

Introduction au Grafcet / SFC	4
1) Partie 1 : Éléments constitutifs du Grafcet	5
1.1) Étape initiale	5
1.2) Étapes	5
1.3) Transitions.....	5
1.4) Liaisons orientées.....	5
1.5) Fonctionnement séquentiel du Grafcet	6
1.6) Rappel de définition	6
1.7) Points à retenir (ingénieurs & techniciens)	6
1.8) Norme internationale IEC 60848	7
1.9) Lien avec IEC 61131-3.....	7
1.10) Bonne pratique – nommer les actions	7
1.11) Avantage en conception	7
1.12) Différence avec un graphe d'état	7
1.13) Synthèse	8
2) Partie 2 – Règles d'évolution du Grafcet.....	9
2.1) Règle n°1 : Initialisation	9
2.2) Règle n°2 : Franchissement d'une transition	9
2.3) Règle n°3 : Évolution des étapes actives	9
2.4) Importance de ces règles	10
2.5) Représentation logicielle	10
2.6) Bonne pratique – valider les réceptivités	10
2.7) Cas particuliers.....	11
2.8) Synthèse	12
3) Partie 3 : Structures de séquence	13
3.1) Grafcet linéaire	13
3.2) Grafcet avec sélection de séquence (Divergence OU)	14
3.3) Grafcet avec reprise de séquence	15
3.4) Grafcet avec séquences simultanées (Divergence ET)	16
3.5) Comparaison des structures	17
3.6) Synthèse	18
4) Partie 4 : Niveaux de description & Exemple.....	19
4.1) Niveaux de description du Grafcet :.....	19
4.2) Exemple : Barrière de parking	20
4.3) Avantages de cette approche par niveaux	21
4.4) Synthèse	22
5) Partie 5 : Systèmes combinatoires vs séquentiels.....	23
5.1) Système combinatoire.....	23
5.2) Système séquentiel	24

5.3)	Exemple typique : bascule RS	25
5.4)	Différences clés.....	26
5.5)	Conclusion.....	26
5.6)	Synthèse	27
6)	Partie 6 : Programmation d'un système séquentiel	28
6.1)	Difficulté de programmation empirique (Technique n°1)	28
6.2)	Méthode structurée avec Grafcet (Technique n°2)	28
6.3)	Exemple simple (cuisson de pâtes)	28
6.4)	Formalisme graphique vs Graphe d'état.....	29
6.5)	Bonnes pratiques	29
6.6)	Lien avec cycle en V.....	30
6.7)	Erreurs courantes	31
6.8)	Synthèse	32
7)	Partie 7 : Programmation avec variables d'état.....	33
7.1)	Méthode 1 – Équations booléennes équivalentes	33
7.2)	Méthode 2 – Algorithme basé sur une variable d'état	33
7.3)	Différence boucle CPU vs boucle process.....	34
7.4)	Bonnes pratiques	34
7.5)	Synthèse	35
8)	Partie 8 : Application complète (Barrière de parking)	36
8.1)	Spécification opérative	36
8.2)	Spécification commande.....	37
8.3)	Grafcet (version simplifiée en texte)	37
8.4)	Points clés à retenir.....	37
8.5)	Synthèse	38
9)	Fonctionnalité Rockwell	39
9.1)	Qualifier	39
9.2)	Boolean Action.....	41
9.3)	Synthèse : Action vs Boolean Action.....	41
9.4)	Etape Stop.....	42
10)	Conclusion générale – Grafcet / SFC	44
10.1)	Atouts du Grafcet pour l'industrie.....	45
10.2)	Synthèse finale	45

Introduction au Grafcet / SFC

Le **Grafcet** (*Graphe Fonctionnel de Commande Étape/Transition*), appelé **SFC** (*Sequential Function Chart*) en anglais, est un outil graphique de description du fonctionnement d'un système automatisé.

- Il modélise le **comportement séquentiel** d'un procédé, étape après étape.
- Il s'applique à tout type de commande **logique d'automatisme industriel** : électrique, pneumatique ou hydraulique.
- Il peut être utilisé aussi bien pour un système **câblé** (relais, contacteurs...) que pour un système **programmé** (automates programmables industriels).

En d'autres termes, le Grafcet décrit **QUOI faire et DANS QUEL ORDRE**, sans préciser comment la technologie réalise ces actions.

1) Partie 1 : Éléments constitutifs du Grafcet

1.1) Étape initiale

- **Représentation** : un **double carré**.
- **Rôle** : c'est l'état actif au **démarrage** (mise sous tension).
- **Remarque** : généralement, aucune action n'y est associée → c'est un état de repos ou d'attente.

1.2) Étapes

- **Symbole** : carré simple numéroté.
- Chaque numéro doit être **unique** dans le Grafcet.
- À chaque étape est **associée une ou plusieurs actions**.
- **Activation** : quand l'étape est active, l'action correspondante est exécutée.
- **Représentation des actions** : rectangle contenant l'action (ex. *MONTER MOTEUR*), relié à l'étape par un trait horizontal.

1.3) Transitions

- **Symbole** : trait vertical reliant deux étapes, coupé par un trait horizontal.
- Associées à une **réceptivité** = condition logique (équation booléenne).
- **Principe** : si la réceptivité = 1 (vraie), la transition est franchie → passage à l'étape suivante.

1.4) Liaisons orientées

- Relient les étapes entre elles, en indiquant le **sens de progression** de la séquence.
- Il existe souvent une **liaison de retour** vers l'étape initiale, permettant de boucler le cycle.

1.5) Fonctionnement séquentiel du Grafcet

- **Au démarrage** : seule l'étape initiale est active.
- **Pendant le cycle** :
 - Une transition est franchie lorsque :
 1. L'étape précédente est active,
 2. La réceptivité associée est vraie.
 - L'étape suivante devient alors active, l'étape précédente est désactivée.
 - L'action liée à la nouvelle étape passe à l'état actif (=1).
- **À la fin** : lorsque la dernière transition est franchie, le Grafcet **revient à l'étape initiale**, pour recommencer un nouveau cycle.

