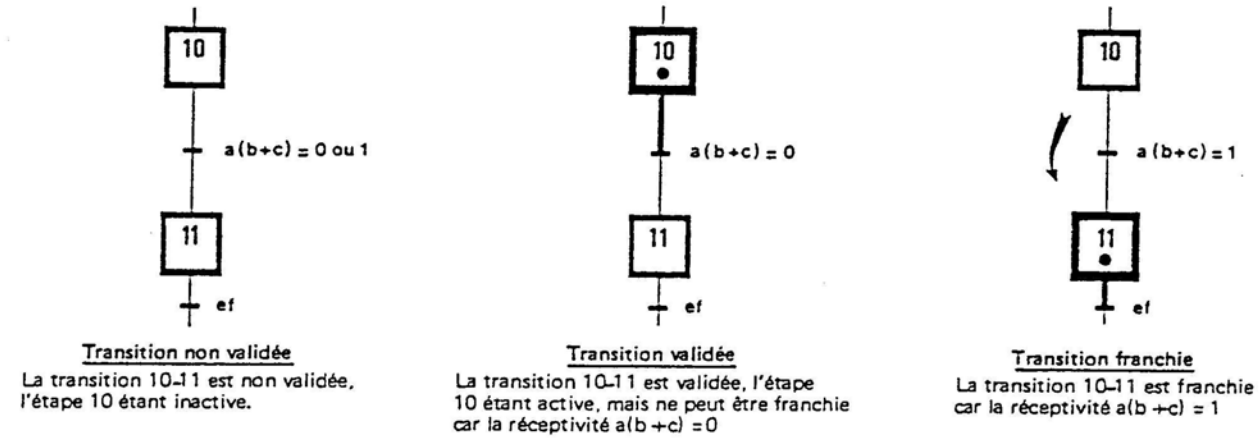
Si au cours du fonctionnement une même étape doit être désactivée et activée simultanément, elle reste activée.

Note importante :

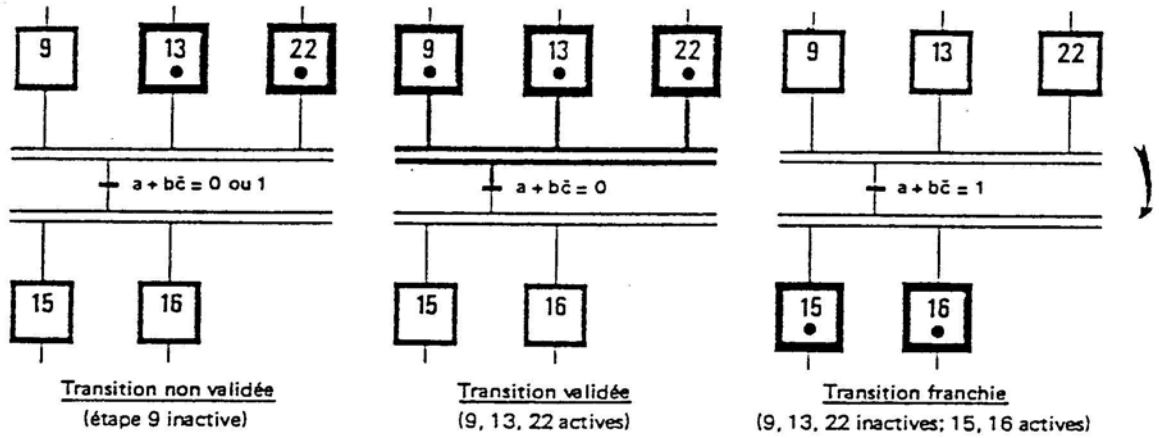
La durée de franchissement d'une transition ne peut jamais être rigoureusement nulle, même si, théoriquement (règles 3 et 4) elle peut être rendue aussi petite que l'on veut. Il en est de même de la durée d'activation d'une étape. En outre, la règle 5 se rencontre très rarement dans la pratique. Ces règles ont été ainsi formulées pour des raisons de cohérence théorique interne au GRAFCET.



a. Etape initiale



b. Franchissement d'une transition



c. Transition entre plusieurs étapes

Figure 6.6. Règles d'évolution du GRAFCET

6.2.5. REPRESENTATION DES SEQUENCES MULTIPLES

Les aiguillages :

Choix conditionnel entre plusieurs séquences

Un GRAFCET est généralement constitué de plusieurs séquences, c'est-à-dire de plusieurs suites d'étapes à exécuter les unes après les autres et il est souvent nécessaire d'effectuer une sélection exclusive d'une parmi ces séquences.

La figure 6.7.a. en donne un exemple.

Dans l'aiguillage formé par le choix de la séquence à réaliser, les différentes transitions correspondant aux réceptivités x, y et z étant simultanément validées par la même étape 5 pourraient, d'après la règle 4 de simultanéité, être franchies simultanément. En pratique, on est souvent amené à rendre ces réceptivités exclusives. On peut également introduire des priorités (figure 6.7.b.)

Saut d'étapes et reprise de séquence

Le saut conditionnel est un aiguillage particulier permettant de sauter une ou plusieurs étapes lorsque les actions à réaliser deviennent inutiles, tandis que la reprise de séquence permet au contraire de reprendre une ou plusieurs fois la même séquence tant qu'une condition fixée n'est pas obtenue (figure 6.7.c.)

Séquences simultanées :

Un GRAFCET peut comporter plusieurs séquences s'exécutant simultanément mais dont les évolutions des étapes actives dans chaque branche restent indépendantes

Pour représenter ces fonctionnements simultanés, une transition UNIQUE et deux traits parallèles indiquent le début et la fin des séquences, c'est-à-dire l'activation simultanée des branches ainsi réalisées et leur attente réciproque vers une séquence commune (figure 6.8.).

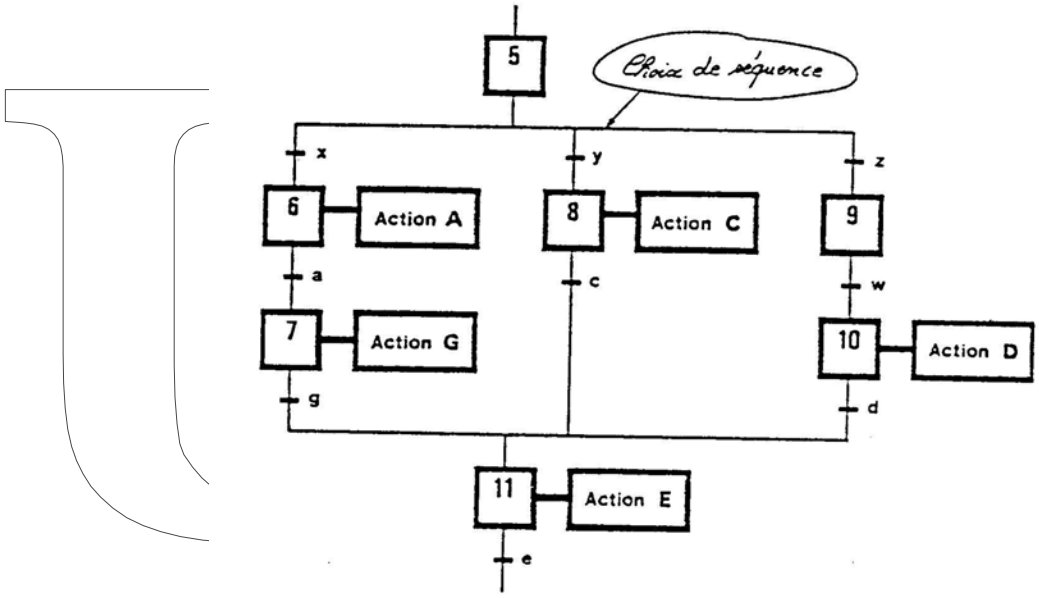
A partir de l'étape 22, la réceptivité p provoque l'activation simultanée des étapes 23 et 26. Ces deux séquences 23-24-25 et 26-27-28-29 évolueront alors de façon totalement indépendante et ce n'est que :
- lorsque les étapes de fin de branche 25 et 29 étant actives,
- lorsque la réceptivité étant vraie (q.r =1),

que la transition sera franchie.

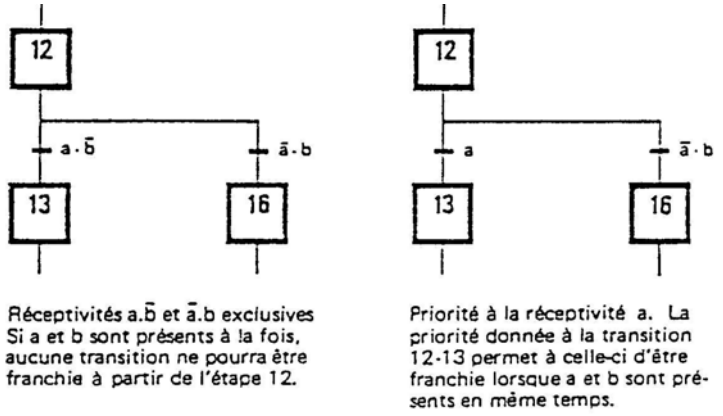
L'étape 30 devient alors active et les étapes 25 et 29 inactives.

Note : les conditions particulières à chaque branche peuvent être notées entre parenthèses au-dessus des traits parallèles de regroupement.

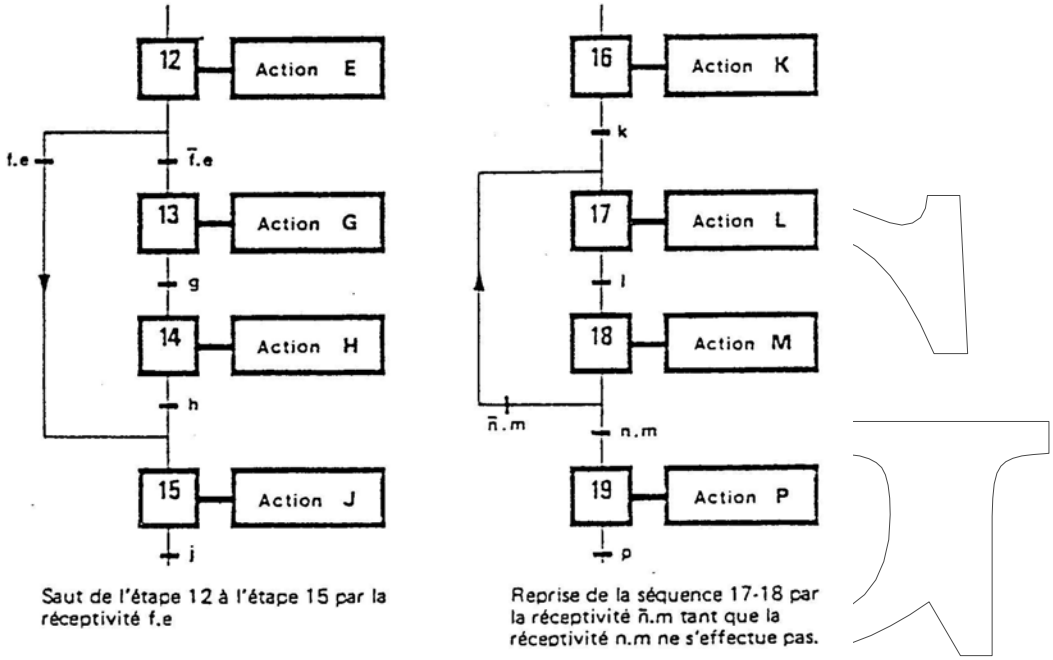
La figure 6.9. donne l'exemple concret d'une unité de perçage - taraudage.



a. Choix conditionnel



b. Exclusivité et priorités



c. Saut d'étapes

Figure 6.7. Aiguillages dans un GRAFCET

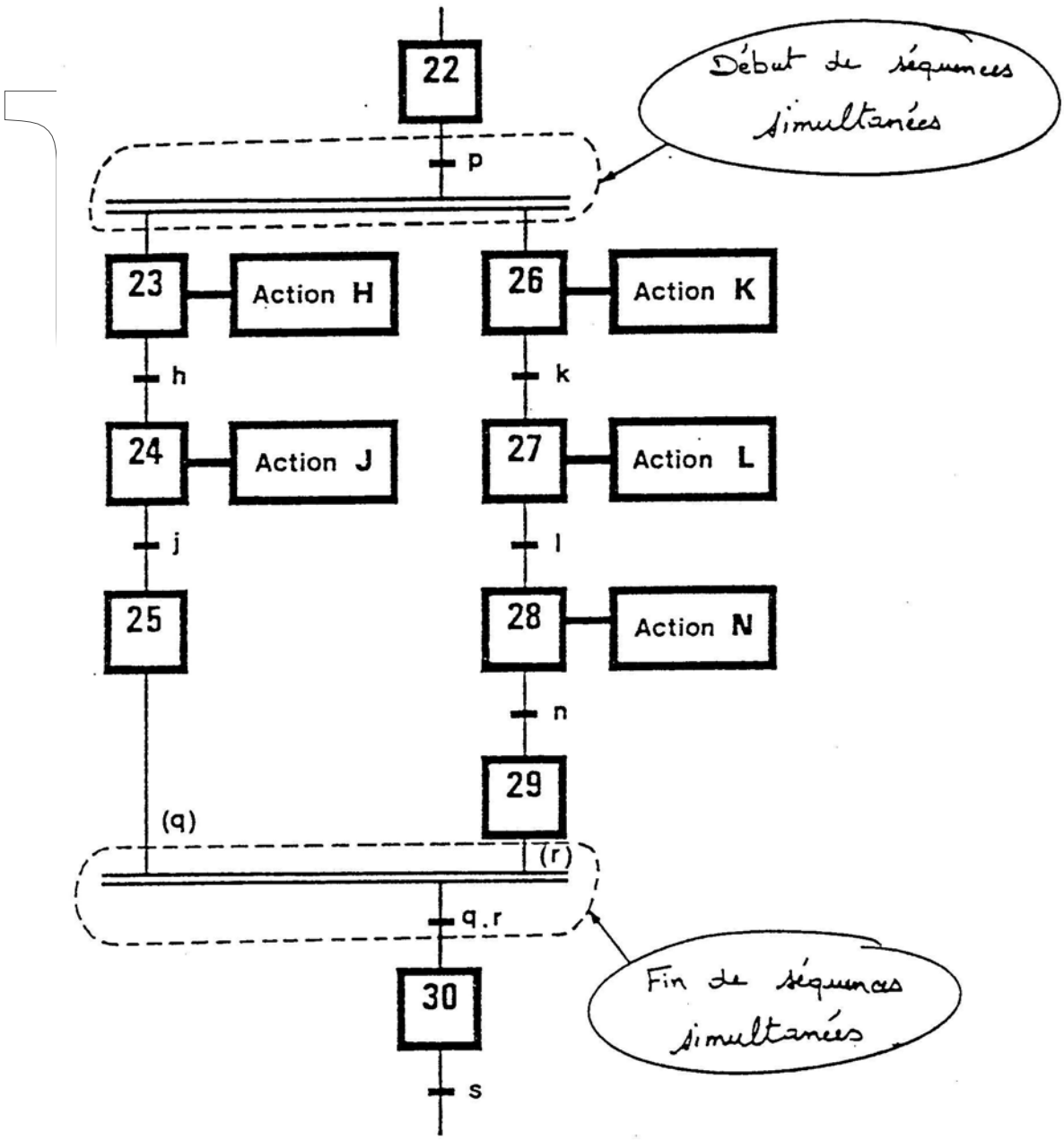


Figure 6.8. Séquences simultanées dans un GRAFCET

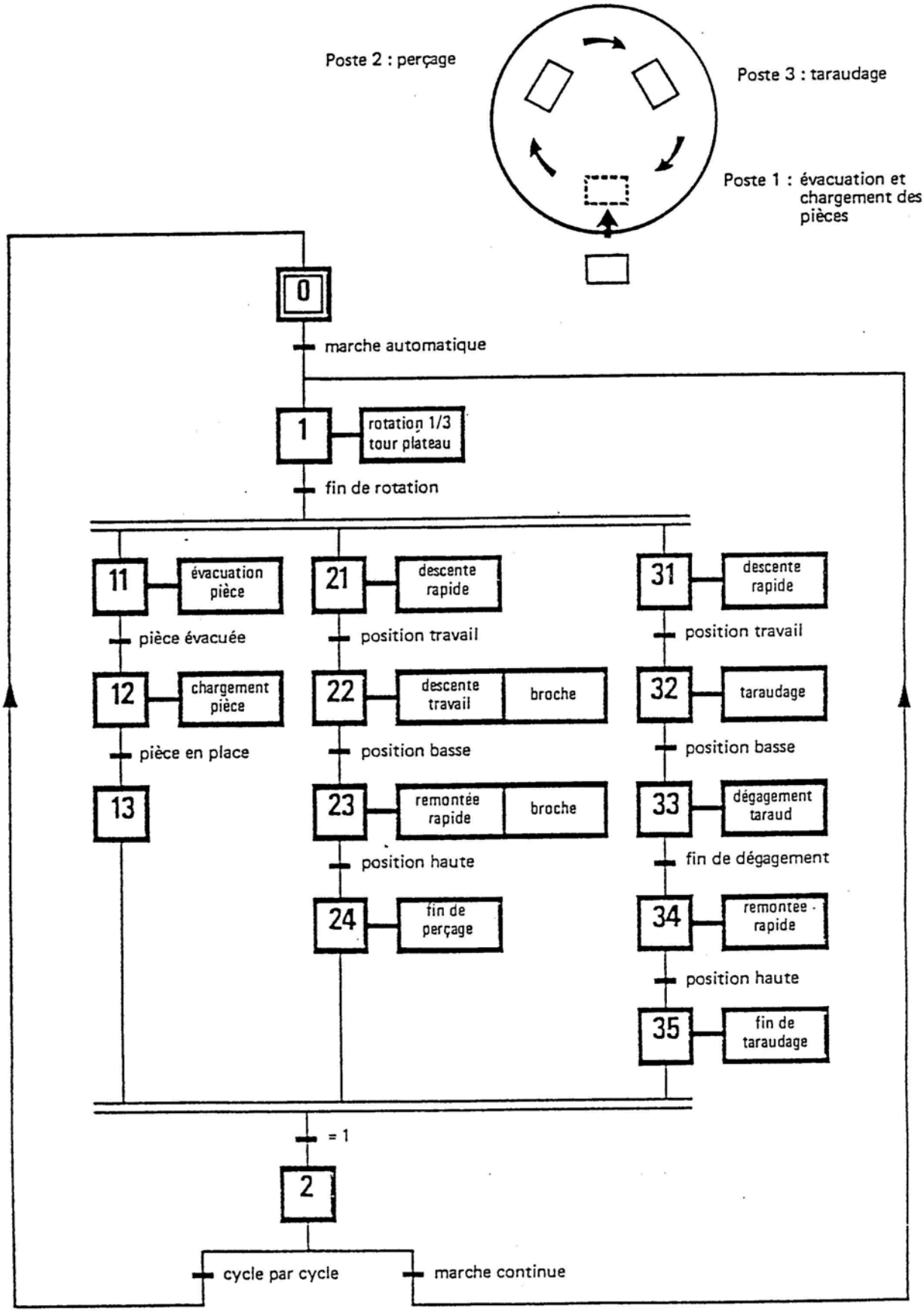


Figure 6.9. Exemple de séquences simultanées : GRAFCET de niveau 1 d'une unité de perçage-taraudage

6.3. ELEMENTS COMPLEMENTAIRES DU GRAFCET

6.3.1. VARIABLES D'ETAPE

Deux variables sont généralement associées aux étapes :

– la variable d'activité : il s'agit d'une variable booléenne gérée automatiquement par le système et qui vaut 1 lorsque l'étape est active. Nous verrons quel usage on peut en faire aux paragraphes 6.4. et 6.5.
Dans la norme CEI 1131-3, elle est désignée par

nom_étape . X

– la variable de durée d'activité : il s'agit d'une variable de type "temps" gérée automatiquement par le système et qui indique depuis combien de temps l'étape est active. Dans la norme CEI 1131-3, elle est désignée par

nom_étape . T

Elle peut être utilisée pour faire intervenir le temps dans une réceptivité, par exemple :

(nom_étape . T > T# 10s)

Notons que, chez beaucoup de constructeurs, cette variable de durée n'est pas disponible d'emblée. C'est au niveau de la configuration du système qu'il faut déclarer pour quelles étapes bien précises, on souhaite qu'une telle variable soit définie. Ceci évidemment dans le but d'économiser du temps de calcul et de la place en mémoire.

Remarquons, enfin, que la présence de ces variables d'étape tenues à jour par le système constituent une facilité pour l'utilisateur mais qu'elles ne sont pas strictement indispensables à la construction d'un GRAFCET. En effet, le programmeur pourrait très bien gérer lui-même des informations équivalentes par le biais de variables internes et de temporisation.

A titre informatif, on donne, dans le tableau qui suit, les solutions adoptées par différents constructeurs.

	CEI 1131	ALLEN-BRADLEY	CADEPA	ISAGRAPH	TELEME-CANIQUE
variable d'étape	étape . X	*SCj:n . SA	Xi	GSi . X	Xi
durée d'étape	étape . T	*SCj:n . TIM	-	GSi . t	*Xi, V
réceptivité associée à la durée	étape . T > T#5s	SCj:n . DN	T/i/5s/	GSi . t > 5s	Xi, V > 50

Légende :
* à confirmer dans une phase de configuration

ALLEN-BRADLEY
j : n° du fichier de contrôle SFC
n : n° de la structure de contrôle attachée à l'étape considérée
SCj:n . PRE : valeur de présélection pour la durée d'étape

CADEPA, ISAGRAPH, TELEMECANIQUE
i : n° de l'étape

6.3.2. TYPES D'ACTION

Au paragraphe 6.2.1., on a implicitement supposé que les actions associées aux étapes étaient effectuées et maintenues durant toute la période d'activité de l'étape.

Ce type d'action, qualifié de "non mémorisé est suffisant pour décrire n'importe quel automatisme. Cependant, de nouveau pour simplifier la tâche du programmeur, les constructeurs d'abord, la norme CEI 1131-3 ensuite, ont introduit une série d'autres types d'action répondant à des besoins fréquemment rencontrés en pratique.
La liste en est donnée à la figure 6.10.

Qualificatif		*Explication*	*Disponibilité chez les constructeurs*			
Norme CEI 1131	Hors norme		A-B	Telem	Isagraf	Cadepa
-		Par défaut - Non mémorisé	X	X	X	X
N		Non mémorisé	X		X	
S		Positionné (mémorisé)	X		X	X
R		Remise à zéro prioritaire	X		X	X
L		Limité dans le temps	X			
D		Temporisé	X			X
P	P1	Impulsion (activation)	X	X		X
	P0	Impulsion (désactivation)	X	X		X
SL		Mémorisé et limité dans le	X			
SD		temps	X			
DS		Mémorisé et et temporisé	X			
		Temporisé et mémorisé				

Fig. 6.10. Les différents types d'action

La figure 6.11. permet de mieux comprendre le sens des différents qualificatifs :

– Pas de qualificatif ou N (Non mémorisé) : l'action commence à l'activation de l'étape et se termine à la désactivation de celle-ci. C'est le mode de fonctionnement originel du GRAFCET.

– S (Positionné - mémorisé) : l'action commence à l'activation de l'étape et se poursuit même après la désactivation de celle-ci; elle doit être explicitement arrêtée dans une étape ultérieure du GRAFCET contenant l'action en question affectée du qualificatif "R"

– R (Remise à zéro prioritaire) : arrête l'action qualifiée "R" dès l'activation de l'étape à laquelle elle est associée

– L (Limité dans le temps) : l'action démarre avec l'activation de l'étape et s'arrête après un délai fixé ou à la désactivation de l'étape

– D (Temporisé) : l'action démarre avec un retard déterminé par rapport à l'activation de l'étape, pour autant que l'étape soit toujours active à ce moment.

– P1 (Impulsion à l'activation) : l'action est effectuée une fois lors de l'activation de l'étape.

– P0 (Impulsion à la désactivation) : l'action est effectuée une fois lors de la désactivation de l'étape.

– SL (Mémorisé et limité dans le temps) : l'action démarre à l'activation de l'étape et s'arrête après un délai fixé ou une remise à zéro prioritaire ("R") et, cela, indépendamment de l'évolution de l'étape elle-même

– SD (Mémorisé et temporisé) : l'action démarre avec un retard déterminé par rapport à l'activation de l'étape et s'arrête après une remise à zéro prioritaire et, cela, indépendamment de l'évolution de l'étape elle-même

– DS (Temporisé et mémorisé) : l'action démarre avec un retard déterminé par rapport à l'activation de l'étape pour autant que celle-ci soit toujours active à ce moment (c'est ce qui différencie ce cas du précédent); l'action s'arrête après une remise à zéro prioritaire

Tous ces types d'action peuvent être recréés à partir du type originel (N). Ainsi, la figure 6.12. montre comment on peut réaliser une action impulsionnelle moyennant l'introduction d'une étape supplémentaire.

S'il est vrai que l'utilisation des actions particulières décrites ci-dessus est de nature à alléger le dessin d'un GRAFCET, par contre elle en rend la lecture beaucoup plus difficile car certaines informations relatives à l'évolution du système n'y apparaissent plus que de manière implicite. Il peut en résulter d'assez sérieuses difficultés pour la mise au point et la maintenance des programmes.

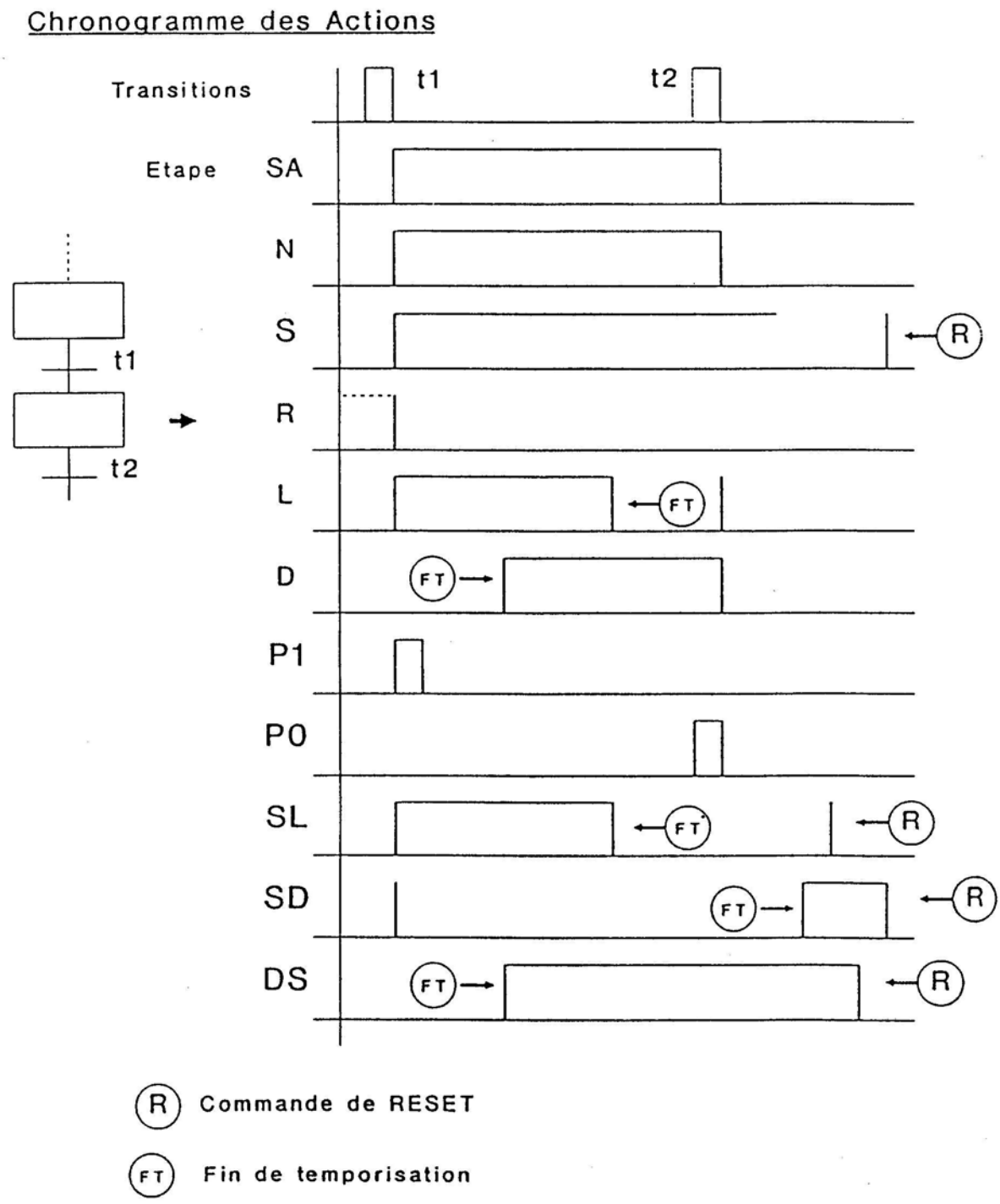


Figure 6.11. Les différents types d'action. Explication.
(Source : ALLEN-BRADLEY)

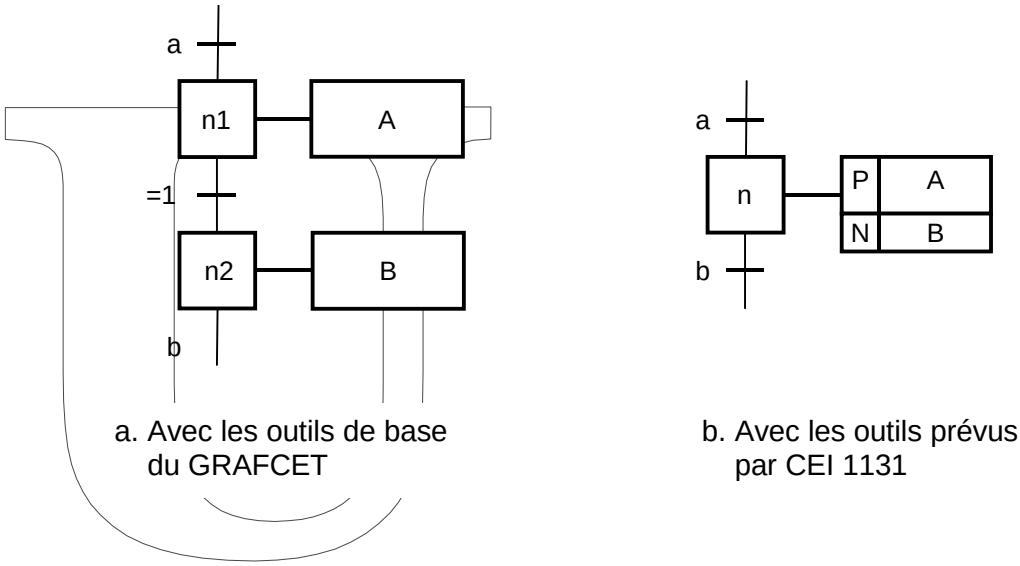


Figure 6.12. Action impulsionnelle A

6.3.3. NATURE DES ACTIONS

Le plus souvent, une action consiste à changer l'état d'une ou plusieurs variables booléennes (sorties, bits internes). En fait, la norme CEI 1131-3 donne une définition beaucoup plus large de l'action : il peut s'agir d'un ensemble d'instructions logiques, arithmétiques ou autres, écrites dans un quelconque des langages de l'automate.

Ces instructions seront exécutées à chaque cycle de l'automate pendant toute la durée de l'action, cette durée dépendant elle-même de l'activation de l'étape associée et du type assigné à l'action (cf. § 6.3.2.).

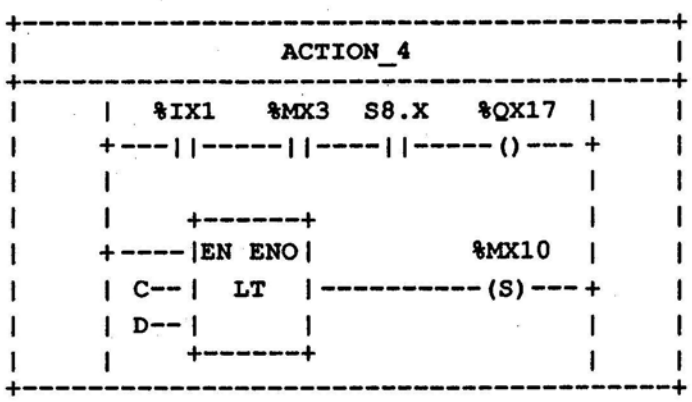
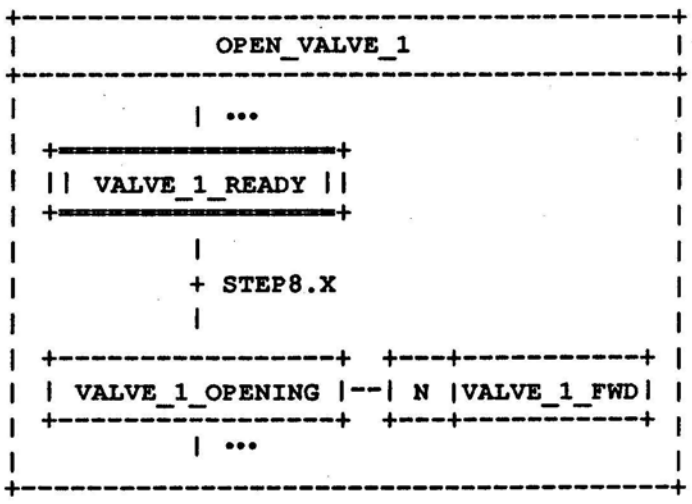
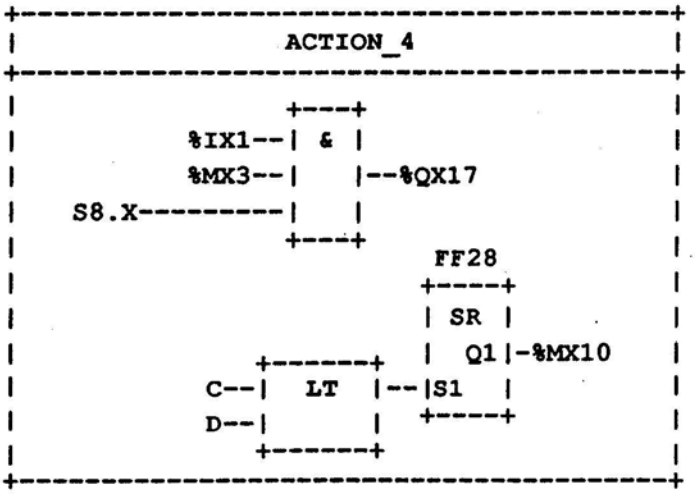
Une action peut même être constituée d'un autre GRAFCET. Celui-ci est lancé (activation de ses étapes initiales) au démarrage de l'action et tué (désactivation de toutes ses étapes) lorsque l'action est arrêtée. La durée de l'action étant régie par les règles du paragraphe 6.3.2.

La figure 6.13. extraite de la norme, donne quelques exemples d'actions.

6.4. STRUCTURATION D'UN GRAFCET

Pour conserver au GRAFCET tout son potentiel de clarté, il est indispensable de pouvoir le structurer. On entend par là que, dans le graphisme, on doit pouvoir utiliser des étapes qui représentent elles-mêmes des GRAFCET complets (on parle de macro-étapes). Si cette possibilité n'existait pas le GRAFCET d'un automatisme quelque peu complexe s'étalerait, en effet, sur des mètres carrés de papier et serait pratiquement inexploitable.

La structuration permettra donc d'aborder la conception d'un automatisme de haut en bas. On pourra partir d'un GRAFCET représentant les objectifs globaux du système. Chaque étape sera ensuite décomposée en GRAFCET plus détaillés et ainsi de suite jusqu'à arriver aux actions élémentaires sur le processus.

N°	Caractéristique	
1	**Toute variable booléenne déclarée dans un bloc VAR ou VAR_OUTPUT, ou ses équivalents graphiques, peut être une action**	
	Exemple	**Caractéristique**
2l		**Déclaration graphique en langage LD (voir 4.2)**
2s		**Inclusion d'éléments SFC dans une action**
2f		**Déclaration graphique en langage FBD (voir 4.3)**

N°	Caractéristique	
1	Toute variable booléenne déclarée dans un bloc VAR ou VAR_OUTPUT, ou ses équivalents graphiques, peut être une action	
	Exemple	Caractéristique
3s	ACTION ACTION_4 : %QX17 = %IX1 & %MX3 & S8.X ; FF28(S1 := (C<D)) ; %MX10 := FF28.Q ; END_ACTION	Déclaration littérale en langage ST (voir 3.3)
3i	ACTION ACTION_4 : LD S8.X AND %IX1 AND %MX3 ST %QX17 LD C LT D S1 FF28 LD FF28.Q ST %MX10 END_ACTION	Déclaration littérale en langage IL (voir 3.2)

Figure 6.13. Exemples de déclarations d'actions selon la norme CEI 1131

Les avantages de cette démarche toute cartésienne sont importants :

– existence de plusieurs niveaux de description de l'automatisme, les niveaux supérieurs étant facilement compréhensibles même par des non-spécialistes.
Le GRAFCET structuré pourra donc constituer un outil de dialogue entre les différents métiers impliqués dans l'automatisation de la production (mécaniciens, électriciens, automaticiens), de même d'ailleurs qu'entre ces métiers et la direction.

– possibilité de répartir les travaux de programmation entre plusieurs équipes, chacune pouvant se situer avec précision dans l'ensemble.

– simplification des opérations de test et de maintenance qui pourront rapidement se circonscrire au sous-ensemble fonctionnel de la structure qui est en défaut.
Pour pousser à fond ce dernier avantage, on imposera qu'à chacun des niveaux de la décomposition, les sous-GRAFCET obtenus tiennent sur une page de papier (ou sur l'écran de la console), de manière à pouvoir en saisir toute la logique d'un seul coup d'oeil.

6.4.1. UTILISATION DES OUTILS STANDARDS DU GRAFCET

La figure 6.14. montre comment on peut créer des macro-étapes en utilisant les variables d'étape X définies au paragraphe 6.3.1. L'activation de l'étape 6 du GRAFCET principal lance l'exécution du sous-GRAFCET, en attente de la réceptivité X6. L'évolution du GRAFCET principal est alors suspendue (réceptivité X103) jusqu'à ce que le sous-GRAFCET ait terminé son cycle (activation de l'étape 103).

