Mise en œuvre des modèles en Ladder Diagram

1 – PRINCIPE GÉNÉRAL
2 – STOCKAGE DE L'ÉTAT
3 – ÉVOLUTION
4 – ACTIONS
5 – STRUCTURE GLOBALE DU PROGRAMME

1 – Principe général

2 – STOCKAGE DE L'ÉTAT

3 – ÉVOLUTION

4 – ACTIONS

5 – STRUCTURE GLOBALE DU PROGRAMME

Principe général

- Le langage Ladder Diagram étant conçu sur un principe tout ou rien, il est naturel d'utiliser une approche similaire à l'électronique numérique
- On se concentrera principalement sur des variables booléennes
- L'état sera stocké dans des vecteurs de bits
  - Chaque bit du vecteur sera piloté par des équations logiques
- Les actions seront pilotées par des équations logiques
  - Actions combinatoires pour les sorties non mémorisées
  - Actions mémorisées (séquentielles) pour les sorties mémorisées

1 – PRINCIPE GÉNÉRAL

2 – Stockage de l'état

3 – ÉVOLUTION

4 – ACTIONS

5 – STRUCTURE GLOBALE DU PROGRAMME

Codage de l'état

- L'état doit être enregistré dans des vecteurs de bits
- Dans une MAE
  - Un seul état actif à la fois
  - Plusieurs possibilités pour représenter celui-ci
    - Voir slide suivant
- Dans un Grafcet
  - Plusieurs étapes peuvent être actives simultanément
  - On utilise un bit pour représenter l'état de chaque étape

Codage de l'état : MAE

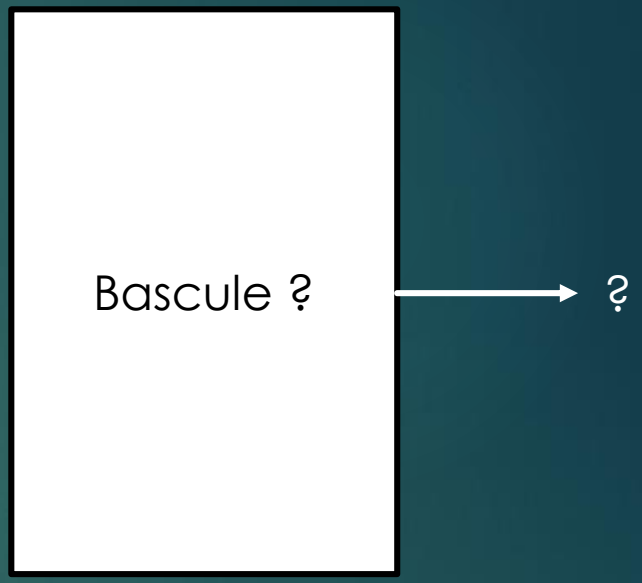
- L'état peut être codé en binaire naturel
  - État 0 = 0000
  - État 1 = 0001
  - État 2 = 0010
  - État 3 = 0011
  - Etc.
- Avantage : moins de bits
  - Nombre de bits nécessaires = $\log_2(N)$ (arrondi à l'entier supérieur), avec N nombre d'états
- Désavantage : plus de portes logiques (équation combinatoire plus complexe)
  - Nécessaire pour coder/décoder l'état courant

- Ou codé avec une bascule par état (« one-hot »)
  - État 0 = 0001
  - État 1 = 0010
  - État 2 = 0100
  - État 3 = 1000
  - Etc.
- Désavantage : plus de bits
  - Nombre de bits nécessaires = N, avec N nombre d'états
- Avantage : moins de portes logiques (équation combinatoire plus simple)
  - L'état est directement disponible sur un bit du vecteur d'état

On utilisera le codage one-hot, plus simple à mettre en œuvre

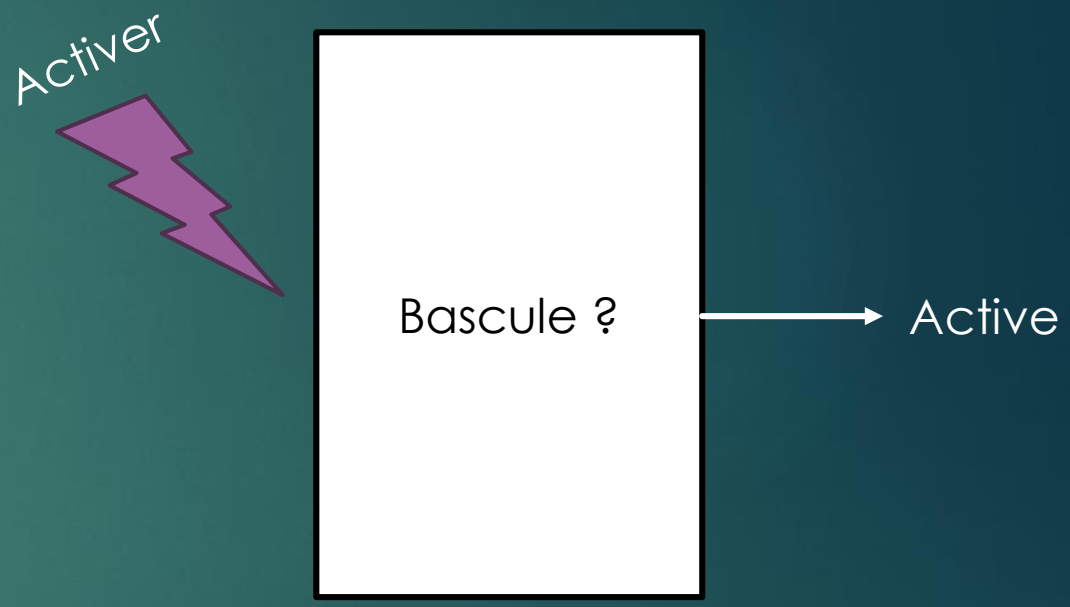
Codage de l'état : Grafcet

- Afin de simplifier la gestion de chaque bit, on utilisera des bascules pour gérer l'activation/désactivation d'une étape
- Question : quel type de bascule est adaptée au stockage de l'état de l'étape ?
  - D, RS, JK, autre ?