1.6) Rappel de définition

Le Grafcet est donc :

- Un **outil graphique de description temporelle**,
- Qui formalise le **fonctionnement séquentiel** d'un système,
- Composé de trois éléments essentiels :
 - **Étapes** (avec actions associées),
 - **Transitions** (avec conditions/réceptivités),
 - **Liaisons orientées** (enchaînement logique entre étapes).

1.7) Points à retenir (ingénieurs & techniciens)

- Le Grafcet est une **langue commune** entre concepteurs, automaticiens, et techniciens de maintenance.
- Chaque **étape** est un état du système, avec son action associée.
- Chaque **transition** est une condition qui permet le passage d'une étape à une autre.
- Une **séquence Grafcet** est toujours **exclusive** : une étape active entraîne l'action correspondante et désactive la précédente.
- Le **retour à l'étape initiale** assure la répétition cyclique du processus.

1.8) Norme internationale IEC 60848

- Imagine le Grafcet comme une **grammaire internationale** pour décrire les séquences.
- Grâce à la norme IEC 60848, un ingénieur allemand, un technicien français et un automaticien chinois comprendront le même schéma.
- Comme la **portée musicale** en musique : peu importe le pays, tout le monde lit les notes de la même manière.

1.9) Lien avec IEC 61131-3

- L'IEC 61131-3 est la **norme des langages d'automates** (comme Ladder, ST, FBD, SFC).
- Elle dit : « Le Grafcet (SFC) est un vrai langage de programmation API ».
- Exemple : dans **TIA Portal** (Siemens), tu peux directement programmer en SFC → chaque étape et transition du Grafcet devient du code exécutable.

1.10) Bonne pratique – nommer les actions

- Toujours utiliser des **verbes d'action** en majuscules (*OUVRIR, FERMER, ATTENDRE*).
- Pourquoi ? Parce que même en maintenance, en lisant une étape, un technicien comprend tout de suite ce qui doit se passer.

1.11) Avantage en conception

- Le Grafcet est un **pont** entre le **cahier des charges** (“La barrière doit s’ouvrir quand la carte est validée”) et le **code automate**.
- On gagne du temps → pas besoin de refaire la logique à chaque étape du projet.

1.12) Différence avec un graphe d'état

- Graphe d'état → les états sont des ronds et l'action est **mélangée** avec la condition.
- Grafcet → étapes (carrés) = actions, transitions (traits) = conditions → **plus clair et organisé**.

1.13) Synthèse

- **Grafcet = SFC** → langage graphique pour décrire un système séquentiel.
- Utilisé en **automatisme industriel** : électrique, pneumatique, hydraulique, câblé ou programmé.
- **Éléments clés** :
 - Étape initiale (double carré, active au départ),
 - Étapes (carrés numérotés avec actions),
 - Transitions (traits avec conditions logiques),
 - Liaisons orientées (enchaînement + retour au départ).
- **Cycle** : Initialisation → franchissement des transitions → activation/désactivation → retour étape initiale.
- **Valeur ajoutée** : lisible par tous, normé, structurant pour la conception et la maintenance.

2) Partie 2 – Règles d'évolution du Grafcet

2.1) Règle n°1 : Initialisation

- Au **démarrage du système**, une étape unique est définie comme **étape initiale**.
- Elle est **active dès la mise sous tension**.
- Elle correspond généralement à un **état de repos** ou à un comportement neutre.

Étape active : une étape est dite active si et seulement si l'action qui lui est associée est en cours d'exécution.

2.2) Règle n°2 : Franchissement d'une transition

Une transition est franchie uniquement si deux conditions sont remplies :

1. L'**étape précédente** est active,
2. La **réceptivité** (condition logique associée à la transition) est vraie.

Exemple : *Si le capteur "fin de course bas" est activé (réceptivité vraie) et que l'étape "descente en cours" est active, alors la transition est franchie → passage à l'étape suivante.*

2.3) Règle n°3 : Évolution des étapes actives

Lorsqu'une transition est franchie :

- L'**étape suivante** devient active,
- L'**étape précédente** est désactivée.

Cela assure une **progression séquentielle claire** : à chaque instant, le système sait **où il en est** et **quelles actions exécuter**.

2.4) Importance de ces règles

- Elles garantissent une **séquence déterministe** : pas de confusion entre plusieurs étapes possibles.
- Elles assurent la **robustesse du cycle**, en empêchant des activations multiples ou incohérentes.
- Elles sont le fondement de la **programmation structurée** en Grafcet/SFC, car elles traduisent la logique en **états** et **transitions**.

2.5) Représentation logicielle

- Dans les automates, si tu programmes en SFC, l'API gère automatiquement :
 - Activation d'une étape,
 - Désactivation de la précédente,
 - Vérification des conditions.
- Tu n'as pas besoin de recoder ces règles → elles sont "incluses dans le langage".

2.6) Bonne pratique – valider les réceptivités

- Exemple : un capteur de fin de course peut avoir un **rebond** électrique → si on ne filtre pas, la transition risque de s'activer plusieurs fois.
- Solution → utiliser des **temporisations anti-rebonds**.

2.7) Cas particuliers

- Divergence OU → choix exclusif (comme prendre une bifurcation de route).
- Divergence ET → deux chemins en parallèle (comme faire tourner la machine et ventiler en même temps).
- Les règles de base restent valables : activation/désactivation étape par étape.

2.8) Synthèse

- **Initialisation** : au démarrage, seule l'étape initiale (repos) est active.
- **Étape active** = l'action correspondante est en cours d'exécution.
- **Franchissement d'une transition** si :
 - Étape précédente active,
 - Réceptivité vraie.
- **Évolution** : transition franchie → étape suivante activée → étape précédente désactivée.
- Ces règles garantissent un fonctionnement **séquentiel, clair et déterministe**.