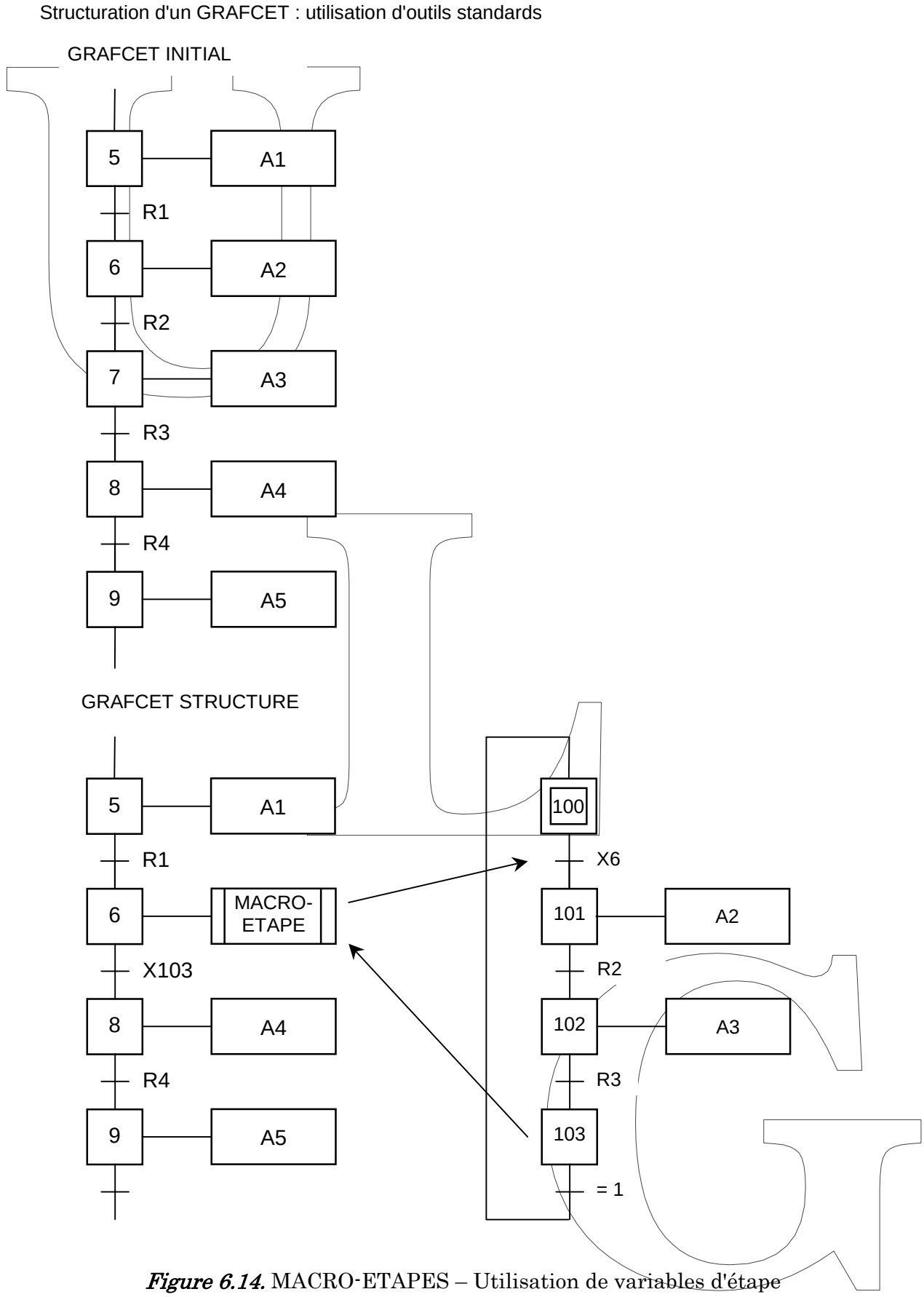


Figure 6.14. MACRO-ETAPES – Utilisation de variables d'étape

6.4.2. UTILISATION D'ACTIONS GRAFCET

La norme CEI 1131-3 ne définit pas la notion de macro-étape. Les actions "GRAFCET" qui ont été présentées au paragraphe 6.3.3. peuvent cependant être utilisées dans un optique de structuration fort semblable, comme cela est représenté à la figure 6.15.

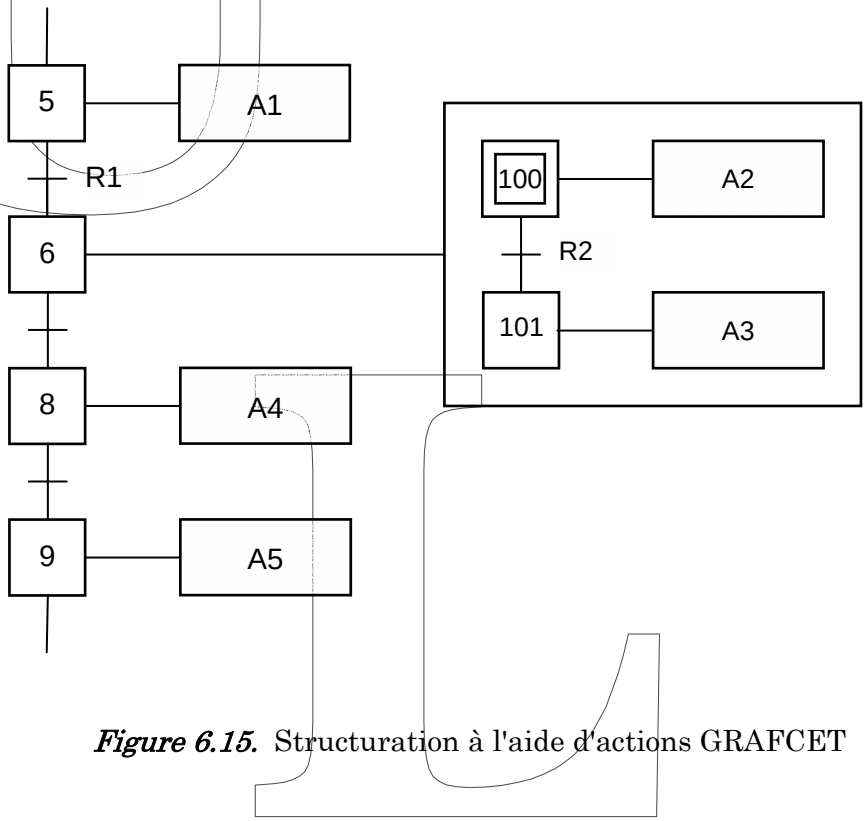


Figure 6.15. Structuration à l'aide d'actions GRAFCET

6.4.3. SYNTAXES SPECIFIQUES

Dans les langages GRAFCET proposés par les constructeurs, on trouve en général des syntaxes spécifiques pour la définition de macro-étapes. La figure 6.16. donne l'exemple du langage TELEMECANIQUE.

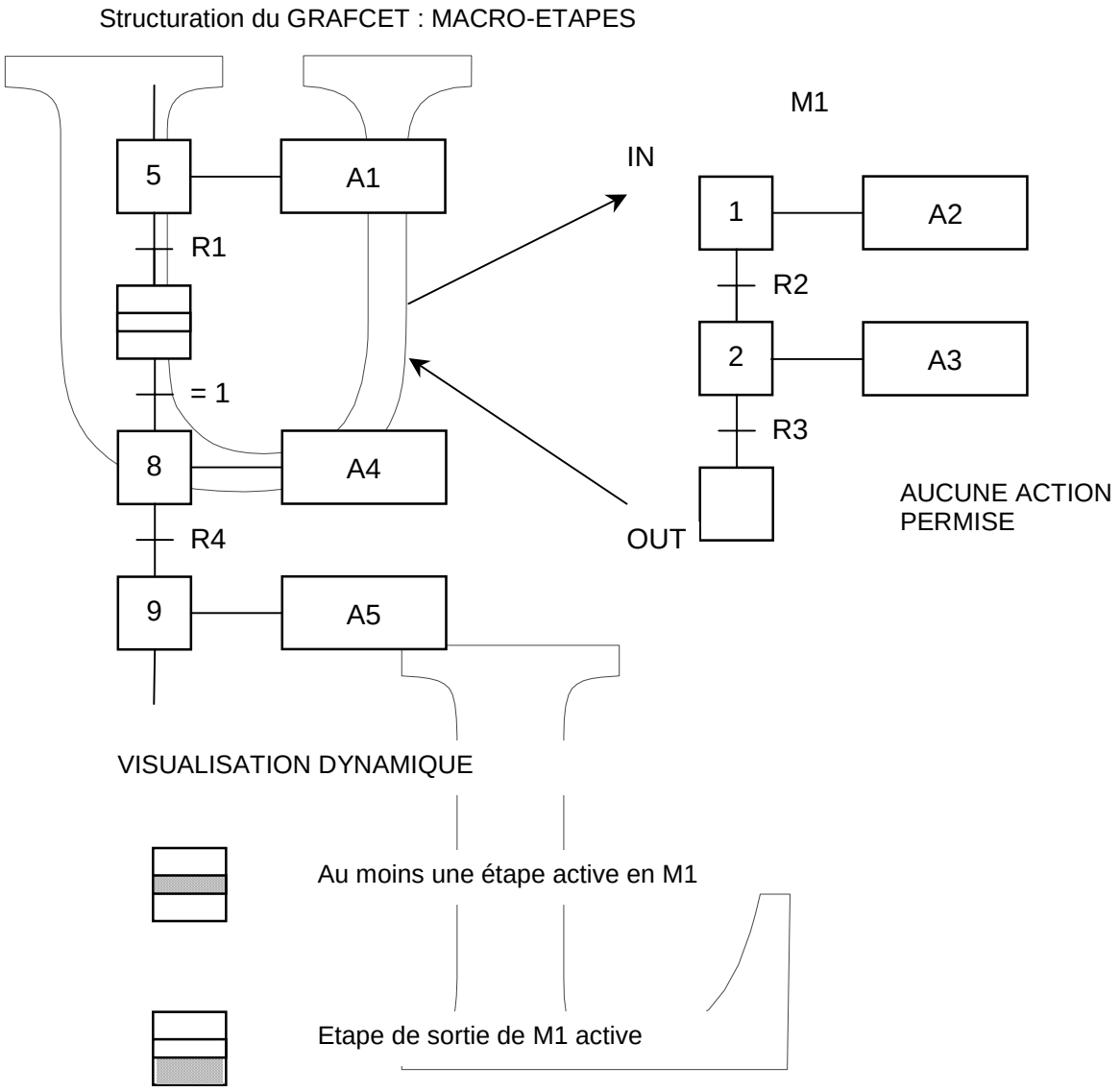


Figure 6.16. MACRO-ETAPES – Syntaxe spécicifique (TELEMECANIQUE)

6.4.4. UTILISATION DU PRINCIPE "CLIENT-SERVEUR"

Le principe consiste ici à utiliser une action du GRAFCET principal pour lancer le sous-GRAFCET (cf. figure 6.17.). En quelque sorte, le GRAFCET principal fait une demande service ("Dem 100" sur la figure) ou sous-GRAFCET. Une fois sa tâche achevée, ce dernier émet un compte-rendu d'activité, également sous forme d'action ("OK 100" sur la figure) qui permet de poursuivre la séquence principale.

Les avantages de cette manière de faire sont les suivantes :

– indépendance vis-à-vis de la numérotation des étapes. Ce point prend tout son sens si l'on sait que certains éditeurs permettent de renuméroter automatiquement les étapes d'un GRAFCET. Ceci est évidemment catastrophique si les variables d'étape du GRAFCET en question sont utilisées dans d'autres GRAFCET.

– facilité de tester séparément le sous-GRAFCET : il suffit en effet de forcer la variable "Dem 100".

– facilité d'introduire des fonctions de diagnostic. En effet, le sous-GRAFCET peut très bien renvoyer un compte-rendu circonstancié de son activité, par exemple, "OK 100" si tout s'est bien passé; "NOK 100" s'il a détecté un problème. Le GRAFCET principal peut alors réagir de manière adéquate.

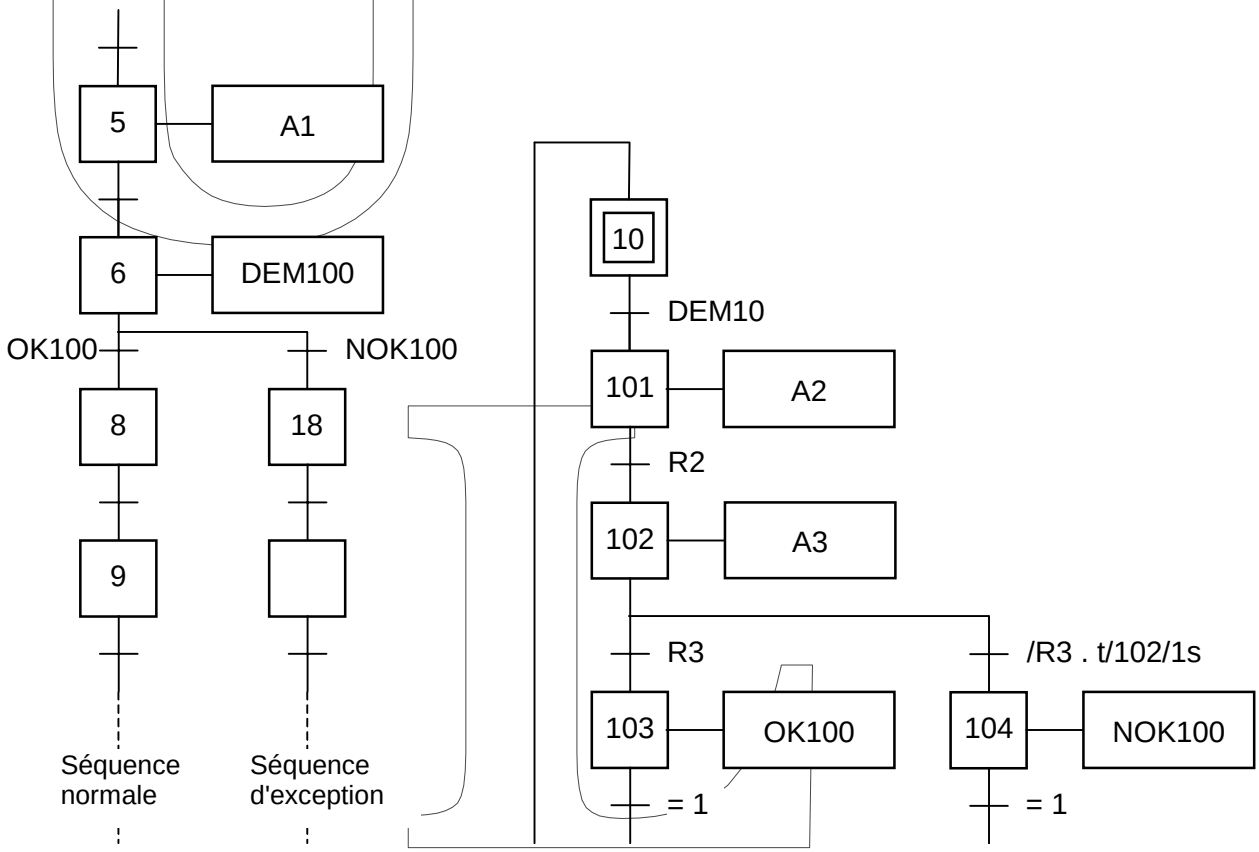


Figure 6.17. Structuration du GRAFCET – Principe du "client-serveur"

6.5. SYNCHRONISATION ET PROTECTION

On a vu, au paragraphe 6.2.5. que le mode de représentation GRAFCET autorisait le lancement d'activités se déroulant en parallèle, de manière complètement indépendante. Il est évidemment parfois nécessaire de resynchroniser de telles activités de même qu'il faut pouvoir protéger des ressources communes contre des accès simultanés.

6.5.1. SYNCHRONISATION

– Synchronisation explicite

Un premier mode de synchronisation a déjà été présenté au paragraphe 6.2.5 : regroupement des branches parallèles sur une double barre horizontale avec une réceptivité unique. Il est rappelé sur le GRAFCET imaginaire de la figure 6.18.

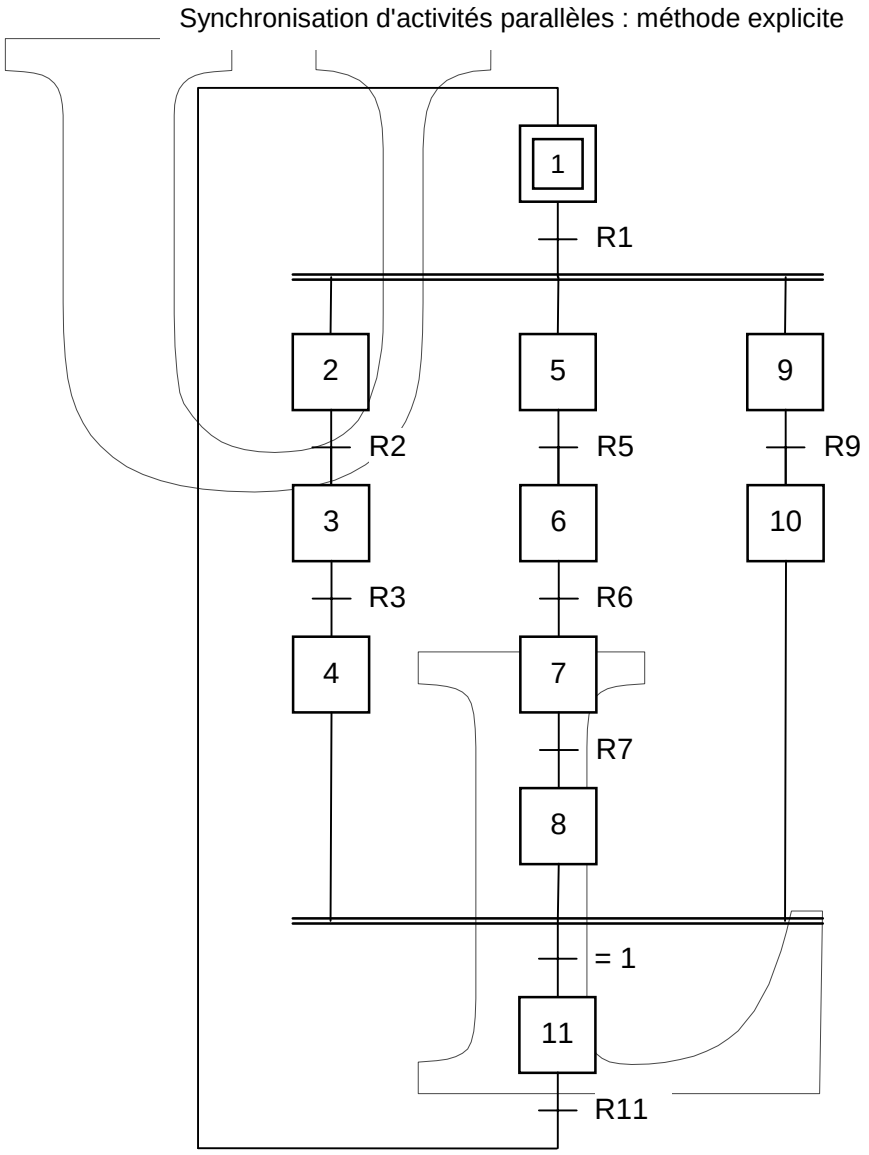


Figure 6.18. Synchronisation explicite

On y remarque trois branches parallèles représentant des activités indépendantes lancées en même temps (réceptivité R1) et une activité, en l'occurrence réduite à l'étape 11, qui ne peut se dérouler que lorsque les trois activités précédentes sont terminées.

Ce mode de synchronisation s'applique dans des cas simples ou bien dans des cas où les activités parallèles sont si étroitement liées qu'une représentation graphique explicite de leurs interactions est indispensable.

Dans des cas plus complexes ou lorsque les activités parallèles ne sont que faiblement couplées, on préfère en général, pour des raisons évidentes de clarté, considérer des GRAFCET graphiquement indépendants et opérer une synchronisation implicite par variables d'étape.

– Synchronisation implicite horizontale

La figure 6.19. montre ce que l'on entend par synchronisation horizontale. Les quatre activités évoquées au paragraphe précédent sont décrites par quatre GRAFCET graphiquement indépendants.

La réceptivité R1 assure le lancement simultané des trois premières activités. La quatrième est verrouillée par la réceptivité X4 . X8 . X10 qui ne peut être vraie que si les trois premières activités sont terminées (étapes 4, 8 et 10 actives). La fin de la quatrième activité (étape 20 active) remet le système dans les conditions initiales par le truchement des réceptivités X20.

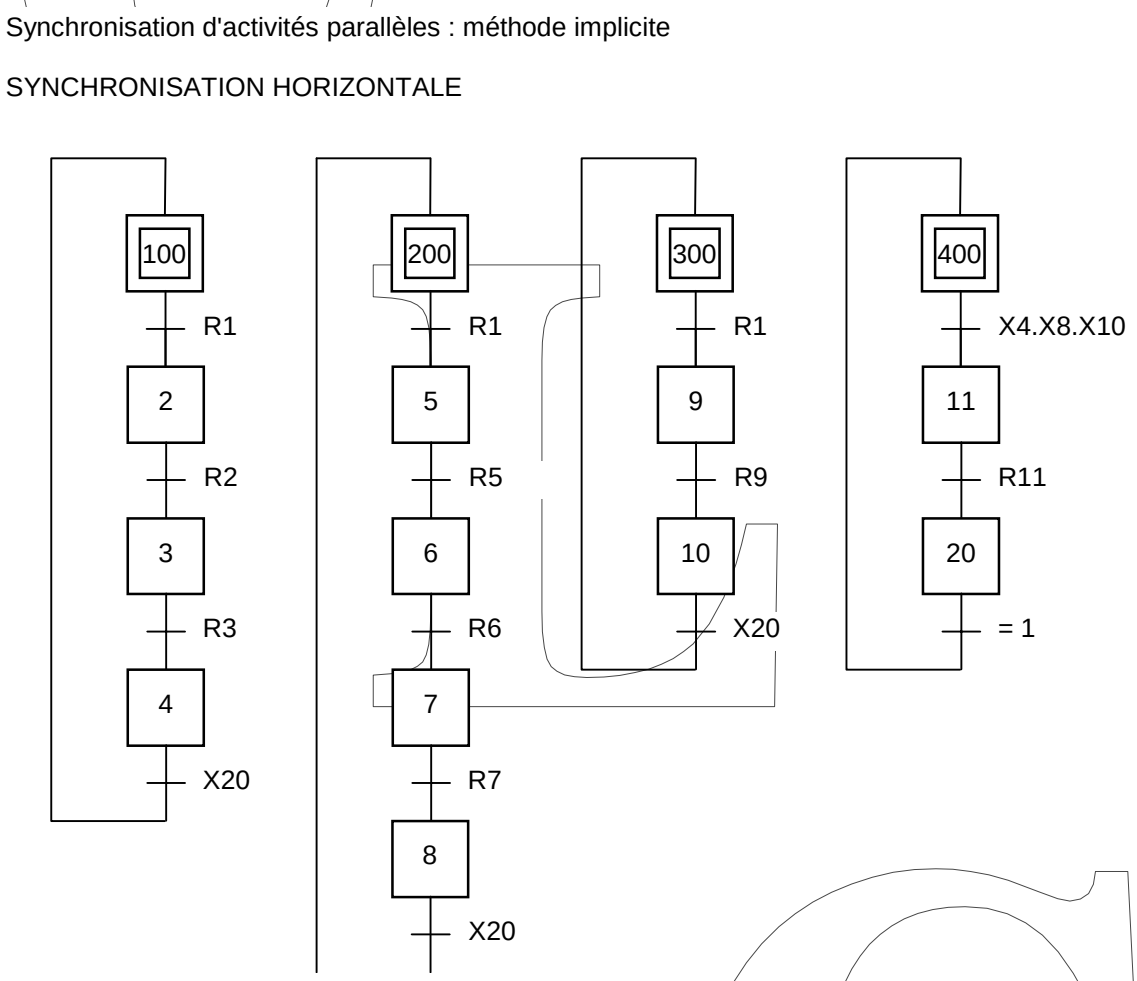


Figure 6.19. Synchronisation horizontale

– **Synchronisation implicite verticale**

Dans des cas complexes, la synchronisation horizontale peut devenir difficile à suivre étant donné l'intervention possible, dans chaque GRAFCET, de variables d'étape provenant d'un ou plusieurs autres GRAFCET.

A cet égard, la synchronisation verticale est beaucoup plus claire comme le montre la figure 6.20. On y retrouve les quatre GRAFCET de l'exemple précédent plus, cette fois, un cinquième GRAFCET de synchronisation. On y a, en outre, introduit le principe "client-serveur" expliqué au paragraphe 6.4.4.

C'est ce GRAFCET qui contrôle l'exécution des différentes activités.

L'intérêt de la présente solution est que les interactions entre GRAFCET ne se font qu'entre les GRAFCET inférieurs et le GRAFCET de synchronisation.

6.5.2. PROTECTION DE RESSOURCES COMMUNES

La protection de ressources communes est un autre problème qui se pose typiquement lorsque l'on a affaire à des activités se déroulant en parallèle. Les ressources communes doivent être dûment verrouillées de manière à éviter des accès simultanés non désirés.

La figure 6.21. présente l'exemple du chargement d'un haut fourneau par deux wagonnets travaillant indépendamment. En l'occurrence, la ressource à protéger est constituée d'un tronçon de rail commun : il faut impérativement éviter que les deux wagonnets ne s'y engagent en même temps. A cet effet, on prévoit, juste avant le tronçon commun, deux positions, WA et WB, où l'on peut arrêter l'un des wagonnets si l'autre se trouve dans la zone commune.

Synchronisation d'activités parallèles : méthode implicite

SYNCHRONISATION VERTICALE

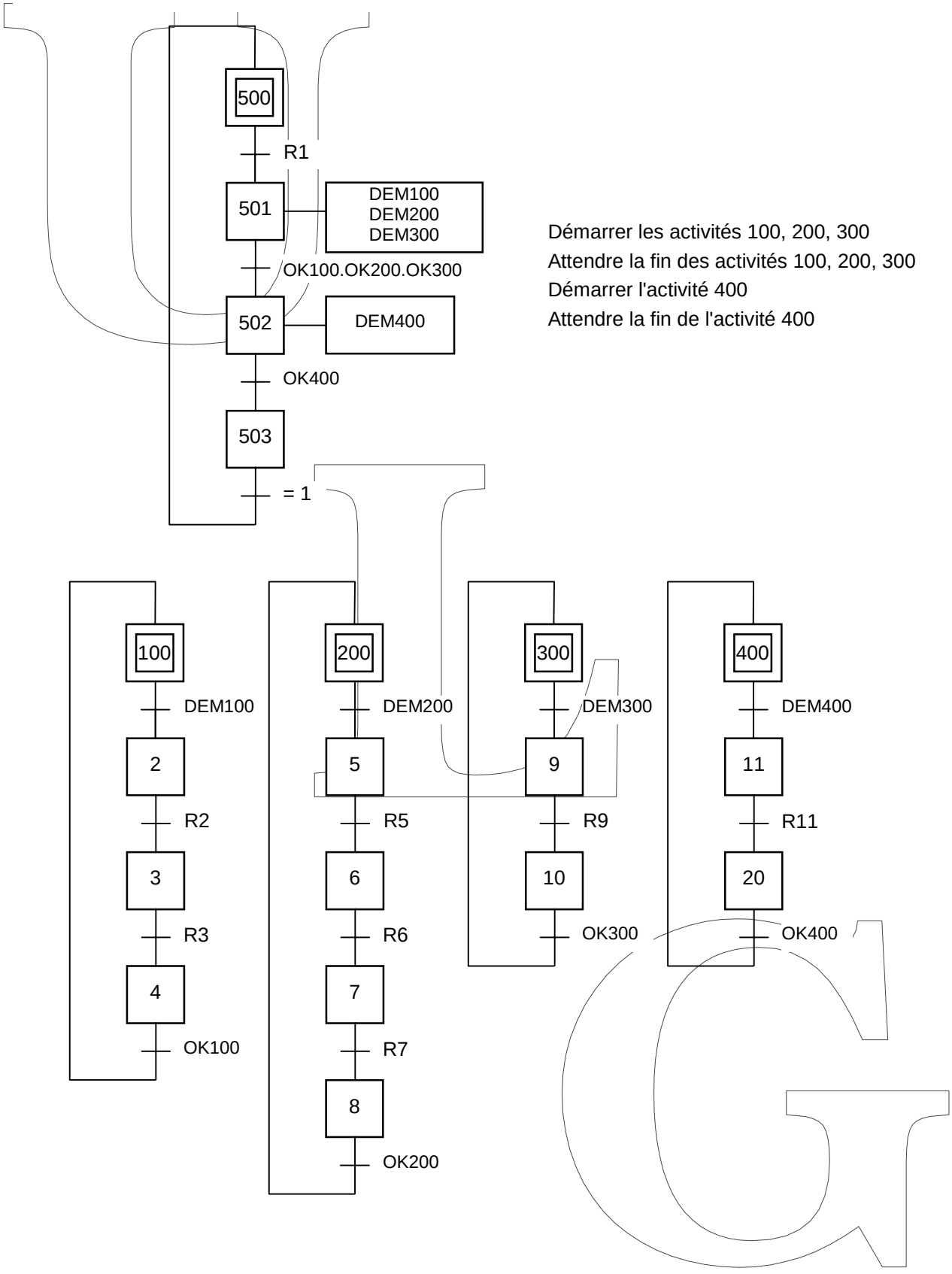


Figure 6.20. Synchronisation verticale

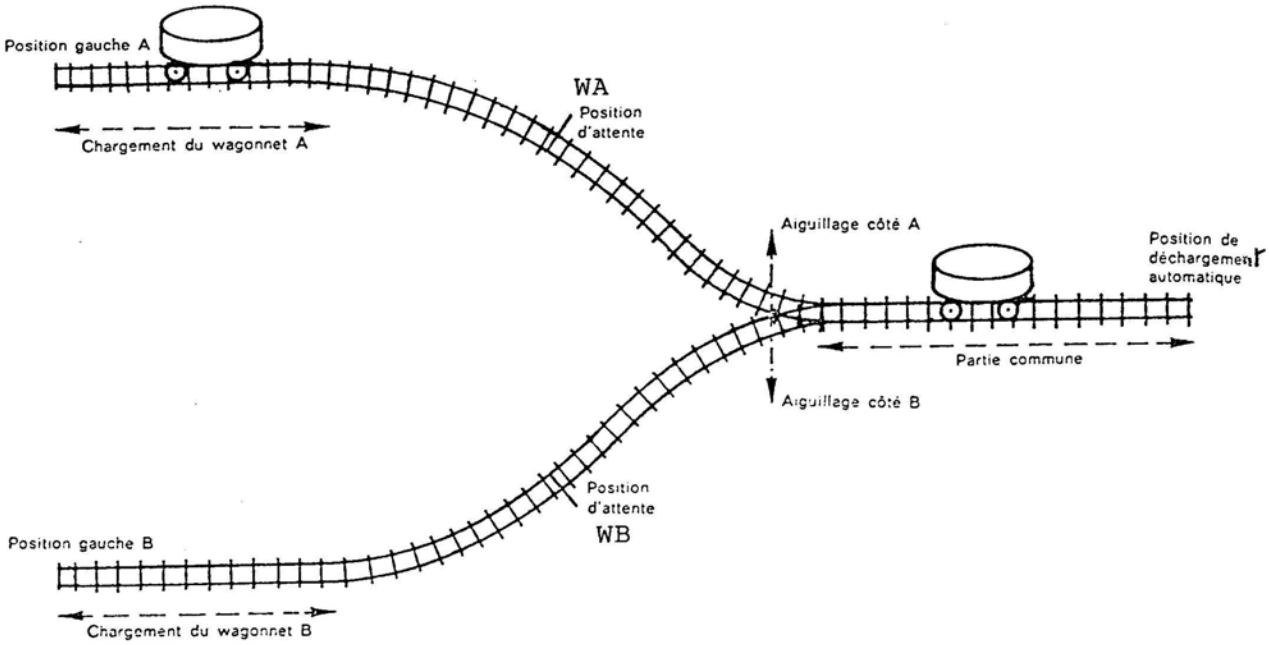


Figure 6.21. Exemple de ressource commune : chargement d'un haut fourneau

Le GRAFCET permet de gérer très élégamment les conflits d'accès comme cela est montré à la figure 6.22. On y trouve deux branches quasi identiques correspondant au fonctionnement des deux wagonnets. L'état libre ou occupé de la ressource commune y est représenté par l'état actif ou inactif de l'étape n 1.

Supposons que la ressource soit libre (étape 1 active) et que le wagonnet A arrive le premier à sa position d'attente WA. Par simple application des règles d'évolution du GRAFCET (étapes 1 et 12 actives), on passe à l'étape 13 et le wagonnet A peut aller se décharger dans le haut fourneau. Ce faisant, l'étape 1 a évidemment été désactivée.

Si, pendant ce temps, le wagonnet B se présente à la position d'attente WB, il va s'y trouver bloqué (étape 22) puisque l'étape 1 n'est plus active. Il ne pourra continuer son chemin que lorsque l'étape 1 sera redevenue active c'est-à-dire, lorsque le wagonnet A sera repassé par sa position d'attente WA et, donc, que la voie sera libre.

Le même raisonnement peut évidemment être fait en commençant par le wagonnet B. Un dernier problème reste à résoudre : celui où les deux wagonnets atteignent simultanément leur position d'attente (étapes 12 et 22 actives). La solution adoptée dans l'exemple consiste à donner la priorité à l'un des wagonnets. En introduisant la réceptivité /X12 dans la branche de droite du GRAFCET, c'est le wagonnet A que l'on privilégie ainsi. En effet, l'étape 23 ne pourrait être activée si l'étape 12 est inactive.

Sur la figure 6.23., le même principe a été exploité mais, cette fois, au travers d'un GRAFCET de synchronisation: le tronçon commun est libre si l'étape 1 est active et occupé si l'étape 2 est active. Remarquons que le rôle joué par l'étape 1 dans le mécanisme de protection décrit ci-dessus est tout à fait similaire au rôle du "sémaphore" utilisé par les informaticiens en programmation multitâche temps réel.

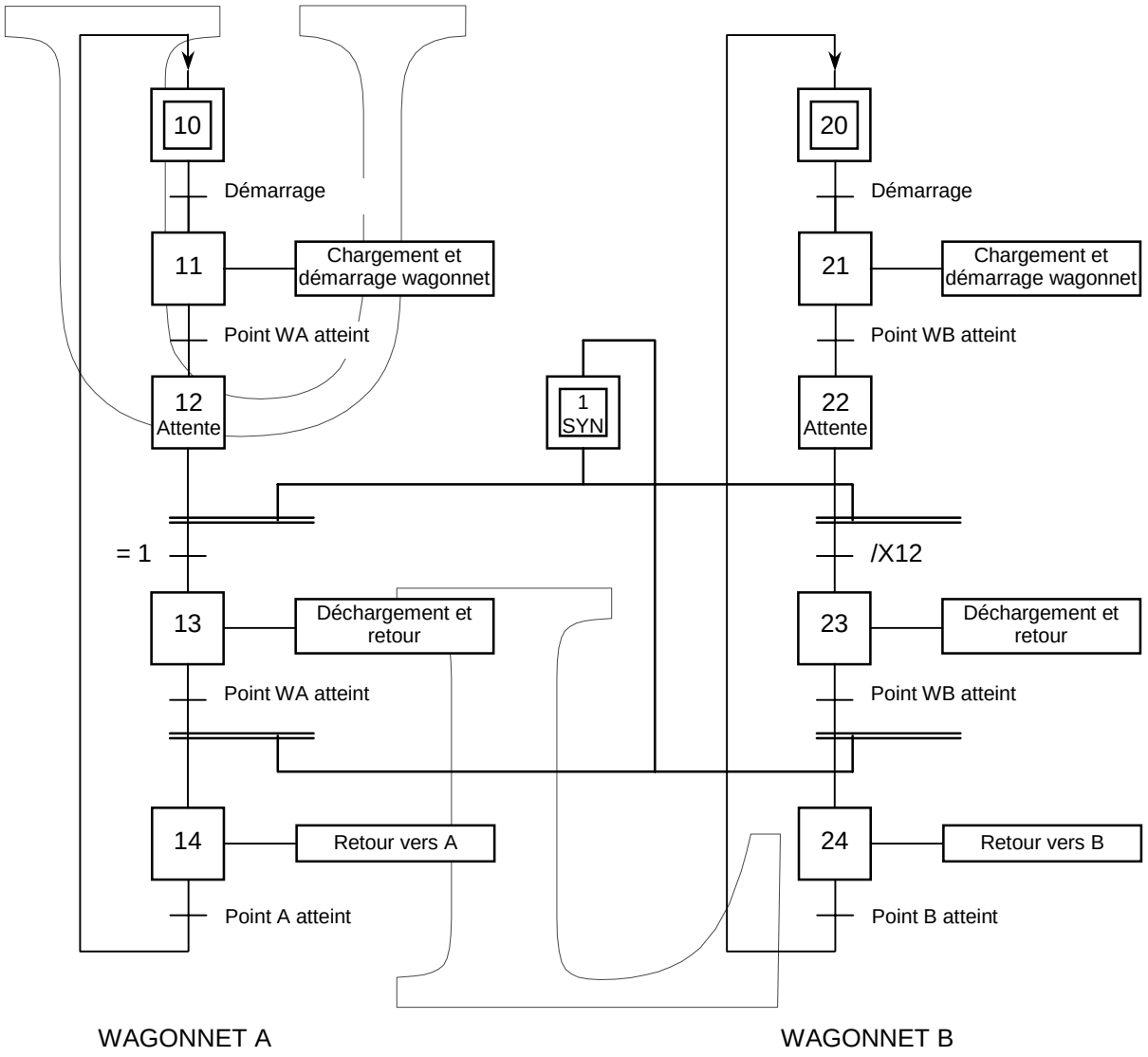


Figure 6.22. Mécanisme de protection de la ressource commune – Méthode explicite

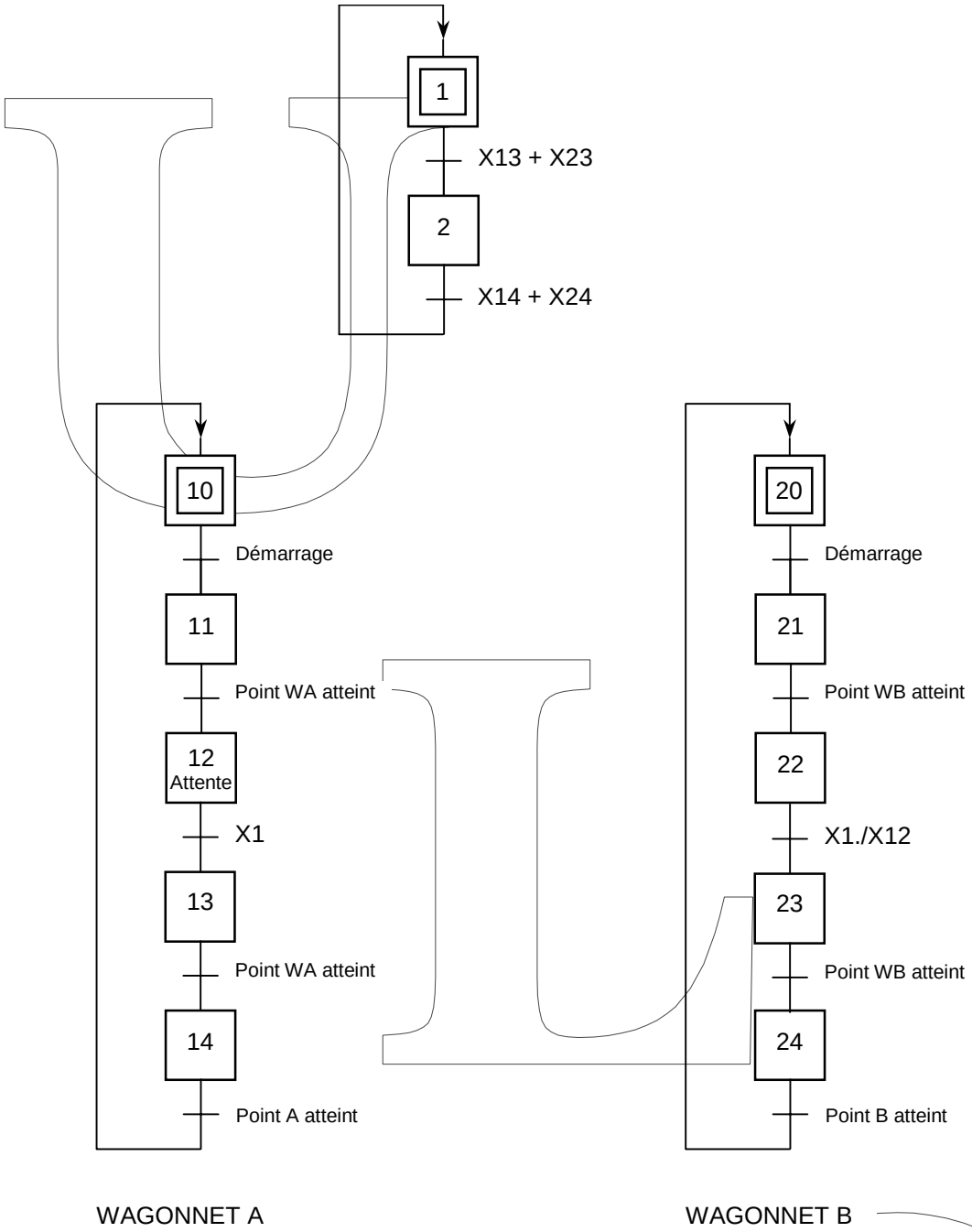


Figure 6.23. Mécanisme de protection de la ressource commune
Utilisation d'un graphe de synchronisation

6.6. TRANSPOSITION DU GRAFCET DANS UN LANGAGE AUTOMATE

6.6.1. PRINCIPES DE LA TRANSPOSITION

Plusieurs méthodes de transposition du GRAFCET sont envisageables. On peut, en gros, les classer en deux catégories :

– **utilisation de bits d'étapes :** l'activité d'une étape est représentée par l'état d'un bit interne. C'est ce qui avait été présenté au paragraphe 4.3.2. Nous verrons ci-après que plusieurs variantes existent qui peuvent évidemment être programmées avec n'importe quel langage d'automate et pas seulement avec le langage à relais.