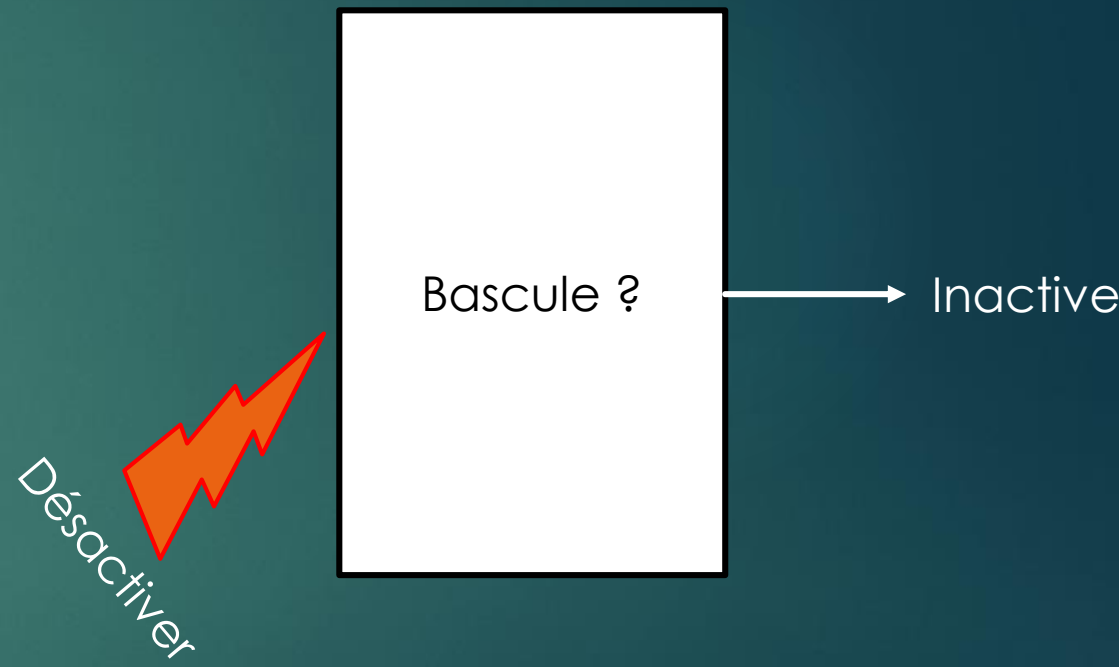
Codage de l'état : Grafcet

- Afin de simplifier la gestion de chaque bit, on utilisera des bascules pour gérer l'activation/désactivation d'une étape
- Question : quel type de bascule est adaptée au stockage de l'état de l'étape ?
  - D, RS, JK, autre ?



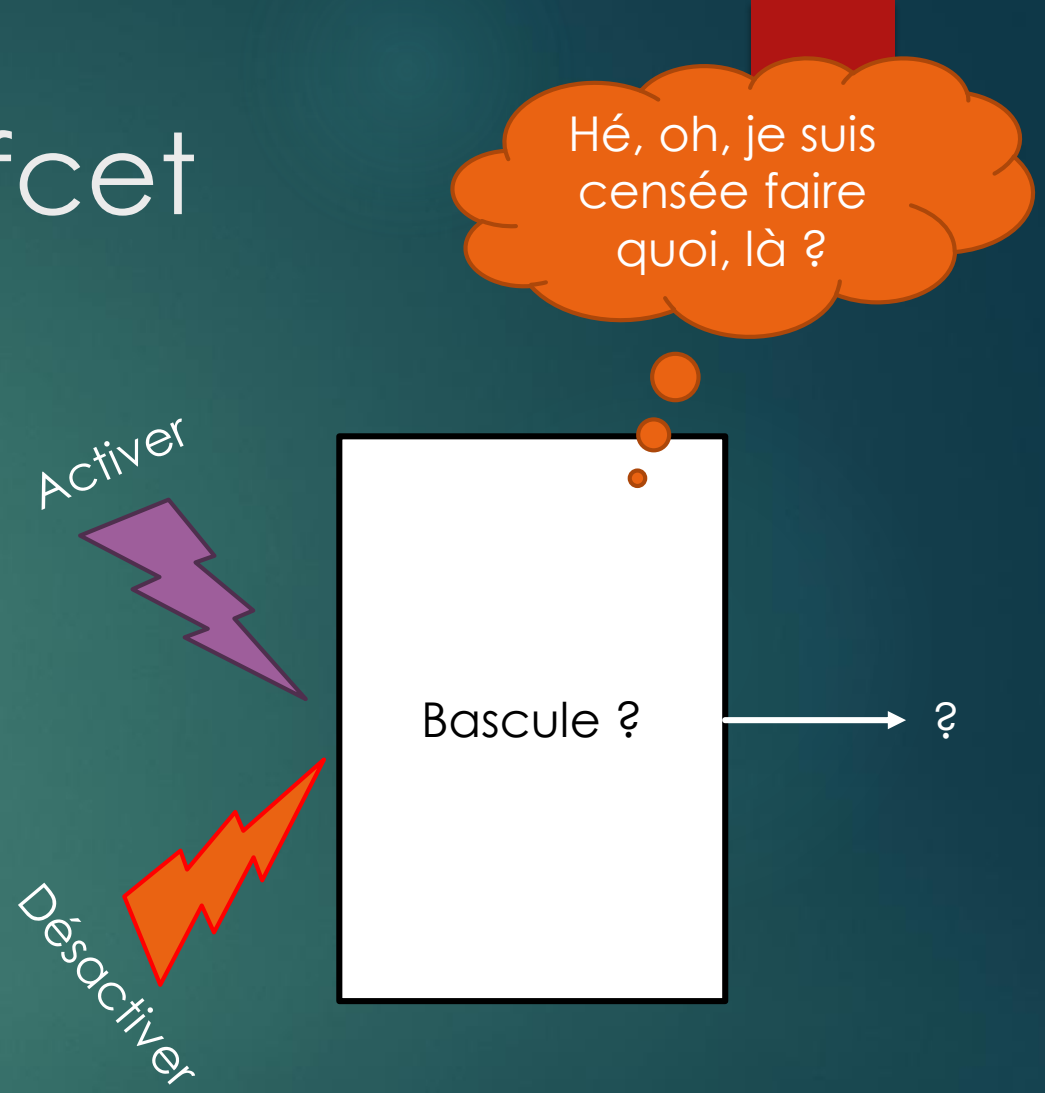
Codage de l'état : Grafcet

- Afin de simplifier la gestion de chaque bit, on utilisera des bascules pour gérer l'activation/désactivation d'une étape
- Question : quel type de bascule est adaptée au stockage de l'état de l'étape ?
  - D, RS, JK, autre ?