3) Partie 3 : Structures de séquence

3.1) Grafcet linéaire

Définition

- Une succession simple et unique d'**étapes** et de **transitions**.
- Le principe fondamental : **alternance stricte** → Étape → Transition → Étape → Transition...

Caractéristiques

- Une seule **trajectoire possible**, sans embranchement ni simultanéité.
- Structure adaptée aux systèmes **simples et déterministes**.

Exemple industriel

- **Convoyeur monotâche :**
 1. Étape 1 : Démarrer le moteur.
 2. Transition : Le capteur "objet détecté" est vrai.
 3. Étape 2 : Arrêter le moteur.
 4. Transition : Le capteur "objet évacué" est vrai.
 5. Retour étape initiale.

Tout le cycle suit un seul chemin, sans choix alternatif.

Avantages

- Simplicité, lisibilité immédiate.
- Facile à programmer, peu sujet aux erreurs.

Limites

- Pas adapté aux systèmes **avec choix conditionnels** ou **actions simultanées**.

3.2) Grafset avec sélection de séquence (Divergence OU)

Définition

- Permet de **choisir une branche parmi plusieurs** à un moment donné.
- Une seule branche est activée en fonction de la **réceptivité**.

Exemple industriel (Percètris)

- Une machine perce des pièces uniquement si nécessaire :
 - Si la pièce comporte déjà un trou → évacuation directe (branche 1).
 - Si la pièce n'a pas de trou → perçage → évacuation (branche 2).

Avantages

- Introduit la **prise de décision** dans le cycle.
- Permet une meilleure **adaptabilité** aux conditions réelles.

Limites

- Complexité accrue : il faut gérer toutes les conditions possibles.
- Risque d'oubli d'un cas → bloquer la séquence.

Bonne pratique

- Toujours vérifier que les **conditions de sélection** sont **mutuellement exclusives** (sinon risque d'ambiguïté).

3.3) Grafcet avec reprise de séquence

Définition

- Utilisé pour **répéter plusieurs fois** un ensemble d'actions identiques.
- Permet d'éviter de dupliquer inutilement un Grafcet.

Exemple industriel (Palan)

- Un palan doit déplacer **deux pièces successivement**.
- Chaque cycle inclut 4 mouvements :
 - Monter, Descendre, Droite, Gauche.
- La séquence est **reprise** pour traiter la deuxième pièce.

Avantages

- Évite la **redondance** dans le Grafcet.
- Simplifie la **lecture** et la **programmation**.

Limites

- Il faut prévoir une **condition de sortie** claire (ex. nombre de répétitions, signal d'arrêt).
- Risque de boucle infinie si la condition n'est pas gérée correctement.

3.4) Grafcet avec séquences simultanées (Divergence ET)

Définition

- Plusieurs branches peuvent être activées **en parallèle**.
- Chaque branche exécute ses actions de manière **indépendante**.

Exemple industriel (Palan)

- Pour **gagner du temps** lors du retour en position initiale :
 - La montée et le déplacement gauche peuvent se faire en **même temps**, au lieu de les exécuter l'un après l'autre.

Avantages

- Réduction du **temps de cycle**.
- Exploite les possibilités des systèmes capables de mouvements simultanés (robots, machines multi-axes...).

Limites

- Attention aux **conflits de ressources** :
 - Deux actions parallèles peuvent nécessiter le même moteur, la même pompe, etc.
 - Risque de blocage ou d'endommagement si non prévu.
- Obligation de prévoir une **re-synchronisation** des branches avant de poursuivre le cycle global.

Bonne pratique

- Toujours définir un **point de convergence** (transition commune) pour attendre la fin des branches parallèles avant de reprendre la séquence globale.

3.5) Comparaison des structures

Structure	Caractéristique	Exemple typique	Avantage	Limite
Linéaire	Séquence unique	Convoyeur simple	Simplicité	Pas de choix, pas de simultanéité
OU (sélection)	Choix exclusif entre branches	Percètris (percer ou non)	Flexibilité, adaptabilité	Risque d'ambiguïté si conditions mal définies
Reprise	Répétition d'actions	Palan manipulant plusieurs pièces	Évite duplication, lisibilité	Risque de boucle infinie
ET (simultanéité)	Branches parallèles indépendantes	Palan retour rapide	Optimisation temps de cycle	Risques de conflit, besoin de synchronisation

3.6) Synthèse

- **Linéaire** → succession simple d'étapes/transitions, clair mais limité.
- **OU (sélection)** → choix entre plusieurs branches, une seule activée.
- **Reprise** → permet de répéter plusieurs fois les mêmes actions.
- **ET (simultanéité)** → exécution parallèle de plusieurs branches, gain de temps.
- **Bonnes pratiques :**
 - Vérifier conditions exclusives (OU),
 - Définir condition de sortie (Reprise),
 - Prévoir synchronisation (ET).
- Ces structures rendent le Grafcet **souple, puissant et adapté à des systèmes industriels complexes.**

4) Partie 4 : Niveaux de description & Exemple

4.1) Niveaux de description du Grafcet :

Niveau 1 – Partie opérative (fonctionnelle)

- Décrit **les fonctions à réaliser**, sans préciser la technologie.
- On utilise uniquement des **phrases simples** :
 - *Autoriser l'accès*
 - *Lever la barrière*
 - *Maintenir ouverte 15 secondes*
 - *Redescendre la barrière*

C'est une vision **abstraite et indépendante** de l'automatisme.