– **utilisation de pointeurs de phase :** l'activité d'une étape est dans ce cas représentée par la valeur d'un mot. Si le mot vaut n, cela signifie que l'étape n est active. Cette méthode implique évidemment de nombreuses opérations sur mots.

Lors du choix d'une méthode de transposition, il convient d'être attentif aux points suivants :

– **facilité d'édition et lisibilité du programme :** celles-ci dépendent essentiellement de l'adéquation entre le langage de programmation utilisé et la méthode de transposition considérée. Ainsi un langage de type ST se prêtera parfaitement à la méthode du pointeur de phase tandis qu'un langage IL, qui gère assez péniblement les opérations sur mots, s'accommodera mieux d'une méthode basée sur les bits d'étape.

– **facilité de mise au point :** en phase de mise au point, il faut être en mesure de faire évoluer un GRAFCET même si toutes les conditions nécessaires ne sont pas réunies. Ceci implique de pouvoir forcer des réceptivités et des étapes. A cet égard, la méthode du pointeur de phase est certainement la plus sûre car elle garantit intrinsèquement qu'il n'y aura jamais qu'une seule étape active à la fois dans une branche de GRAFCET. Les méthodes basées sur les bits d'étape requièrent beaucoup plus de vigilance de la part du programmeur.

– **facilité de diagnostic :** ici aussi la méthode du pointeur de phase s'avère très commode. Il suffit en effet de visualiser dynamiquement la valeur des pointeurs de phase des différentes branches du GRAFCET pour connaître, à tout moment, le numéro des étapes actives.

– **performances :** pour une méthode donnée, les performances dépendent fortement du type d'automate utilisé et du nombre d'étapes du GRAFCET. Il est donc difficile de dégager des règles générales. On peut cependant s'attendre à ce que la méthode du pointeur de phase soit la plus défavorable puisqu'elle utilise des opérations sur mots.
Pour une étude systématique de cet aspect des choses, le lecteur consultera avec profit [FORET, 1996].

6.6.2. DIFFICULTES POTENTIELLES

Dans ce paragraphe, on attirera l'attention sur certains problèmes qui peuvent apparaître en pratique. Ceux-ci résultent, pour l'essentiel, du fait que les instructions décrivant le GRAFCET sont exécutées de manière séquentielle et cyclique.

– **Durée d'activation minimale d'une étape**

Il faut se souvenir(cf. § 2.3.6.) que les sorties calculées durant un cycle d'automate sont en fait inscrites dans une table image et ne sont effectivement envoyées vers les interfaces de sortie qu'en fin de cycle.

Si, dès lors, une étape est activée et désactivée au cours d'un même cycle de l'automate, les actions éventuellement associées à cette étape n'auront aucun effet sur le processus. Le cas se présente, par exemple, si l'on veut réaliser une action impulsionnelle par le moyen décrit au paragraphe 6.3.2. (figure 6.12.)

Il est assez facile de résoudre ce problème en obligeant les étapes à rester actives durant au moins un cycle de l'automate. Nous verrons au paragraphe 6.6.3. quelques artifices de programmation qui permettent d'arriver à ce résultat.

– **Problèmes de parallélisme d'évolution**

Considérons, à la figure 6.24., trois branches de GRAFCET sensées évoluer en parallèle mais avec certaines interactions. Plaçons-nous d'abord dans le cas d'un GRAFCET idéal.

La figure 6.24.a montre l'état du GRAFCET à un instant donné. Si la réceptivité R1 devient vraie, les règles d'évolution du GRAFCET l'amènent à l'état 6.24.c. après un passage fugitif par l'état 6.24.b. (cf. la remarque faite à la fin du paragraphe 6.2.4.).

Prenons maintenant le cas réel d'une exécution séquentielle et supposons que la scrutation soit effectuée dans l'ordre 1, 2, 3 (figure 6.25.). Après un cycle de l'automate, on se retrouve dans la situation 6.25.b. Après un second cycle, on atteint la situation stable et correcte de la figure 6.25.c.

Supposons maintenant que la scrutation se fasse dans l'ordre 3, 2, 1 (figure 6.26). Après un cycle de l'automate, on se retrouvera dans la situation 6.26.b. A1 est devenu vrai mais trop tard pour être vu par les branches 2 et 3.

Le cycle suivant conduit à la situation de la figure 6.26.c. A2 est devenu vrai mais trop tard pour être vu par la branche 3. Entre temps, A11 est devenu vrai, si bien que la réceptivité $A2 \cdot \overline{A1}\,\overline{1}$ est fausse et que la branche 3 ne peut plus évoluer. Le GRAFCET est donc stabilisé mais dans un état incorrect.

A notre connaissance, il n'y a pas de solution radicale au problème. C'est à l'utilisateur de détecter les situations potentiellement critiques et à prendre les mesures nécessaires pour éviter les risques de dysfonctionnement.

On notera que ce problème est également sous-jacent lorsque l'on utilise les langages GRAFCET proposés par les constructeurs. La différence est que ces derniers ne donnent que rarement des indications sur la manière dont les GRAFCET sont scrutés. Il est de ce fait plus difficile pour l'utilisateur de gérer les situations critiques.

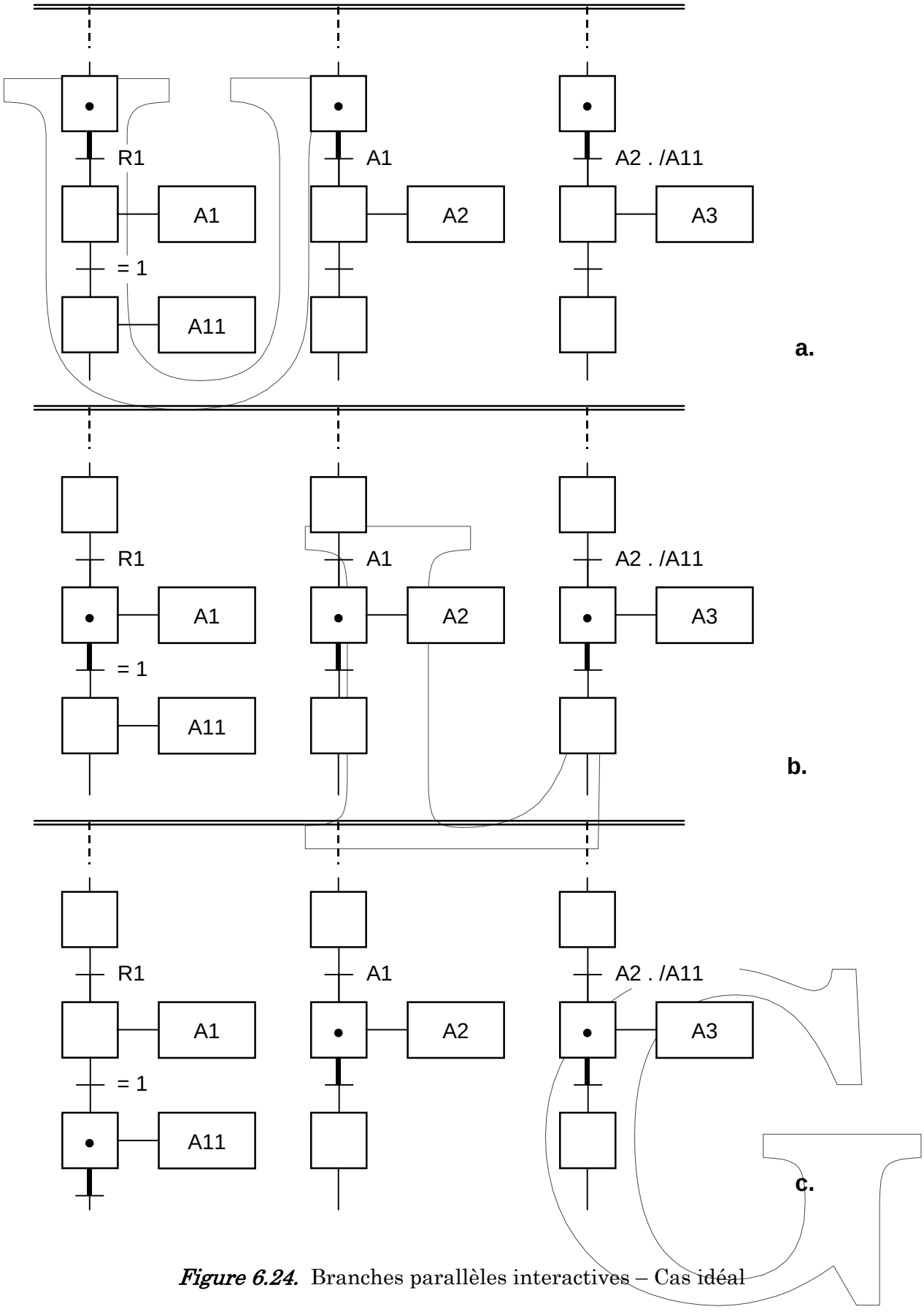


Figure 6.24. Branches parallèles interactives – Cas idéal

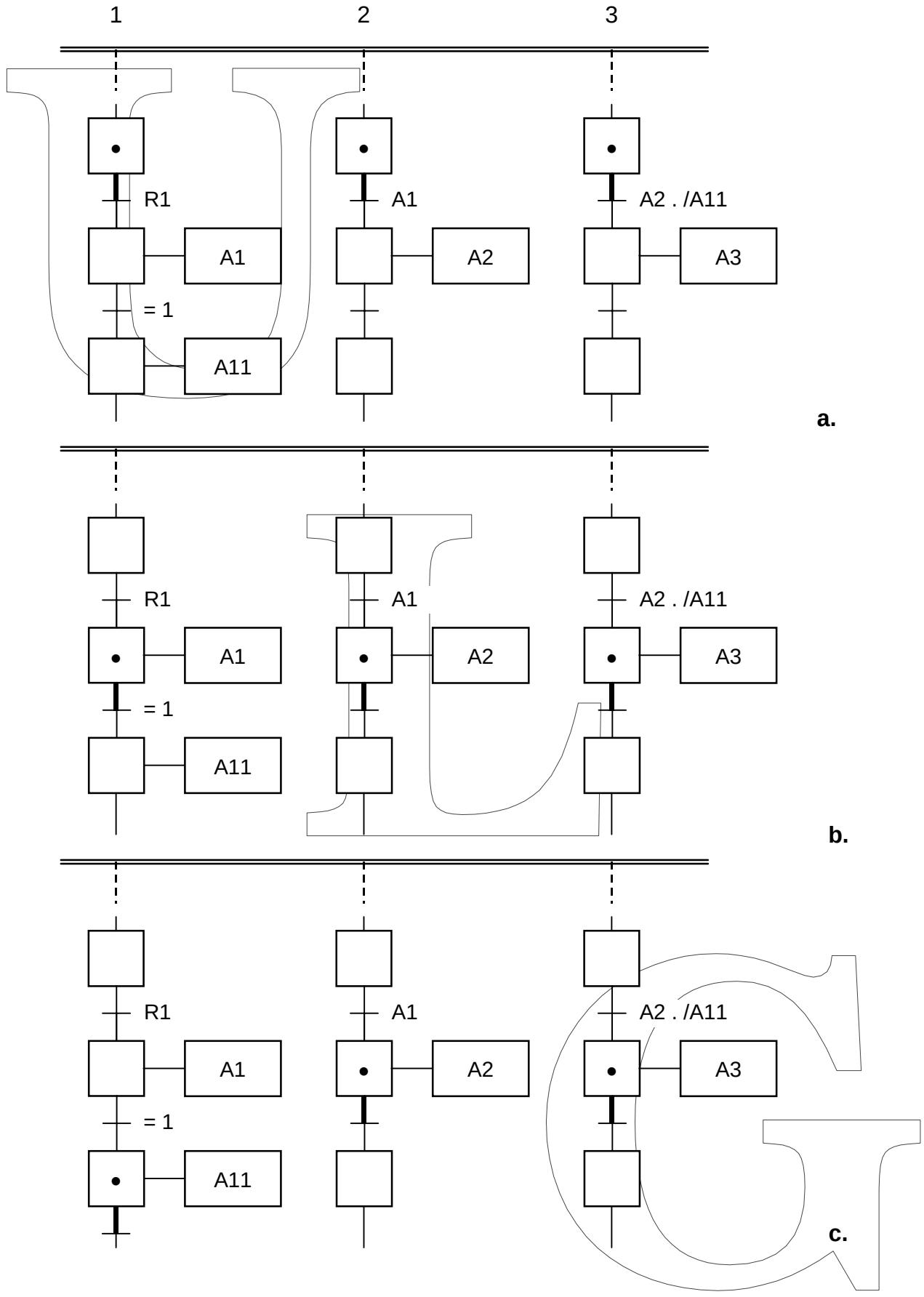


Figure 6.25. Branches parallèles interactives – Scrutation dans l'ordre 1 – 2 – 3

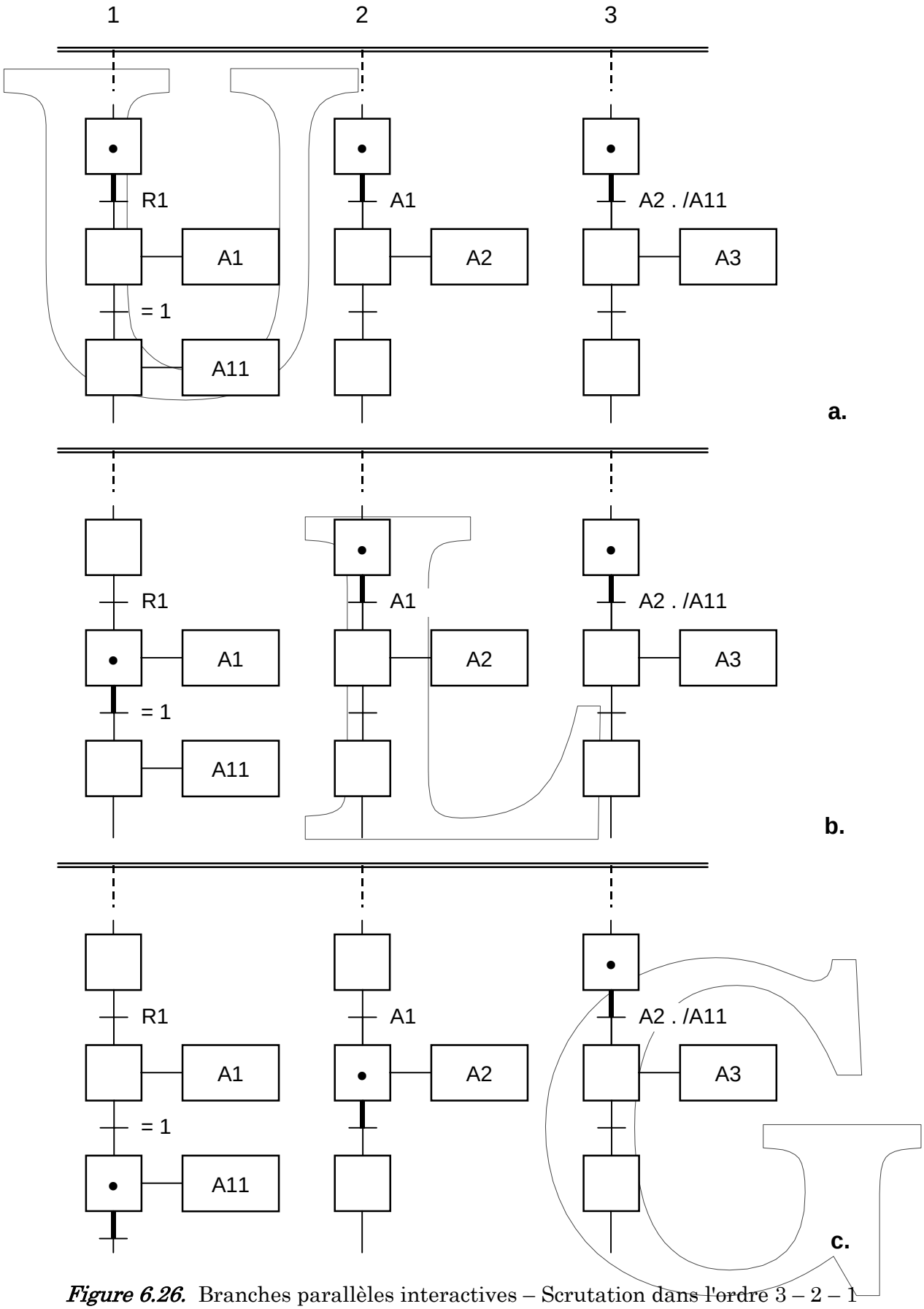


Figure 6.26. Branches parallèles interactives – Scrutation dans l'ordre 3 – 2 – 1

– **Problèmes de synchronisation**

Un problème tout à fait semblable au précédent se pose lorsque l'on utilise des variables d'étape pour synchroniser des GRAFCET se déroulant en parallèle ou pour assurer la protection de ressources communes.

Nous reprendrons, ici, le cas du GRAFCET de synchronisation exposé au paragraphe 6.5.2. (figure 6.27.). Supposons qu'on vienne d'arriver dans la situation indiquée à la figure 6.27.a. Supposons aussi que l'ordre de scrutation des GRAFCET soit 1, 2, 3. Au cycle d'automate qui suit, le GRAFCET 1 commence par passer à l'étape 13 puisque la réceptivité X1 est vraie, le GRAFCET 2, scruté en second, trouve aussi une réceptivité vraie puisque X1 est toujours vraie et que /X12 est devenue vraie. Enfin le GRAFCET 3, scruté en dernier, passe à l'étape 2 mais il est trop tard, les deux chariots sont en route vers la voie commune !

Dans le cas présent, il est assez facile de contourner la difficulté. Il suffit, en effet, lorsqu'un des GRAFCET prend le contrôle de la ressource commune, qu'il fasse lui-même et dans le même temps évoluer le GRAFCET de synchronisation.

En d'autres termes, les instructions qui assurent la transition 1 2 du GRAFCET devront être écrites deux fois : une fois après celles qui assurent la transition 12 13 du GRAFCET 1 et une fois après celles qui assurent la transition 22 23 du GRAFCET 2. De cette manière, on s'affranchit complètement de l'ordre de scrutation des GRAFCET, du moins en ce qui concerne la protection de la ressource commune.

6.6.3. ORGANISATION DES PROGRAMMES

D'une manière générale, on préfère séparer la partie du programme relative à l'évolution du GRAFCET et la partie relative au calcul des actions. Celle-ci sera évidemment exécutée en second, afin de refléter l'état le plus récent du GRAFCET.

Cette manière de faire permet de regrouper, en une seule équation, toutes les conditions qui affectent une même sortie. Ceci est évidemment de nature à faciliter la mise au point du programme. De même, cela permet d'introduire plus facilement les différents types d'actions décrites au paragraphe 6.3.2..

La figure 6.28.a. montre le cas d'une action mémorisée. Au lieu de devoir redéfinir l'action A à chacune des étapes 2 à 4, il suffira d'écrire l'équation de sortie unique de la figure 6.28.b.

De plus, avec cette organisation, il sera aussi plus facile de contrôler le comportement des sorties dans des situations particulières telles que l'arrêt d'urgence. La figure 6.28.c. correspond au cas d'une remise à zéro de la sortie A en cas d'arrêt d'urgence. Dans l'hypothèse de la figure 6.28.a., il aurait fallu introduire la condition d'arrêt d'urgence à chacune des étapes 2, 3 et 4.

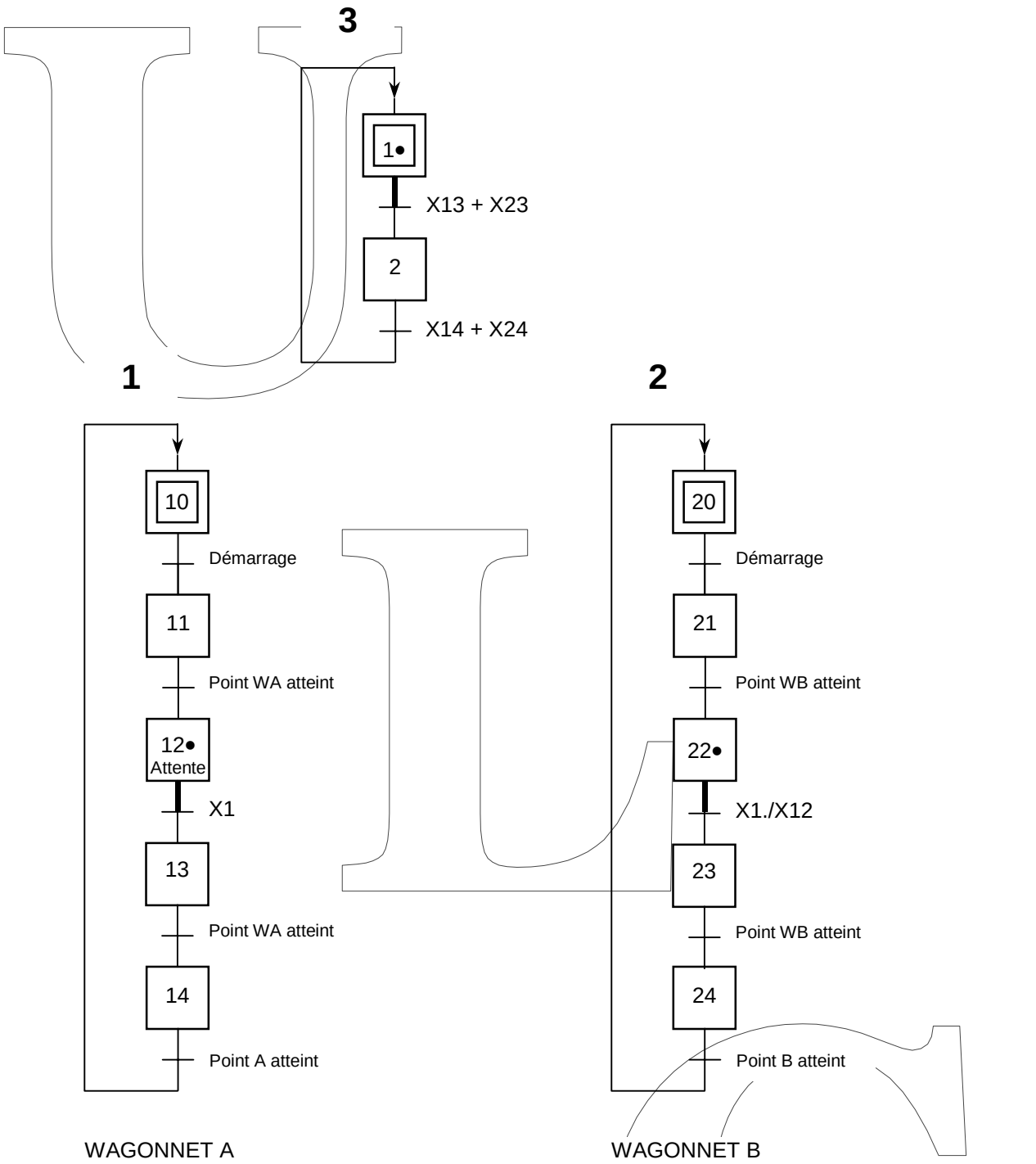


Figure 6.27. GRAFCET de synchronisation
a. Etat initial

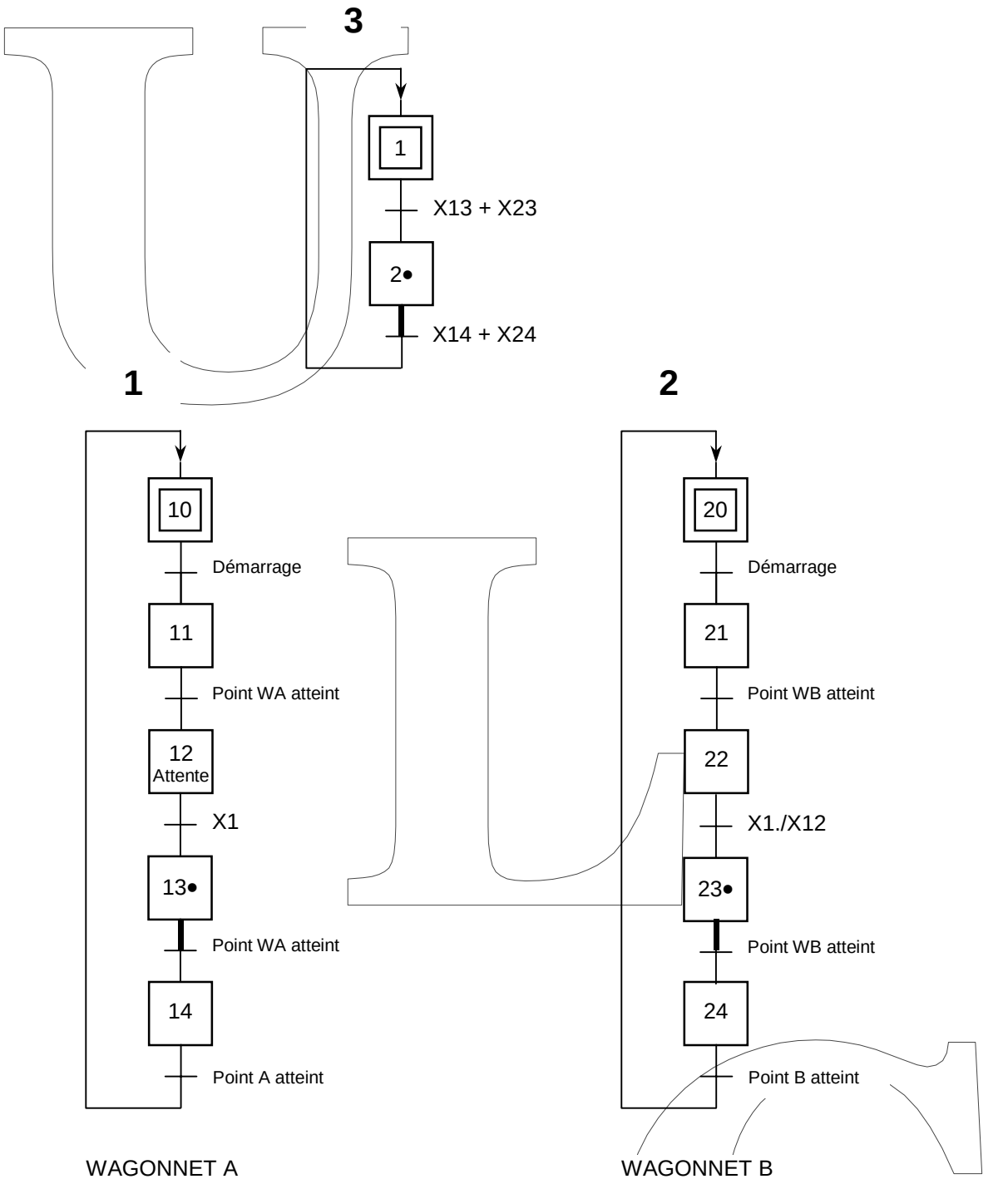


Figure 6.27. GRAFCET de synchronisation
b. Evolution (fautive) dans le cas d'une scrutation dans l'ordre 1 – 2 - 3

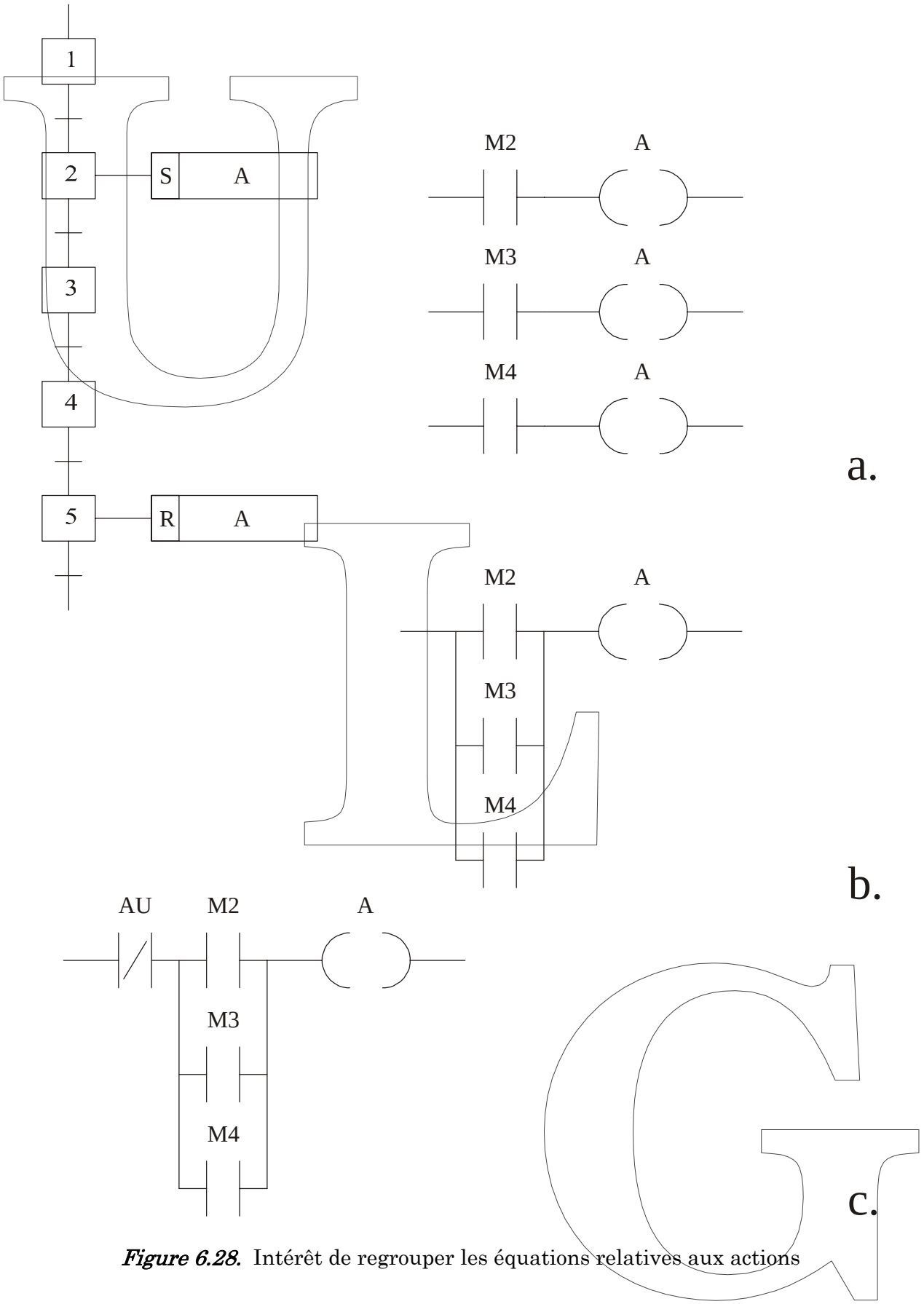


Figure 6.28. Intérêt de regrouper les équations relatives aux actions

6.6.4. PRINCIPALES METHODES DE TRANSPOSITION

L'organisation décrite ci-dessus étant admise, différentes approches sont alors envisageables pour gérer l'évolution du GRAFCET. Nous présentons ci-dessous quelques exemples utilisés dans la pratique mais qui ne sont certainement pas exhaustifs.

a. Calcul préalable des conditions de transition

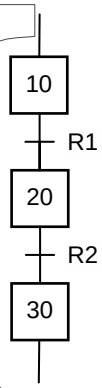
Dans cette solution, les conditions de transition pour toutes les étapes sont calculées en début de programme et mémorisée dans des bits internes. Ces bits internes sont ensuite utilisés en tant que réceptivités dans la description de l'évolution du GRAFCET.

Ainsi, si une étape est activée, on est assurée qu'elle ne pourrait être désactivée dans le même cycle d'automate. En effet, la réceptivité qui la verrouille est nécessairement fausse puisque, au moment où celle-ci a été calculée, l'étape n'était pas active par hypothèse.

La présente méthode offre aussi l'avantage de pouvoir "forcer" facilement l'évolution d'un GRAFCET en phase de mise au point. Au lieu de devoir "forcer" toutes les variables intervenant dans une réceptivité, il suffira en effet de "forcer" le bit de transition.

La figure 6.29. donne un exemple de la méthode programmée en langage IL normalisé.

– Exemple de GRAFCET



– Calcul des conditions de transition

condition de transition 10 - 20

LD	M10	Si l'étape 10 est active
AND	R1	et que la réceptivité R1 est vraie
ST	M11	activer la condition de transition M11
LD	M20	condition de transition 20 - 30
AND	R2	Si l'étape 20 est active
ST	M21	et que la réceptivité R2 est vraie
•		activer la condition de transition M21
•		
•		

– Calcul de l'évolution

transition 10, 20

LD	M10	Si l'étape 10 est active
AND	M11	et que la condition de transition est vraie
S	M20	alors activer l'étape 20
R	M10	et désactiver l'étape 10

transition 20, 30

LD	M20	Si l'étape 20 est active
AND	M21	et que la condition de transition est vraie
S	M30	alors activer l'étape 30
R	M20	et désactiver l'étape 20
•		
•		
•		

– Calcul des actions

•
•
•

Figure 6.29. Calcul préalable des conditions de transition

b. Utilisation de sauts conditionnels

Si le langage automate comporte des instructions de sauts conditionnels, on peut les utiliser pour essayer d'éviter les calculs inutiles. Comme le montre l'exemple de la figure 6.30., on saute d'étape en étape dans le GRAFCET jusqu'à tomber sur une étape active. Dans ce cas, on calcule la réceptivité associée; si elle est vraie, on fait évoluer le GRAFCET. Qu'elle soit vraie ou fausse, on saute ensuite directement à la partie du programme qui calcule les sorties, ce qui garantit que, si une nouvelle étape est activée, elle le restera au moins pendant un cycle de l'automate.

Cette méthode est moins favorable que la précédente en ce qui concerne le forçage des réceptivités car il faudra en effet ici forcer de manière adéquate toutes les variables qui interviennent dans R1.