Codage de l'état : Grafcet

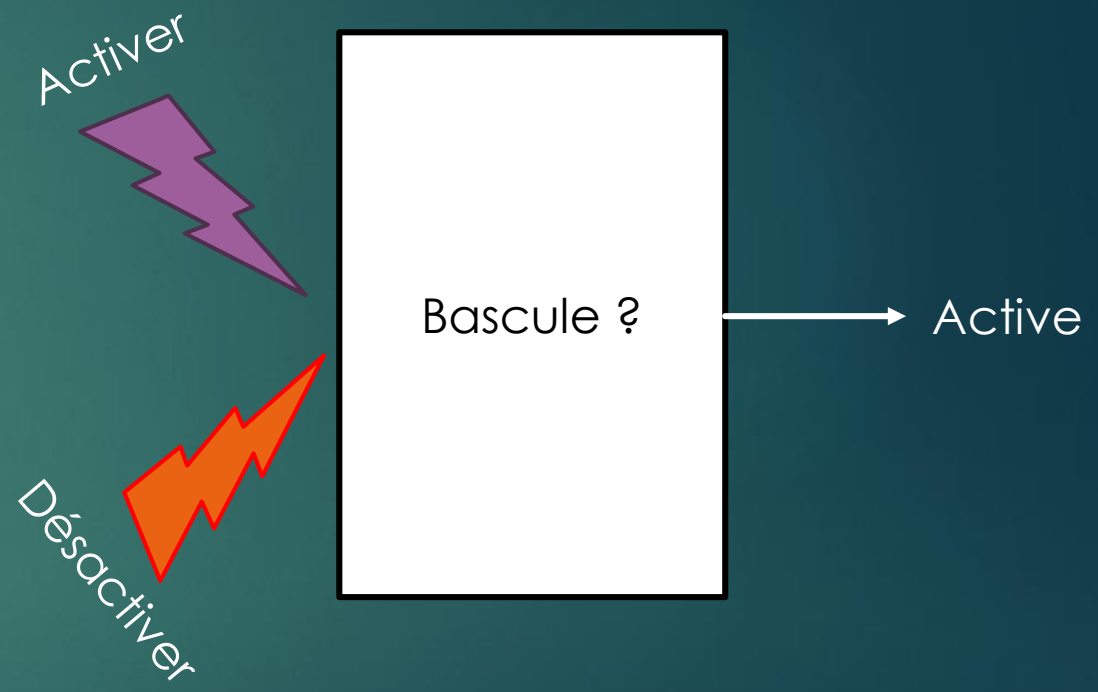
- Afin de simplifier la gestion de chaque bit, on utilisera des bascules pour gérer l'activation/désactivation d'une étape
- Question : quel type de bascule est adaptée au stockage de l'état de l'étape ?
  - D, RS, JK, autre ?



Codage de l'état : Grafcet

- Afin de simplifier la gestion de chaque bit, on utilisera des bascules pour gérer l'activation/désactivation d'une étape
- Question : quel type de bascule est adaptée au stockage de l'état de l'étape ?
  - D, RS, JK, autre ?

Si, au cours du même cycle, on souhaite à la fois activer et désactiver une étape, c'est l'activation qui « gagne »

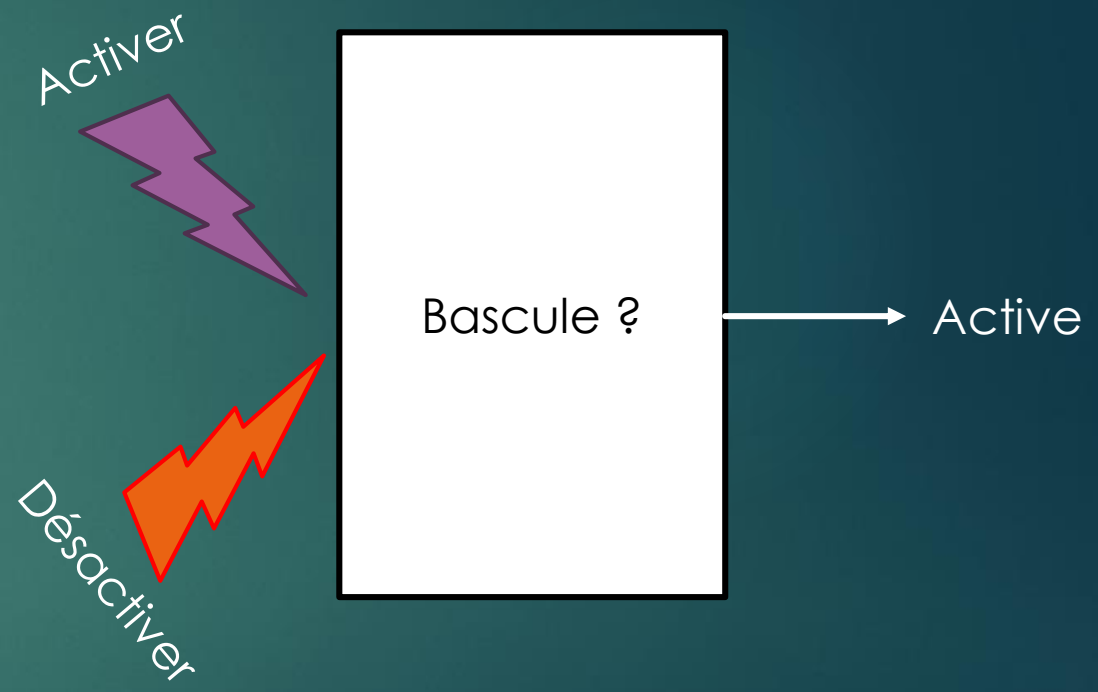


Codage de l'état : Grafcet

- On veut donc :

Activer	Désactiver	État
0	0	Valeur conservée
0	1	Désactiver
1	0	Activer
1	1	Activer

- Quelle bascule est adaptée ?