Niveau 2 – Partie commande (technologique)

- Décrit le système **avec ses composants réels**.
- Exemple barrière de parking :
 - **Moteur** réversible (piloté par deux contacteurs : KM1 pour montée, KM2 pour descente).
 - **Lecteur de carte magnétique** (autorisation d'accès).
 - **Capteurs de position** :
 - FC1 = position haute,
 - FC2 = position basse.

Ici, on fait le lien entre la logique fonctionnelle et les **éléments matériels concrets**.

Niveau 3 – Partie automate (programmation API)

- Décrit le système avec ses **adresses entrées/sorties automates**.
- Exemple barrière de parking :
 - %I1.0 → Signal carte magnétique (Accès).
 - %I1.1 → Détecteur haut (FC1).
 - %I1.2 → Détecteur bas (FC2).
 - %M1 → Mémoire interne (temporisation, état intermédiaire, etc.).

Ce niveau sert de **plan direct pour la programmation** dans l'automate.

4.2) Exemple : Barrière de parking

Fonctionnement opératif

1. Un utilisateur présente sa carte → accès autorisé.
2. La barrière s'ouvre (moteur en montée).
3. Quand la barrière atteint la position haute (capteur FC1 actif) → attente de 15 secondes.
4. Après le délai → la barrière redescend (moteur en descente).
5. Retour en position basse (capteur FC2 actif).
6. Cycle terminé → retour à l'étape initiale.

Fonctionnement commande

- Le moteur est commandé par **KM1 (montée)** et **KM2 (descente)**.
- La carte active un signal d'autorisation.
- Les fins de course FC1 et FC2 sécurisent les positions.
- Une temporisation interne maintient la barrière ouverte pendant 15 s.

Fonctionnement automate

- L'entrée %I1.0 valide la carte → condition d'ouverture.
- Les entrées %I1.1 (FC1) et %I1.2 (FC2) sécurisent les positions.
- Les sorties %Q0.0 et %Q0.1 commandent les contacteurs (KM1/KM2).
- Une mémoire interne %M1 gère la temporisation de 15 s.

4.3) Avantages de cette approche par niveaux

- **Lisibilité** : chaque acteur (opérateur, automaticien, programmeur) a une vision adaptée.
- **Cohérence** : le passage opératif → commande → automate se fait sans perte d'information.
- **Programmation facilitée** : le Grafcet automate devient la **traduction directe** en langage SFC dans l'API.
- **Méthode descendante**
 - On part du **plus simple** → *Qu'est-ce que la machine doit faire ?* (opératif).
 - Puis → *Avec quels composants ?* (commande).
 - Enfin → *Avec quelles adresses automate ?* (automate).
- **Vérification progressive**
 - Niveau opératif → validé avec l'utilisateur final (il comprend la logique sans technique).
 - Niveau commande → validé avec les automaticiens (choix des capteurs, moteurs).
 - Niveau automate → validé par simulation (API et code).
- **Lien avec cycle en V**
 - C'est exactement le même principe que la conception logicielle → spécification → conception → codage → test.

4.4) Synthèse

- **Trois niveaux Grafcet :**
 - **Opératif** : décrit *ce que fait le système* (sans technologie).
 - **Commande** : décrit *comment la commande est réalisée* (contacteurs, capteurs, moteurs).
 - **Automate** : décrit *comment programmer l'API* (adresses entrées/sorties).
- Exemple barrière de parking :
 - Carte magnétique (%I1.0),
 - Moteur montée (KM1), moteur descente (KM2),
 - Fin de course haut (%I1.1), bas (%I1.2),
 - Temporisation 15 s (%M1).
- **Cycle complet** : Carte → Ouverture → Attente 15s → Fermeture → Retour initial.
- **Avantage** : permet de passer **de la fonction** → **à la technologie** → **à la programmation** sans ambiguïté.

5) Partie 5 : Systèmes combinatoires vs séquentiels

5.1) Système combinatoire

Définition

Un système est **combinatoire** si la valeur de ses sorties dépend uniquement de la **valeur instantanée des entrées**.

- Aucune mémoire n'est nécessaire.
- On peut décrire complètement son fonctionnement par une **table de vérité**.

Exemple : équation logique

Un système où la sortie **Q1** s'allume si **I1** est actif OU si les deux entrées sont actives simultanément.

Table de vérité correspondante :

I2	I1	Q1
0	0	0
0	1	1
1	0	0
1	1	1

Analyse :

- Si aucune entrée n'est active → $Q1 = 0$.
- Si I1 seul est actif → $Q1 = 1$.
- Si I2 seul est actif → $Q1 = 0$.
- Si I1 et I2 sont actifs ensemble → $Q1 = 1$.

Applications industrielles

- Bouton-poussoir → lampe : si bouton appuyé, lampe ON.
- Porte logique AND/OR dans des circuits câblés.
- Commandes simples (ex. : ventilateur ON si capteur température > 30 °C).

5.2) Système séquentiel

Définition

Un système est **séquentiel** si la valeur de ses sorties dépend :

- Des **entrées actuelles**,
- **Et de l'historique des états précédents.**

Il intègre une notion de **temps et d'ordre** → une mémoire est indispensable pour conserver les états antérieurs.

Exemple : table de vérité incomplète

Considérons un système dont la sortie Q1 dépend non seulement de I1 et I2 mais aussi de ce qui s'est passé avant.

Table (simplifiée) :

I2	I1	Q1
0	0	??
0	1	1
1	0	0
1	1	0

Analyse :

- Lorsque I1=0 et I2=0 → impossible de définir Q1 directement, il dépend de l'**état précédent** (??).
- Cela prouve que la sortie ne peut pas être décrite par une simple équation logique.

5.3) Exemple typique : bascule RS

- **Entrées** : R (Reset), S (Set).
- **Sortie** : Q.
- **Fonctionnement** :
 - S=1, R=0 → Q=1 (on mémorise un état haut).
 - S=0, R=1 → Q=0 (on mémorise un état bas).
 - S=0, R=0 → Q conserve son état précédent (mémoire).
 - S=1, R=1 → état indéfini (cas interdit).