On pourrait évidemment envisager un calcul préalable des réceptivités avec mémorisation dans des bits internes mais dans ce cas, évidemment, on perdrait une grande partie de l'intérêt des sauts conditionnels, surtout si les réceptivités sont des expressions complexes.

```
                                    transition 10 - 20
ET10 :    LDN       M10             Si l'étape 10 n'est pas active
          JMPC      ET20            sauter à l'étape suivante
          LD        R1              Sinon tester la réceptivité et, si elle est vraie
          S         M20             activer l'étape 20
          R         M10             désactiver l'étape 10
          JMP       ACTION          sauter au calcul des actions

                                    transition 20 - 30
ET20 :    LDN       M20             Si l'étape 20 n'est pas active
          JMPC      ET30            sauter à l'étape suivante
          LD        R2              Sinon tester la réceptivité et, si elle est vraie
          S         M30             activer l'étape 30
          R         M20             désactiver l'étape 20
          JMP       ACTION          sauter au calcul des actions
           •
           •
           •

ACTION :                            calcul des actions
           •
           •
           •
```

Figure 6.30. Utilisation de sauts conditionnels

c. Utilisation d'un pointeur de phase

Pour les automates disposant d'instructions de sauts conditionnels et de calcul sur mots, on peut encore procéder d'une autre manière. Il faut commencer par s'assurer que les GRAFCET décrivant l'automatisme ne comportent pas de branches parallèles. Si c'est le cas, on les remplacera par autant de GRAFCET simples, synchronisés comme expliqué au paragraphe 6.5.

A chaque GRAFCET, on associera alors un mot de donnée, appelé pointeur de phase, dont la valeur courante indiquera le numéro de l'étape active. Les conditions de transition comporteront donc une comparaison entre le numéro de l'étape active et la valeur courante du pointeur de phase. En cas d'évolution, c'est le numéro de la nouvelle étape active qui deviendra la valeur courante du pointeur de phase.

Cette méthode est certainement plus gourmande en temps de calcul que la précédente, par exemple, puisqu'elle implique des opérations sur mots. Par contre, elle présente des avantages intéressants qui ont déjà été évoqués au paragraphe 6.6.1.

D'une part, elle garantit de manière intrinsèque qu'il n'y aura jamais qu'une seule étape active à la fois dans un GRAFCET. Ceci peut s'avérer utile en phase de test lorsque l'on procède à des forçages d'étapes et/ou de réceptivités. Dans les autres méthodes, basées sur les bits, il faut en effet être très vigilant lors de tels forçages pour garder le GRAFCET dans un état cohérent.

D'autre part, elle permet de suivre très facilement l'évolution d'un GRAFCET : par simple consultation de la valeur de son pointeur de phase, on connaît le numéro de l'étape active.

Une remarque analogue à celle du cas précédent peut être faite à propos du forçage des réceptivités, avec les mêmes conclusions.

La figure 6.31. montre la transposition du GRAFCET de l'exemple en utilisant un pointeur de phase d'abord en langage IL puis en langage ST. Ce dernier est manifestement mieux adapté au problème.

a. Programmation en langage IL

Pointeur de phase MW1

```
                                        transition 10 - 20
ET10 :   LD      10        } Si le pointeur de phase n'est pas égal à
         NE      MW1       } 10
         JMPC    ET20      sauter à l'étape suivante
         LD      R1        sinon, tester la réceptivité
         JMPCN   ACTION    sauter au calcul des actions si elle est fausse
         LD      20        } sinon mettre 20 dans le pointeur de
         ST      MW1       } phase
         JMP     ACTION    sauter au calcul des actions

                                        transition 20 - 30
ET20 :   LD      20        } Si le pointeur de phase n'est pas égal à
         NE      MW1       } 20
         JMPC    ET30      sauter à l'étape suivante
         LD      R2        sinon, tester la réceptivité
         JMPCN   ET20      sauter au calcul des actions si elle est fausse
         LD      30        } sinon mettre 30 dans le pointeur de
         ST      MW1       } phase
         JMP     ACTION    sauter au calcul des actions
         •
         •
         •

ACTION :                   calcul des actions
         •
         •
         •
```

b. Programmation en langage ST

```
                        transition 10 - 20
    ! IF [MW1=10] . R1
            THEN 20 → MW1 ; JUMP ACTION

                        transition 20 - 30
    ! IF [MW1=20] . R2
            THEN 30 → MW1 ; JUMP ACTION
        •
        •
        •
ACTION :
        •
        •
        •
```

Figure 6.31. Illustration de la méthode du pointeur de phase

Chapitre 7

GEMMA

Guide d'Etude des Modes de Marches et d'Arrêts

Ce chapitre est quasi intégralement repris du document original de l'ADEPA (Association pour le Développement de la Production Automatisée – France) – 1981.

7.1. INTRODUCTION

Le GEMMA est un « outil-méthode » permettant de mieux définir les Modes de Marches et d'Arrêts d'un système industriel automatisé.

Il est constitué pour l'essentiel d'un Guide Graphique qui est rempli progressivement lors de la conception du système.

On commence par recenser les « Modes » ou « Etats » de fonctionnement du système en utilisant des critères clairement définis, indépendants à la fois du type de système étudié et de la technologie de commande.

On établit ensuite les liaisons possibles entre ces « Modes » ou « Etats », en explicitant les conditions d'évolution.

Le document ainsi établi matérialise l'analyse détaillée des Modes de Marches et d'Arrêts du système: le GEMMA est un outil d'aide à l'analyse.

Il reste alors à en déduire le GRAFCET « complété » afin de terminer la définition des spécifications de la Partie Commande, y compris le pupitre et les capteurs supplémentaires éventuellement nécessaires : le GEMMA est un outil d'aide à la synthèse du cahier des charges. Enfin, ce document accompagne la vie du système: le GEMMA est un outil d'aide à la conduite de la machine, à sa maintenance ainsi qu'à son évolution.

7.1.1. LE BESOIN D'OUTILS-METHODES

L'automatisation de la production exige la réalisation de machines automatiques de plus en plus complexes mais en même temps offrant plus de sécurité et une plus grande souplesse d'emploi.

Par ailleurs, un système automatisé est un carrefour : pour le concevoir, le réaliser, le mettre au point, le réparer, le faire évoluer, des hommes ayant des compétences et des motivations diverses doivent se comprendre.

Par conséquent, le Développement de la Production Automatisée passe par la création, la promotion et la pratique d'outils-méthodes utilisant des concepts clairement définis et facilitant la conception, la réalisation et l'exploitation des machines automatiques.

Ressenti d'abord au niveau de la description du fonctionnement, ce besoin d'outils-méthodes a donné naissance au GRAFCET. Des outils-méthodes complémentaires se révèlent maintenant nécessaires.

7.1.2. LE BESOIN D'UN VOCABULAIRE PRECIS

Dans le domaine des Modes de Marches et d'Arrêts, le vocabulaire pratiqué est imprécis et parfois même contradictoire : il conduit à des incompréhensions graves.

Par exemple, très utilisés des praticiens les termes "MARCHE AUTOMATIQUE, SEMI-AUTOMATIQUE, MANUELLE" recouvrent des notions toutes relatives : selon son expérience et son environnement, chaque technique, chaque Société et chaque individu leurs donnent des significations différentes. Ainsi, une Marche semi-automatique sera souvent considérée comme une Marche automatique pour des systèmes moins complexes.

Par ailleurs, la liste des termes utilisés pour dénommer les Modes de Marches s'allonge sans pour autant apporter d'idée unificatrice.

Quelques termes utilisés, à titre d'exemple : Marche "pas à pas", Marche "coup par coup", Marche "d'intervention", Marche "réglage", Marche "cycle par cycle", ... Arrêt "d'urgence", Arrêt "figeage", Arrêt "fin de mouvement", Arrêt "à l'étape" ...

Par conséquent, pour que toutes les personnes concernées se comprennent, il est indispensable de définir un vocabulaire précis en le rattachant a des critères fondamentaux, indépendants du genre de l'équipement et de la technologie de réalisation.

7.1.3. LE BESOIN D'UNE APPROCHE GUIDEE

Le plus souvent lors de l'étude d'un système automatisé, les besoins en Modes de Marches et d'Arrêts sont peu ou mal exprimés. Après réalisation du système, c'est alors au prix de modifications et tâtonnements coûteux qu'il faut répondre à ces besoins essentiels.

Le concepteur a donc besoin d'une approche guidée et systématique, du genre "check-list", pour tout prévoir dès l'étude et envisager les conséquences, tant pour la partie opérative que pour la partie commande du système à réaliser.

Le GRAFCET permet une expression précise de certains Modes de Marches et d'Arrêts. Seul, il ne permet cependant pas l'approche systématique et globale nécessaire.

Pour les Modes de Marches et d'Arrêts, le GEMMA répond à ces besoins : c'est un outil-méthode qui définit un vocabulaire précis, en proposant une approche guidée systématique pour le concepteur : le guide graphique GEMMA.

7.2. LES CONCEPTS DE BASE

Le GEMMA constitue une méthode d'approche des Modes de Marches et d'Arrêts, fondée sur quelques concepts de base matérialisés par un Guide graphique.

La démarche proposée comporte 2 temps :

1. le recensement des différents modes envisagés, et la mise en évidence des enchaînements qui les relient;

2. la détermination des conditions de passage d'un mode à l'autre.

Commençons par découvrir les concepts de base du GEMMA.

7.2.1. CONCEPT N° 1 : LES MODES DE MARCHES SONT VUS PAR UNE PARTIE COMMANDE EN ORDRE DE MARCHE

Tout système peut être décomposé fonctionnellement en 2 parties qui coopèrent : la partie opérative et la partie commande.

On entendra ici par "partie opérative", tout ce qui n'est pas l'automatisme en cours d'étude (que l'on appellera "partie commande"). La partie opérative peut donc inclure, outre les mécanismes ou dispositifs divers du système, tout ce qui relève de l'environnement du système (opérateurs humains ou autres parties commande).

Les Modes de Marches ou d'Arrêts concernent le système, c'est-à-dire l'ensemble "Partie opérative" + "Partie commande", mais tels qu'ils sont vus par la partie commande.

Nous nous limitons volontairement à cette hypothèse d'une description vue d'une partie commande en ordre de marche, comme étant de loin la plus fréquente et surtout la moins complexe à aborder. Ceci revient à écarter les cas des parties commande à sécurité passive ou tolérant des pannes, ainsi que les interventions humaines directement sur la partie opérative sans que, par un capteur, la commande en soit informée.

Par conséquent, ceci suppose que la partie commande est en ordre de marche, avec tous ses organes convenablement alimentés, même si la partie opérative est hors énergie, ou en défaut, ou à l'arrêt.

Le Guide graphique GEMMA est donc constitué de 2 zones (figure 7.1.) :

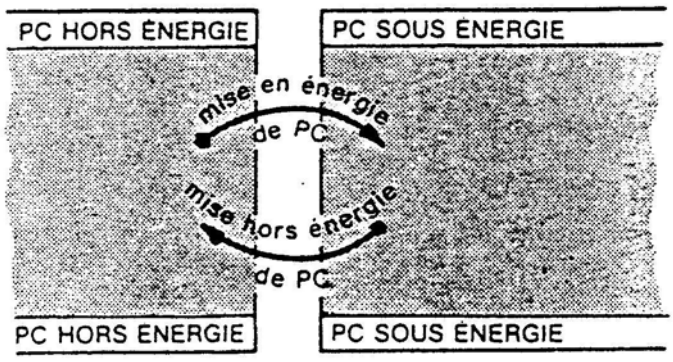


Figure 7.1. Les 2 zones du GEMMA

– une zone correspondant à l'état inopérant de la partie commande (PC) → elle ne figure que pour la forme;

– une zone permettant de décrire ce qui se passe lorsque la partie commande (PC) fonctionne normalement → c'est la zone qui couvre la quasi-totalité du guide graphique.

La mise sous énergie et dans un état initial de la partie commande permet de franchir la frontière entre les 2 zones. On trouve alors la partie opérative (PO) hors ou sous énergie. On peut franchir la frontière dans l'autre sens par coupure d'énergie sur PC.

7.2.2. CONCEPT N° 2 : LE CRITERE "PRODUCTION"

Un système industriel automatisé est conçu fondamentalement pour produire une certaine valeur ajoutée. C'est la justification principale de la construction du système.

Cette production de valeur ajoutée peut être très variée : modification des produits, contrôle, manutention, stockage, et cependant, l'expérience montre qu'on peut toujours la caractériser pour un système donné de façon précise et unique.

Ce sera donc notre premier critère : on dira que le système est "en production" si la valeur ajoutée pour laquelle le système a été conçu est obtenue, on dira que le système est "hors production" dans le cas contraire (figure 7.2.).

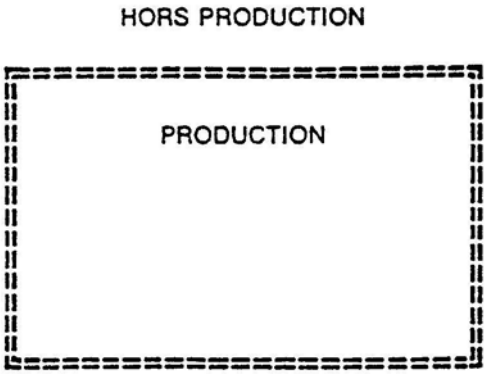


Figure 7.2. Le critère "production"

7.2.3. CONCEPT N° 3 : LES 3 GRANDES FAMILLES DE MODES DE MARCHES ET D'ARRETS

On peut classer en 3 grandes familles les Modes de Marches et d'Arrêts d'un système automatisé :

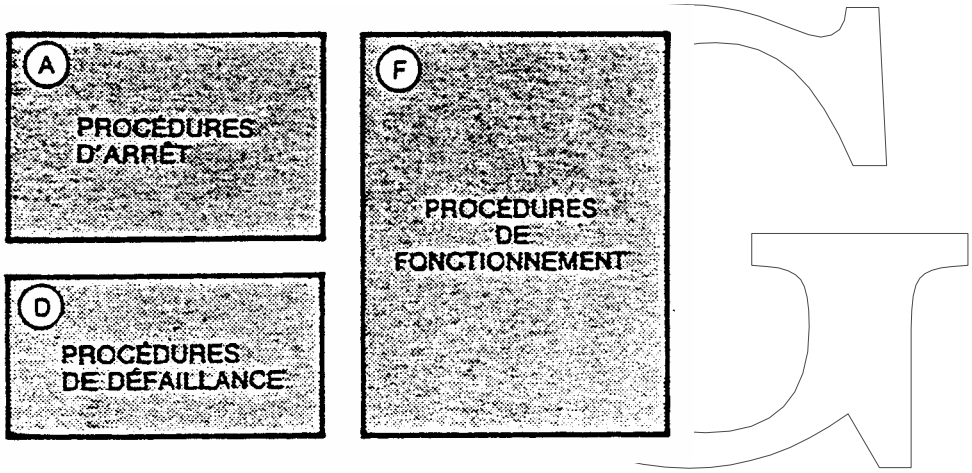


Figure 7.3. Les 3 grandes familles de modes de marches et d'arrêts

1 - Famille (F)

On groupe dans cette famille tous les Modes ou Etats qui sont indispensables à l'obtention de la valeur ajoutée, ou, autrement dit, tous ceux sans lesquels on ne sait pas techniquement ou fonctionnellement obtenir la valeur ajoutée pour laquelle la machine est prévue.

Ces modes sont regroupés dans le guide graphique dans une zone F "Procédures de **F**onctionnement".

Notons que l'on ne "produit" pas forcément dans tous les modes de cette famille : ils peuvent être préparatoires à la production, ou servir aux réglages ou aux tests par exemple. Ils n'en demeurent pas moins indispensables : on ne sait pas faire du moulage en coquille sans préchauffer l'outillage (marche de préparation), ni effectuer des opérations d'usinage sans réglages ni contrôles périodiques.

2 - Famille (A)

Une machine automatique fonctionne rarement 24 heures sur 24 : il est nécessaire de l'arrêter de temps à autre, pour des raisons **extérieures** au système, tout simplement parce que la journée est finie, ou bien par manque d'approvisionnement, par exemple.

L'expérience montre qu'il est souvent délicat de concevoir les équipements automatiques pour qu'ils arrêtent correctement le processus qu'ils contrôlent.

On classera dans cette famille tous les Modes conduisant à (ou traduisant) un état d'arrêt du système pour des raisons extérieures. Ils sont regroupés dans une zone A "procédures d'**A**rrêt" du guide graphique.

Notons qu'on peut produire en étant dans une procédure d'arrêt, si cet arrêt est demandé à la fin d'une opération, par exemple.

3 - Famille (D)

Il est rare qu'un Système fonctionne sans incident pendant toute sa vie : il est indispensable de prévoir les défaillances.

On regroupera dans cette famille tous les modes conduisant à (ou traduisant) un état d'arrêt du système pour des raisons **intérieures** au système, autrement dit, à cause de défaillances de la partie opérative. Ces modes sont représentés dans une zone D "procédures en **D**éfaillance" du guide graphique.

On a écarté d'emblée les cas de défaillance de la PC, pour lesquels on considère qu'on ne peut rien dire, dans la majorité des cas.

7.2.4. LES "RECTANGLES-ETATS"

Sur le guide graphique GEMMA (voir § 7.3.), chaque Mode de Marche ou d'Arrêt désiré peut être décrit dans l'un des "rectangles-états" prévus à cette fin (figure 7.4.).

La position d'un rectangle-état sur le guide graphique définit :

– son appartenance à l'une des 3 familles, procédure de fonctionnement, d'arrêt ou de défaillance;

– le fait qu'il soit "en" ou "hors production".

Le rectangle-état porte une désignation de Marche ou d'Arrêt utilisant un vocabulaire ne pouvant prêter à confusion : c'est à dessein qu'ont été écartées, pour ces dénominations générales, les expressions communément utilisées pour dénommer les Modes de Marches ou d'Arrêts, qui sont souvent comprises différemment de chacun.

Par contre, ces expressions usitées pourront être employées dans le "langage machine", précisant dans le cadre des "rectangles-états" les Modes de Marches ou d'Arrêts pour une machine déterminée. Par exemple, dans le "rectangle-état" < Production normale >, on trouvera la précision : "moulage semi-automatique".

Citons des exemples :

- L'état F1, < Production normale >, est celui pour lequel la machine a été conçue. Représenté par un grand rectangle renforcé, il se trouve bien entendu dans l'intersection des zones "procédures de fonctionnement" et "en production".

- L'état A2, < Arrêt demandé en fin de cycle > est représenté par un rectangle dans l'intersection des zones "procédures d'arrêt" et "en production". En effet, dans cet état, la machine continue encore de produire, bien que l'arrêt soit demandé.

- L'état Al, < Arrêt dans l'état initial >, est représenté par un rectangle dans la zone "procédures d'arrêt" et hors de la zone "production", puisque la machine est maintenant arrêtée.

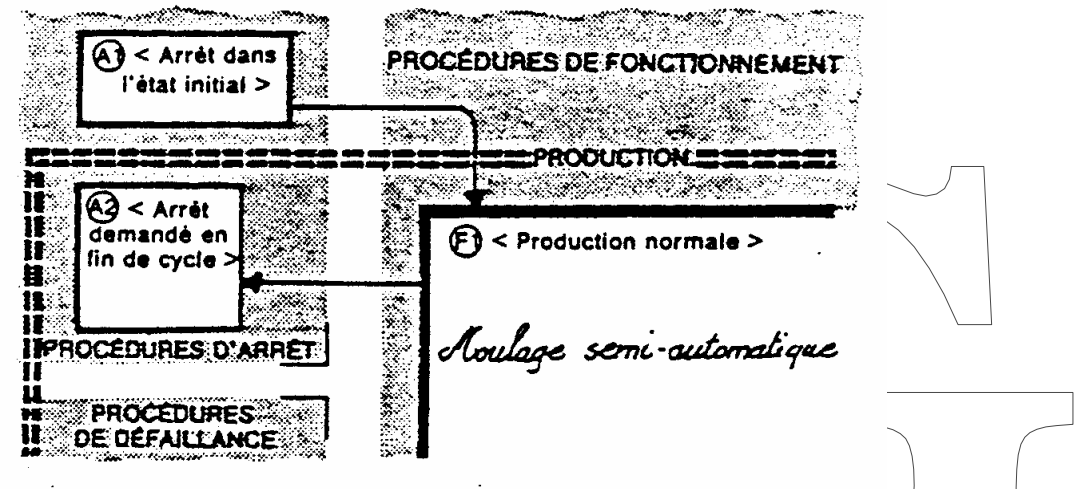


Figure 7.4. Les rectangles-états

7.3. LES ETATS DE MARCHES ET D'ARRETS

Le Guide graphique GEMMA (voir ci-dessous) porte les "rectangles-états" dans lesquels seront exprimés les différents états de Marches et d'Arrêts pris par la machine.

En pratique, pour une machine donnée, on ne choisira parmi les états proposés par le guide que ceux qui sont nécessaires, et on précisera le nom de chacun des états retenus, à l'intérieur du "rectangle-état" correspondant.

Pour effectuer ce choix, il est nécessaire de bien comprendre la signification de chacun des états de Marches et d'Arrêts proposés par le guide graphique : c'est l'objet de ce chapitre.

7.3.1. UN "RECTANGLE-ÉTAT" TYPE

F2 est le repère du "rectangle-état", F signifie que l'état proposé fait partie des procédures de Fonctionnement. (A pour procédures d'Arrêt, et D pour procédures en Défaillance) (figure 7.5).

< Marche de préparation > est la dénomination générale de l'état proposé. Les <—> indiquent l'emploi du "langage général".
Les principales possibilités de liaisons d'état à état sont suggérées.

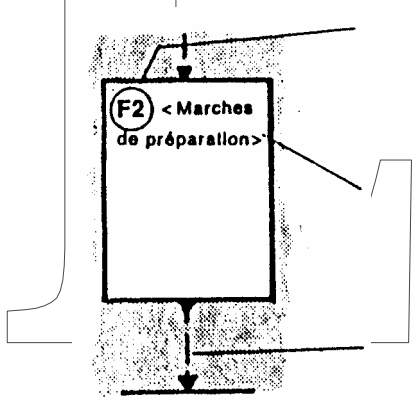


Figure 7.5. Exemple de rectangle-état < Marche de préparation >

7.3.2. UTILISATION D'UN "RECTANGLE-ÉTAT"

I - L'état est retenu pour la machine. Il est précisé dans le "langage machine" (en écriture cursive sur la figure).

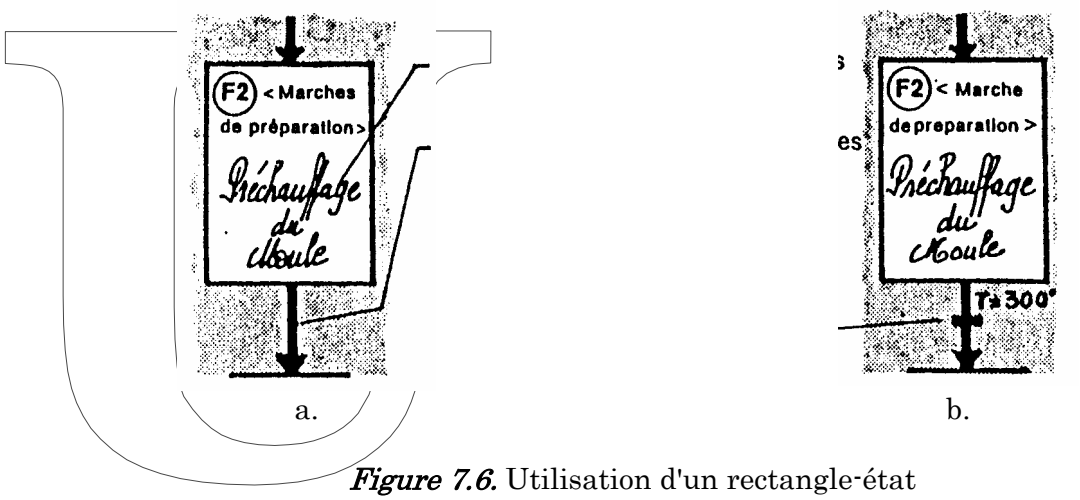
Les liaisons retenues sont indiquées en traits forts (figure 7.6.a).

Le "langage général" est celui proposé par le guide graphique : il s'applique à toutes les machines et systèmes automatisés. Le "langage machine" est celui qui est communément pratiqué pour la machine particulière que l'on considère.

"Langage général" et "langage machine" se complètent dans chaque "rectangle-état", pour obtenir une définition précise et claire pour tous, des Modes de Marches et d'Arrêts.

II - Les conditions d'évolution entre états sont indiquées sur les liaisons (figure 7.6.b).

Dans le cas présent, température de 300°.



a. b.

Figure 7.6. Utilisation d'un rectangle-état

7.3.3. LES ÉTATS "F "

Ce sont les états de marches situés dans la zone "procédures de fonctionnement" du guide graphique GEMMA.

F1 < Production normale >

Dans cet état, la machine produit normalement : c'est l'état pour lequel elle a été conçue. C'est à ce titre que le "rectangle-état" a un cadre particulièrement renforcé. On peut souvent faire correspondre à cet état un GRAFCET que l'on appelle GRAFCET de base.
Remarque : à cet état ne correspond pas nécessairement une marche automatique.

F2 < Marche de préparation >

Cet état est utilisé pour les machines nécessitant une préparation préalable à la production normale : préchauffage de l'outillage, remplissage de la machine, mises en routes diverses, etc.

F3 < Marche de clôture >

C'est l'état nécessaire pour certaines machines devant être vidées, nettoyées, etc., en fin de journée ou en fin de série.

F4 < Marche de vérification dans le désordre>

Cet état permet de vérifier certaines fonctions ou certains mouvements sur la machine, sans respecter l'ordre du cycle.

F5 < Marche de vérification dans l'ordre >

Dans cet état, le cycle de production peut être exploré au rythme voulu par la personne effectuant la vérification, la machine pouvant produire ou ne pas produire.

F6 < Marche de test >

Les machines de contrôle, de mesure, de tri..., comportent des capteurs qui doivent être réglés ou étalonnés périodiquement : la < Marche de test > F6 permet ces opérations de réglage ou d'étalonnage.

7.3.4. LES ÉTATS "A "

Situés dans la zone "procédures d'Arrêt de la partie opérative", ces états correspondent à des arrêts normaux ou à des marches conduisant à des arrêts normaux.

A1 <Arrêt dans l'état initial >

C'est l'état "repos" de la machine. Il correspond en général à la situation initiale du GRAFCET : c'est pourquoi, comme une étape initiale, ce "rectangle-état" est entouré d'un double cadre.
Pour une étude plus facile de l'automatisme, il est recommandé de représenter la machine dans cet état initial.

A2 <Arrêt demandé en fin de cycle >

Lorsque l'arrêt est demandé, la machine continue de produire jusqu'à la fin du cycle. A2 est donc un état transitoire vers l'état A1.

A3 < Arrêt demandé dans état déterminé >

La machine continue de produire jusqu'à un arrêt en une position autre que la fin de cycle : c'est un état transitoire vers A4.

A4 < Arrêt obtenu >

La machine est alors arrêtée en une autre position que la fin de cycle.

A5 < Préparation pour remise en route après défaillance >

C'est dans cet état que l'on procède a toutes les opérations (dégagements, nettoyages,..) nécessaires à une remise en route après défaillance.

A6 < Mise P.O. dans état initial>

La machine étant en A6, on remet manuellement ou automatiquement la Partie Opérative en position pour un redémarrage dans l'état initial.

A7 < Mise P.O. dans état déterminé >

La machine étant en A7, on remet la P.O. en position pour un redémarrage dans une position autre que l'état initial.

7.3.5. LES ÉTATS "D "

Ce sont les états de Marches et d'Arrêts situés dans la zone "procédures ou Défaillances" de la partie opérative.

D1 <Arrêt d'urgence >

C'est l'état pris lors d'un arrêt d'urgence: on y prévoit non seulement les arrêts, mais aussi les cycles de dégagements, les procédures et précautions nécessaires pour éviter ou limiter les conséquences dues à la défaillance.

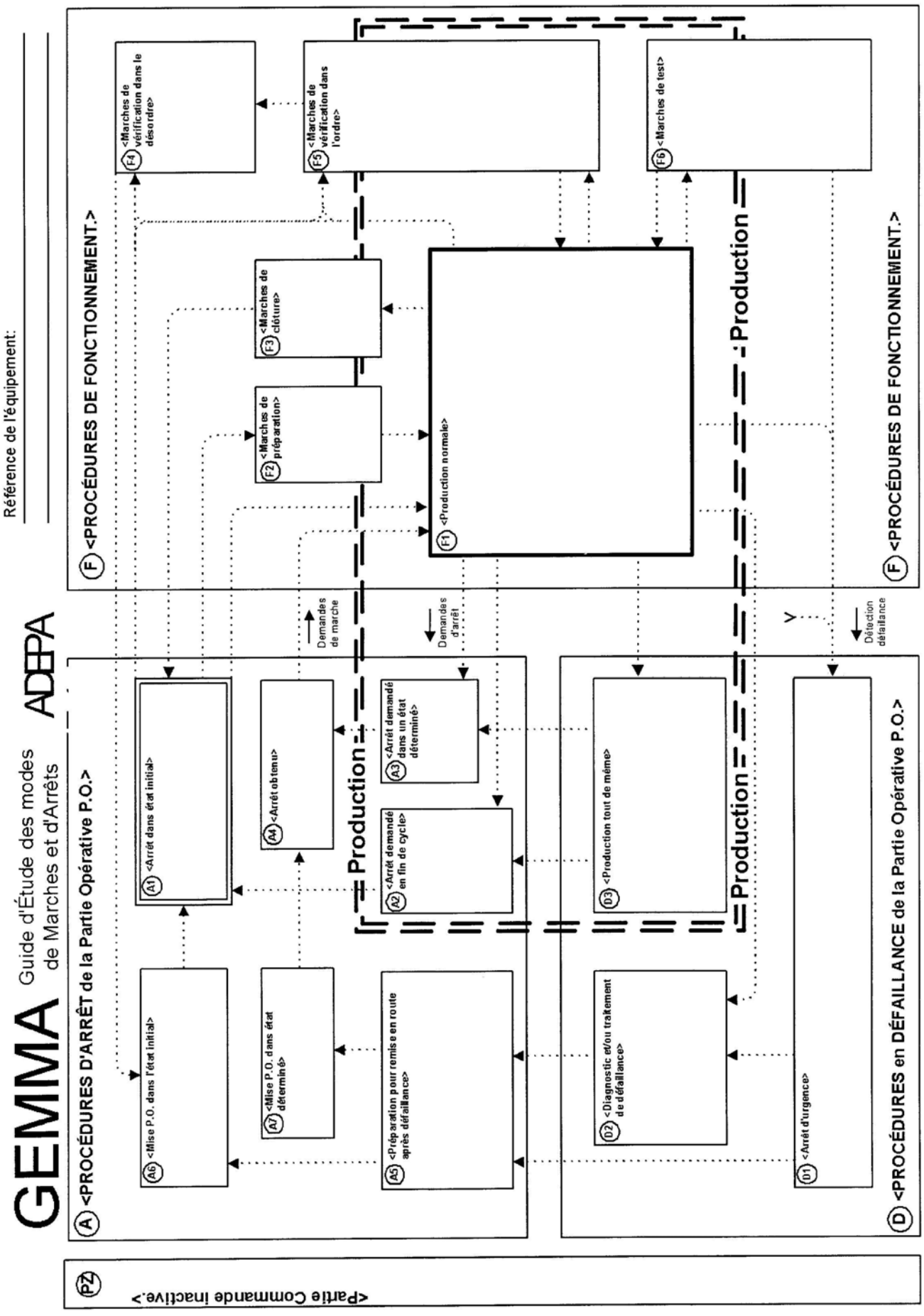
D2 < Diagnostic et/ou traitement de défaillance >

C'est dans cet état que la machine peut être examinée après défaillance et qu'il peut être apporté un traitement permettant le redémarrage.

D3 < Production tout de même >

Il est parfois nécessaire de continuer la production même après défaillance de la machine : on aura alors une "production dégradée", ou une "production forcée", ou une production aidée par des opérateurs non prévus en < Production normale >.

Le tableau de la page suivante donne la structure complète du GEMMA.


GEMMA
Guide d'Étude des modes de Marches et d'Arrêts
ADEPA
Référence de l'équipement:
(F) <PROCÉDURES DE FONCTIONNEMENT.>
(F4) <Marches de vérification dans le désordre>
(F5) <Marches de vérification dans l'ordre>
(F6) <Marches de test>
(F3) <Marches de clôture>
(F2) <Marches de préparation>
(F1) <Production normale>
Production
(A) <PROCÉDURES D'ARRÊT de la Partie Opérative P.O.>
(A1) <Arrêt dans état initial>
(A4) <Arrêt obtenu>
(A3) <Arrêt demandé dans un état déterminé>
(A2) <Arrêt demandé en fin de cycle>
(A6) <Mise P.O. dans l'état initial>
(A7) <Mise P.O. dans état déterminé>
(A5) <Préparation pour remise en route après défaillance>
Demandes de marche
Demandes d'arrêt
(D3) <Production tout de même>
(D2) <Diagnostic et/ou traitement de défaillance>
(D1) <Arrêt d'urgence>
Détection défaillance
(D) <PROCÉDURES en DÉFAILLANCE de la Partie Opérative P.O.>
(PZ) <Partie Commande inactive.>

7.3.6. EXEMPLES D'UTILISATION DU GEMMA

On trouvera ci-après deux exemples d'utilisation du GEMMA d'après des documents du CETIM (Centre Technique des Industries Mécaniques – France).

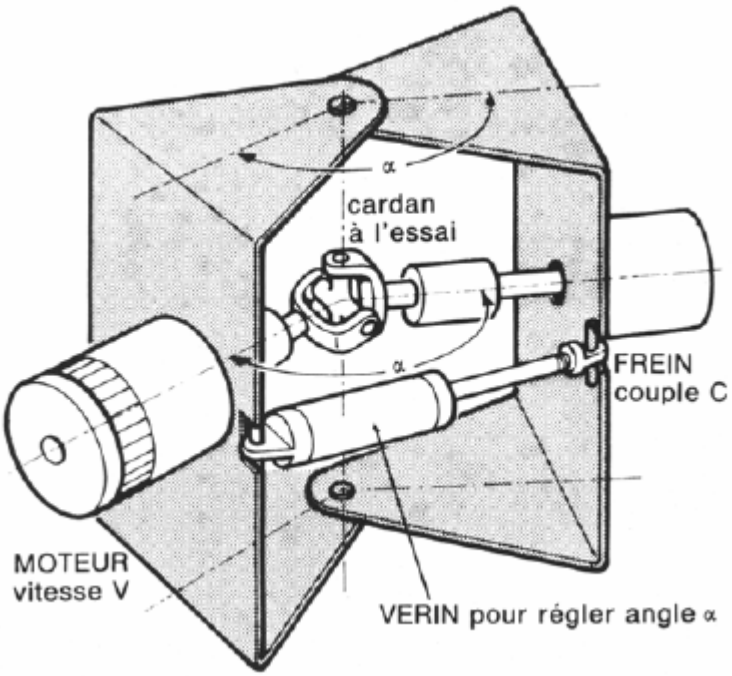
1er exemple introductif : banc d'essai pour transmissions cardan

La machine :

Entièrement automatique, ce banc d'essai est conçu pour simuler le travail de transmissions cardan pour véhicules. La transmission cardan est montée sur une armature articulée permettant de lui donner l'angle α souhaité grâce à l'action d'un vérin hydraulique. Un moteur fait tourner la transmission à la vitesse V souhaitée, et un frein permet de la solliciter avec le couple C voulu.

Les trois facteurs ainsi maîtrisés pour tester la transmission cardan sont donc :
– l'angle α ;
– la vitesse V ;
– le couple C.

D'une très longue durée, les essais tournent 24 h sur 24, selon un programme préenregistré, donnant à tous moments les valeurs de la vitesse V, du couple C, et de l'angle α pris par la transmission au long d'un parcours type : démarrages, virages, arrêts, etc. Périodiquement, le programme demande une dépose et un contrôle de la transmission avant d'autoriser la poursuite des essais.



Banc d'essai d'une transmission cardan

Utilisation du GRAFCET :

Dans cet exemple assez particulier, la production étant assurée uniquement par la lecture d'un programme, le GRAFCET n'a pas été utilisé. Nous verrons dans l'exemple suivant les rôles respectifs du GRAFCET et du GEMMA. Cette application a été choisie comme premier exemple introductif pour faire comprendre plus rapidement l'intérêt du GEMMA dans une première approche.

Utilisation du GEMMA :

Afin de faire tout de suite découvrir l'objectif poursuivi, nous présentons ici directement le GEMMA final (voir page suivante).

A partir du guide graphique présenté plus haut, ce GEMMA a été en fait obtenu en deux temps :

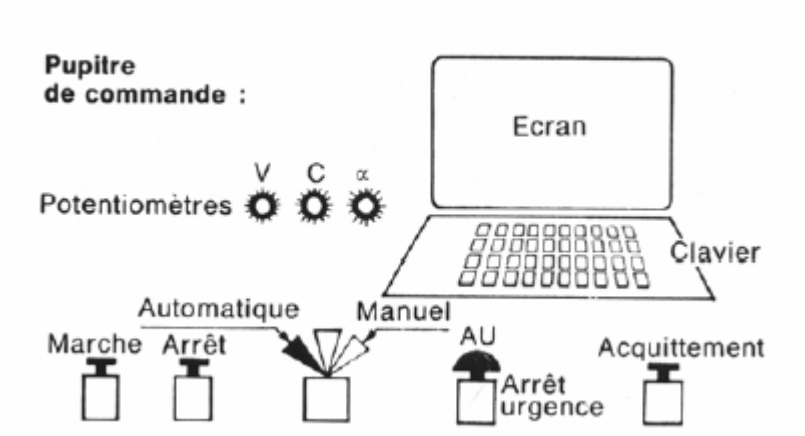
1. Sélection des modes de marches et d'arrêts, en utilisant les rectangles états et en définissant les liaisons.