Codage de l'état : Grafcet

- On veut donc :

Activer	Désactiver	État
0	0	Valeur conservée
0	1	Désactiver
1	0	Activer
1	1	Activer

- Quelle bascule est adaptée ?

- Rappels

Bascule D

D	Q_n
0	0
1	1

Bascule RS

S	R	Q_n
0	0	Q_{n-1}
0	1	0
1	0	1
1	1	X

Bascule JK

J	K	Q_n
0	0	Q_{n-1}
0	1	0
1	0	1
1	1	$/Q_{n-1}$

Codage de l'état : Grafcet

- On veut donc :

Activer	Désactiver	État
0	0	Valeur conservée
0	1	Désactiver
1	0	Activer
1	1	Activer

- Quelle bascule est adaptée ?
  - On utilise une bascule RS dans laquelle on s'assure qu'en cas de conflit, c'est l'activation qui « gagne »

Bascule RS avec set prioritaire

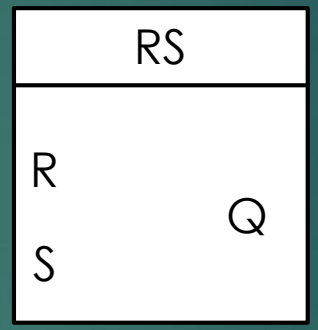
S	R	Q_n
0	0	Q_{n-1}
0	1	0
1	0	1
1	1	1

Bascule RS

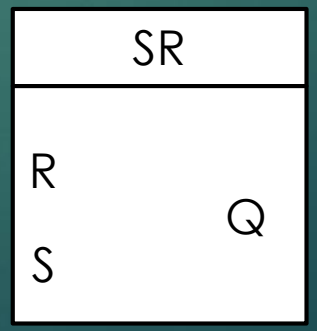
S	R	Q_n
0	0	Q_{n-1}
0	1	0
1	0	1
1	1	X

Stockage de l'état

- Sur les automates, il existe deux types de bascules RS
  - RS : set prioritaire
  - SR : reset prioritaire
- On utilise des bascule RS pour le stockage de l'état d'un Grafcet
- Pour les MAE, le type importe peu car on n'aura jamais d'activation et de désactivation au cours du même cycle
  - Sauf en cas de reset ! Mais on traitera ce cas à part



R	S	Q_n
0	0	Q_{n-1}
0	1	1
1	0	0
1	1	1



R	S	Q_n
0	0	Q_{n-1}
0	1	1
1	0	0
1	1	0

1 – PRINCIPE GÉNÉRAL

2 – STOCKAGE DE L'ÉTAT

3 – Évolution

4 – ACTIONS

5 – STRUCTURE GLOBALE DU PROGRAMME

Théorie vs. application

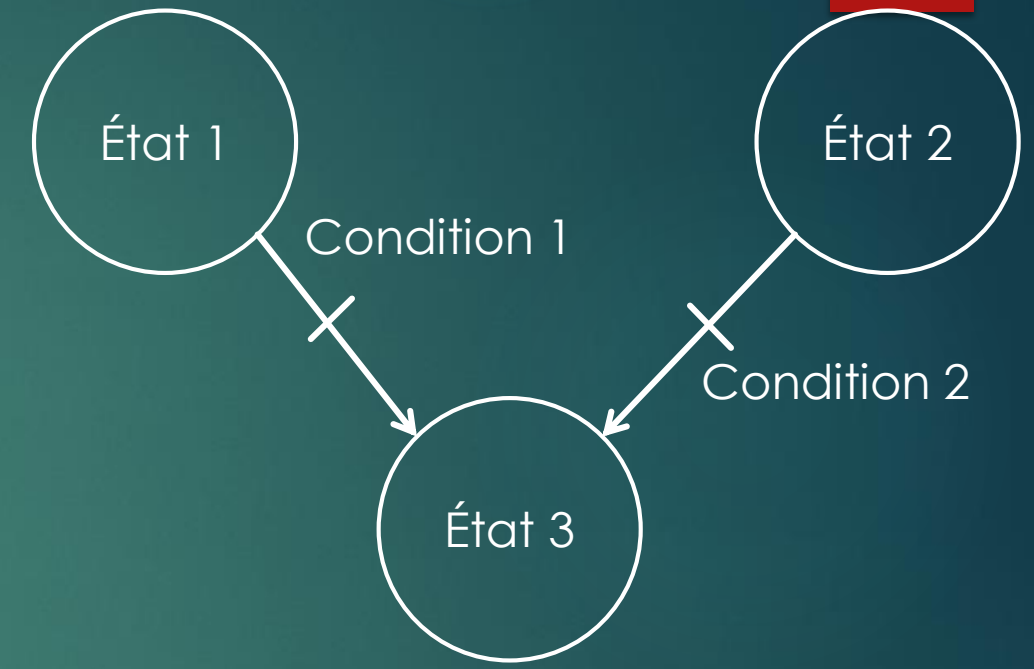
- Dans les slides qui suivent, nous allons présenter la théorie à l'aide de bobines SET et RESET
- On représentera ainsi l'équation nécessaire pour activer ou désactiver un état/une étape
  - Cela simplifiera la lisibilité des slides
- Mais en pratique, on découpera ce calcul en utilisant des variables intermédiaires avant d'appliquer les équations sur les bascules
  - Les programmes présentés sur les slides qui suivent ne sont donc pas à appliquer tel quel !