Table de fonctionnement de la bascule RS :

S	R	Q (nouveau)
0	0	Q (ancien)
0	1	0
1	0	1
1	1	Indéfini

Analyse :

- Ici, la sortie Q ne dépend pas seulement des entrées S et R, mais aussi de l'**ancien état de Q**.
- C'est exactement la logique d'un **Grafcet** : la séquence dépend toujours de l'**étape active précédente**.

5.4) Différences clés

Critère	Système combinatoire	Système séquentiel
Dépendance	Sortie = f(entrées instantanées)	Sortie = f(entrées + états précédents)
Mémoire	Aucune	Obligatoire (registre, bascule, étape active)
Représentation	Table de vérité complète possible	Table de vérité incomplète, besoin d'automate fini
Exemple simple	Porte logique AND/OR	Bascule RS, Grafcet
Exemple industriel	Allumer une lampe avec un capteur	Barrière de parking (avec temporisation et séquence)

Lien avec la théorie des automates

- Combinatoire = “réponse instantanée”.
 - Exemple : *Si bouton appuyé → lampe ON.*
- Séquentiel = “réponse qui dépend aussi du passé”.
 - Exemple : *Si carte validée et barrière fermée avant → alors ouvrir.*

Programmation API

- Les blocs combinatoires (AND, OR, comparateurs) servent à écrire les conditions logiques (réceptivités).
- Le Grafcet (séquentiel) gère la logique d'enchaînement.

Applications industrielles

- Combinatoire : *démarrer un ventilateur si $T^{\circ}\text{C} > 30^{\circ}\text{C}$.*
- Séquentiel : *remplir une cuve → attendre niveau haut → fermer la vanne.*

5.5) Conclusion

- Le **Grafcet** est **forcément séquentiel** :
 - Chaque **étape** représente un **état mémorisé**,
 - Les **transitions** dépendent des entrées + de l'étape active (mémoire).
- Un automate programmable industriel combine en réalité les deux :
 - Logique **combinatoire** pour les conditions instantanées (ET, OU...),
 - Logique **séquentielle** pour les enchaînements d'étapes.

5.6) Synthèse

- **Combinatoire**

- Sortie dépend uniquement des entrées instantanées.
- Représentable par une table de vérité complète.
- Pas de mémoire → pas d'historique.

- **Séquentiel**

- Sortie dépend des entrées + de l'état précédent.
- Nécessite une mémoire (bascule, étape active).
- Impossible à décrire par simple table de vérité.

- **Grafcet = système séquentiel**

- Chaque étape active est une mémoire de l'état en cours.
- La transition dépend à la fois de l'entrée (réceptivité) et de l'étape active.

- **Exemple concret**

- Combinatoire → lampe avec capteur.
- Séquentiel → barrière de parking avec temporisation.

6) Partie 6 : Programmation d'un système séquentiel

6.1) Difficulté de programmation empirique (Technique n°1)

- **Méthode** : coder directement en blocs (contacts, mémoires, tempos).
- **Inconvénients** :
 - On multiplie les tests à l'infini.
 - Chaque correction peut introduire un nouveau problème.
 - Impossible de vérifier toutes les situations possibles.
 - Conséquence → perte de temps, manque de fiabilité.

Exemple : un automaticien qui programme directement une machine complexe en Ladder sans Grafcet devra tester des centaines de scénarios imprévus → source d'erreurs.

6.2) Méthode structurée avec Grafcet (Technique n°2)

- **Principe** : partir du **Grafcet** comme plan directeur.
- **Étapes** :
 1. Dessiner le Grafcet à partir du cahier des charges.
 2. Traduire directement chaque étape et transition en langage API (SFC, Ladder, ST...).
 3. Tester la séquence → validation rapide.
- **Avantages** :
 - Programme **structuré et lisible**.
 - Réduction drastique du temps de mise au point.
 - Sécurité → aucune étape oubliée, transitions claires.

6.3) Exemple simple (cuisson de pâtes)

- Étape 1 : **PRÉPARER** l'eau et les pâtes.
- Étape 2 : **CUIRE** pendant un temps défini.
- Étape 3 : **ÉGOUTTER** après cuisson.

Ce cycle correspond à un Grafcet : une suite d'**étapes + transitions**.

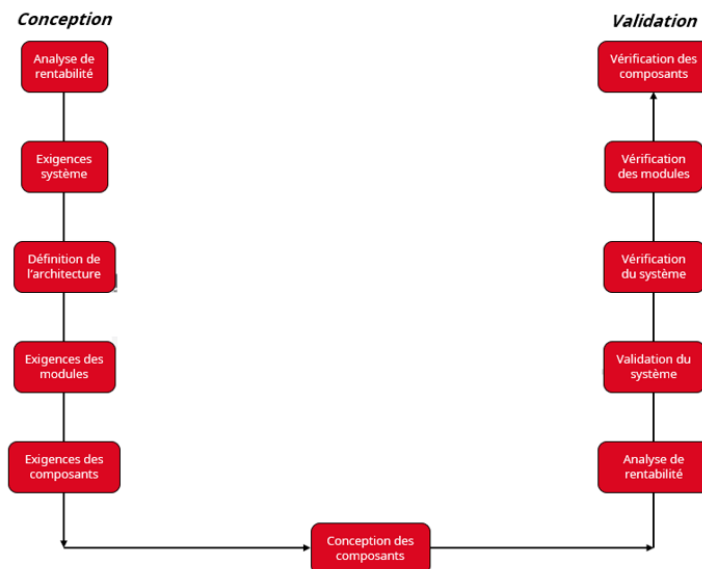
6.4) Formalisme graphique vs Graphe d'état

- **Grafcet** : étapes = carrés, actions = rectangles associés, transitions = traits avec réceptivités.
- **Graphe d'état** : étapes = ronds, actions notées dedans, transitions = flèches.
- Différence graphique seulement : le principe logique reste le même.