2. Recherche des conditions d'évolution entre états, en parallèle avec la définition du pupitre de commande.

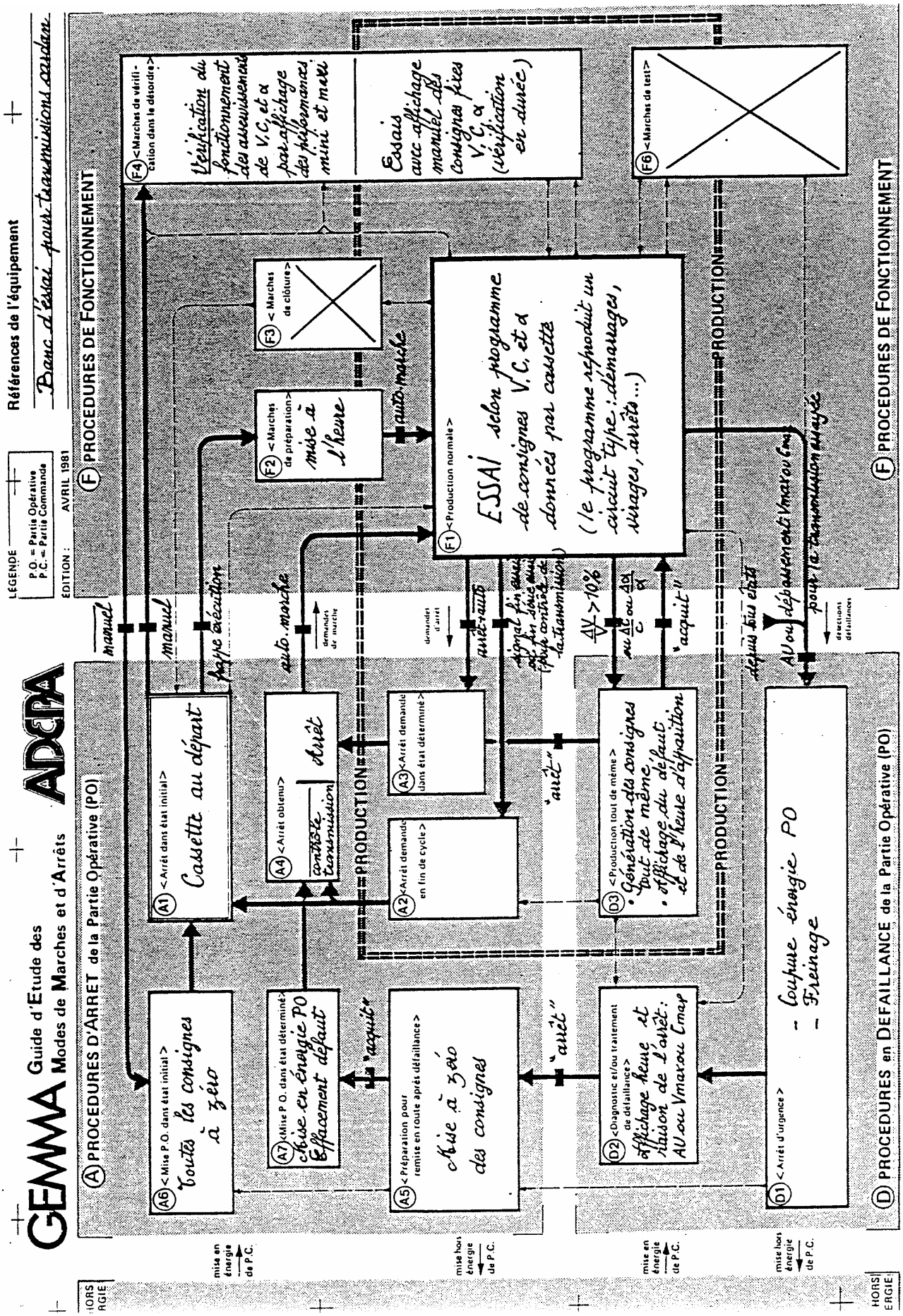
Pour ce premier exemple, contentons-nous d'expliquer les choix effectués, sans suivre dans le détail la méthode d'obtention.

L'état F1, < Production normale >, est la mise en oeuvre du programme, pour déroulement de l'essai. Cet état F1 est atteint depuis l'état initial A1, par la frappe du mot "exécution" sur le clavier du pupitre (voir figure ci-dessous), la traversée de l'état F2 qui permet une mise à l'heure, puis la mise sur "auto" du sélecteur de marche et l'action sur le bouton "marche".

A tout moment, l'arrêt de la machine peut être provoqué manuellement par passage dans les états A3 et A4, en agissant sur le bouton "arrêt" du pupitre.



Le pupitre de commande comporte un clavier et un écran, afin de permettre l'écriture des programmes sur cassette.


GEMMA Guide d'Etude des Modes de Marches et d'Arrêts
ADEPA
Références de l'équipement
Banc d'essai pour transmissions cardan
LÉGENDE
P.O. = Partie Opérative
P.C. = Partie Commande
EDITION : AVRIL 1981
A PROCEDURES D'ARRET de la Partie Opérative (PO)
F PROCEDURES DE FONCTIONNEMENT
D PROCEDURES en DEFAILLANCE de la Partie Opérative (PO)
A1 <Arrêt dans état initial>
Cassette au départ
A6 <Mise P.O. dans état initial>
Toutes les consignes à zéro
A7 <Mise P.O. dans état déterminé>
Mise en énergie PO
Effacement défaut
A5 <Préparation pour remise en route après défaillance>
Mise à zéro des consignes
A4 <Arrêt obtenu>
contrôle transmission
Arrêt
A3 <Arrêt demandé dans état déterminé>
A2 <Arrêt demandé en fin de cycle>
PRODUCTION
manuel
auto
Aqpe exécution
auto-maint
demandes de marche
demandes d'arrêt
signal fin cycle (pour contrôle de la transmission)
"arrêt"
"acquit"
F1 <Production normale>
ESSAI selon programme de consignes V.C. et à données par cassette
(le programme reproduit un cycle type : démarrages, virages, arrêts ...)
F2 <Marches de préparation>
mise à l'heure
F3 <Marches de clôture>
F4 <Marches de vérification dans le désordre>
Vérification du fonctionnement des asservissements de V.C. et à par affichage des performances mini et maxi
Essais avec affichage manuel des consignes fixes V.C. à (vérification en durée)
F6 <Marches de test>
D3 <Production tout de même>
• Génération des consignes tout de même
• Affichage du défaut et de l'heure d'apparition
D2 <Diagnostic et/ou traitement de défaillance>
Affichage heure et raison de l'arrêt: AV ou Vmax ou Cmax
D1 <Arrêt d'urgence>
- Coupure énergie PO
- Freinage
AV ou dépassement Vmax ou Cmax
pour la commutation auxiliaire
défauts détectés
mise en énergie de P.C.
mise hors énergie de P.C.
HORS ENERGIE

Lorsque le programme demande l'opération de dépose et de contrôle de la transmission, un signal "fin de série essais" nous conduit en A2, puis en A4 lorsque l'arrêt est obtenu. Après contrôle et remontage, il suffit d'agir sur le bouton "marche", le sélecteur de marche étant en "auto" pour repartir en F1 avec la série suivante d'essais dans le programme.

Lorsque les essais sont terminés, un signal "Fin essais" nous conduit en A2, puis en A1, la cassette étant terminée.

Lorsque la machine s'écarte de plus de 10 % des valeurs demandées par le programme pour V, C ou α, nous passons en D3, < Production tout de même >. Les essais continuent, mais défaut et heure sont affichés sur l'écran. Lorsqu'un opérateur survient, il peut alors arrêter la machine vers A3 ou acquitter et revenir en F1.

Atteinte par la sélection "manuel", la < Marche de vérification > F4 permet au choix :

– de vérifier le fonctionnement des asservissements de V, C et α, par affichage des performances mini et maxi, à l'aide des potentiomètres prévus au pupitre ;

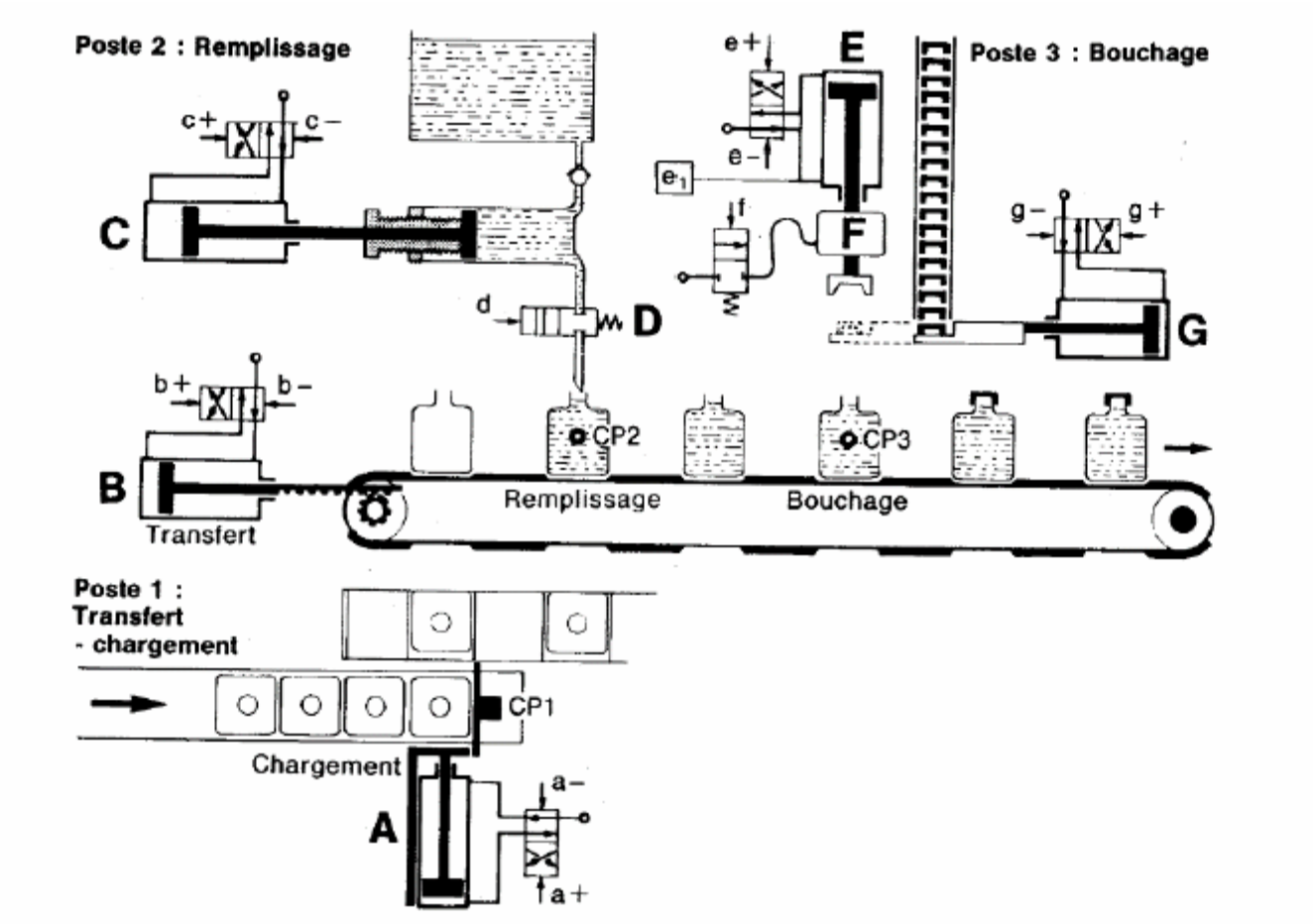
– de procéder à des vérifications de comportement par des essais prolongés avec consignes fixes V, C et α ; dans ce cas, la marche de vérification est productive puisqu'elle essaie la transmission : elle est donc située dans le rectangle "en production".

On remarquera que, à partir du guide graphique, le rectangle état F4 a été obtenu par le regroupement des rectangles-états F4 et F5 d'origine. De telles adaptations du guide graphique donnent souvent la souplesse nécessaire pour l'appliquer à des exemples très différents.

S'il y a action sur le bouton d'arrêt d'urgence (AU) ou s'il y a dépassement de V ou C maximum autorisés pour la transmission essayée, nous passons en D1 < Arrêt d'Urgence > ce qui provoque la coupure d'énergie de la PO et le freinage. En D2, l'heure et la raison de l'arrêt sont affichées. Lorsqu'un opérateur survient, il peut alors agir sur le bouton "arrêt" pour passer en A5, remettre les consignes à zéro et, par action sur le bouton acquittement, passer en A7 pour redémarrer depuis A4.

Au travers de cet exemple, l'étudiant peut commencer à percevoir les objectifs et résultats du GEMMA. La méthode de mise en œuvre sera plus explicite dans les pages suivantes.

2e exemple introductif : machine à remplir et à boucher



La machine :

Sous des formes variées, ce type de machine se rencontre souvent dans l'industrie, pour remplir automatiquement toutes sortes de contenants : flacons, bidons, boîtiers mécaniques tels que boîtes de vitesses, des réducteurs…

Les contenants sont présentés sous un poste de remplissage, puis sous un poste de bouchage.

Pour la machine ci-dessus :

– le poste de remplissage comporte un doseur volumétrique réglable, mû par le vérin C, et une vanne de fermeture D, monostable;

– le poste de bouchage comporte un transfert de bouchon par vérin G, un moteur pneumatique F pour tourner le bouchon à visser, et un vérin d'avance E. Le vérin E avance jusqu'au bouchon présenté, recule avec ce bouchon pendant le retrait de G, puis avance à nouveau avec rotation du moteur F, pour visser le bouchon.

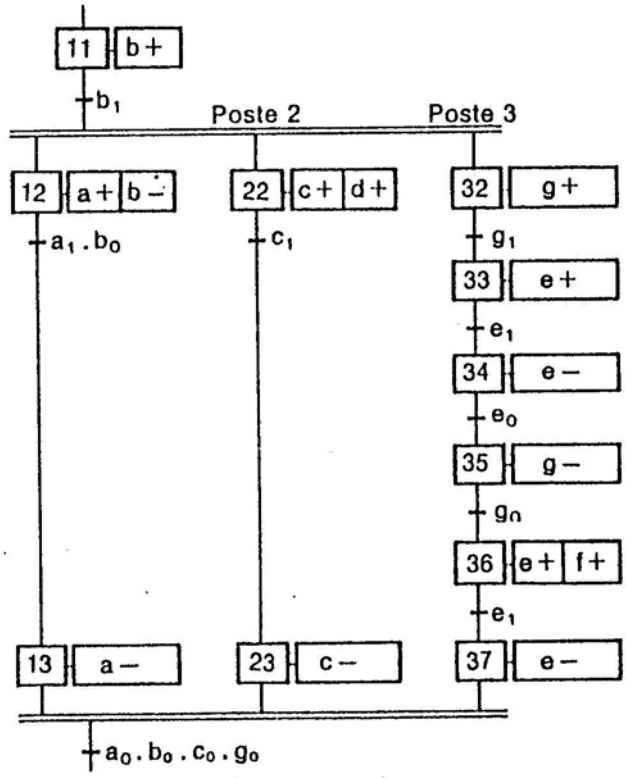
Les capteurs

Ces capteurs permettent la <Production normale> et interviennent dans le GRAFCET de BASE ci-dessous. Chaque vérin est muni d'un capteur fin de course.
Exemple : pour le vérin A, capteurs a_{11} et a_1, a_{11} étant actionné à l'état repos de la machine.

Le vérin E est équipé d'un capteur à chute de pression e1, détectant l'arrêt du vérin en butée en un point quelconque de sa course. La vanne D et le moteur F ne comportent pas de capteurs de position, car ils sont difficiles à implanter.

GRAFCET de Base

C'est le GRAFCET décrivant la <Production normale>.
Remarque : ce GRAFCET est incomplet; il sera complété après la réflexion GEMMA qui conduira entre autres à préciser la situation initiale et le rebouclage.



Définition des Modes de Marches et d'Arrêts

Le guide graphique GEMMA ci-dessous permet la détermination des états de marches et d'arrêts nécessaires à cette machine :

A1 - La case A1, < Arrêt dans l'état Initial >, représente l'état de la machine défini par le croquis (page 150).

F1 - De la case A1, lorsqu'on met la machine en route, on passe à la case Fl, <Production normale>, c'est-à-dire "Remplissage et bouchage automatiques" selon GRAFCET ci-dessus.

A2 - L'arrêt de production peut être demandé en tout point du cycle. Le cycle en cours se termine, état décrit case A2.

F2 - Lorsque la machine est vide, elle doit être mise en route progressivement, chaque poste démarrant lorsque le premier bidon se présente en position : ceci est la < Marche de préparation > de la case F2.

F3 - La < Marche de clôture> de la case F3 permet l'opération inverse, c'est-à-dire l'arrêt progressif de la machine avec vidange des bidons.

D3 - Point délicat de la machine, le poste "Bouchage" est parfois défaillant. Il peut être alors décidé de produire tout de même, en bouchant manuellement les bidons au fur et à mesure de leur remplissage automatique : c'est l'objet de la case D3 : < Production tout de même >.

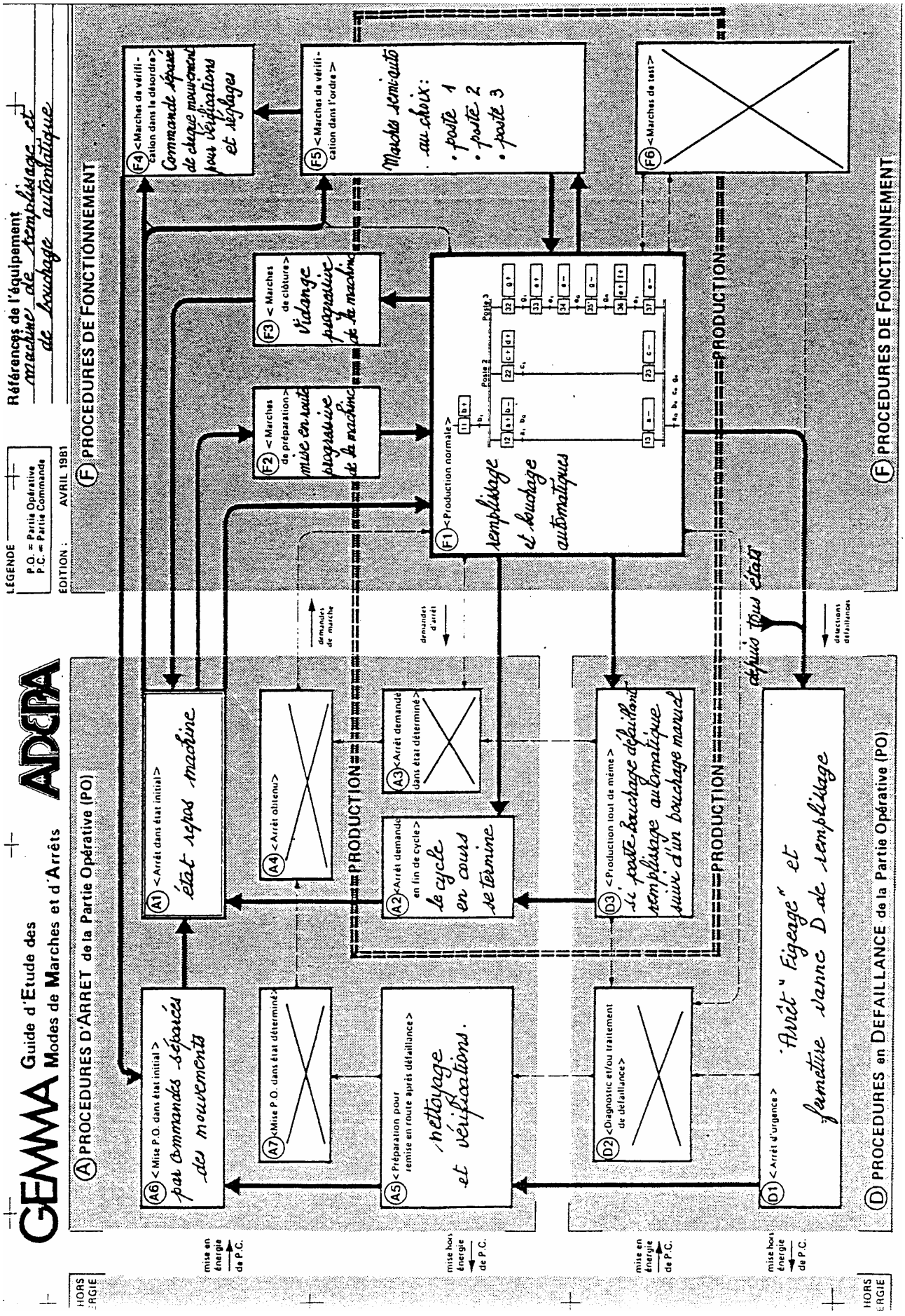
D1 - En cas d'arrêt d'urgence, la case D1, < Arrêt d'urgence >, est atteinte : elle prévoit un arrêt de tous les mouvements en cours, et la fermeture de la vanne D pour arrêter tout écoulement de liquide.

A5 - Après arrêt d'urgence, nettoyage et vérification sont souvent nécessaires : c'est l'objet de la case A5 < préparation pour remise en route après défaillance >.

A6 - Après toute défaillance ou toute vérification, une remise à l'état initial est nécessaire : case A6.

F4 - Pour le réglage du doseur, la vérification du distributeur de bouchons.... une commande séparée des mouvements est prévue : case F4.

F5 - Pour vérifications et mises au point, une commande semi-automatique (un seul cycle à la fois) est nécessaire pour chaque poste et pour l'ensemble case F5, < Marche de vérification dans l'ordre >.


GEMMA Guide d'Etude des Modes de Marches et d'Arrêts
ADEPA
Références de l'équipement
machine de remplissage et de bouchage automatique
LÉGENDE
P.O. = Partie Opérative
P.C. = Partie Commande
ÉDITION : AVRIL 1981
(A) PROCEDURES D'ARRET de la Partie Opérative (PO)
(F) PROCEDURES DE FONCTIONNEMENT
(A1) <Arrêt dans état initial> état repos machine
(A2) <Arrêt demandé en fin de cycle> le cycle en cours se termine
(A3) <Arrêt demandé dans état déterminé>
(A4) <Arrêt obtenu>
(A5) <Préparation pour remise en route après défaillance> nettoyage et vérifications.
(A6) <Mise P.O. dans état initial> par commandes séparés des mouvements
(A7) <Mise P.O. dans état déterminé>
(F1) <Production normale> remplissage et bouchage automatiques
(F2) <Marches de préparation> mise en route progressive de la machine
(F3) <Marches de clôture> Vidange progressive de la machine
(F4) <Marches de vérification dans le désordre> Commande séparé de chaque mouvement pour vérifications et réglages
(F5) <Marches de vérification dans l'ordre> Marche semi-auto au choix: • poste 1 • poste 2 • poste 3
(F6) <Marches de test>
PRODUCTION
(D) PROCEDURES en DEFAILLANCE de la Partie Opérative (PO)
(D1) <Arrêt d'urgence> "Arrêt" figeage" et fermeture vanne D de remplissage
(D2) <Diagnostic et/ou traitement de défaillance>
(D3) <Production tout de même> si poste bouchage défaillant remplissage automatique suivi d'un bouchage manuel
depuis tout état
demandes de marche
demandes d'arrêt
détections défaillances
mise en énergie de P.C.
mise hors énergie de P.C.
HORS ENERGIE

Conditions d'évolution entre Modes de Marches et d'Arrêts

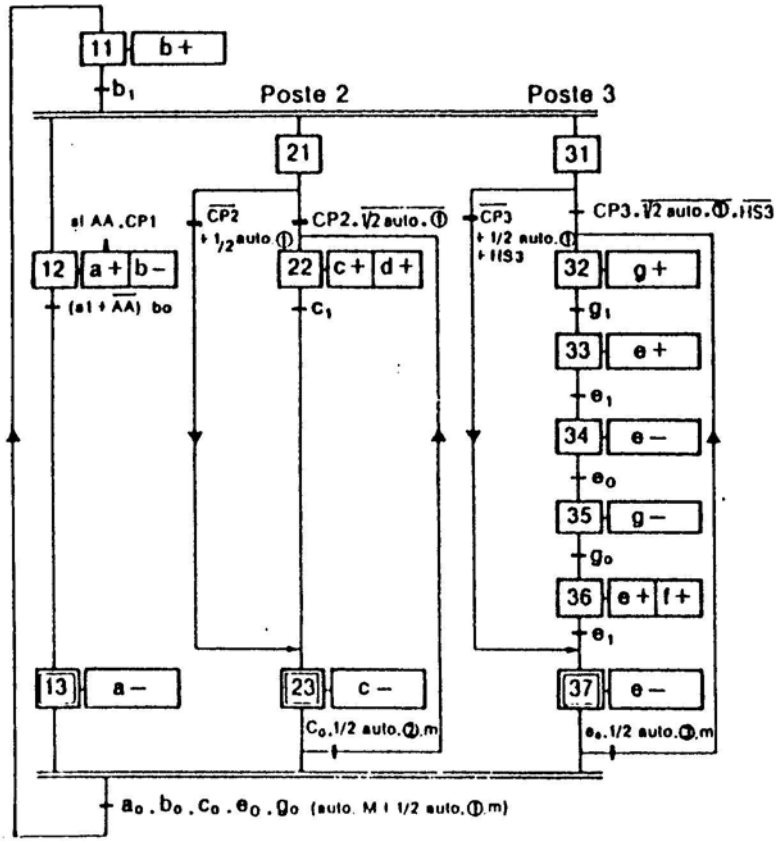
Ces conditions d'évolution sont maintenant portées sur le GEMMA, le faisant évoluer comme montré ci-dessous.

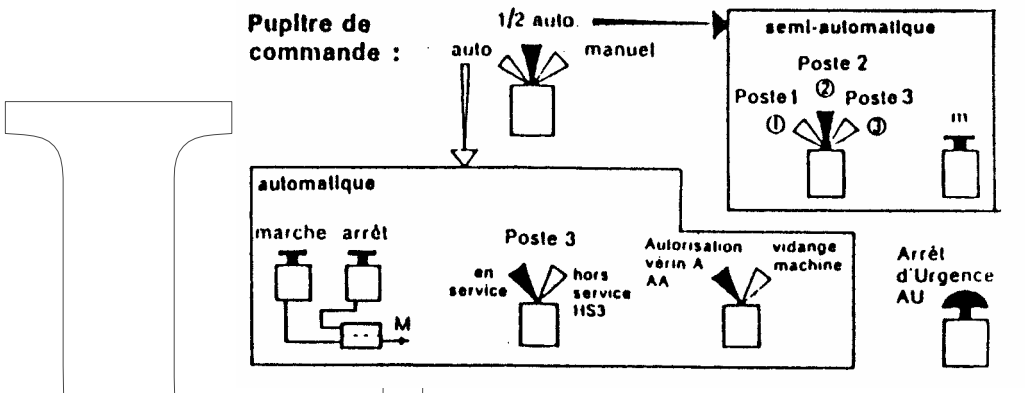
En déterminant ces conditions d'évolution nous prenons conscience :

1 - de la nécessité de capteurs complémentaires détectant la présence de bidons sous chacun des postes : CP1. CP2, CP3 permettront la mise en route progressive demandée en F2 et la vidange progressive demandée en F3.
2 - des besoins au niveau du pupitre de commande : boutons fournissant les conditions d'évolution données par l'opérateur.
3 - de compléter le GRAFCET de base avec les préoccupations de Marches et d'Arrêts.

GRAFCET complété

Le GRAFCET de base est ici complété par les besoins en Modes de Marches et d'Arrêts exprimés par le GEMMA.





Sur le pupitre de commande, un sélecteur principal permet d choisir entre les options "auto", "½ auto" et "manuel".

A l'option "automatique" correspondent :

– 2 boutons "marche" et "arrêt" dont l'action est mise en mémoire (signal M),
– un sélecteur lié au poste 3 : "en service" ou "hors service" = HS3,
– un sélecteur AA d'Autorisation du vérin A, pour permettre la vidange machine.
Ces boutons, et les capteurs CP1, CP2 et CP3, permettent d'explorer les états A1, F1. F2, F3, A2 et D3, comme montré sur le GEMMA ci-dessous.

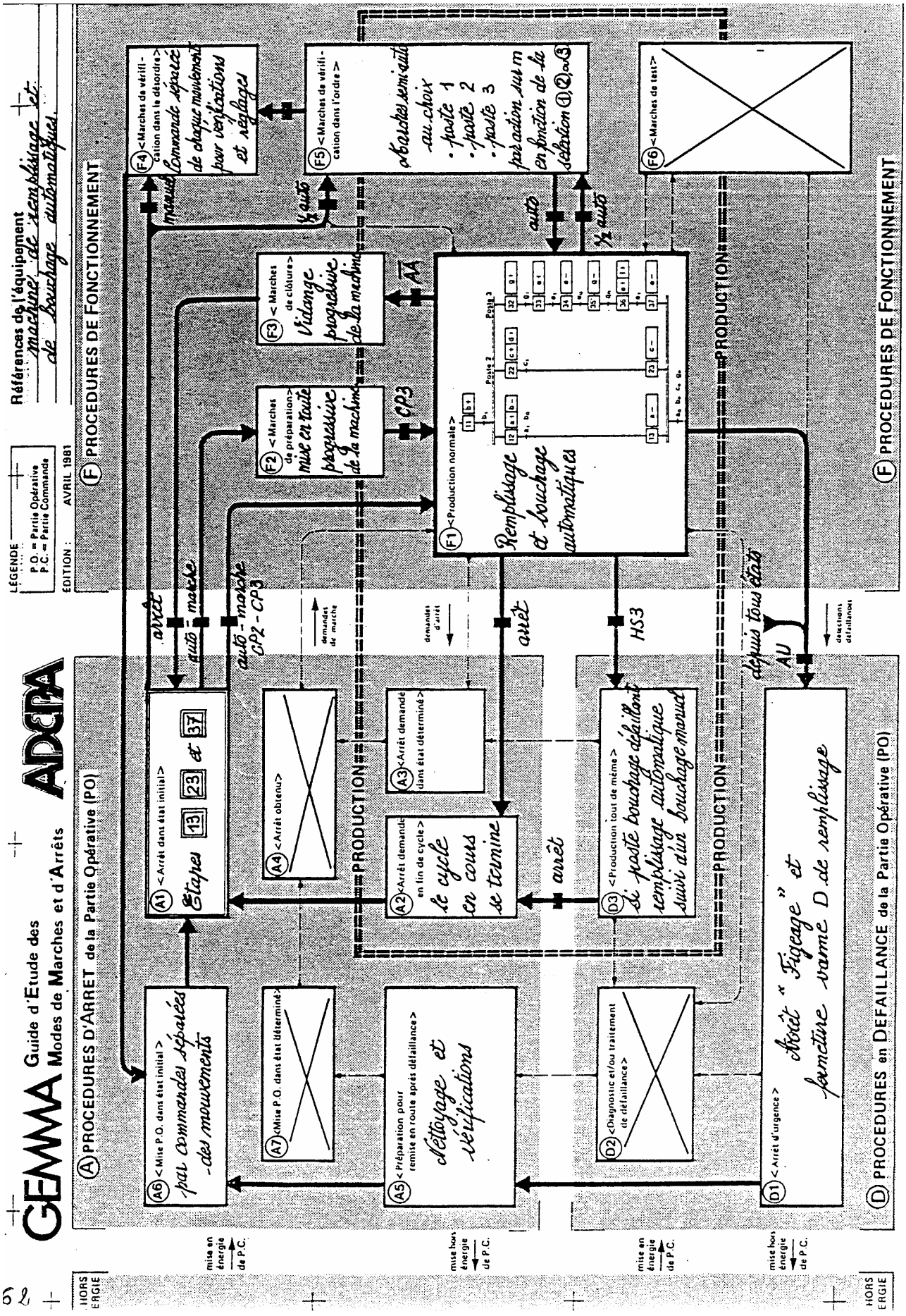
Sur le GRAFCET complété, est ménagée la possibilité de sauter les étapes avec actions des postes 2 et 3 dans le cas d'absence de bidons ($\overline{CP2}$, $\overline{CP3}$), afin de satisfaire aux besoins de mise en route progressive F2, et de vidange progressive F3 si signal $\overline{AA}$.
Les étapes avec actions du poste 3 sont également sautées HS3 (poste 3 Hors Service), car nous nous trouvons alors en D3, production tout de même sans le poste 3.

A l'option "semi-automatique" correspond F5, < marche de vérification dans l'ordre >, qui permet en appuyant sur le bouton m, l'exploration du cycle d'un seul poste, 1 ou 2 ou 3, en fonction de la position du sélecteur "semi-automatique" en ① ou ② ou ③.

Sur le GRAFCET complété, les reprises d'étapes adéquates sont prévues aux postes 2 et 3 pour dérouler cette marche. Pour le poste 1, le GRAFCET est décrit complètement, mais avec saut des étapes agissantes des postes 2 et 3. C'est afin que cette marche semi-auto se déroule sans précaution particulière que les étapes initiales du GRAFCET sont 13, 23 et 37.

A l'option "manuel" correspondent les états F4, A5 et A6, qui exigent la commande séparée des mouvements (par action directe) sur les distributeurs ou par action sur boutons s'il en est prévu.

Enfin, le bouton d'Arrêt d'Urgence AU permet de passer en D, depuis tous états.


GEMMA Guide d'Etude des Modes de Marches et d'Arrêts
ADEPA
Références de l'équipement
machine de remplissage et de bouchage automatiques
LÉGENDE
P.O. = Partie Opérative
P.C. = Partie Commande
EDITION : AVRIL 1981
F PROCEDURES DE FONCTIONNEMENT
A PROCEDURES D'ARRET de la Partie Opérative (PO)
D PROCEDURES en DEFAILLANCE de la Partie Opérative (PO)
PRODUCTION
F1 <Production normale> Remplissage et bouchage automatiques
F2 <Marches de préparation> mise en route progressive de la machine
F3 < Marches de clôture> Vidange progressive de la machine
F4 <Marches de vérification dans le désordre> Commande séparée de chaque mouvement pour vérifications et réglages
F5 <Marches de vérification dans l'ordre> Marches semi-auto au choix • poste 1 • poste 2 • poste 3 fonctionnement auto en fonction de la sélection 1, 2, 3
F6 <Marches de test>
A1 <Arrêt dans état initial> Etapes 13 23 et 37
A2 <Arrêt demandé en fin de cycle> Le cycle en cours se termine
A3 <Arrêt demandé dans état déterminé>
A4 <Arrêt obtenu>
A5 <Préparation pour remise en route après défaillance> Nettoyage et vérifications
A6 <Mise P.O. dans état initial> par commandes séparées des mouvements
A7 <Mise P.O. dans état déterminé>
D1 <Arrêt d'urgence> Arrêt "figeage" et fermeture vanne D de remplissage
D2 <Diagnostic et/ou traitement de défaillance>
D3 <Production tout de même> Si poste bouchage défaillant remplissage automatique suivi d'un bouchage manuel
arrêt
auto - marche
auto - marche CP2 - CP3
CP3
HS3
AU
depuis tous états
demandes d'arrêt
demandes de marche
directions défaillances
mise en énergie de P.C.
mise hors énergie de P.C.
HORS ERGIE

7.4. METHODE DE MISE EN ŒUVRE

Comme nous l'avons montré, le GEMMA est non seulement une méthode systématique pour sélectionner les Modes de Marches et d'Arrêts lors de la conception d'une machine automatique, mais aussi un moyen pratique pour les présenter et les exploiter.

Les deux exemples introductifs que nous avons explorés nous permettent de dégager les règles pratiques pour l'emploi du GEMMA dans la conception d'un équipement automatisé.

7.4.1. UTILISATION DU GEMMA POUR L'ETUDE D'UNE MACHINE DE PRODUCTION AUTOMATISEE

Comme nous l'avons montré, la pratique courante de l'étude des machines de Production Automatisée n'aborde pas méthodiquement la sélection des Modes de Marches et d'Arrêts, ce qui entraîne souvent des modifications longues et coûteuses de la machine après réalisation. En mettant en œuvre le GEMMA dans l'étude, les Modes de Marches et d'Arrêts sont prévus dès la conception et intégrés dans la réalisation.
Voici une séquence d'étude typique.

I parallèlement →	– étude du processus d'action – définition du cycle de production (GRAFCET fonctionnel)
II parallèlement →	– définition de la partie opérative et des capteurs – établissement du GRAFCET opérationnel de base
III ⟶	– mise en œuvre du guide graphique GEMMA pour la sélection des Modes de Marches et d'Arrêts avec mise en évidence des liaisons entre ces modes
IV parallèlement →	– définition à l'aide du GEMMA des conditions d'évolution entre les états de Marches et d'Arrêts – définition des fonctions du pupitre de commande – établissement du GRAFCET complété
V ⟶	Choix d'une technologie de commande : électrique, électronique ou pneumatique, câblée ou programmée…
VI ⟶	Conception du schéma ou du programme de commande dans la technologie choisie

7.4.2. SELECTION DES MODES DE MARCHES ET D'ARRETS

La partie opérative de la machine étant définie, ainsi que le GRAFCET du cycle de production normale, le guide graphique GEMMA est mis en œuvre, dans un premier temps, pour sélectionner et préciser les Modes de Marches et d'Arrêts nécessaires.

Deux préoccupations simultanées permettent d'aboutir facilement :

1. Envisager tous les "rectangles-états" proposés par le GEMMA

Avec ses rectangles-états, le guide graphique constitue une "check-list" des différents types de Modes de Marches et d'Arrêts nécessaires en automatisation industrielle courante. Pour une machine donnée, il est donc important d'examiner le cas de chaque "rectangle-état" :

– si le mode proposé est retenu, il sera précisé en "langage-machine" dans le "rectangle-état"; au besoin, plusieurs variantes de ce mode seront distinguées;

– si le mode proposé n'est pas nécessaire pour la machine, une croix sera portée dans le "rectangle-état", pour bien signifier qu'il n'est pas retenu.

2. Rechercher les évolutions d'un état à l'autre

Deux états essentiels, définis dès le début de l'étude, se retrouvent sur toutes les machines :

– l'état A1, dit <état initial>, ou "état repos" de la machine (1)
– l'état F1, mode de <production normale>, pour lequel la machine a été conçue (1)

En parlant de chacun des deux états essentiels, A1 et F1, il est intéressant de rechercher les évolutions vers d'autres états, dont la nécessité est moins évidente au premier abord.

- On pourra commencer par démarrer la machine, c'est-à-dire passer de A1 à F1, en se posant la question : une <marche de préparation> F2 est-elle nécessaire ?

- On arrêtera alors la machine, au choix :
  - en fin de cycle → circuit F1 → A2 → A1
  - dans une autre position → circuit F1 → A3 → A4

- On examinera les cas de défaillance,
  - avec <Arrêt d'urgence>, D1
  - avec <Production tout de même>, D3

- etc.

En pratique, on mettra en évidence des boucles passant par les états A1 ou F1, et appartenant à l'une des deux familles :

- les boucles "démarrage – arrêt normal"
- les boucles "démarrage – défaillance – retour à l'état initial"

Le guide graphique GEMMA permet donc une sélection des Modes de Marche et d'Arrêts pour une machine donnée; la méthode consiste à examiner le cas de chacun des états proposés, en recherchant les évolutions d'un état dans l'autre, en particulier à partir des deux états essentiels sur toute machine, l'état initial A1 et l'état de production normale F1.