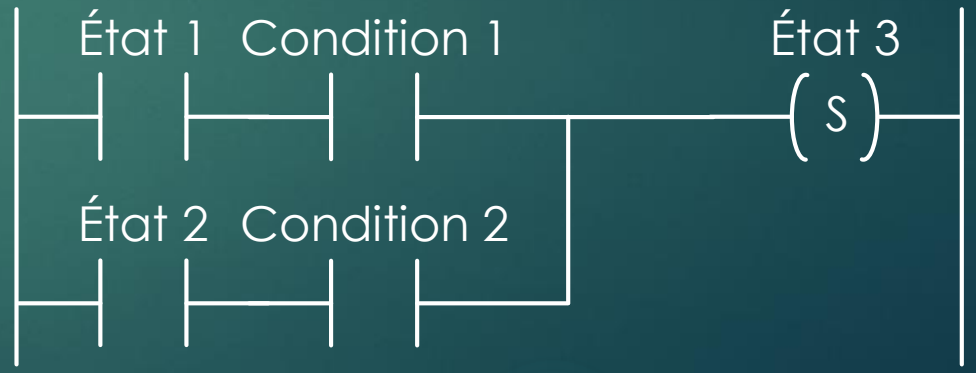
Activation d'un état

- Pour déterminer l'équation nécessaire pour activer un état, il faut regarder quelles sont les transitions entrantes

Activation d'un état

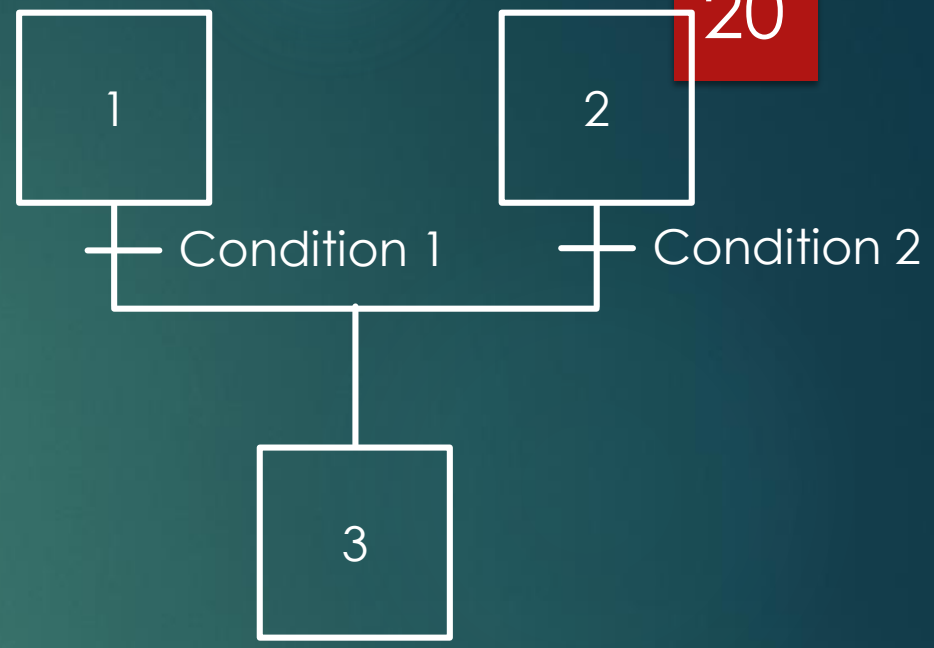
- Pour déterminer l'équation nécessaire pour activer un état, il faut regarder quelles sont les transitions entrantes
  - Dans une MAE
    - Plusieurs transitions entrantes : OU logique

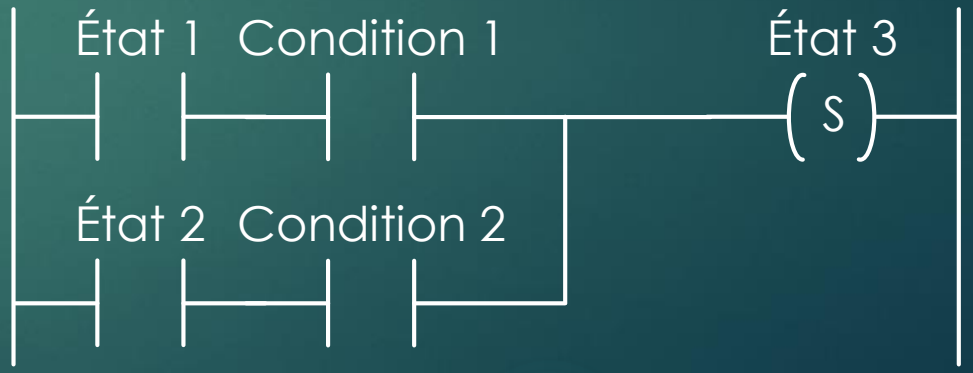




Activation d'un état

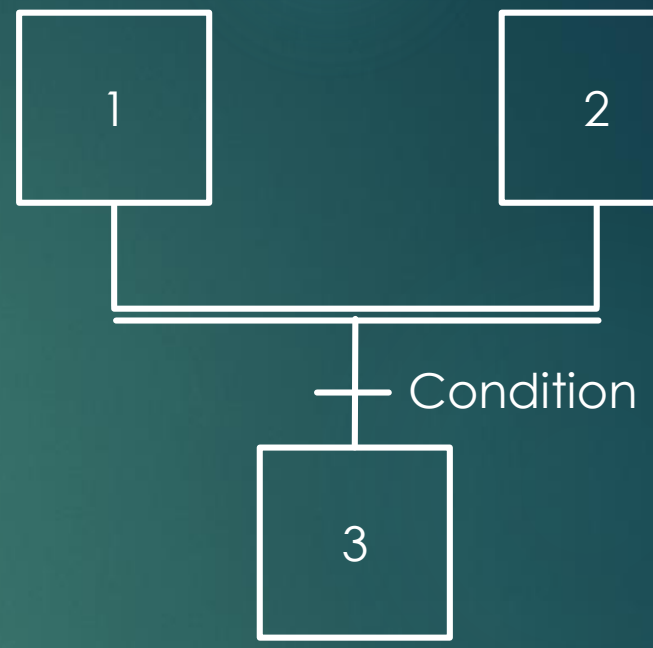
- Pour déterminer l'équation nécessaire pour activer un état, il faut regarder quelles sont les transitions entrantes
  - Dans une MAE
    - Plusieurs transitions entrantes : OU logique
  - Dans un Grafcet
    - Après une convergence en OU : OU logique