6.5) Bonnes pratiques

- Toujours nommer les **actions en majuscules et à l'infinitif** (ex. : *OUVRIER*, *ATTENDRE*, *FERMER*).
- Définir clairement une **étape initiale** (repos).
- Traduire chaque étape → une action, chaque transition → une condition logique.
- Toujours **valider le Grafcet sur papier** avant de coder.

6.6) Lien avec cycle en V



1. Conception

- **Exigences système → Grafcet opératif** (fonctions attendues, langage simple, lisible par le client).
- **Définition de l'architecture / modules → Grafcet commande** (intégration moteurs, capteurs, actionneurs).
- **Exigences composants / conception → Grafcet automate** (entrées/sorties API, %I/%Q, programmation SFC/ST/LD).

2. Validation

- **Vérification des composants** → test I/O (capteurs et moteurs).
- **Vérification des modules** → validation séquences locales (Grafcet par module).
- **Vérification / Validation système** → cohérence globale, conformité avec le Grafcet opératif initial.

6.7) Erreurs courantes

- Étape non désactivée → 2 actions en même temps → conflit.
- Transition impossible → Grafcet bloqué.
- Mauvaise réceptivité → boucle infinie.

6.8) Synthèse

- **Technique 1 (empirique)** : coder directement → beaucoup de tests, risques élevés, temps perdu.
- **Technique 2 (structurée)** : partir du Grafcet → traduction directe, fiabilité, rapidité.
- **Exemple** : cuisson de pâtes → séquence simple = Grafcet basique.
- **Graphe d'état** ≈ Grafcet, seule la représentation change (ronds vs carrés).
- **Bonnes pratiques** : actions en majuscules/infinif, étape initiale claire, validation papier avant codage.
- **Avantage Grafcet** : structuration, fiabilité, normalisation internationale (IEC 60848).

7) Partie 7 : Programmation avec variables d'état

7.1) Méthode 1 – Équations booléennes équivalentes

- Chaque étape et transition est traduite en équations logiques.
- Exemple simple :
 - Étape active $\leftrightarrow$ mémoire booléenne (%M).
 - Transition $\leftrightarrow$ condition booléenne.
- Implémentation possible en **Ladder** ou en **Texte structuré**.
- Limite $\rightarrow$ très vite illisible si le Grafcet est long.

7.2) Méthode 2 – Algorithme basé sur une variable d'état

- On utilise une variable **numérique** (par ex. Etat) pour mémoriser la position actuelle dans la séquence.
- Chaque transition met à jour Etat.

Exemple pseudo-code :

```
IF (Etat = 1) AND (I1 = TRUE) THEN
```

```
    Etat := 2;
```

```
END_IF;
```

```
IF (Etat = 2) AND (I2 = TRUE) THEN
```

```
    Etat := 3;
```

```
END_IF;
```

```
IF (Etat = 3) AND (I3 = TRUE) THEN
```

```
    Etat := 1; // retour à l'état initial
```

```
END_IF;
```

Avantages

- **Clarté** : un seul numéro définit l'état courant.
- **Robustesse** : impossible de rester bloqué dans une boucle CPU.
- **Efficacité** : le programme se boucle en quelques ms → le cycle CPU reste <150 ms.

7.3 Exemple d'erreur à éviter – Boucle bloquante

While (I1 = FALSE) DO

Q1 := TRUE;

End_While;

Mauvaise pratique :

- La CPU attend indéfiniment que I1 devienne TRUE.
- Résultat : le temps de cycle dépasse la limite (ex. 150 ms sur Siemens S7-1200) → CPU STOP.

7.3) Différence boucle CPU vs boucle process

- **Boucle CPU** : vitesse très élevée (ms). Ne doit jamais être bloquée.
- **Boucle process** : vitesse lente, dépend du système réel (sec, min).
- Exemple : attendre 15 secondes pour une barrière de parking → doit être géré par une **temporisation**, pas par un While.

7.4) Bonnes pratiques

- Toujours utiliser une **variable d'état** pour représenter l'étape active.
- Les transitions doivent être **non bloquantes**.
- Gérer les temporisations avec des **timers automates**, pas avec des boucles logicielles.
- Vérifier que le **cycle CPU** reste largement inférieur à la limite constructeur.

Lien avec Grafcet

- Etat = 1 → étape initiale,
- Etat = 2 → étape suivante, etc.
- Chaque valeur numérique représente une étape active.

Traduction en SFC

- Dans TIA Portal (ou autre API moderne), ce lien est automatique.

Implémentation avancée

- Ladder → SET/RESET des étapes.
- ST → CASE Etat OF ... → plus lisible.

Concept universel

- C'est exactement ce qu'on fait en **robotique** (FSM pour marche avant/arrière),
- En **jeux vidéo** (état du joueur : courir, sauter, tomber).

7.5) Synthèse

- **Deux méthodes** :
 - Équations booléennes → peu lisible si séquence longue.
 - Variable d'état → robuste et claire.
- **Erreur classique** : boucles While → bloquent la CPU (STOP automatique).
- **Bonne pratique** : transitions non bloquantes + timers pour gérer le temps.
- **Variable d'état** = correspondance directe avec les étapes du Grafcet.
- **Lien industriel** : toutes les plateformes API modernes intègrent cette logique en SFC.

8) Partie 8 : Application complète (Barrière de parking)**8.1) Spécification opérative**

- Étape 1 : Attente d'une carte magnétique.
- Étape 2 : Ouverture de la barrière (moteur montée).
- Étape 3 : Maintien de la barrière en position haute (15 s).
- Étape 4 : Fermeture de la barrière (moteur descente).
- Étape 5 : Retour position initiale (repos).