(1) Sur le guide graphique GEMMA,
- le "rectangle-état" A1 <Arrêt dans l'état initial> comporte un double cadre pour montrer sa correspondance avec la situation initiale du GRAFCET
- le "rectangle-état" F1 <Production normale> a un cadre particulièrement renforcé pour mettre en évidence l'importance de cet état sur toute machine

Cette sélection étant achevée et mise au net sur le guide graphique GEMMA, il y a lieu de procéder au 2e temps de l'exploitation du GEMMA : la recherche des conditions d'évolution entre les Modes de Marche et d'Arrêts, expliquée aux pages suivantes.

7.4.3. CONDITIONS D'EVOLUTION ENTRE MODES DE MARCHES ET D'ARRETS

Ainsi personnalisé avec les Modes de Marche et d'Arrêts retenus pour la machine, le GEMMA est alors utilisé pour préciser les informations nécessaires pour passer d'un état à l'autre : c'est la mise en évidence de ces conditions de passage qui permet alors de concevoir le pupitre de commande (actions de l'opérateur), de prévoir éventuellement des capteurs supplémentaires sur la machine, de compléter le GRAFCET, etc.

On peut passer d'un état à l'autre de 2 manières (figure 7.7.) :

1 - avec une condition d'évolution : elle est portée sur la liaison orientée entre états; la condition peut être liée à l'action sur un bouton du pupitre de commande, ou à l'actionnement d'un capteur situé sur la machine.

2 - sans condition explicite : dans certaines évolutions entre états, l'écriture d'une condition n'apporterait aucune information utile : c'est le cas lorsque celle-ci est évidente (exemple, passage de A2 à A1 ci-dessous), ou parce que l'état atteint dépend de l'intervenant.

Avec les conditions d'évolution provenant de l'opérateur, apparaissent les besoins en boutons : le pupitre de commande peut alors être défini : boutons à 2 ou 3 positions, bistables ou monostables, etc.

Aux conditions d'évolution ne provenant pas de l'opérateur doivent correspondre des capteurs sur la machine : détecteurs de défauts, détecteurs de présence pièce, etc., dont le besoin est ainsi mis en évidence.

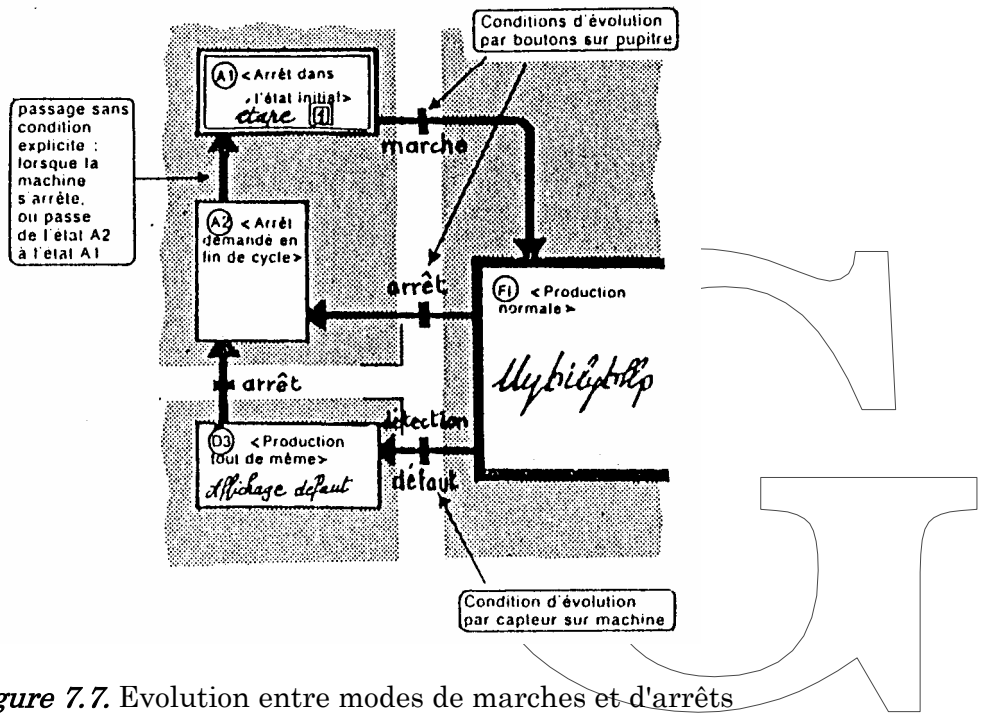


Figure 7.7. Evolution entre modes de marches et d'arrêts

Enfin, avec ces éléments, le GRAFCET de base peut alors être complété de façon à donner une description précise de l'ensemble du fonctionnement de la partie commande.

7.4.4. CONCLUSION

Comme montré dans les exemples introductifs, on peut apercevoir les conséquences de l'intervention du GEMMA dans la séquence d'étude de la machine :

– pupitre, capteurs supplémentaires, GRAFCET complété, découlant de la réflexion GEMMA sont mieux prévus;

– la machine est mieux conçue, donc sa réalisation et sa mise en route se font avec moins de tâtonnements et de modifications;

– comme le GRAFCET, le GEMMA suivra ensuite la machine, servant aux dépannages ou modifications.

Remarque : unicité du Mode

Le GEMMA est conçu pour une partie commande unique pilotant une partie opérative unique. Autrement dit, on considère le système globalement, ce qui implique qu'à tout instant on est dans un mode et un seul ou, ce qui revient au même, dans un "rectangle-état" et un seul. Les conséquences de cette unicité sont expliquées à la fin du chapitre.

7.4.5. CAS PARTICULIER : MACHINES NECESSITANT L'INTERVENTION D'UN OPERATEUR A CHAQUE CYCLE

Communément appelées "semi-automatiques", ces machines doivent être servies par un opérateur à chaque cycle. De façon à ne pas astreindre l'opérateur à suivre la cadence de la machine, il est préférable de le rendre maître de cette cadence avec un bouton qui lui permet de déclencher chaque cycle lorsque sa tâche est achevée.(1)

L'intervention de l'opérateur peut être traduite sur un GEMMA, de deux manières, selon la machine :

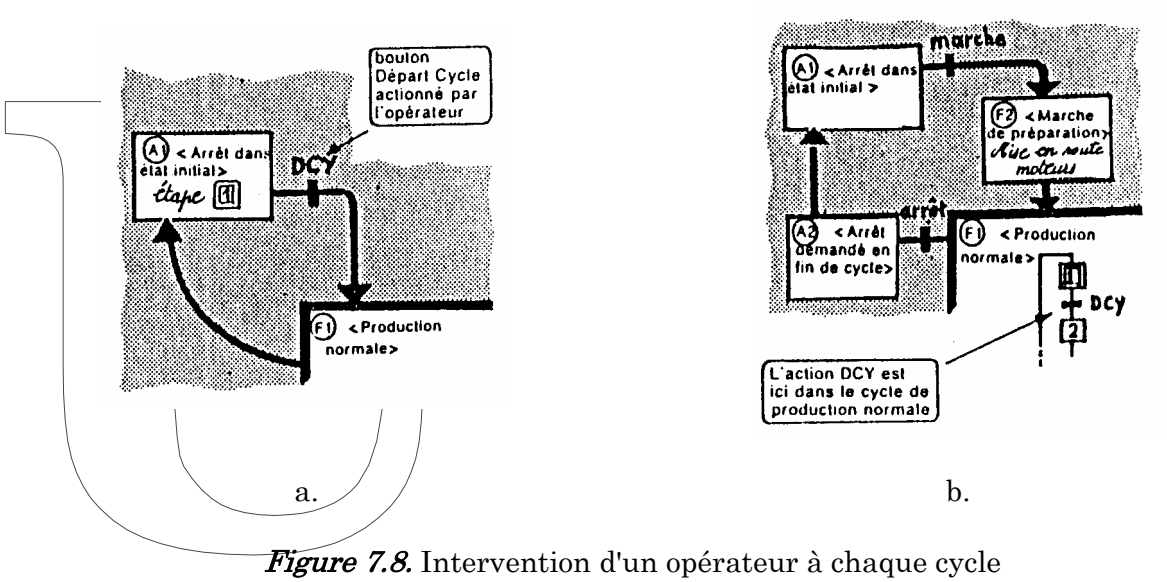
1. Action "Départ Cycle" extérieure à F1

A chaque cycle, la machine part de l'état initial A1 et, par la condition DCY, signal "Départ Cycle" donné par l'opérateur, passe dans l'état F1. Lorsque le cycle est terminé, la machine reprend l'état A1 sans intervention de l'opérateur. Cette méthode convient lorsque tout peut être arrêté sur la machine, même lors d'une courte pause (figure 7.8.a).

2. Action "Départ Cycle" intérieure à F1

La machine est mise en route via l'état F2 et est alors prête à produire dans l'état F1. DCY est ici intégré dans le GRAFCET de production normale : pour produire, la machine attendra l'action de l'opération en restant en F1. Cette méthode convient lorsqu'il faut laisser se continuer des actions (moteur, chauffage…) lors des pauses courtes (figure 7.8.b).

(1) La tâche de l'opérateur, à chaque cycle, peut être, par exemple, de charger la pièce ou de la décharger, de nettoyer l'outillage, de vérifier la pièce avant ou après opérations, etc.



a. b.

Figure 7.8. Intervention d'un opérateur à chaque cycle

7.5. PROGRAMMATION

Le GEMMA d'un système ayant été obtenu, on dispose alors des éléments nécessaires pour établir le cahier des charges définitif de la partie commande et en particulier son GRAFCET.

L'objet de ce paragraphe est de montrer comment intégrer les précisions obtenus sur les Modes de Marches et d'Arrêts, dans le GRAFCET d'ensemble de la partie commande.

On appelle GRAFCET COMPLETE, le GRAFCET d'un automatisme intégrant l'ensemble des modes de marche, par opposition au GRAFCET de BASE qui décrit seulement le comportement du système en mode Fl <Production Normale>.

Pour aboutir au GRAFCET' COMPLÉTÉ, on peut envisager 2 types de méthodes :

1 - Enrichissement du GRAFCET de BASE.
2 - Structuration en "TACHES".

7.5.1. ENRICHISSEMENT DU GRAFCET DE BASE

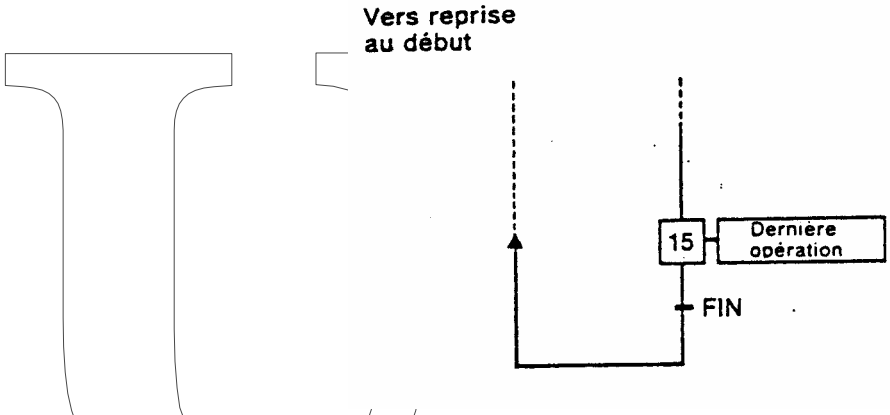
C'est la méthode utilisée dans l'exemple introductif de la machine à remplir et à boucher (cf. § 7.3.6.). A partir du GRAFCET de BASE correspondant au rectangle-état Fl < Production Normale >, on examine ce qu'il faut rajouter pour que les autres modes puissent être assurés. Ceci revient souvent à ajouter des "branches" ou "séquences" exclusives les unes des autres, où les conditions d'aiguillages sont les conditions de passage d'un rectangle-état à un autre.

EXEMPLE

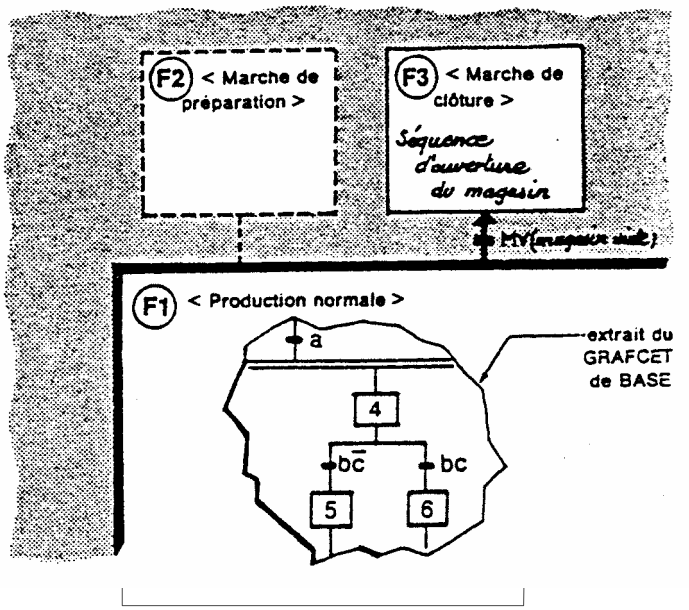
Considérons un Système doté d'une <marche de clôture> F3 constituée d'une séquence d'ouverture d'un magasin pour réapprovisionnement.

Nous trouvons les éléments suivants (figure 7.9.)

a) Extrait du "GRACET de BASE"



b) Extrait du GEMMA du Système



c) Extrait du GRAFCET COMPLETE
(obtenu par adjonction d'une branche au GRAFCET de BASE)

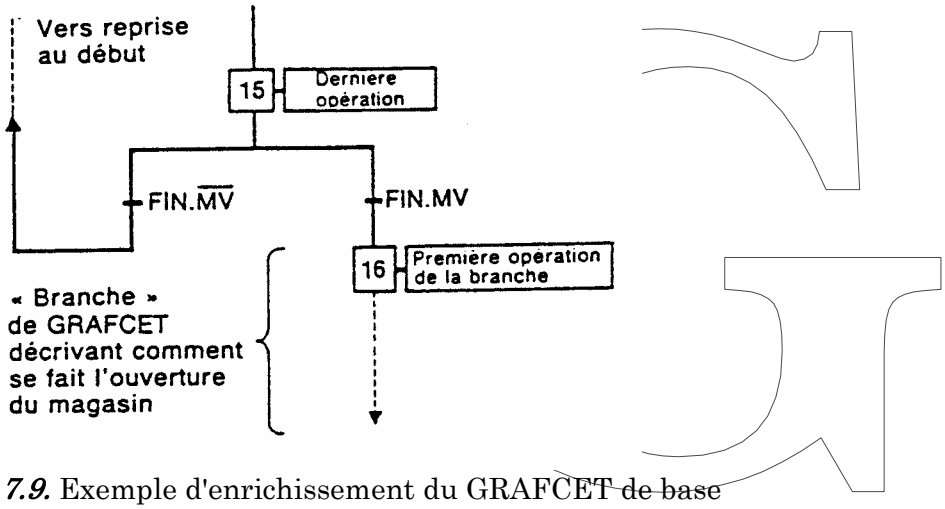


Figure 7.9. Exemple d'enrichissement du GRAFCET de base

7.5.2. DECOUPAGE EN TACHES

Il arrive fréquemment qu'un ou plusieurs rectangles-états autres que Fl <Production normale> exigent la mise en oeuvre de cycles assez complexes justifiant pour chacun d'eux une étude séparée.

Il peut être alors intéressant de structurer chacun de ces modes, y compris Fl <Production normale> en tâches autonomes, dont on étudiera ensuite les conditions de coordination.

Le GRAFCET permet facilement de représenter une telle structuration en tâches.

Une fois que chacun des rectangles-états intéressés est décrit par GRAFCET sous forme de tâche autonome (tâche de préparation, tâche de clôture, tâche de test...) il convient d'examiner attentivement les conditions de coordination entre tâches.

Pour cela, on peut recourir à 2 types de méthodes :

- coordination "horizontale"
- coordination "verticale" ou "hiérarchisée"

A - Coordination HORIZONTALE

Ce procédé est intéressant lorsque aucune tâche n'est prééminente et que chacune peut en lancer une autre.

Reprenons l'exemple donné plus haut où l'on trouve une <marche de clôture> F3 que l'on structure en tâche F3. Le GRAFCET de base correspondant à Fl est également structuré en tâches (figure 7.10.).

Cet exemple montre qu'on peut facilement enchaîner les tâches selon le schéma apparaissant sur le GEMMA.

Ce procédé est cependant à réserver lorsqu'il y a peu de tâches et que les liaisons entre elles sont assez limitées. De plus, il est souhaitable qu'à tout instant une tâche et une seule soit "en cours d'exécution".

Ceci est le cas pour un système régi par un GEMMA unique, puisqu'il y a à tout instant unicité du mode.

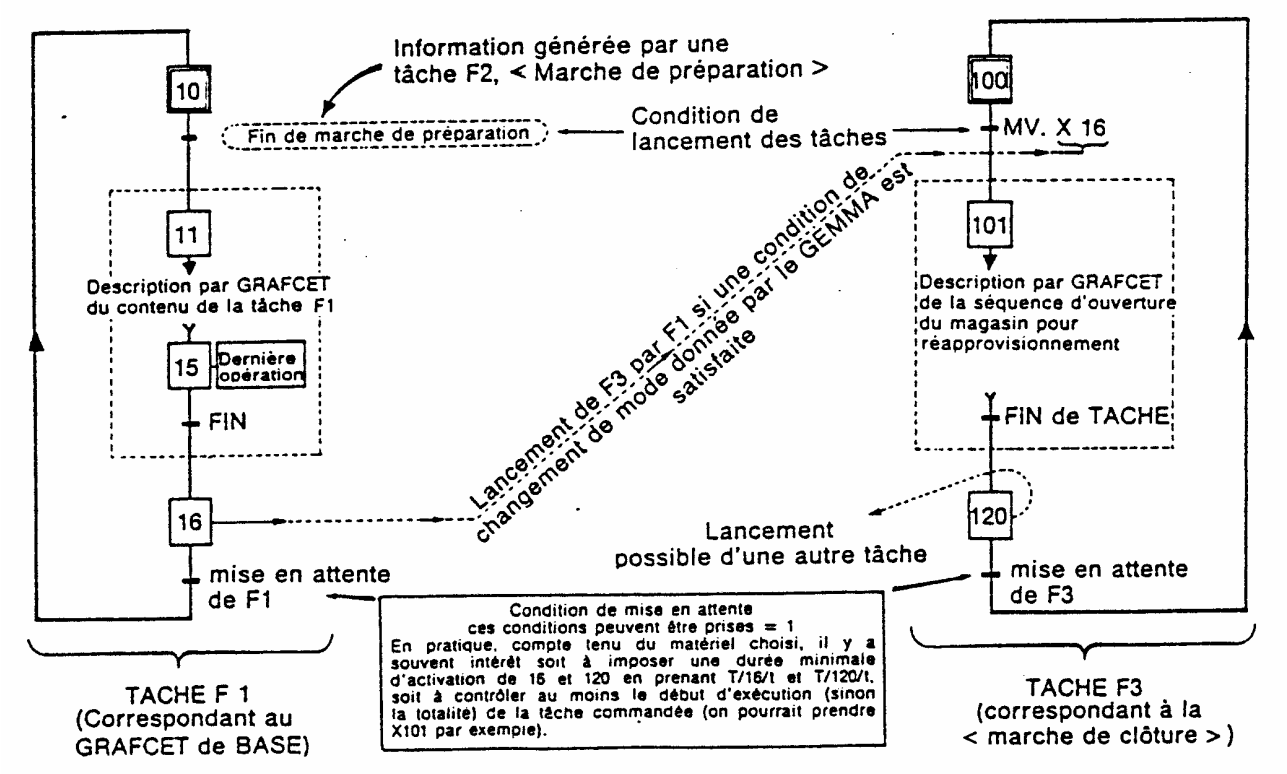
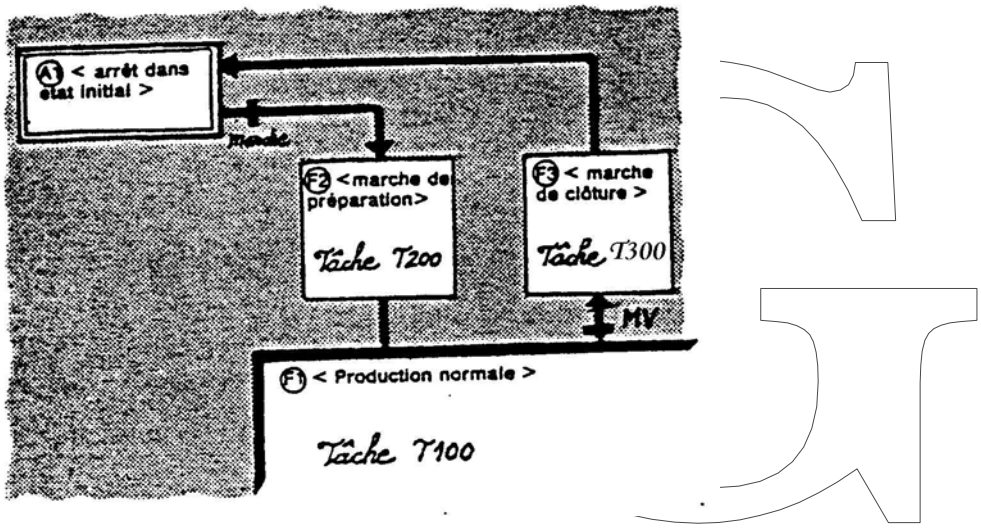


Figure 7.10. Coordination horizontale

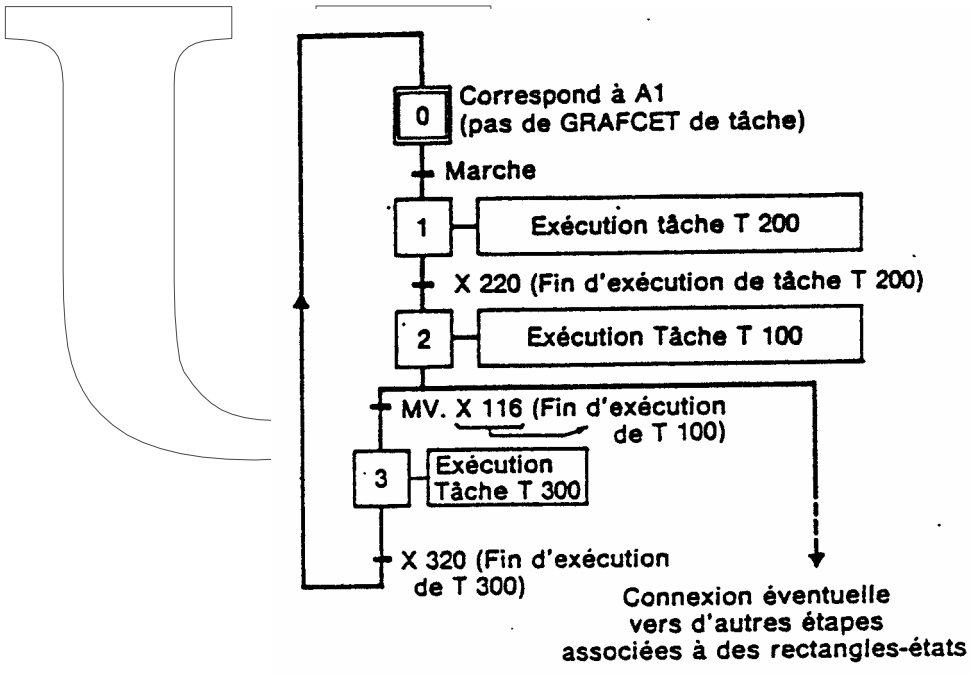
B - Coordination VERTICALE ou HIERARCHISEE

Reprenons l'exemple déjà évoqué plus haut et dotons le système d'une <marche de préparation> F2 correspondant à une séquence de fermeture du magasin après réapprovisionnement. il a de plus un mode A1, <arrêt dans un état initial>. Il peut avoir d'autres rectangles-états, mais nous ne considérerons que A1, F1, F2, F3.

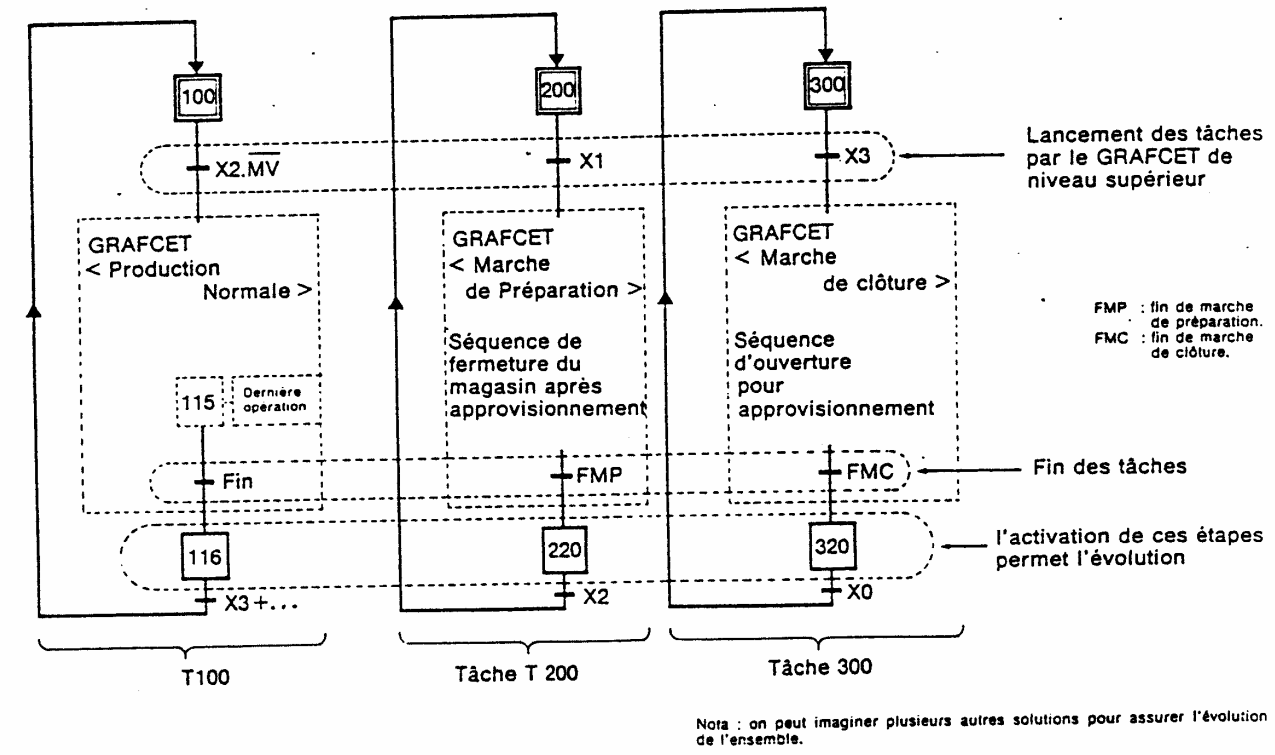
La partie correspondante du GEMMA est montrée ci-dessous :



Il est alors aisé d'en déduire la partie correspondante du GRAFCET de niveau supérieur :



La figure suivante donne alors une vue schématisée des GRAFCETS des tâches T100, T200, T300 (de niveau inférieur) :



7.6. TRAITEMENT DES ARRETS D'URGENCE

7.6.1. PROBLEMES

On sait qu'un des intérêts du GRAFCET est de n'indiquer à chaque situation que les seules circonstances qui la font évoluer.

L'ARRET D'URGENCE devant toujours faire évoluer la situation de la partie commande, on devrait en toute rigueur le faire apparaître à chaque étape du GRAFCET. Dès que le cas traité devient un peu important le GRAFCET devient complètement illisible. Il en découle qu'il est malaisé de considérer l'arrêt d'urgence AU comme une information d'entrée comme les autres. C'est de plus contraire à l'esprit même de l'urgence accordant à cet arrêt une sorte de "Super Priorité". Pourtant il est intéressant (mais ce n'est pas toujours possible) de pouvoir formaliser le comportement attendu d'un système lors de l'apparition de l'information AU. Deux approches sont généralement envisagées :

– figeage du GRAFCET et annulation des actions;

– lancement d'une procédure d'urgence qui peut évidemment dépendre du moment où l'arrêt d'urgence se produit.

Dans le cas du figeage du GRAFCET, la commande des actionneurs doit être particulièrement bien étudiée selon le type de réaction souhaitée en cas d'arrêt d'urgence. Par exemple :

Pour des distributeurs commandant des vérins :
- commande monostable, si 1'on désire un retour dans une position donnée.
- commande bistable, si l'on désire un arrêt en fin de mouvement,
- distributeurs à 3 positions, si l'on désire un arrêt sur place...

Pour des contacteurs commandant des moteurs :
- commande monostable.
- câblage incluant des "surcourses" en sécurité...

De plus, le problème de la reprise après arrêt d'urgence est un problème particulièrement délicat.

7.6.2. SOLUTIONS PRAGMATIQUES

Pour éviter la programmation explicite du comportement en cas d'arrêt d'urgence, les constructeurs offrent des fonctions spéciales. Si elles allègent considérablement la programmation, ces fonctions restreignent évidemment les possibilités à ce qui a été prévu par le constructeur.

Nous donnerons ci-dessous quelques exemples d'origine TELEMECANIQUE.
Les fonctions en question sont mises en oeuvre par l'intermédiaire de bits "système".

- Bit SY21 : sa mise à 1 provoque l'initialisation du GRAFCET (activation des étapes initiales)

- Bit SY22 : sa mise à 1 provoque la remise à zéro générale du GRAFCET

- Bit SY23 : sa mise à 1 provoque le figeage du GRAFCET et l'inhibition de toutes les actions

De plus, les étapes du GRAFCET peuvent être activées à volonté dans la partie combinatoire du traitement en forçant à 1 les variables d'étape Xi correspondantes. On a ainsi le moyen de relancer le GRAFCET à partir d'un état quelconque défini par le programmeur.

La figure 7.11. montre un exemple d'utilisation du bit SY23 et son équivalent GRAFCET.

La figure 7.12. montre comment on peut lancer une séquence d'urgence en jouant sur le bit SY22 et sur le positionnement des variables d'étape.

SY 21 : initialisation du GRAFCET

SY22 : désactivation du GRAFCET

SY23 : fige le GRAFCET et annule les actions

Possibilité de forcer les variables d'étape X :

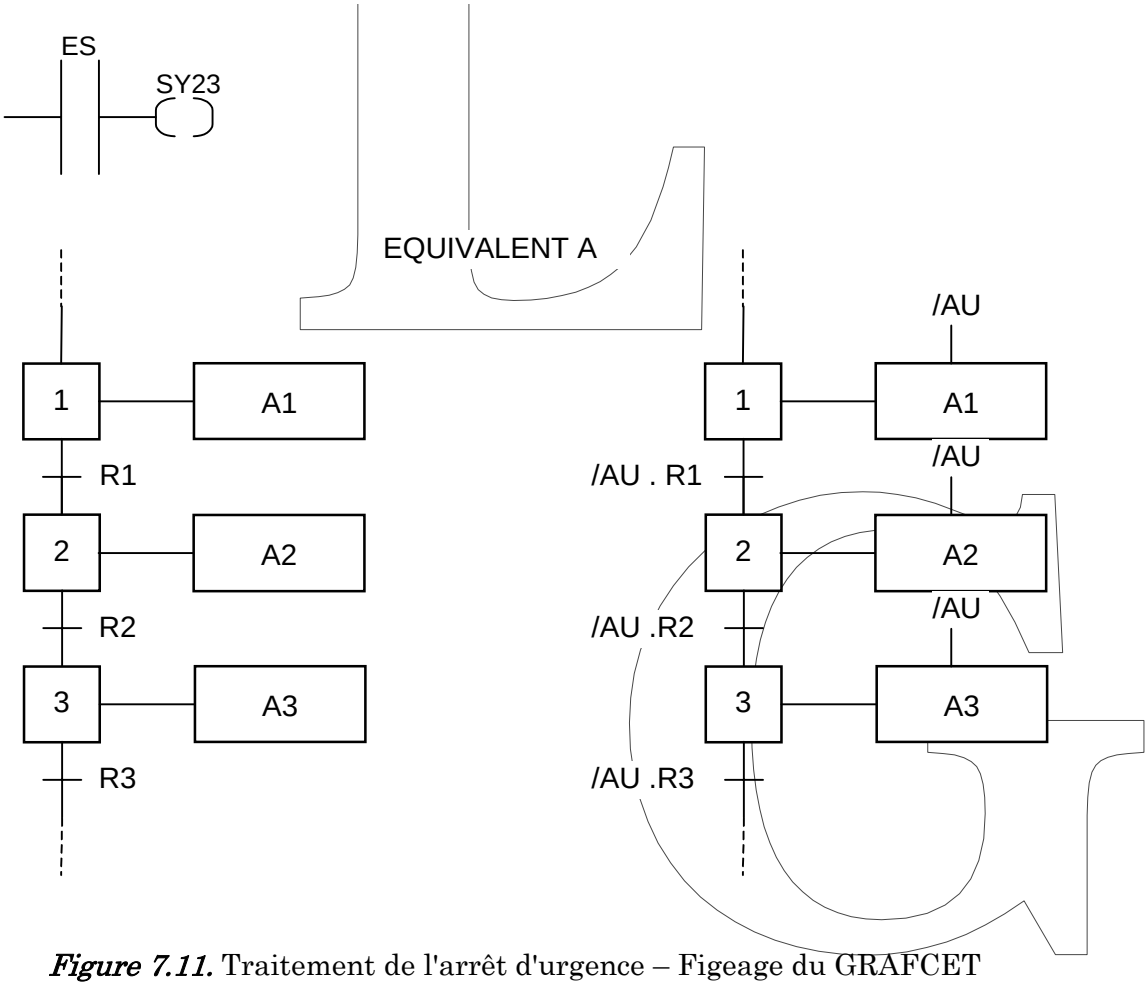


Figure 7.11. Traitement de l'arrêt d'urgence – Figeage du GRAFCET

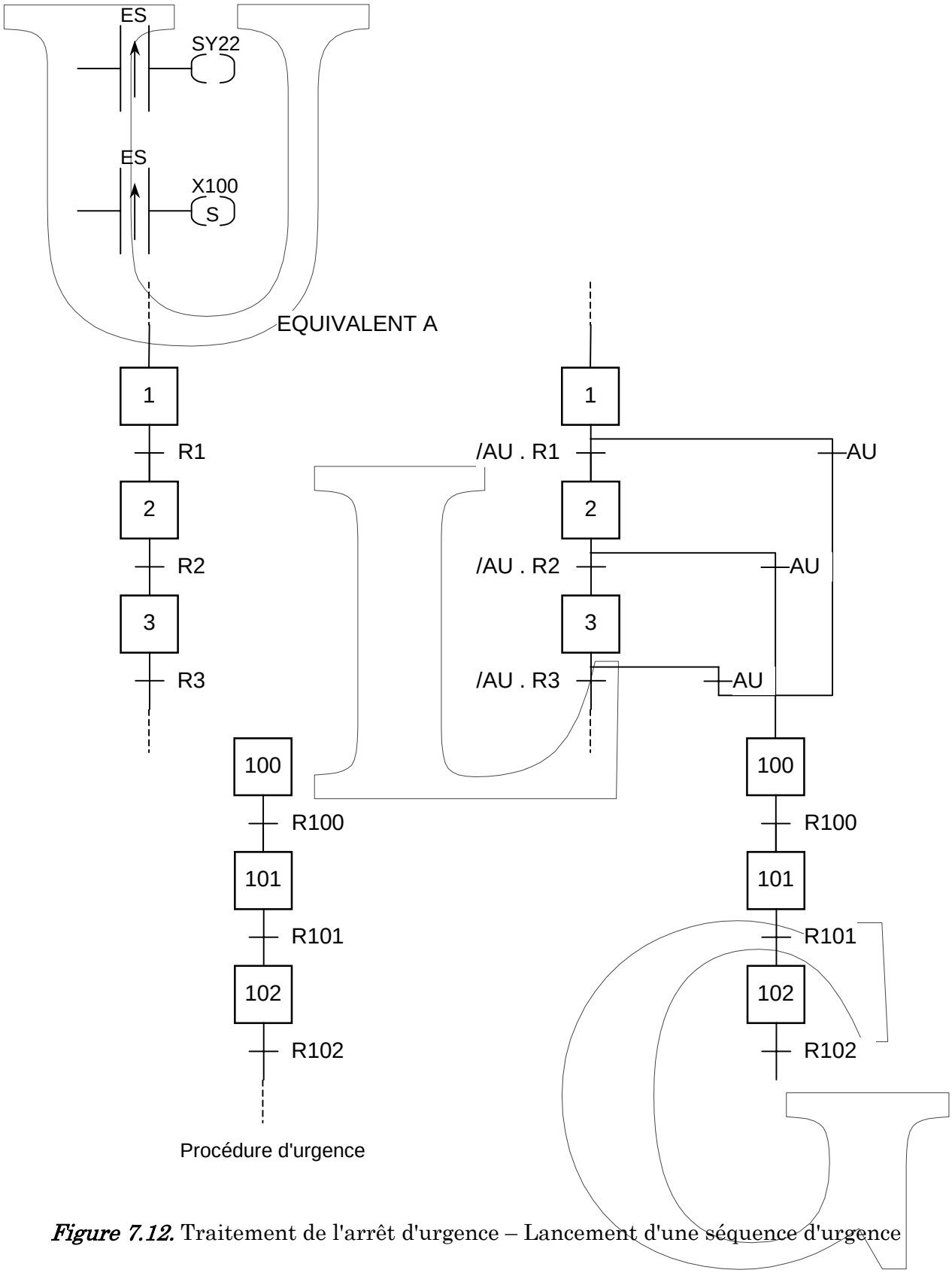


Figure 7.12. Traitement de l'arrêt d'urgence – Lancement d'une séquence d'urgence

7.7. LIMITES ET EXTENSIONS POSSIBLES

Rappelons tout d'abord que le GEMMA est conçu pour une partie commande unique pilotant une partie opérative UNIQUE. Autrement dit, on considère le système globalement, ce qui implique qu'à tout instant on est dans un mode et un seul, ou, ce qui revient au même, dans un RECTANGLE-ETAT et un seul.