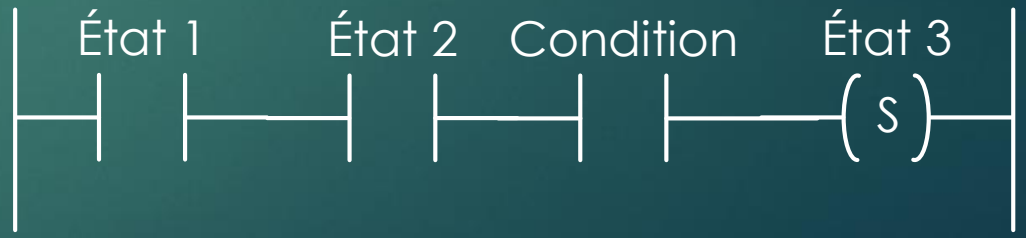




Activation d'un état

- Pour déterminer l'équation nécessaire pour activer un état, il faut regarder quelles sont les transitions entrantes
  - Dans une MAE
    - Plusieurs transitions entrantes : OU logique
  - Dans un Grafcet
    - Après une convergence en OU : OU logique
    - Après une convergence en ET : ET logique



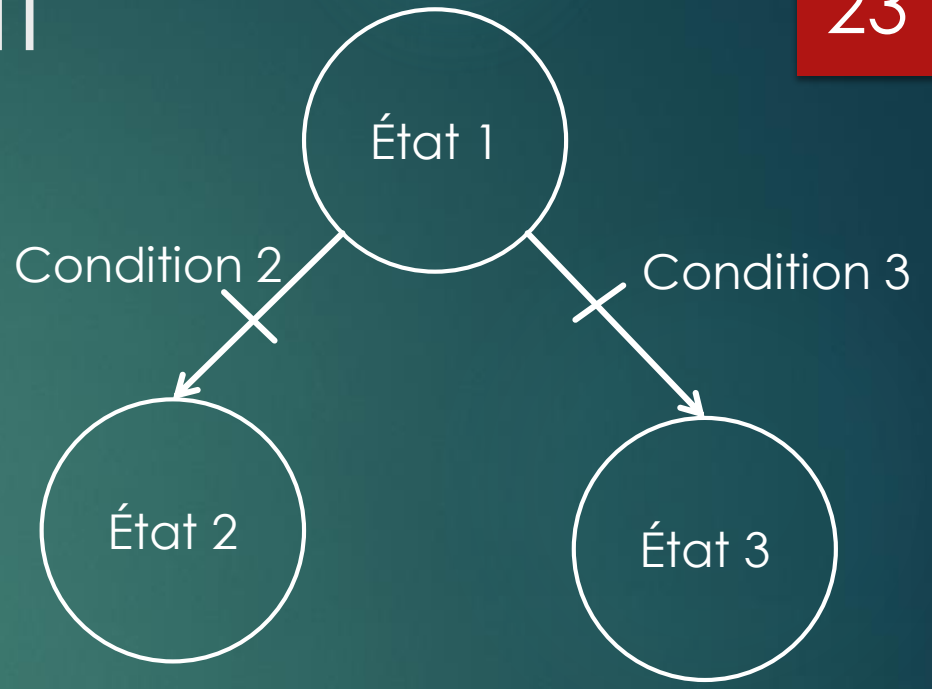


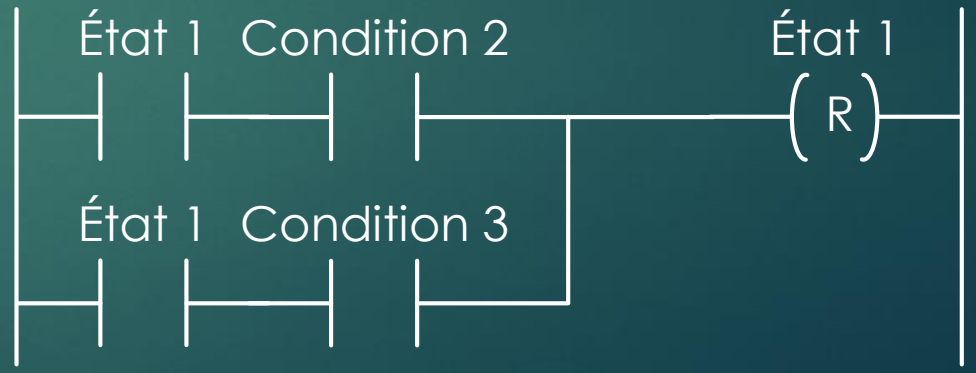
Désactivation d'un état

- Pour déterminer l'équation nécessaire pour désactiver un état, il faut regarder quelles sont les transitions sortantes

Désactivation d'un état

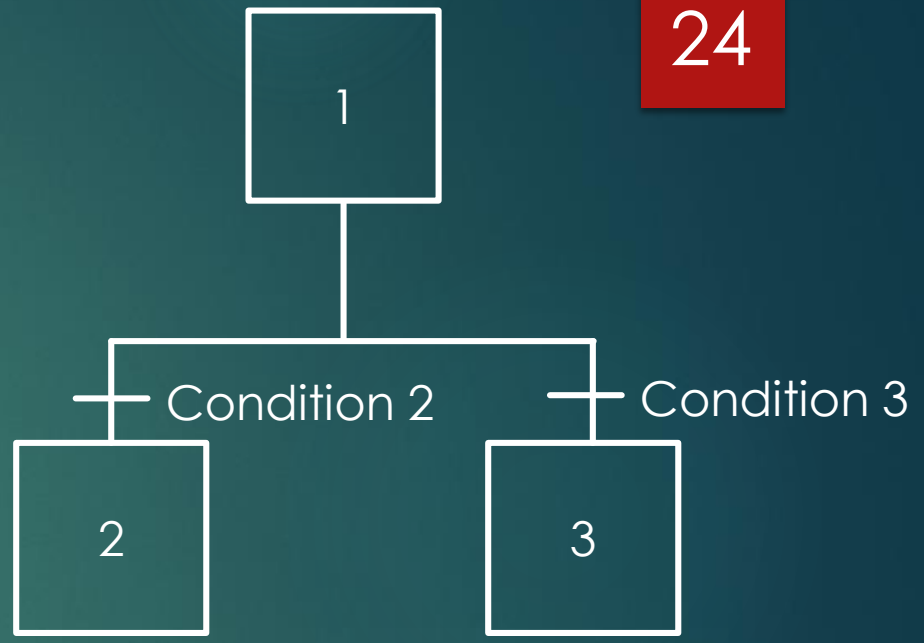
- Pour déterminer l'équation nécessaire pour désactiver un état, il faut regarder quelles sont les transitions sortantes
  - Dans une MAE
    - Plusieurs transitions sortantes : OU logique