8.2) Spécification commande

- **Entrées :**
 - I0.0 = Carte validée (Accès).
 - I0.1 = Capteur position haute (FC1).
 - I0.2 = Capteur position basse (FC2).
- **Sorties :**
 - Q0.0 = Moteur montée (KM1).
 - Q0.1 = Moteur descente (KM2).
- **Mémoire :**
 - M1 = Timer 15 s.

8.3) Grafcet (version simplifiée en texte)

- 1** Étape initiale : Attente d'accès
→ Transition : carte validée
- 2** Étape 2 : Monter la barrière (KM1 ON)
→ Transition : FC1 actif
- 3** Étape 3 : Maintien (temporisation 15 s)
→ Transition : temps écoulé
- 4** Étape 4 : Descendre la barrière (KM2 ON)
→ Transition : FC2 actif
- 5** Retour étape initiale

8.4) Points clés à retenir

- Une variable Etat suffit pour gérer toute la séquence.
- Les actions sont activées uniquement quand l'état est actif.
- Les transitions sont non bloquantes → pas de While, pas de CPU bloquée.
- La temporisation est gérée par un **timer interne**, pas par une boucle.

8.5) Synthèse

- Exemple complet : **barrière de parking**.
- Étapes opératives : Attente → Ouverture → Maintien → Fermeture → Retour.
- Commande :
 - Entrées = Carte, FC1, FC2.
 - Sorties = Moteur montée/descente.
 - Mémoire = Timer 15 s.
- Variable Etat = cœur du programme séquentiel.
- Programmation ST → structure claire, transitions non bloquantes, CPU jamais bloquée.
- **Avantage** : méthode évolutive, facile à maintenir et à diagnostiquer.

9) Fonctionnalité Rockwell

9.1) Qualifier

Dans un **Sequential Function Chart (SFC)**, chaque étape peut avoir des **actions** associées (par exemple : activer un moteur, allumer une lampe, ouvrir une vanne...). Cependant, il ne suffit pas de savoir **quoi faire**, il faut aussi définir **quand** et **combien de temps** l'action doit s'exécuter.

C'est le rôle des **qualifiers d'action**.

Un qualifier détermine **la durée de vie de l'action** :

- Est-ce qu'elle s'arrête quand l'étape s'arrête ?
- Est-ce qu'elle continue même après la désactivation de l'étape ?
- Est-ce qu'elle démarre avec un délai ?
- Est-ce qu'elle doit envoyer une impulsion unique ?

Le tableau ci-dessous résume les qualifiers disponibles dans **Studio 5000**, leur signification et un exemple d'utilisation avec un moteur d'agitateur.

Qualifier	Nom complet	Fonctionnement	Exemple industriel
N	Non-Stored	Action active uniquement tant que l'étape est active.	Lampe de signalisation allumée uniquement pendant l'étape « Cycle en cours ».
S	Stored	Action reste active même après désactivation de l'étape, nécessite un Reset .	Convoyeur qui continue à tourner après la phase de chargement, jusqu'à reset.
L	Time Limited	Action démarre avec l'étape mais s'arrête automatiquement après un temps défini.	Pompe de lubrification qui s'arrête automatiquement après 15 s, même si l'étape continue.
SL	Stored and Time Limited	Action démarre avec l'étape, dure un temps défini, puis reste active tant qu'on ne fait pas de reset.	Ventilateur de refroidissement qui tourne 30 s puis reste en marche jusqu'à reset manuel.
D	Time Delayed	Action démarre après un délai si l'étape est toujours active.	Chauffage d'un four qui démarre 10 s après l'ouverture de la porte si celle-ci reste ouverte.
DS	Delayed and Stored	Action démarre après un délai et reste active même si l'étape s'arrête. Nécessite un reset.	Convoyeur tampon qui démarre 5 s après la demande et continue jusqu'à reset.
SD	Stored and Time Delayed	Action démarre après un délai et reste active même si l'étape est désactivée. Nécessite un reset.	Extracteur de poussières qui démarre 5 s après activation et continue jusqu'à reset, même si l'étape est terminée.
P	Pulse	Action exécutée une seule fois lors de l'activation de l'étape.	Électrovanne qui envoie une impulsion d'air pour éjecter une pièce.
P1	Pulse (Rising Edge)	Action exécutée une seule fois au front montant de l'étape.	Sirène qui donne un bip sonore au lancement d'un cycle.
PO	Pulse (Falling Edge)	Action exécutée une seule fois au front descendant (désactivation de l'étape).	Éclairage d'atelier qui s'éteint automatiquement avec un bip sonore en fin de cycle.
R	Reset	Sert à désactiver une action de type S, DS ou SD.	Étape « Arrêt d'urgence » envoie un reset pour couper convoyeur et moteurs stockés.

9.2) Boolean Action

Dans un **Sequential Function Chart (SFC)**, un **Boolean Action** est une action simplifiée qui ne contient **aucune logique interne**.

- Elle se contente de **mettre à ON ou OFF un bit** associé à l'étape.
- Ce bit se trouve dans une structure de tag (SFC_ACTION).
- C'est ensuite au **programme Ladder ou ST externe** de surveiller ce bit et d'exécuter la logique correspondante.
- Comme il n'y a pas de lien direct entre le SFC et la logique associée, le **reset automatique** n'affecte pas les Boolean Actions : tu dois gérer manuellement les arrêts et réinitialisations.