7.7.1. UNICITE DU MODE : POURQUOI ?

Elle permet de considérer indépendamment les unes des autres les tâches (ou actions) associées aux différents modes sans risque d'interférence fâcheuse. En effet, si plusieurs tâches étaient actives au même moment, il pourrait se trouver des cas où les ordres émis seraient contradictoires.

Il est donc non seulement souhaitable de respecter cette contrainte mais recommandé de placer tous les "verrous" nécessaires pour que seules les évolutions respectant l'unicité du rectangle-état actif soient possibles.

Cette contrainte devient gênante lorsque le système prend une dimension telle qu'il est difficile de le considérer comme un tout ou bien lorsque ses parties peuvent être dotées d'une certaine autonomie, l'une pouvant être à l'arrêt et les autres en marche par exemple. On peut cependant conserver l'approche GEMMA à condition d'ajouter un niveau supplémentaire comme indiqué ci-après.

7.7.2. COMMENT UTILISER LE GEMMA LORSQU'IL N'Y A PLUS UNICITE DU MODE ?

La restriction précédente n'étant plus satisfaite, on peut conseiller d'utiliser un GEMMA par système ou sous-système pouvant être considéré comme ayant à tout instant un mode (ou rectangle-état) actif et un seul.

On a alors le schéma suivant (figure 7.13.) :

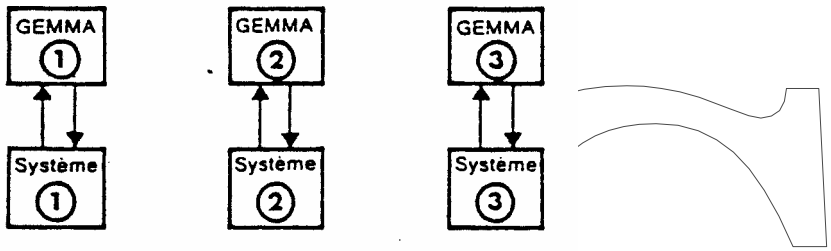


Figure 7.13. Décomposition du système en plusieurs GEMMA à mode unique

Il faut alors assurer la coordination entre ces GEMMA afin d'éviter les risques évoqués plus haut. Là encore deux possibilités s'offrent.

Coordination "HORIZONTALE"

On écrit sur les conditions de transition entre rectangles-états de chacun des GEMMA, les rectangles-états où peuvent être les autres GEMMA.

Exemple : une machine-transfert linéaire est formée par la réunion de trois systèmes :

- un système A d'approvisionnement,
- un système B : machine proprement dite,
- un système C de déchargement.

Chacun des systèmes a son GEMMA : A et C peuvent être arrêtés en défaillance (états D1 ou D2). Si B continue à fonctionner avec un chargement/déchargement à la main, GEMMA B peut être en D3, <production tout de même>.

On pourra mettre pour la condition d'entrée du GEMMA B ou Fl, <production normale> que GEMMA A et C soient également en Fl.

Coordination "VERTICALE"

On fait alors un "GEMMA de GEMMA". C'est alors un ensemble hiérarchisé : les systèmes ne communiquent plus directement entre eux, mais via un GEMMA de niveau supérieur (figure 7.14.).

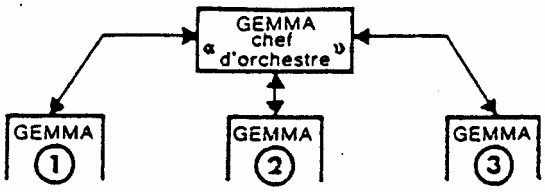


Figure 7.14. Utilisation d'un GEMMA "chef d'orchestre"

Chapitre 8

NORMALISATION DES LANGAGES DE PROGRAMMATION

On a déjà fait abondamment état dans ce qui précède des travaux de normalisation relatifs aux automates programmables (norme CEI 1131) et plus spécialement aux langages de programmation (norme CEI 1131-3). Dans ce chapitre, nous entrerons un peu plus dans le détail de la norme CEI 1131-3.

8.1. CONCEPTS DE BASE

La norme n'a pas essayé de bouleverser les habitudes des automaticiens en imposant un langage de programmation unique. En fait, elle reprend les principaux types de langages déjà existants et autorise même leur cohabitation en sein d'une application.

Quatre langages ont ainsi été normalisés :

– langages graphiques
  - le langage à contacts sous le sigle LD (Ladder Diagram)
  - le langage en blocs fonctionnels sous le sigle FBD (Function Block Diagram).

– langages littéraux
  - le langage à liste d'instruction sous le sigle IL (Instruction List)
  - le langage littéral structuré sous le sigle ST (Structured Text).

Par ailleurs, c'est le GRAFCET, normalisé sous le sigle SFC (Sequential Function Chart), qui constitue l'outil de base pour la structuration des commandes de nature séquentielle. La présentation qui en a été faite au chapitre 6 est déjà largement inspirée de la norme et nous n'y reviendrons donc plus ici.

Par contre, à côté des langages de programmation proprement dits, la norme s'intéresse aussi à des notions plus pratiques telles que

- les éléments de configuration
- les unités d'organisation des programmes
- les types de données
- les variables
- l'utilisation de caractères imprimés

Il est bien certain que leur normalisation est tout aussi importante pour la portabilité des programmes que celle des langages de programmation eux-mêmes.

Nous commencerons ce chapitre par un aperçu de ces aspects de la norme dans la mesure où ils sont communs à tous les langages.

8.2. MODELE LOGICIEL D'UN AUTOMATE PROGRAMMABLE

La figure 8.1. montre le modèle logiciel d'un automate programmable tel qu'il apparaît au stade de la configuration du système dans la norme CEI 1131. On peut y distinguer des éléments dits de configuration (ressources, variables globales et directement représentées, chemins d'accès, tâches) et des unités d'organisation des programmes (programmes, blocs fonctionnels).

8.2.1. ELEMENTS DE CONFIGURATION

– **ressource** : il s'agit d'une entité de traitement de l'automate comportant ses interfaces homme-machine et capteurs-actionneurs propres. La notion de "ressource" prend tout son sens dans les automates multiprocesseurs. Pour les autres, la configuration se réduit à une seule ressource.

– **variables globales** : il s'agit des variables symboliques accessibles par toutes les ressources du système.

– **variables "directement représentées"** : il s'agit de variables spécifiées directement par leur adresse physique; elles sont, de ce fait, également accessibles par toutes les ressources. Les deux types de variables précitées constituent un moyen d'échange d'information entre ressources.

– **chemins d'accès** : ils spécifient les variables symboliques auxquelles peuvent accéder les services de communication dans le cas d'utilisation de réseaux de communication.

– **tâches** : il s'agit d'éléments de commande d'exécution capables de lancer soit de façon périodique, soit sur activation d'une variable, l'exécution d'un ensemble d'unités d'organisation de programmes (cf. § 8.2.2.) telles que programmes et blocs fonctionnels. Des priorités peuvent être accordées aux tâches.

Ces possibilités existent déjà dans beaucoup d'automates où l'on peut déclarer des tâches rapides, des tâches lentes, des sous-routines d'interruption, etc

8.2.2. UNITES D'ORGANISATION DE PROGRAMMES

Les unités d'organisation de programmes correspondant à la notion d'"objet" en informatique classique. Elles encapsulent des données et les procédures de traitement associées dans des "boîtes noires" auxquelles on ne peut accéder que par des interfaces (entrées et sorties) limitées et bien définies. Ce concept d'"objet" est essentiel pour la structuration, la réutilisation et la qualité des logiciels.

La norme définit trois types d'unités d'organisation : les fonctions, les blocs fonctionnels et les programmes. Celles-ci peuvent être fournies par le constructeur et/ou être programmées par l'utilisateur. La norme donne des directives très précises pour la constitution de ces unités d'organisation. Notons qu'elles peuvent être utilisées avec n'importe lequel des langages normalisés mentionnés ci-dessus et, cela, quel que soit par ailleurs le langage utilisé pour les programmes.

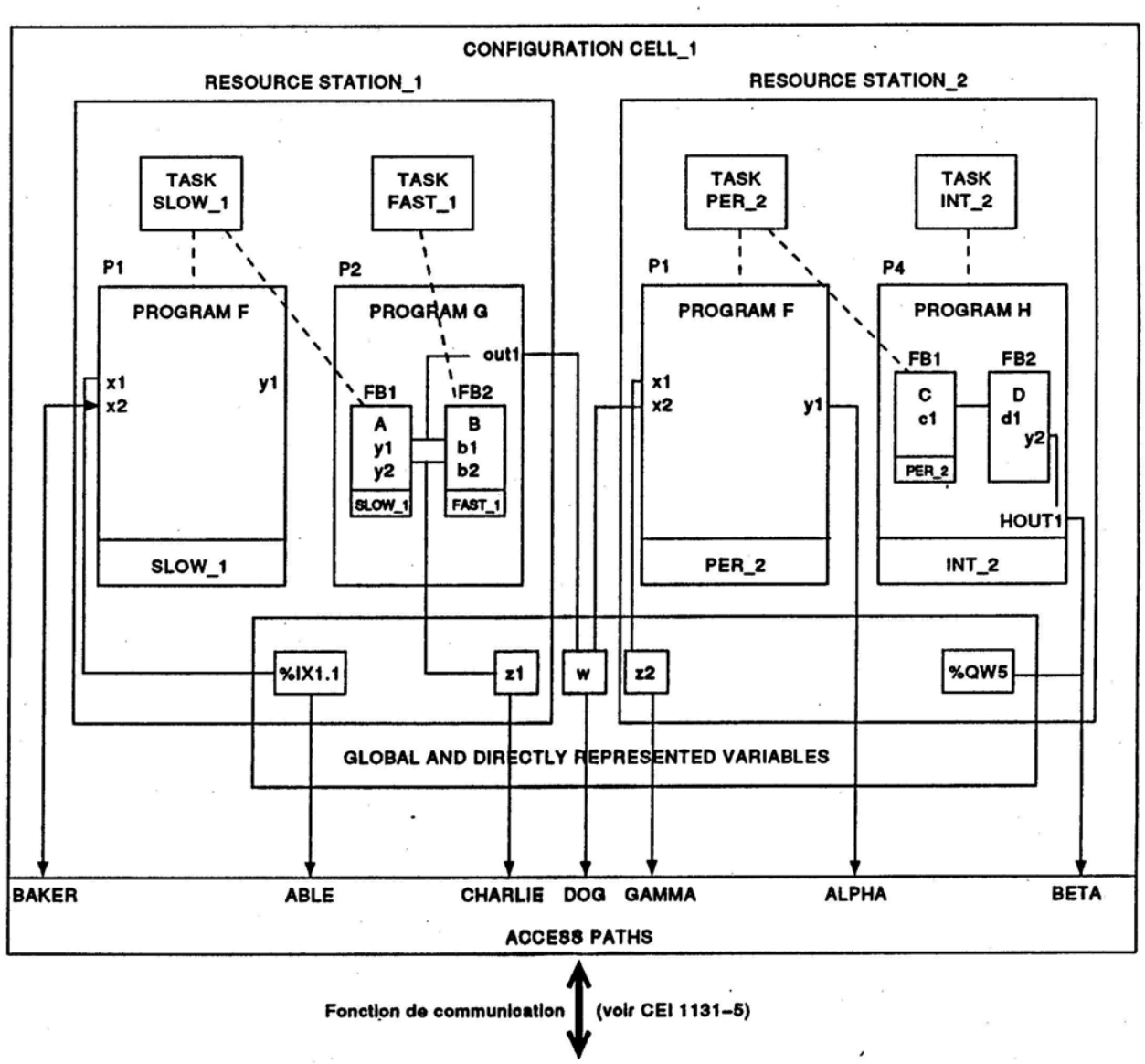


Figure 8.1. Modèle logiciel d'un automate programmable

– **fonctions** : une fonction est une unité d'organisation de programme comportant une ou plusieurs entrées mais une seule sortie. De plus la valeur de cette sortie doit correspondre de manière univoque à la valeur des entrées. En d'autres termes, cela signifie que la valeur d'une fonction ne peut pas dépendre d'états internes inaccessibles à l'utilisateur.

Les fonctions arithmétiques, numériques, de conversion de type, de comparaison ... sont des exemples typiques.

La figure 8.2. montre la mise en oeuvre de la fonction A = sin(B) dans les différents langages normalisés. Rappelons que, dans le langage IL, toutes les instructions font implicitement référence à un accumulateur.

– **blocs fonctionnels** : un bloc fonctionnel est une unité d'organisation de programme comportant une ou plusieurs entrées et une ou plusieurs sorties. De plus, un bloc fonctionnel peut comporter des variables internes, cachées à l'utilisateur, qui peuvent influencer les

sorties. Contrairement au cas de la fonction, les sorties d'un bloc fonctionnel pourront donc être différentes pour un même état de ses entrées.

Pour cette raison, si un bloc fonctionnel est utilisé en différents endroits d'un programme, il faudra créer autant de copies (on parle d'instances) du bloc, chaque copie comportant un identificateur (le nom de l'instance) et une structure de données capable de contenir ses variables internes et ses sorties.

Les temporisateurs et les compteurs sont des exemples simples mais fréquents utilisés de blocs fonctionnels. La figure 8.3 montre la mise en oeuvre d'un temporisateur dans les différents langages normalisés.

– **programmes** : un programme est défini comme un ensemble logique de tous les éléments et constructions des langages de programmation nécessaires pour le traitement des signaux destinés à commander une machine ou un processus.

La déclaration et l'utilisation de programmes sont assez semblables à celles des blocs fonctionnels excepté que les programmes ne peuvent être instanciés que dans des ressources alors que les blocs fonctionnels le sont dans des programmes ou dans d'autres blocs fonctionnels.

De plus, des chemins d'accès peuvent être associés à des programmes (cf § 8.2.1.) permettant donc à ceux-ci d'utiliser des services de communication dans le cas d'une mise en réseau.

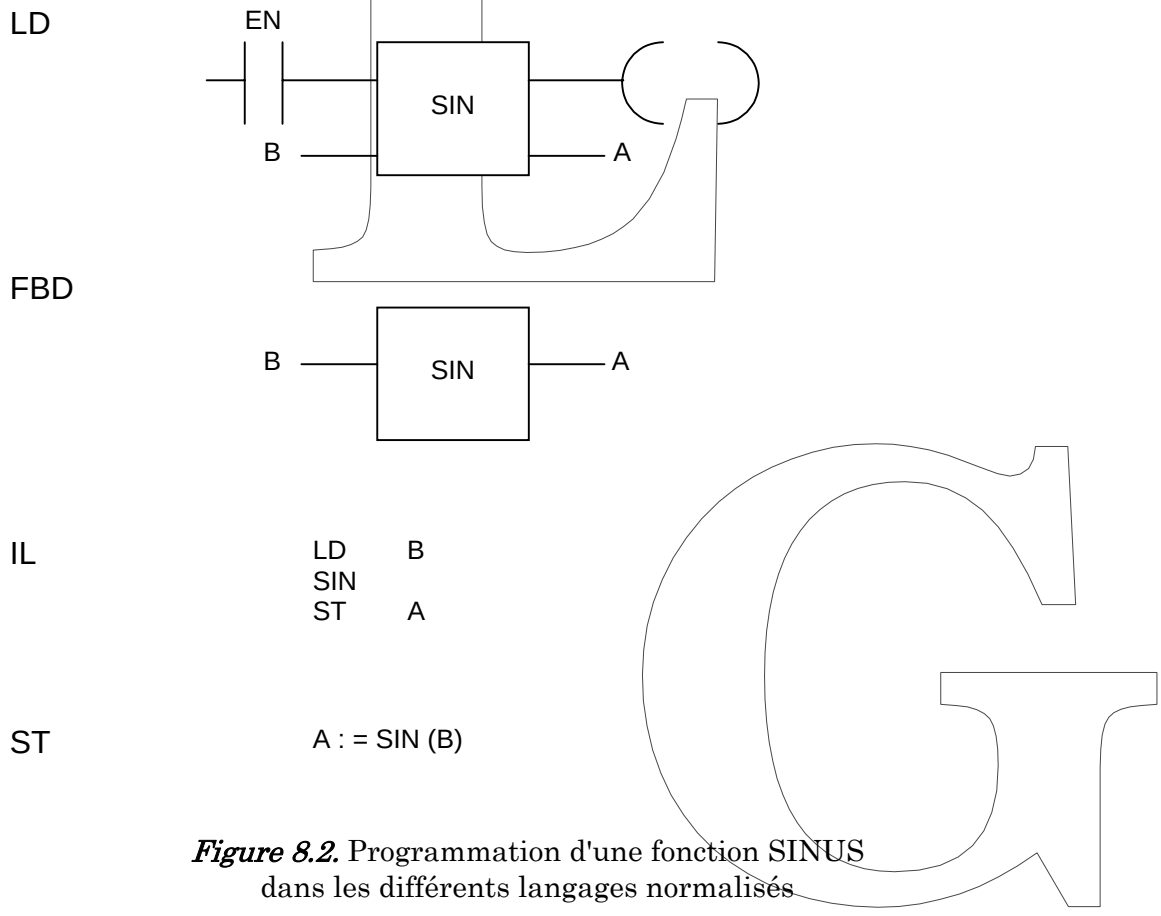


Figure 8.2. Programmation d'une fonction SINUS dans les différents langages normalisés

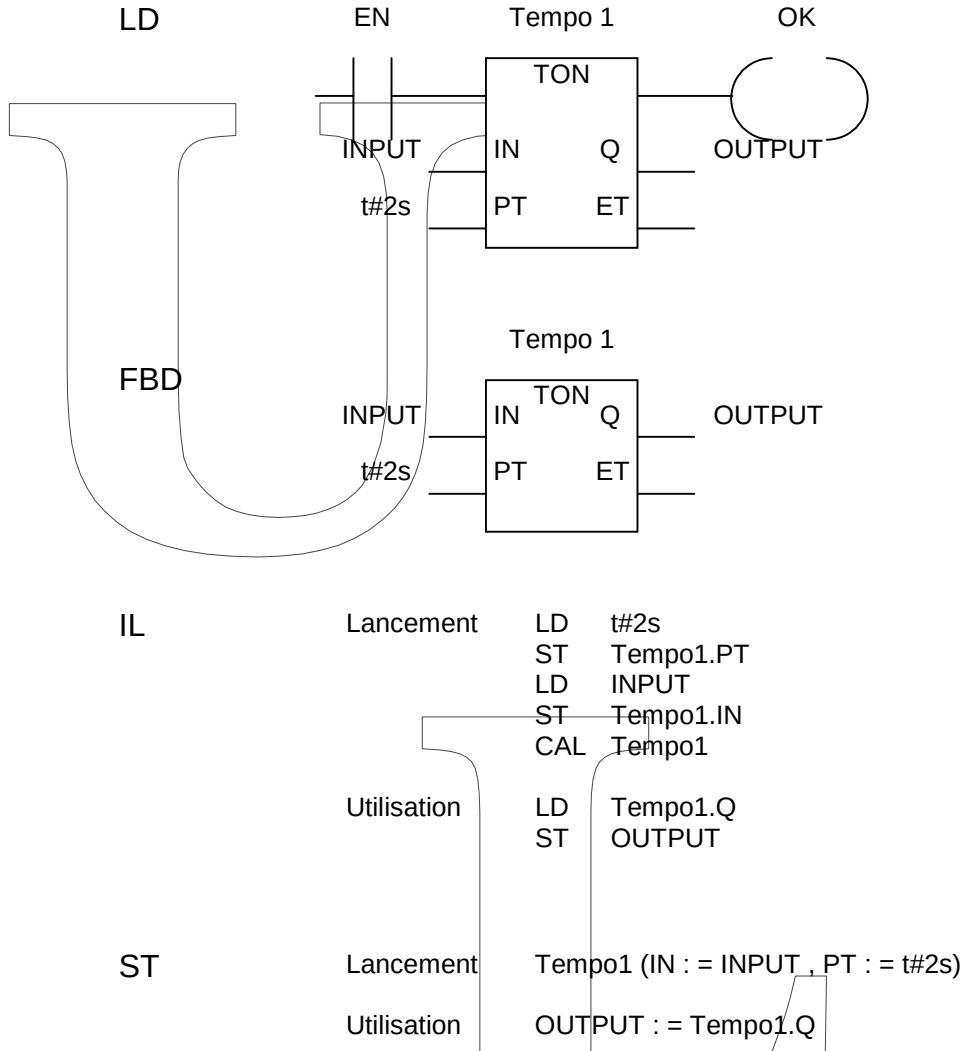


Figure 8.3. Programmation d'un bloc fonctionnel de temporisation dans les différents langages normalisés

8.2.3. DONNEES ET VARIABLES

En ce qui concerne les données et variables, la norme prévoit pratiquement les mêmes possibilités que pour les langages informatiques évolués. Le début de chaque déclaration d'unité d'organisation de programme (fonction, blocs fonctionnels, programmes) doit comporter une partie de déclaration qui spécifie le type, le champ d'application, la valeur initiale et, si nécessaire, l'emplacement physique ou logique des variables utilisées dans l'unité d'organisation. Que le lecteur se rassure cependant tout de suite, les systèmes de développement fournissent, en général, un support convivial pour ces déclarations qui se réduisent alors pratiquement à un jeu de questions/réponses.

– **type de données** : le tableau de la figure 8.4 reprend les types élémentaires de données prévus par la norme. A partir de ces types élémentaires, on peut définir des types dérivés tels que des tableaux (ensemble de données de même type) et des structures (ensemble de données de type différent). La figure 8.5. en donne quelques exemples. De même, la figure 8.6. montre comment on peut spécifier la valeur initiale par défaut pour un type de données.

Tableau 10 – Types de données élémentaires

N°	Mot clé	Type de donnée	Bits	Étendue
1	BOOL	Booléen	1	Note 8
2	SINT	Entier court	8	Note 2
3	INT	Entier	16	Note 2
4	DINT	Entier double	32	Note 2
5	LINT	Entier long	64	Note 2
6	USINT	Entier court non signé	8	Note 3
7	UINT	Entier non signé	16	Note 3
8	UDINT	Entier double non signé	32	Note 3
9	ULINT	Entier long non signé	64	Note 3
10	REAL	Nombre réels	32	Note 4
11	LREAL	Réels longs	64	Note 5
12	TIME	Durée	Note 1	Note 6
13	DATE	Date (uniquement)	Note 1	Note 6
14	TIME_OF_DAY ou TOD	Heure du jour (uniquement)	Note 1	Note 6
15	DATE_AND_TIME ou DT	Date et heure du jour	Note 1	Note 6
16	STRING	Cordon de caractères de longueur variable	Note 1	Note 7
17	BYTE	Cordon de bits de longueur 8	8	Note 7
18	WORD	Cordon de ~~caractères~~ de longueur 16 bits	16	Note 7
19	DWORD	Cordon de ~~caractères~~ de longueur 32 bits	32	Note 7
20	LWORD	Cordon de ~~caractères~~ de longueur 64 bits	64	Note 7

Figure 8.4. Types de données élémentaires

N°	Caractéristique/exemple littéral
1	Dérivation directe, à partir de types élémentaires, par exemple: TYPE R : REAL ; END_TYPE
2	Types de données énumérés, par exemple: TYPE ANALOG_SIGNAL_TYPE : (SINGLE_ENDED, DIFFERENTIAL) ; END_TYPE
3	Types de données sous-étendue, par exemple: TYPE ANALOG_DATA : INT (-4095..4095) ; END_TYPE
4	Types de données tableau, par exemple: TYPE ANALOG_16_INPUT_DATA : ARRAY [1..16] OF ANALOG_DATA ; END_TYPE
5	Types de données structurés, par exemple: TYPE ANALOG_CHANNEL_CONFIGURATION : STRUCT RANGE : ANALOG_SIGNAL_RANGE ; MIN_SCALE : ANALOG_DATA ; MAX_SCALE : ANALOG_DATA ; END_STRUCT ; ANALOG_16_INPUT_CONFIGURATION : STRUCT SIGNAL_TYPE : ANALOG_SIGNAL_TYPE ; FILTER_PARAMETER : SINT (0..99) ; CHANNEL : ARRAY [1..16] OF ANALOG_CHANNEL_CONFIGURATION ; END_STRUCT ; END_TYPE

Figure 8.5. Déclaration de types de données dérivés

N°	Caractéristique/exemple littéral
1	Initialisation de types directement dérivés; par exemple: TYPE FREQ : REAL :- 50.0 ; END_TYPE
2	Initialisation des types de données énumérés; par exemple; TYPE ANALOG_SIGNAL_RANGE : (BIPOLAR_10V, (* –10 à +10 Vc.c. *) UNIPOLAR_10V, (* 0 à +10 Vc.c. *) UNIPOLAR_1_5V, (* +1 à +5 Vc.c. *) UNIPOLAR_0_5V, (* 0 à +5 Vc.c. *) UNIPOLAR_4_20_MA, (* +4 à +20 mAc.c. *) UNIPOLAR_0_20_MA, (* 0 à +20 mAc.c. *)) := UNIPOLAR_1_5V ; END_TYPE
3	Initialisation de types de données sous-étendue; par exemple: TYPE ANALOG_DATAZ : INT (-4095..4095) := 0 ; END_TYPE
4	Initialisation de types de données tableau; par exemple: TYPE ANALOG_16_INPUT_DATAI : ARRAY [1..16] OF ANALOG_DATA := 8(–4095), 8(4095) ; END_TYPE
5	Initialisation d'éléments de types de données structurés; par exemple: TYPE ANALOG_CHANNEL_CONFIGURATIONI : STRUCT RANGE : ANALOG_SIGNAL_RANGE ; MIN_SCALE : ANALOG_DATA := –4095 ; MAX_SCALE : ANALOG_DATA := 4095 ; END_STRUCT ; END_TYPE
6	Initialisation de types de données structurés dérivés; par exemple: TYPE ANALOG_CHANNEL_CONFIGZ : ANALOG_CHANNEL_CONFIGURATIONI (MIN_SCALE := 0, MAX_SCALE := 9999) ; END_TYPE

Figure 8.6. Déclaration de valeurs initiales par défaut

– **nature des variables** : la figure 8.7 présente les mots clés qui interviennent dans la déclaration des variables. Ils en précisent le champ d'application, par exemple VAR pour une variable interne à l'unité d'organisation, VAR_GLOBAL pour une variable accessible par toutes les ressources du système. Le mot clé RETAIN déclare la variable non "volatile" c'est-à-dire que la variable doit être sauvegardée en cas de coupure d'alimentation.

Enfin, le mot clé AT permet de faire référence à des adresses physiques d'entrée/sortie ou de mémoire interne. Ces dernières sont qualifiées de variables directement représentées et ont une syntaxe particulière montrée à la figure 8.8.

La figure 8.9., enfin, donne toute une série d'exemples de déclaration de variable. On notera, à la rubrique n° 3, comment le mot clé AT permet de faire correspondre une variable symbolique et une adresse physique.

Mot clé	Utilisation de variables
VAR	Interne à l'unité d'organisation
VAR_INPUT	Fournie de l'extérieur, non modifiable dans l'unité d'organisation
VAR_OUTPUT	Fournie par l'unité d'organisation aux entités externes
VAR_IN_OUT	Fournie par des entités externes Peut être modifiée dans l'unité d'organisation NOTE - Des exemples de l'utilisation de ces variables sont donnés dans les figures 11b et 12.
VAR_EXTERNAL	Fournie par la configuration, par l'intermédiaire de VAR_GLOBAL (2.7.1) Peut être modifiée dans l'unité d'organisation
VAR_GLOBAL	Déclaration de variable globale (2.7.1)
VAR_ACCESS	Déclaration de chemin d'accès (2.7.1)
RETAIN	Variables non volatiles (voir texte précédent)
CONSTANT	Constante (variable qui ne peut être modifiée)
AT	Enoncé d'emplacement (voir 2.4.3.1)

Figure 8.7. Mots clés pour la déclaration de variables

Caractéristiques des préfixes d'emplacement et de taille pour des variables représentées directement

N°	Préfixe	Signification
1	I	Emplacement d'entrée
2	Q	Emplacement de sortie
3	M	Emplacement de mémoire
4	X	Taille d'un seul bit
5	Aucun	Taille d'un seul bit
6	B	Taille d'un octet (8 bits)
7	W	Taille d'un mot (16 bits)
8	D	Taille d'un double mot (32 bits)
9	L	Taille d'un mot long (Quad) (64 bits)

NOTES

1 Sauf déclaration contraire, le type de donnée d'une variable directement adressable, de la taille d'un "seul bit" doit être BOOL.

2 Les organismes nationaux de normalisation peuvent publier des tables de traduction de ces préfixes.

exemples de variables directement représentées

%QX75 et %Q75	Bit de sortie 75
%IW215	Emplacement du mot d'entrée 215
%QB7	Emplacement de l'octet de sortie 7
%MD48	Double mot à l'emplacement en mémoire 48
%IW2.5.7.1	Voir explication ci-après

Figure 8.8. Variables directement représentées

N°	Caractéristique/exemples	
1	Déclaration de variables volatiles, directement représentées	
	VAR AT %IW6.2 : WORD ; AT %MW6 : INT ; END_VAR	Cordon à 16 bits (note 2) Entier à 16 bits, valeur initiale = 0
2	Déclaration de variables non volatiles, directement représentées	
	VAR RETAIN AT %QW5 : WORD ; END_VAR	A la reprise à froid, %QW5 sera initialisé à un cordon à 16 bits avec la valeur 0
3	Déclaration d'emplacements de variables symboliques	
	VAR_GLOBAL LIM_SW_5 AT %IX27 : BOOL ; CONV_START AT %QX25 : BOOL ; TEMPERATURE AT %IW28 : INT ; END_VAR	Affecte le bit d'entrée 27 à la variable booléenne LIM_SW_5 (note 2) Affecte le bit de sortie 25 à la variable booléenne CONV_START Affecte le mot d'entrée 28 à la variable entière TEMPERATURE (note 2)
4	Affectation d'emplacements de tableaux	
	VAR INARY AT %IW6 : ARRAY [0..9] OF INT ; END_VAR	Déclare un tableau de 10 entiers à affecter à des emplacements d'entrée contigus, commençant à %IW6 (note 2)
5	Affectation automatique en mémoire de variables symboliques	
	VAR CONDITION_RED : BOOL ; IBOUNCE : WORD ; MYDUB : DWORD ; AWORD, BWORD, CWORD : INT ; MYSTR : STRING(10) ; END_VAR	Affecte un bit de mémoire à la variable booléenne CONDITION_RED Affecte un mot de mémoire à la variable IBOUNCE à chaîne de 16 bits Affecte un double mot de mémoire à la variable MYDUB à chaîne de 32 bits Affecte 3 mots de mémoire séparés pour les variables entières AWORD, BWORD et CWORD Affecte la mémoire qui doit contenir un cordon d'une longueur maximale de 10 caractères. Après l'initialisation, le cordon a une longueur égale à 0 et contient le cordon vide".
6	Déclaration de tableaux	
	VAR THREE : ARRAY[1..5,1..10,1..8] OF INT ; END_VAR	Affecte 400 mots de mémoire pour un tableau tridimensionnel d'entiers
7	Déclaration de tableaux non volatiles	
	VAR RETAIN RTBT : ARRAY[1..2,1..3] OF INT ; END_VAR	Déclare le tableau non-volatile RTBT, avec des valeurs initiales 0 pour tous les éléments lors d'une "reprise à froid".
8	Déclaration de variables structurées	
	VAR MODULE_8_CONFIG : ANALOG_16_INPUT_CONFIGURATION ; END_VAR	Déclaration d'une variable de type de donnée dérivé (voir tableau 12)

NOTES:

1 Les caractéristiques 1 à 4 ne peuvent être utilisées que dans des déclarations PROGRAM et VAR_GLOBAL, respectivement définies aux paragraphes 2.5.3 et 2.7.1.

2 L'initialisation des entrées du système est propre à la mise en oeuvre; se reporter au paragraphe 2.4.2.

Figure 8.9. Exemple de déclaration de variables

8.2.4. TEXTES ET LIBELLES

Dans son souci d'assurer la portabilité des programmes, la norme CEI 1131-3 va même jusqu'à spécifier les jeux de caractères à utiliser et la forme des libellés de données et de variables.

– jeux de caractères : ceux-ci doivent être conformes à la norme ISO/IEC 646.

– libellés : la figure 8.10. donne quelques exemples de libellés relatifs aux variables, aux constantes et aux commentaires; la figure 8.11. au temps et à la date.

Caractéristiques des identificateurs

N°	Description des caractéristiques	Exemples
1	Majuscules et nombres	IW215 IW215Z QX75 IDENT
2	Majuscule et minuscule, nombres, caractères de soulignement intégrés	Tous les exemples ci-dessus, plus: LIM_SW_5 LimSw5 abcd ab_Cd
3	Majuscule et minuscule, nombres, caractères de soulignement en tête ou intégrés	Tous les exemples ci-dessus, plus: _MAIN _12V7

Libellés numériques

N°	Description des caractéristiques	Exemples
1	Libellés entiers	–12 0 123_456 +986
2	Libellés réels	–12.0 0.0 0.4560 3.14159_26
3	Libellés réels avec des des exposants	–1.34E-12 ou –1.34e-12 1.0E+6 ou 1.0e+6 1.234E6 ou 1.234e6
4	Libellés en base 2	2#1111_1111 (255 décimal) 2#1110_0000 (240 décimal)
5	Libellés en base 8	8#377 (255 décimal) 8#340 (240 décimal)
6	Libellés en base 16	16#FF ou 16#ff (255 décimal) 16#E0 ou 16#e0 (240 décimal)
7	Zéro et un booléens	0 1
8	FAUX et VRAI booléens	FAUX VRAI
NOTE - Les mots clés FAUX et VRAI correspondent respectivement aux valeurs booléennes de 0 et 1.		

La caractéristique commentaire

N°	Description des caractéristiques	Exemples
1	Commentaires	(**********) (* Un commentaire encadré *) (**********)

Figure 8.10. Libellés et commentaires

Tableau 7 – Caractéristique de libellés de temps

N°	Description de la caractéristique	Exemples
1a	Libellés de temps sans caractères de soulignement: Préfixe court	T#14ms T#14.7s T#14.7m T#14.7h t#14.7d t#25h15m t#5d14h12m18s3.5ms
1b	Préfixe long	TIME#14ms time#14.7s
2a	Libellés de temps avec caractères de soulignement: Préfixe court	t#25h_15m t#5d_14h_12m_18s_3.5ms
2b	Préfixe long	TIME#25h_15m time#5d_14h_12m_18s_3.5ms

Tableau 9 – Exemples de libellés de date et heure du jour

Notation de préfixes longs	Notation de préfixes courts
DATE#1984-06-25 date#1984-06-25	D#1984-06-25 d#1984-06-25
TIME_OF_DAY#15:36:55.36 time_of_day#15:36:55.36	TOD#15:36:55.36 tod#15:36:55.36
DATE_AND_TIME#1984-06-25-15:36:55.36 date_and_time#1984-06-25-15:36:55.36	DT#1984-06-25-15:36:55.36 dt#1984-06-25-15:36:55.36

Figure 8.11. Libellés de temps et de date

8.3. LE LANGAGE A LISTE D'INSTRUCTIONS IL

Les opérateurs du langage IL normalisés sont repris au tableau de la figure 8.12. Le modificateur N provoque l'inversion logique de l'opérande qui suit. Le modificateur C indique que l'opération considérée n'est effectuée que si le résultat logique courant est VRAI.

N°	Opérateur	Modificateurs	Opérande	Sémantique
1	LD	N	Note 2	Rendre le résultat courant égal à l'opérande
2	ST	N	Note 2	Mémoriser le résultat à l'emplacement de l'opérande
3	S R	Note 3 Note 3	BOOL BOOL	Positionner l'opérande booléen à 1 Remettre l'opérande booléen à 0
4 5 6 7	AND & OR XOR	N, (N, (N, (N, (	BOOL BOOL BOOL BOOL	AND booléen AND booléen OR booléen OR exclusif booléen
8 9 10 11	ADD SUB MUL DIV	((((	Note 2 Note 2 Note 2 Note 2	Addition Soustraction Multiplication Division
12 13 14 15 16 17	GT GE EQ NE LE LT	((((((	Note 2 Note 2 Note 2 Note 2 Note 2 Note 2	Comparaison: >; Comparaison: >= Comparaison: = Comparaison: <>; Comparaison: <= Comparaison: <
18 19 20	JMP CAL RET	C, N C, N C, N	LABEL NAME	Saut vers l'étiquette Appel d'un bloc fonctionnel (note 4) Retour d'une fonction appelée ou d'un bloc fonctionnel
21	)			Evaluation d'une opération différée

Figure 8.12. Opérateurs de base du langage à instructions IL

8.4. LE LANGAGE LITTTERAL STRUCTURE ST

La figure 8.13. présente les opérateurs du langage ST tandis que la figure 8.14. décrit les énoncés disponibles dans ce langage.

N°	Opération	Symbole	Priorité
1	Mise entre parenthèses	(Expression)	MAXIMALE
2	Evaluation de Fonction Exemples:	Identificateur (liste d'arguments) LN(A), MAX(X,Y), etc.	
3	Exponentiation	**	
4 5	Négation Complément	– NOT	
6 7 8	Multiplication Division Modulo	* / MOD	
9 10	Addition Soustraction	+ –	
11	Comparaison	< , > , <= , >=	
12 13	Egalité Inégalité	= <>	
14 15	AND booléen AND booléen	& AND	
16	OR exclusif booléen	XOR	
17	OR booléen	OR	MINIMALE

Figure 8.13. Opérateurs de base du langage littéral structuré ST

N°	Type d'énoncé/référence	Exemples
1	Affectation (3.3.2.1)	A := B ; CV := CV+1 ; C := SIN(X) ;
2	Lancement d'un bloc fonctionnel et utilisation de sortie FB (3.3.2.2)	CMD_TMR(IN := %IX5, PT := T#300ms) ; A := CMD_TMR.Q ;
3	RETURN (3.3.2.2)	RETURN ;
4	IF (3.3.2.3)	D := B*B - 4*A*C ; IF D < 0.0 THEN NROOTS := 0 ; ELSIF D = 0.0 THEN NROOTS := 1 ; X1 := - B/ (2.0*A) ; ELSE NROOTS := 2 ; X1 := (-B+SQRT(D))/(2.0*A) ; X2 := (-B-SQRT(D))/(2.0*A) ; END_IF ;
5	CASE (3.3.2.3)	TW := BCD_TO_INT(THUMBWHEEL) ; TW_ERROR := 0 ; CASE TW OF 1,5 : DISPLAY := OVEN_TEMP ; 2 : DISPLAY := MOTOR_SPEED ; 3 : DISPLAY := GROSS_TARE ; 4,6..10 : DISPLAY := STATUS (TW-4) ; ELSE DISPLAY := 0 ; TW_ERROR := 1 ; END_CASE ; QW100 := INT_TO_BCD(DISPLAY) ;

6	FOR (3.3.2.4)	J := 101 ;
		FOR I := 1 TO 100 BY 2 DO
		IF WORDS[I] = 'KEY' THEN
		J := I ;
		EXIT ;
		END_IF ;
		END_FOR ;

7	WHILE (3.3.2.4)	J := 1 ; WHILE J <= 100 & WORDS[J] <> 'KEY' DO J := J+2 ; END_WHILE ;
8	REPEAT (3.3.2.4)	J := -1 ; REPEAT J := J+2 ; UNTIL J = 101 OR WORDS[J] = 'KEY' END_REPEAT ;
9	EXIT (3.3.2.4)	EXIT ;
10	Enoncé Vide	;

Figure 8.14. Enoncés du langage littéral structuré ST

8.5. LES LANGAGES GRAPHIQUES

Dans ce paragraphe, on commence par présenter des conventions valables pour tous les langages graphiques :

– représentation des lignes et des blocs : figure 8.15.
– éléments de commande d'exécution graphiques : figure 8.16.