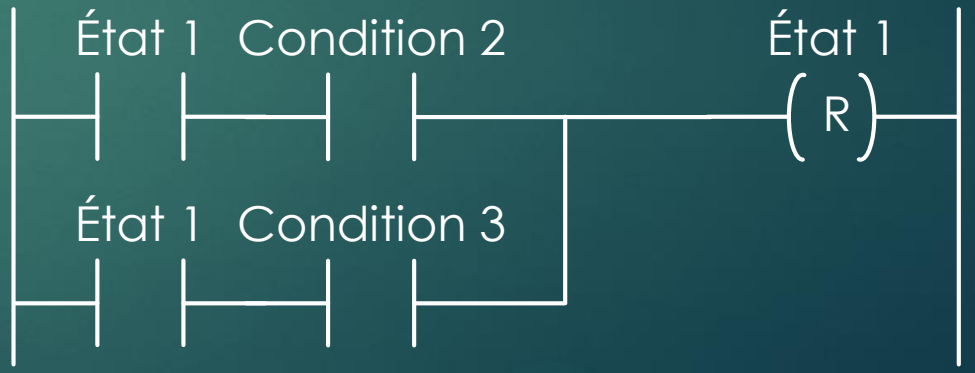




Désactivation d'un état

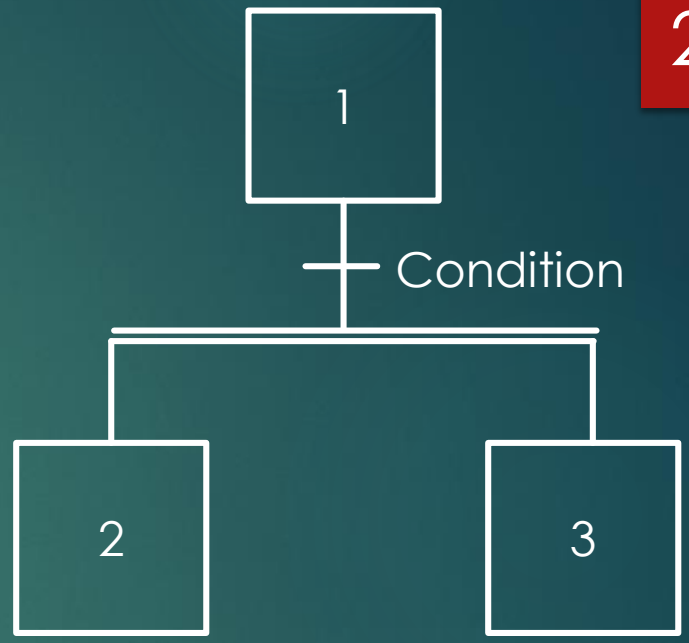
- Pour déterminer l'équation nécessaire pour désactiver un état, il faut regarder quelles sont les transitions sortantes
  - Dans une MAE
    - Plusieurs transitions sortantes : OU logique
  - Dans un Grafcet
    - Avant une divergence en OU : OU logique





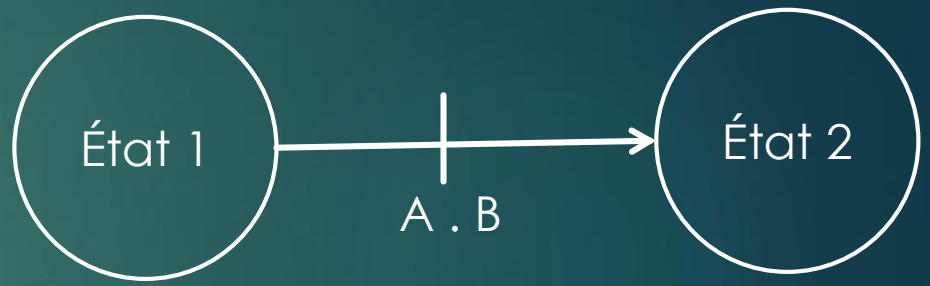
Désactivation d'un état

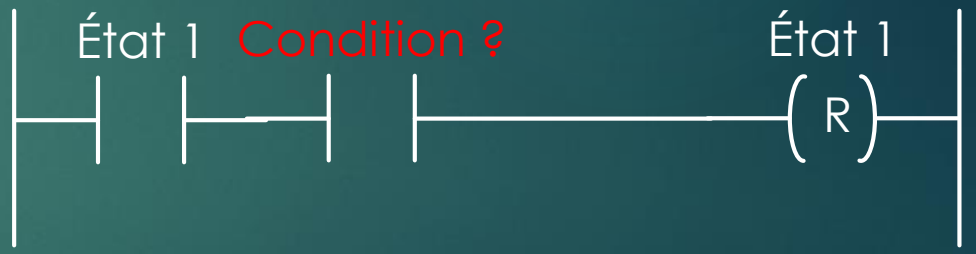
- Pour déterminer l'équation nécessaire pour désactiver un état, il faut regarder quelles sont les transitions sortantes
  - Dans une MAE
    - Plusieurs transitions sortantes : OU logique
  - Dans un Grafcet
    - Avant une divergence en OU : OU logique
    - Avant une divergence en ET : une seule condition



Et si la condition est une équation logique ?

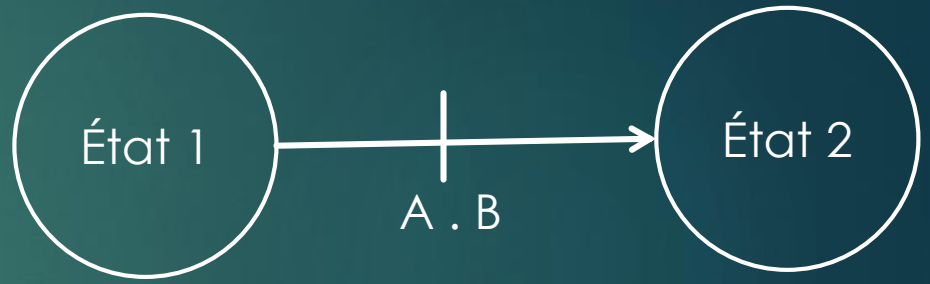
- Et là, je fais comment ?