Exemple :

- Une étape active une Boolean Action Agitateur.
- Le bit Agitateur.ActionActive passe à 1.
- Dans Ladder, on surveille ce bit :

9.3) Synthèse : Action vs Boolean Action

Aspect	Action	Boolean Action
Contenu	Contient de la logique (ST, Ladder, appel de routine)	Ne contient aucune logique, met juste un bit ON/OFF
Exécution	Le code s'exécute directement dans l'action	Le code est exécuté ailleurs en surveillant le bit
Reset automatique	Oui, géré par les qualifieurs (N, S, D...)	Non, doit être géré manuellement
Complexité	Plus lourd, mais puissant et autonome	Plus léger, utile pour signaux simples
Usage typique	Contrôler un moteur, ouvrir une vanne, exécuter une séquence	Activer un flag binaire pour informer une autre partie du programme

9.4) Etape Stop



1. Qu'est-ce qu'une étape « Stop » dans un GRAFCET

- C'est une étape terminale : elle indique que le cycle est fini et que le GRAFCET n'exécutera plus aucune action tant qu'il n'est pas réinitialisé.
- C'est l'opposé de l'étape initiale (qui démarre automatiquement au lancement du grafcet).
- Graphiquement, elle est représentée comme une étape classique mais souvent avec un nom explicite (Stop_000) et parfois un marquage particulier dans les normes.

2. Pourquoi l'utiliser

Il y a plusieurs raisons pour lesquelles un automaticien choisit d'ajouter une étape "Stop" :

a) Fin d'un cycle unique

Exemple : une machine effectue un cycle complet (prise de pièce → usinage → éjection).

- Sans étape stop → le grafcet repartira au début et fera un nouveau cycle si les conditions restent vraies.
- Avec étape stop → il attend un nouvel ordre avant de recommencer.

b) Mode maintenance ou mise en sécurité

Dans certains procédés, on veut que tout s'arrête clairement à la fin d'un process, avec désactivation des sorties.

- L'étape stop sert de "zone neutre" : on coupe les actionneurs, on arrête les moteurs, on met les vérins en position repos.

c) Éviter les boucles involontaires

Si le grafcet est conçu pour des cycles déclenchés à la demande, sans étape stop, une condition d'amorçage qui reste vraie pourrait relancer le cycle immédiatement.

L'étape stop impose un reset manuel ou programmé → pas de relance accidentelle.

d) Point de reprise contrôlé

Dans certaines applications complexes (fabrication batch, test en laboratoire), on peut placer plusieurs étapes stop pour indiquer différents points de reprise selon les besoins du process.

Comment on le gère dans le programme automate

En pratique, quand on arrive sur une étape stop :

- Le grafcet ne passe plus à aucune transition suivante (il n'y en a souvent pas, ou bien elles sont désactivées).
- Pour relancer :
 1. **Reset logiciel** → réactivation de l'étape initiale par un bit dans le programme.
 2. **Reset matériel** → bouton physique ou commande HMI pour réinitialiser la séquence.
- En norme IEC 60848 (GRAFCET), l'étape stop n'est pas obligatoire mais elle est parfaitement légale.

3. Exemple concret dans l'automatisme industriel

Imagine un poste d'assemblage :

1. Démarrage → prise de la pièce.
 2. Assemblage → vissage automatisé.
 3. Contrôle qualité.
 4. Stop → fin du cycle.
- Si on veut relancer, l'opérateur doit appuyer sur "Cycle Start".
 - On évite ainsi que la machine enchaîne les cycles toute seule sans intervention humaine.

10) Conclusion générale – Grafcet / SFC

Le Grafcet (SFC) est bien plus qu'un simple langage graphique : c'est une méthodologie universelle pour concevoir, analyser et programmer des systèmes automatisés séquentiels.

Points essentiels à retenir

- Définition : outil graphique normalisé (IEC 60848) décrivant le fonctionnement séquentiel d'un système.
- Éléments constitutifs :
 - Étapes (avec actions associées),
 - Transitions (avec réceptivités),
 - Liaisons (enchaînement logique).
- Règles d'évolution :
 - Initialisation → étape de départ active,
 - Franchissement → condition vraie + étape active,
 - Évolution → activation suivante + désactivation précédente.
- Structures avancées :
 - Linéaire (séquence simple),
 - OU (sélection d'une branche),
 - Reprise (répétition d'actions),
 - ET (simultanéité de séquences).
- Niveaux de description :
 - Opératif (fonctionnel),
 - Commande (technologique),
 - Automate (programmation API).
- Combinatoire vs Séquentiel :
 - Combinatoire → sorties = $f(\text{entrées instantanées})$,
 - Séquentiel → sorties = $f(\text{entrées} + \text{état précédent})$.
- Programmation :
 - Éviter l'approche empirique (tests infinis),
 - Utiliser la méthode structurée → Grafcet → traduction directe,
 - Implémenter par variable d'état ou directement en SFC.

10.1) Atouts du Grafcet pour l'industrie

- ♦ **Lisibilité universelle** : compréhensible par tous (concepteurs, automaticiens, techniciens).
- ♦ **Fiabilité** : structure claire → réduit les risques d'oubli ou de bug.
- ♦ **Souplesse** : adaptable à toutes les technologies (câblé ou programmé).
- ♦ **Puissance** : permet de modéliser des systèmes simples ou complexes (branches, simultanéité, reprises).
- ♦ **Interopérabilité** : directement intégré aux automates modernes (SFC – IEC 61131-3).

Le Grafcet est la colonne vertébrale de la logique séquentielle industrielle :

- Il transforme un cahier des charges en un modèle fonctionnel.
- Il permet une validation rapide avec les parties prenantes.
- Il sert de plan de programmation fiable pour l'automate.
- Il offre un outil de diagnostic en maintenance (lecture directe de l'étape active).

10.2) Synthèse finale

- Grafcet = langage universel du séquentiel industriel.
- Normé, lisible, structurant → du cahier des charges à la programmation API.
- Évite les approches empiriques → garantit la fiabilité.
- S'adapte à tous les niveaux : opératif, commande, automate.
- Permet de modéliser aussi bien un convoyeur simple qu'une ligne de production complexe.
- Outil indispensable pour tout ingénieur en automatisme et tout technicien de maintenance.