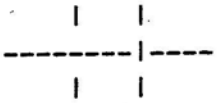
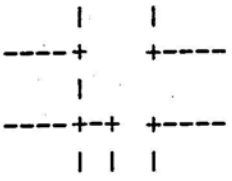
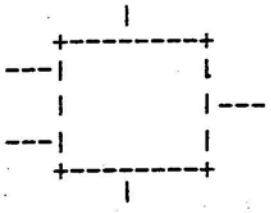
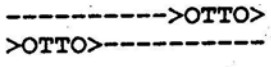
N°	Caractéristique	Exemple
1 2	Lignes horizontales: Caractère "moins" ISO/IEC 646 Graphique ou semi-graphique	
3 4	Lignes verticales: Caractère "ligne verticale" ISO/IEC 646 Graphique ou semi-graphique	
5 6	Jonction ligne horizontale/ligne verticale: Caractère "plus" ISO/IEC 646 Graphique ou semi-graphique	
7 8	Croisements de lignes sans jonction: Caractères ISO/IEC 646 Graphique ou semi-graphique	
9 10	Coins connectés et non connectés: Caractères ISO/IEC 646 Graphique ou semi-graphique	
11 12	Blocs avec lignes connectées: Caractères ISO/IEC 646 Graphique ou semi-graphique	
13 14	Connecteurs utilisant des caractères ISO/IEC 646: Connecteur Suite d'une ligne connectée Connecteurs graphiques ou semi-graphiques	

Figure 8.15. Langages graphiques – Eléments communs

N°	Symbole/exemple	Signification
1	1----->>LABELA	Saut inconditionnel: Langage FBD
2	| +----->>LABELA |	Langage LD
3	X----->>LABELB +---+ %IX20---| & |--->>NEXT %MX50---| | +---+ NEXT: +---+ %IX25---|>=1|---%QX100 %MX60---| | +---+	Saut conditionnel (Langage FBD) Exemple: Condition de saut Cible du saut
4	| X +-| |----->>LABELB | | | %IX20 %MX50 +---| |-----| |--->>NEXT | | NEXT: | %IX25 %QX100 | +----| |----+----( )---+ | %MX60 | +----| |----+ | | |	Saut conditionnel (Langage LD) Exemple: Condition de saut Cible du saut
5	| X +--| |---<RETURN> |	Retour conditionnel: Langage LD
6	X---<RETURN>	Langage FBD
7	END_FUNCTION END_FUNCTION_BLOCK	Retour inconditionnel: d'une FUNCTION d'un FUNCTION_BLOCK
8	| +---<RETURN> |	Autres représentations possibles en langage LD

Figure 8.16. Langages graphiques – Eléments de commande d'exécution

8.5.1. LANGAGES A CONTACTS LD

La figure 8.17. présente les symboles normalisés des contacts et la figure 8.18. ceux des bobinages.

N°	Symbole	Description
	Contacts statiques	
1 2	*** --| |-- ou *** --! !--	***Contact normalement au repos*** L'état de la liaison gauche est copié sur la liaison droite si l'état de la variable booléenne associée (désignée par "*****") est ON. Dans tous les autres cas, l'état de la liaison droite est OFF.
3 4	*** --|/|-- ou *** --!/!--	***Contact normalement au travail*** L'état de la liaison gauche est copié sur la liaison droite si l'état de la variable booléenne associée (désignée par "*****") est OFF. Dans tous les autres cas, l'état de la liaison droite est OFF.
	Contacts détecteurs de transition	
5 6	*** --|P|-- ou *** --!P!--	***Contact détecteur de transition positive*** L'état de la liaison droite est ON d'une évaluation de cet élément jusqu'à l'évaluation suivante, lorsqu'une transition de OFF à ON de la variable associée est détectée, alors que la liaison gauche est à l'état ON. Dans tous les autres cas, l'état de la liaison droite doit être OFF.
7 8	*** --|N|-- ou *** --!N!--	***Contact détecteur de transition négative*** L'état de la liaison droite est ON d'une évaluation de cet élément jusqu'à l'évaluation suivante, lorsqu'une transition de ON à OFF de la variable associée est détectée, alors que la liaison gauche est à l'état ON. Dans tous les autres cas, l'état de la liaison droite doit être OFF.
	NOTE - Comme spécifié en 2.1.1, le point d'exclamation "!" doit être utilisé lorsque la variante nationale du jeu de caractères ne comporte pas la barre verticale "|".	

Figure 8.17. Langage à contacts LD – Symboles des contacts

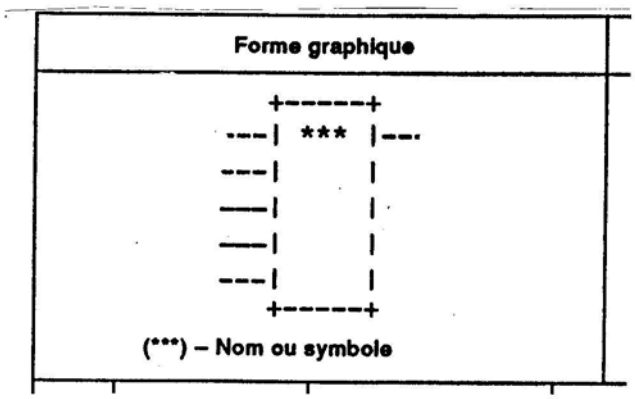
Bobinages fugitifs		
N°	**Symbole**	**Description**
1	*** --( )--	***Bobinage*** **L'état de la liaison gauche est copié dans la variable booléenne associée et vers la liaison droite.**
2	*** --(/)--	***Bobinage "négativé"*** **L'état de la liaison gauche est copié vers la liaison droite. L'inverse de l'état de la liaison gauche est copié dans la variable booléenne associée, c'est-à-dire que, si par exemple, l'état de la liaison gauche est OFF, l'état de la variable associée est ON, et vice versa.**
Bobinages à maintien		
3	*** --(S)--	***Bobinage SET (verrouillage)*** **La variable booléenne associée est mise à l'état ON lorsque la liaison gauche est à l'état ON, et elle conserve cet état jusqu'à remise à zéro par le bobinage RESET.**
4	*** --(R)--	***Bobinage RESET (déverrouillage)*** **La variable booléenne associée est remise à l'état OFF lorsque la liaison gauche est à l'état ON, et elle conserve cet état jusqu'à son positionnement par un bobinage SET.**
Bobinages à mémorisation (voir note)		
5	*** ----(M)----	**Bobinage de mémorisation**
6	*** ----(SM)----	**Bobinage de mémorisation SET**
7	*** ----(RM)----	**Bobinage de mémorisation RESET**
Bobinage détecteurs de transition		
8	*** --(P)--	***Bobinage détecteur de transition positive*** **L'état de la variable booléenne associée est ON d'une évaluation de cet élément jusqu'à l'évaluation suivante, lorsqu'une transition de OFF à ON de la liaison gauche est détectée. L'état de la liaison gauche est toujours copié sur la liaison droite.**
9	*** --(N)--	***Bobinage détecteur de transition négative*** **L'état de la variable booléenne associée est ON de l'évaluation de cet élément jusqu'à l'évaluation suivante, lorsqu'une transition de ON à OFF de la liaison gauche est détectée. L'état de la liaison gauche est toujours copié sur la liaison droite.**
NOTE - Les bobinages 5, 6 et 7 ont respectivement le même effet que les bobinages 1, 3 et 4, excepté que la variable booléenne associée est automatiquement déclarée être dans la mémoire non volatile, sans qu'il soit nécessaire d'utiliser explicitement la déclaration VAR RETAIN définie en 2.4.2.		

Figure 8.18. Langage à contacts LD – Symboles des contacts

8.5.2. LANGAGE EN BLOCS FONCTIONNELS FBD

La programmation en langage FBD consiste à relier graphiquement entre eux des blocs symbolisant les fonctions souhaitées. La forme graphique générale de ces blocs est montrée à la figure 8.19.a. L'inversion logique de signaux booléens d'entrée ou de sortie peut être indiquée par un petit cercle comme montré à la figure 8.19.b.

Cela étant, toute unité d'organisation de programme du type fonction ou bloc fonctionnel (voir § 8.6.) peut être utilisée dans le langage FBD.



a.

Inversion graphique de signes booléens

N°	Caractéristiques	Représentation
1	Entrée inversée	
2	Sortie inversée	

b.

Figure 8.19. Langage à blocs fonctionnels FBD
a. Forme graphique générale b. Inversion de signaux booléens

8.6. FONCTIONS ET BLOCS FONCTIONNELS STANDARDS

La norme CEI 1131-3 définit un nombre déjà assez important d'unités d'organisation standards du type fonction et bloc fonctionnel. A côté des opérations élémentaires faisant l'objet d'une symbolique propre, les différents langages décrits aux paragraphes précédents peuvent utiliser ces unités d'organisation standards par l'intermédiaire d'une syntaxe appropriée qui a été évoquée au paragraphe 8.2.2.

Il serait impossible de les passer toutes en revue ici et nous nous contenterons donc d'une simple énumération.

8.6.1. FONCTIONS STANDARDS

Type	Nom	Symbole éventuel	Explication
Logique sur bit	AND	&	
	OR	>=1	
	XOR	=2k+1	
	NOT		
Arithmétique	ADD	+	
	MUL	*	
	SUB	-	
	DIV	/	
	MOD		
	EXPT	**	
	MOVE	:=	
Numérique	ABS		
	SQRT		
	LN		logarithme naturel
	LOG		logarithme en base 10
	EXP		
	SIN		
	COS		
	ASIN		
	ACOS		
	ATAN		
Comparaison	GT	>	
	GE	>=	
	EQ	=	
	LE	<=	
	LT	<	
	NE	<>	
Conversion de type	*_TO_**		conversion du type * vers le type **
Décalage	SHL		décalage à gauche
	SHR		décalage à droite
	ROR		rotation à droite
	ROL		rotation à gauche
Sélection	SEL		sélection binaire
	MAX		sélection de maximum
	MIN		sélection de minimum
	LIMIT		limiteur
	MUX		multiplexeur
Chaîne de caractères	LEN		longueur d'une chaîne
	LEFT		caractères le plus à gauche
	RIGHT		caractères le plus à droite
	MID		caractères intermédiaires
	CONCAT		
	INSERT		
	DELETE		
	REPLACE		
	FIND		

8.6.2. BLOCS FONCTIONNELS STANDARDS

Type	Nom	Explication
Bistables	SR	Forcé dominant
	RS	Réinitialisé dominant
	SEMA	Sémaphore
Détecteur de front	R-TRIG	Front montant
	F-TRIG	Front descendant
Compteurs	CTU	Compteur
	CTD	Décompteur
	CTUD	Compteur/Décompteur
Temporisateurs	TP	Impulsion
	TON	Temporisation à l'enclenchement
	TOF	Temporisation au déclenchement
	RTC	Horloge temps réel (donne la date et l'heure du jour)

Chapitre 9

OUTIL D'AIDE AU DEVELOPPEMENT DES PROGRAMMES

9.1. CONCEPT D'ATELIER DE GENIE LOGICIEL

9.1.1. CYCLE DE VIE D'UN SYSTEME AUTOMATISE

Le développement d'un système automatisé comprend généralement une série de phases successives allant de la spécification à la mise en route suivant un modèle en "V" proche de celui normalisé par l'AFCIQ (Association Française pour la Qualité) pour les informaticiens et appelé "cycle de vie" (figure 9.1.).

L'originalité du triple V présenté ici est de prendre en compte l'intégration des cycles parallèles de réalisation de trois métiers : le programmeur, l'électricien et le câbleur.

– **Spécification générale** : Première phase du cycle de vie d'une application automatisée, elle consiste à établir le cahier des charges fonctionnel à partir des documents contractuels fournis par le client.

– **Spécification détaillée** : Les fonctions énoncées dans le cahier des charges fonctionnel sont détaillées ici et décomposées en fonctions élémentaires suivant différentes méthodes, graphiques pour la plupart, afin de produire un document de spécifications externes.

– **Conception générale** : Cette phase consiste à implanter les différentes fonctions et structures correspondantes sur un matériel cible en réalisant des architectures matérielle et logicielle. L'architecture matérielle décrit l'ensemble des équipements utilisés, leur positionnement géographique dans l'usine ainsi que leur logique de connexion. Par équipement, elle décrit aussi la configuration des différents composants d'automatismes utilisés et des automates programmables en particulier.

– **Conception détaillée** : A ce stade de développement, le projet d'automatisation fait intervenir trois métiers en parallèle : celui de l'automaticien devant concevoir, réaliser et tester les différents modules logiciels identifiés plus haut, celui de l'électricien devant concevoir, réaliser et tester les équipements, celui de l'installateur devant concevoir, réaliser et tester les raccordements inter-équipements du site.

La conception détaillée d'un logiciel pour automates programmables se fait en décrivant chaque tâche identifiée en conception générale à l'aide des différents langages disponibles. La conception détaillée d'un équipement électrique et des câblages correspondants est réalisée à l'aide des différents logiciels de schématique existants sur le marché, intégrables de préférence à cette démarche de développement afin d'en assurer sa cohérence.

– **Réalisation** : La réalisation logicielle consiste à lancer la compilation du programme source de conception pour obtenir un programme objet automate. Les phases de réalisation

matérielle et chantier sont exécutées manuellement en plate-forme et sur le site d'après les schémas de principe et de réalisation obtenues en conception détaillée.

– **Tests unitaires** : Les modules logiciels ainsi réalisés sont testés individuellement puis globalement sur des automates de tests, suivant le cahier de tests unitaires établi lors de la conception détaillée. Il en va de même pour les modules matériels et de chantier réalisés à tester individuellement suivant les cahiers de tests unitaires correspondants.

– **Intégration et essais en plate-forme** : Après intégration des modules logiciels dans le matériel cible réalisée conformément aux cahiers d'intégration établi en conception générale, la phase d'essais en plate-forme consiste à valider le système d'automatisme dans un environnement provisoire.

– **Mise en route** : Cette phase finale d'intégration et de mise en route sur le site comprend les essais et la mise en service des différents organes de puissance et de commande associée ainsi que la mise en service globale de l'installation.

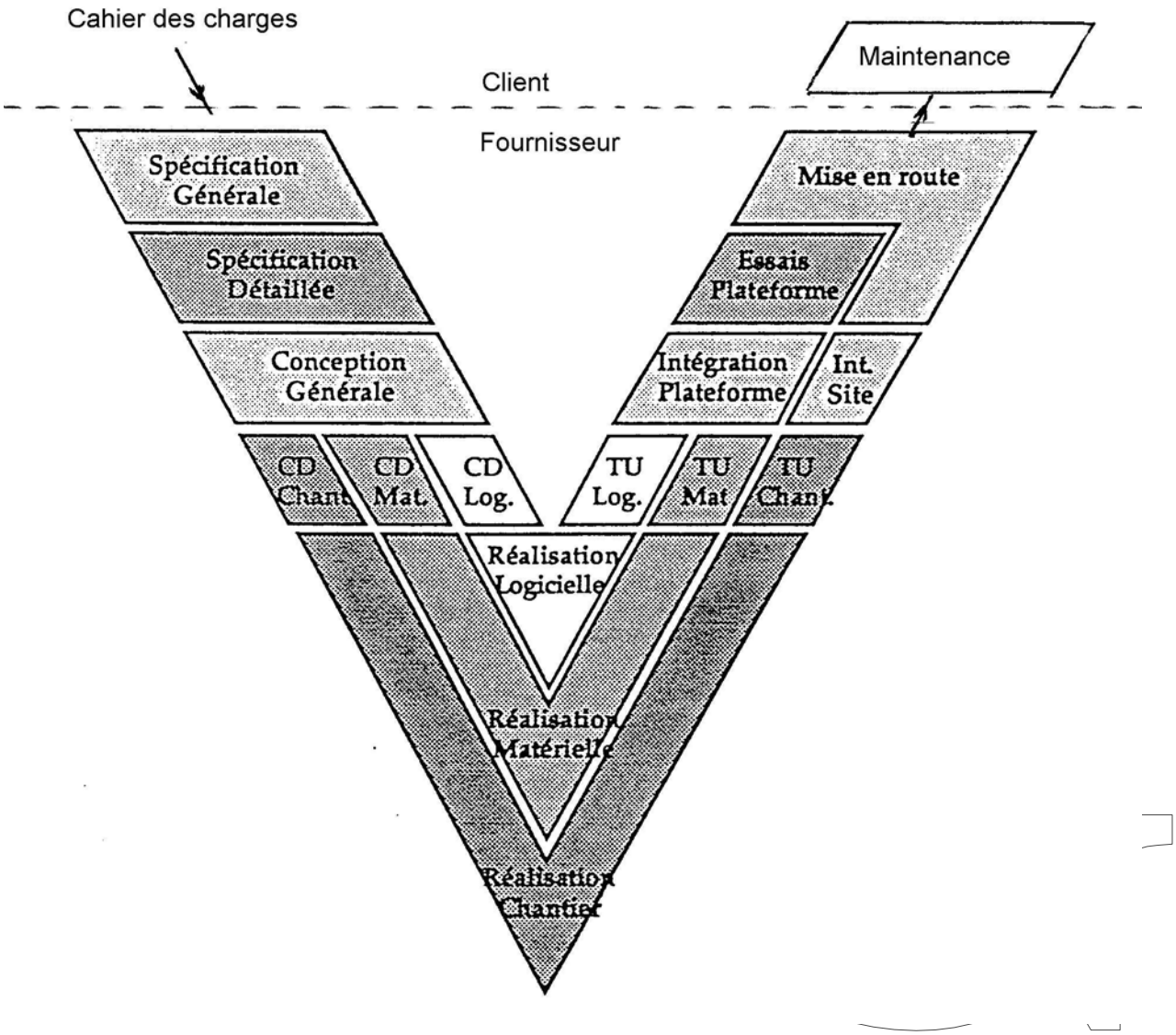


Figure 9.1. Cycle de vie d'un système automatisé (Source : GRAL)

9.1.2. LE POSTE DE TRAVAIL DE L'AUTOMATICIEN

Dans le contexte général présenté ci-dessus, la notion de Poste de Travail de l'Automaticien (PTA), apparue en France en 1984, vise à mettre à la disposition des concepteurs et des mainteneurs d'applications automatisées un environnement de développement et de maintenance informatique unique et cohérent.

C'est ainsi que le PTA devrait intégrer, autour d'une base de données commune, les outils, aujourd'hui dispersés, de programmation des automates, de simulation, de schématique électrique, d'aide au diagnostic et de supervision. (cf. figure 9.2.)

De plus, les applications automatisées développées sur le PTA devraient pouvoir tourner sur n'importe quel équipement programmable, automate ou ordinateur de processus.

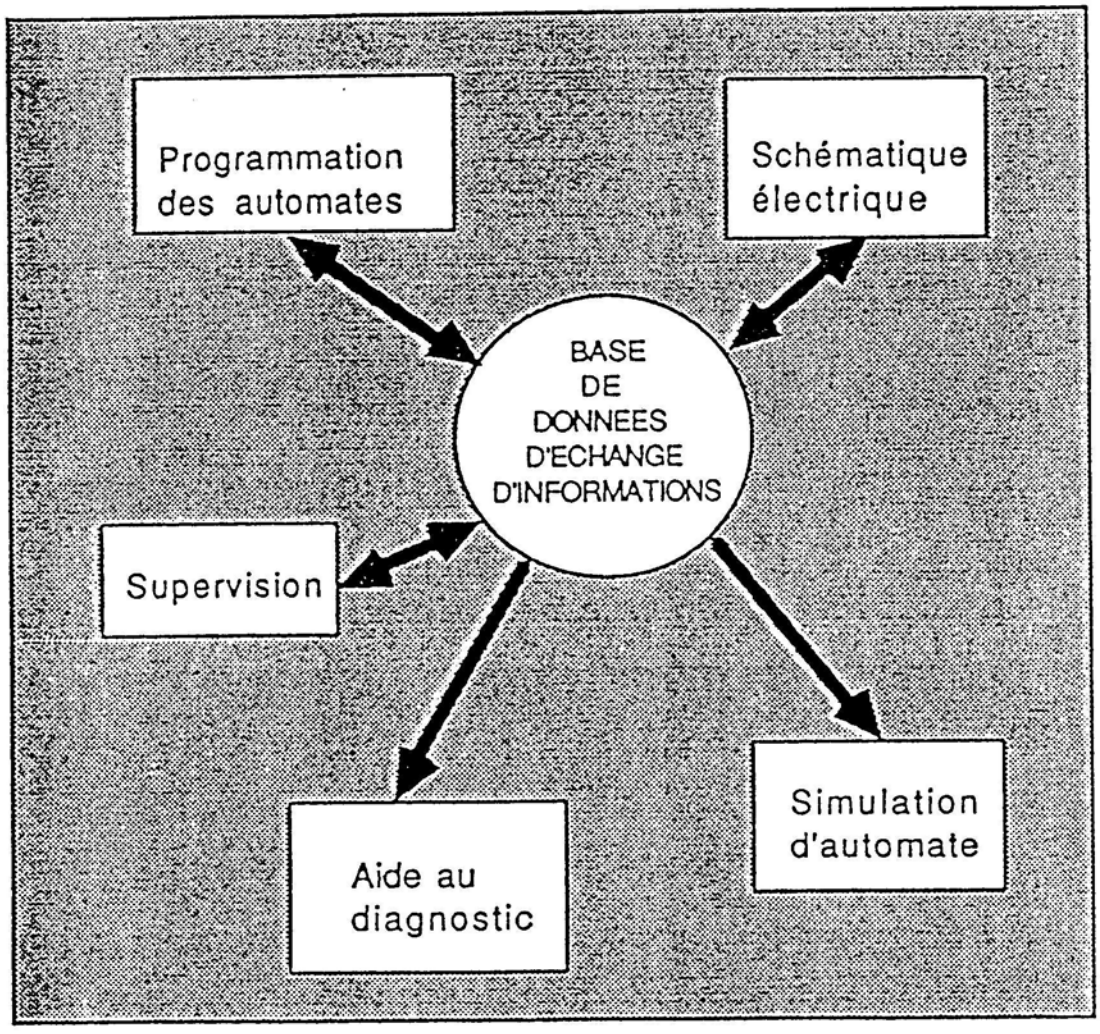


Figure 9.2. Poste de Travail de l'Automaticien (PTA)

9.1.3. ETAT DES REALISATIONS

A l'heure actuelle, on est encore fort éloigné de l'intégration prônée au paragraphe précédent. Force est cependant de reconnaître que la plupart des constructeurs d'automates ont déployé, ces dernières années, des efforts considérables pour améliorer l'environnement de programmation offert aux utilisateurs. Il est vrai qu'ils ont été singulièrement aidés dans cet effort par le développement des micro-ordinateurs et des systèmes d'exploitation conviviaux tels que WINDOWS. Malheureusement, ces environnements de programmation sont totalement incompatibles d'un constructeur à l'autre.

A côté de cela, et malgré un succès commercial mitigé, des maisons de logiciel indépendantes continuent de développer des ateliers logiciels dont le principal atout réside précisément dans l'universalité. Ces ateliers permettent en effet de structurer, de programmer et même de tester une application indépendamment de l'automate cible.

Il s'agit évidemment d'une universalité toute relative qui se limite assez logiquement aux automates dominants du marché. D'autre part, pour atteindre cette universalité, les concepteurs ont forcément dû se restreindre, sur le plan fonctionnel, au commun dénominateur des automates cibles prévus.

C'est pourquoi, fort pragmatiquement, les ateliers logiciels universels offrent la possibilité d'intégrer dans une application, des sous-ensembles programmés dans les langages spécifiques de l'automate cible de manière à avoir accès aux fonctions spéciales prévues par le constructeur.

Le recours à ces fonctions spéciales est souvent indispensable dans la mesure où elles touchent au contrôle de l'exécution (interruptions, traitement des erreurs, ...), aux communications, à la gestion de messages, etc.

Leur utilisation entraîne évidemment la perte de portabilité de l'application. De plus, elle oblige l'automaticien à acquérir une maîtrise approfondie de l'automate cible puisque ces fonctions spéciales sont souvent aussi les plus compliquées à mettre en oeuvre !

La normalisation présentée au chapitre 8 est évidemment de nature à conforter la position des développeurs d'ateliers logiciels sur le marché, en leur offrant une base de travail universellement reconnue. De plus, comme on l'a vu, la norme aborde aussi les aspects du contrôle de l'exécution des programmes et des communications ce qui, à plus ou moins long terme, devrait résoudre le problème des "fonctions spéciales".

De toute évidence, il serait assez difficile, à ce stade, de prédire l'évolution des choses. Nous croyons cependant qu'à côté des constructeurs d'automates, il y a place pour des "constructeurs" indépendants d'ateliers logiciels, de même d'ailleurs, et cela est déjà admis depuis longtemps, qu'il y a place pour des développeurs indépendants de superviseurs. Il s'agit en effet de "métiers" certes complémentaires mais qui font appel à des compétences très sensiblement différentes.

L'exemple de l'informatique classique avec le succès incontestable des plates-formes de développement universelles WINDOWS et UNIX devrait nous inciter à un certain optimisme.

Dans la suite de ce chapitre, nous présenterons, de manière générique, les services de base que l'on est en droit d'attendre aujourd'hui d'un système moderne de développement de programmes pour automates.

9.2. PROGRAMMATION

9.2.1. STRUCTURATION DES PROGRAMMES ET DES DONNEES

Au paragraphe 6.4., on a déjà expliqué l'importance de la structuration au niveau du développement, de la mise au point et de la réutilisabilité des programmes. Tout logiciel d'automate doit ainsi pouvoir être décomposé en une hiérarchie de modules représentant des entités de traitement cohérentes, que ce soit d'un point de vue fonctionnel (décomposition d'une fonction complexe en sous-fonctions plus simples) ou d'un point de vue topologique (commande de sous-ensembles du processus) ou des deux à la fois.
Ces modules doivent pouvoir être manipulés séparément mais doivent aussi pouvoir être archivés et, restaurés globalement, avec toutes leurs liaisons, sous un nom générique d'application.

9.2.2. MIXITE DES LANGAGES

La plupart des systèmes de développement offrent maintenant le choix à l'utilisateur entre plusieurs langages de programmation, langages qui peuvent d'ailleurs être mélangés au sein d'une même application.

Le langage à relais est toujours présent, dans la mesure où c'est pratiquement le seul en usage aux Etats-Unis. Notons que les fonctions spéciales (temporisations, opérations arithmétiques, etc.) s'introduisent dans ce langage sous forme de blocs fonctionnels, si bien que le langage à relais n'est pas très éloigné, en pratique, d'un langage à blocs fonctionnels.

L'important, en l'occurrence, est la nature graphique très visuelle et concrète de ces langages. Ils facilitent sans conteste la compréhension du fonctionnement d'un automatisme même par du personnel peu qualifié. Par contre, leur édition s'avère en général particulièrement fastidieuse.

En Europe, les langages littéraux, plus abstraits mais infiniment plus compacts, sont assez largement entrés dans les moeurs, principalement le langage en liste d'instructions. (type IL).

Enfin, en ce qui concerne le langage GRAFCET ou SFC, on a déjà discuté, au paragraphe 6.1.1., de sa pénétration encore relativement réduite dans l'industrie. Soulignons néanmoins que les constructeurs américains eux-mêmes commencent à le proposer, en tant qu'outil de structuration générale des programmes, le corps de ceux-ci continuant d'être écrit en langage à relais.

Sur un plan strictement pratique cependant, c'est assurément le langage littéral structuré (type ST) qui s'associe le plus harmonieusement au GRAFCET pour la description des réceptivités et des actions. Sa compacité permet en effet d'introduire les équations correspondantes directement dans le graphisme du GRAFCET et, de plus, sa proximité du langage parlé améliore grandement la lisibilité du GRAFCET. Le GRAFCET de niveau 2 peut, de ce fait, devenir pratiquement identique au GRAFCET de niveau 1 !

9.2.3. VARIABLES SYMBOLIQUES ET COMMENTAIRES

– Variables symboliques

La majorité des systèmes de développement actuels permettent au programmeur d'attacher des mnémoniques, c'est-à-dire des noms symboliques rappelant leur fonction, aux entrées, sorties et variables internes de l'automate. Il peut alors utiliser ces mnémoniques dans l'écriture des équations en lieu et place des adresses physiques des variables. On comprend aisément que ceci est de nature à améliorer la lisibilité des programmes.

Il faut cependant prendre garde qu'en général c'est l'adresse physique qui reste prioritaire. Ainsi, si, après avoir écrit un programme à l'aide de mnémoniques, on affecte un mnémonique donné à une autre adresse physique, (à la suite d'une modification du câblage par exemple), le programme effectif ne sera absolument pas modifié !

En d'autres termes, le mnémonique doit être compris comme étant une sorte de commentaire associé à une variable physique et non à un substitut de celle-ci.

Si, dès lors, on veut introduire une meilleure indépendance entre programmes et adresses physiques, il est conseillé de créer, dans les bits internes de l'automate, une table image des entrées et une table image des sorties. Les programmes seront alors écrits sur base de ces bits internes ou de mnémoniques associés. En cas de modification de câblage, il suffira alors d'ajuster quelques instructions au niveau de la copie des tables images, le reste du programme demeurant inchangé.

– Commentaires

Une documentation fournie est évidemment indispensable pour la mise au point et la maintenance des programmes. Les systèmes de développement doivent donc, au minimum, permettre d'associer un commentaire à chaque instruction de programme. De plus, il est souhaitable de pouvoir inclure dans le logiciel des notices explicatives décrivant les fonctionnalités des différents modules de programme.

Toutes ces informations doivent obligatoirement faire partie intégrante de l'application, c'est-à-dire être archivées sous le nom générique de celle-ci.

9.2.4. FACILITES D'EDITION

De plus en plus habitués à la convivialité de l'environnement WINDOWS, les utilisateurs attendent maintenant des systèmes de développement pour automates, des facilités d'édition telles que, par exemple :

– menus déroulants et aide en ligne qui évitent de devoir mémoriser une multitude de commandes d'édition

– multi-fenêtrage et fonction copier/coller très utiles en l'occurrence, car les programmes d'automates sont souvent répétitifs du fait que les possibilités d'indexation des variables sont fort limitées voire inexistantes.

Cependant, ces mêmes utilisateurs souhaitent aussi conserver les côtés positifs des anciennes consoles de programmation spécialisées, à savoir la vérification immédiate de la

syntaxe, voire même la possibilité de programmer ON-LINE, avec toutes les sécurités que cela implique (cf. paragraphe 9.2.5.)

9.2.5. PROGRAMMATION ON-LINE

Depuis toujours, la possibilité de programmer ON-LINE, c'est-à-dire avec exécution immédiate des instructions au fur et à mesure de leur écriture, a constitué une des originalités les plus marquantes des automates programmables dans le contexte informatique général.

Elle est commode en phase de mise au point d'une application. Elle peut s'avérer indispensable pour modifier ou étendre des programmes contrôlant des équipements qui ne peuvent être arrêtés sous peine d'être endommagés (certains fours par exemple).

Il est clair qu'il s'agit là d'une pratique extrêmement dangereuse qui doit être réservé à du personnel particulièrement averti.

Elle pose également un problème au niveau de la gestion des commentaires. En effet, comme l'automate ne possède normalement pas de mémoire de masse propre et que la mémoire centrale reste malgré tout un élément coûteux, il n'est pas pensable, à l'heure actuelle, d'y stocker les commentaires en même temps que les programmes. Ces commentaires vont donc rester physiquement localisés sur la console. Lors de la connexion de la console avec l'automate, les commentaires seront virtuellement remis en relation avec les programmes de manière transparente pour l'utilisateur. Ce dernier devra cependant veiller à travailler avec la même version de l'application dans la console et dans l'automate sous peine de perdre la cohérence entre commentaires et programmes.

9.3. AIDE A LA MISE AU POINT DES PROGRAMMES

Des outils plus ou moins élaborés d'aide à la mise au point des programmes sont prévus dans tous les systèmes de développement. Des aides ON-LINE doivent toujours exister. Certains constructeurs ont aussi prévu des tests OFF-LINE par simulation.

9.3.1. VISUALISATION DYNAMIQUE

C'est l'outil de base des tests ON-LINE. Il permet de suivre dynamiquement le déroulement d'un programme ou d'un ensemble de variables prédéterminé.

– **Programme** : ce mode de visualisation est particulièrement bien adapté aux langages de type graphique (SFC, LD, FBD). Dans un langage à contacts par exemple, les relais passants seront indiqués en surbrillance. De même, dans le cas d'un GRAFCET, les étapes actives seront marquées par une coloration particulière.

La visualisation dynamique des programmes est aussi possible en langage liste d'instructions, car chaque ligne de programme ne contient qu'une variable. On peut donc sans problème afficher la valeur courante de la variable sur la même ligne.

– **Table des variables** : avec ce type d'outil, il est possible de constituer sur l'écran de la console une table de variables relatives, par exemple, à un sous-ensemble du processus et d'en suivre dynamiquement l'évolution. Plusieurs tables peuvent être ainsi constituées et mémorisées. L'utilisateur peut les rappeler à sa guise.

– **Histogramme de variables** : enfin, certains systèmes de développement offrent même la possibilité de tracer sur l'écran des histogrammes relatifs à un groupe déterminé de variables.

9.3.2. MODES D'EXECUTION

Pour faciliter l'analyse des phénomènes, il est en général possible d'interrompre le déroulement cyclique des programmes. Ceci peut se faire de plusieurs manières :

– fonctionnement cycle à cycle : l'automate s'arrête après chaque cycle de scrutation

– points d'arrêt : le programmeur place des points d'arrêt à des endroits du programme qui l'intéressent particulièrement. Arrivé à ces points, l'automate s'arrête, permettant à l'utilisateur d'analyser à son aise l'état des variables.

– pas à pas : le programme s'exécute ici instruction par instruction sous le contrôle du programmeur.

9.3.3. FORÇAGE DE VARIABLES

La plupart des systèmes de développement offre encore à l'utilisateur la possibilité de fixer manuellement l'état de certaines variables.

Le forçage des sorties, par exemple, va permettre de tester le câblage et le fonctionnement des actuateurs avant même que les programmes de l'automate ne soient opérationnels.

Le forçage des entrées, quant à lui, permettra de tester les programmes même si tous les capteurs ne sont pas encore installés.

Dans le cas d'une programmation en langage GRAFCET, il peut être utile de pouvoir forcer l'évolution du graphe de manière à pallier l'absence momentanée de certains signaux intervenant dans les réceptivités.

Comme la programmation ON-LINE, le forçage ON-LINE est un outil à manier avec beaucoup de précaution car il peut amener le système dans des états incohérents potentiellement dangereux. C'est pourquoi certains constructeurs ont prévu de pouvoir inhiber les sorties de l'automate lors des forçages destinés à la mise au point des programmes.

9.3.4. SIMULATION

La possibilité de forcer les entrées et d'observer les sorties sans que celles-ci n'agissent sur le processus (inhibition) constitue déjà une manière de test en simulation.

Certains systèmes de développement permettent de faire ce genre de test complètement OFF-LINE, c'est-à-dire sans que la console ne soit raccordée à l'automate. Dans la foulée, la convivialité des manipulations et des observations a été améliorée. On retrouve, par exemple, à l'écran l'image réaliste de voyants lumineux (pour les sorties) et de boutons poussoirs (pour les entrées) sur lesquels il suffit de cliquer avec la souris pour en changer l'état logique (ISAGRAF).

Remarquons cependant que la procédure décrite ici diffère totalement des techniques de test qui seront présentées dans le tome 2 des présentes notes. Dans ce dernier cas, en effet,

la partie opérative (c'est-à-dire le processus) est émulée par un PC avec sa dynamique propre et tout se passe dès lors comme si l'automate travaillait effectivement sur le processus.

Dans le cas présent, c'est l'utilisateur lui-même qui doit reproduire les réactions du processus en actionnant les entrées au vu de l'état des sorties.
On comprend bien qu'on ne puisse aller fort loin en complexité avec la méthode de simulation présentée dans ce paragraphe.

9.4. GENERATION DU DOSSIER TECHNIQUE

De gros efforts ont été consentis, dans les systèmes de développement, pour automatiser autant que faire se peut la génération du dossier technique d'un projet.

Au départ d'une table des matières standard, l'utilisateur peut personnaliser le dossier en réarrangeant l'ordre des rubriques et en éliminant celles qui ne lui sont pas nécessaires.

De même, la mise en page et la police de caractères peuvent être adaptées au goût de chacun. Enfin, un cartouche spécifique peut être défini.

A titre d'exemple, on reproduit ci-dessous, la table des matières standard de l'atelier logiciel ISAGRAF qui est certainement, à ce jour, un des plus complets du marché.

- Descripteur du projet
- Arbre hiérarchique (liens entre les programmes)

- Code source pour chaque programme
- Fichier "agenda" pour chaque programme

- Définitions communes
- Définitions globales
- Définitions locales pour chaque programme
- Variables globales
- Variables locales pour chaque programme

- Paramètres de l'application
- Câblage des E/S
- Listes de variables
- Tables de conversion
- Références croisées - représentation condensée
- Références croisées - représentation détaillée
- Résumé des déclarations
- Carte d'adressage "réseau" des variables

- Historique des modifications