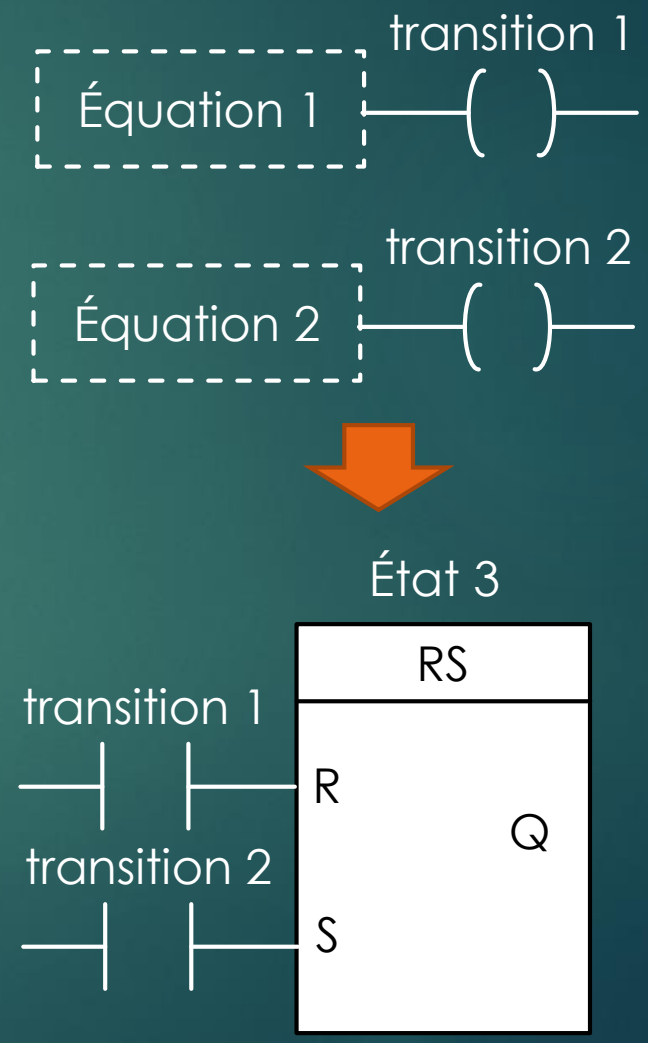
Et si la condition est une équation logique ?

- Et là, je fais comment ?
- On décompose simplement la condition

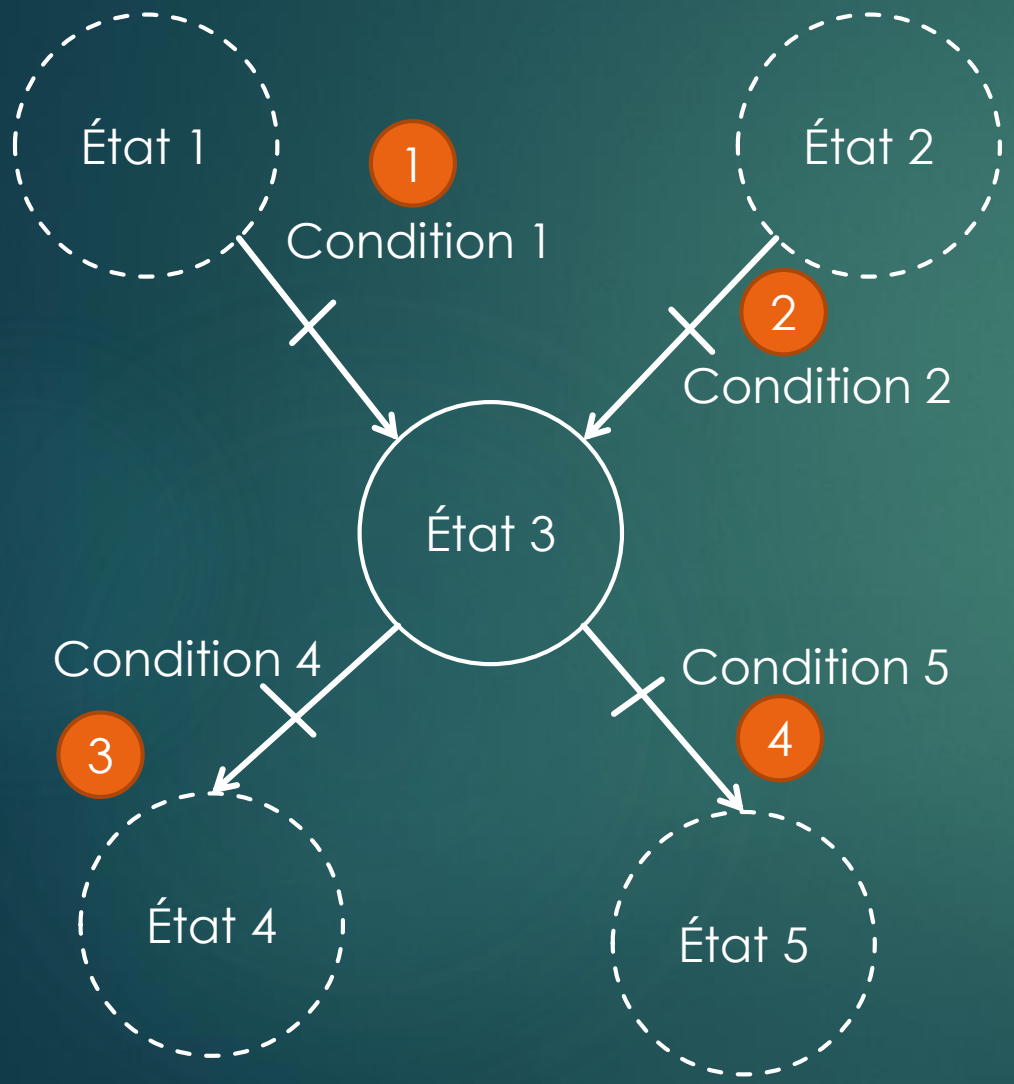


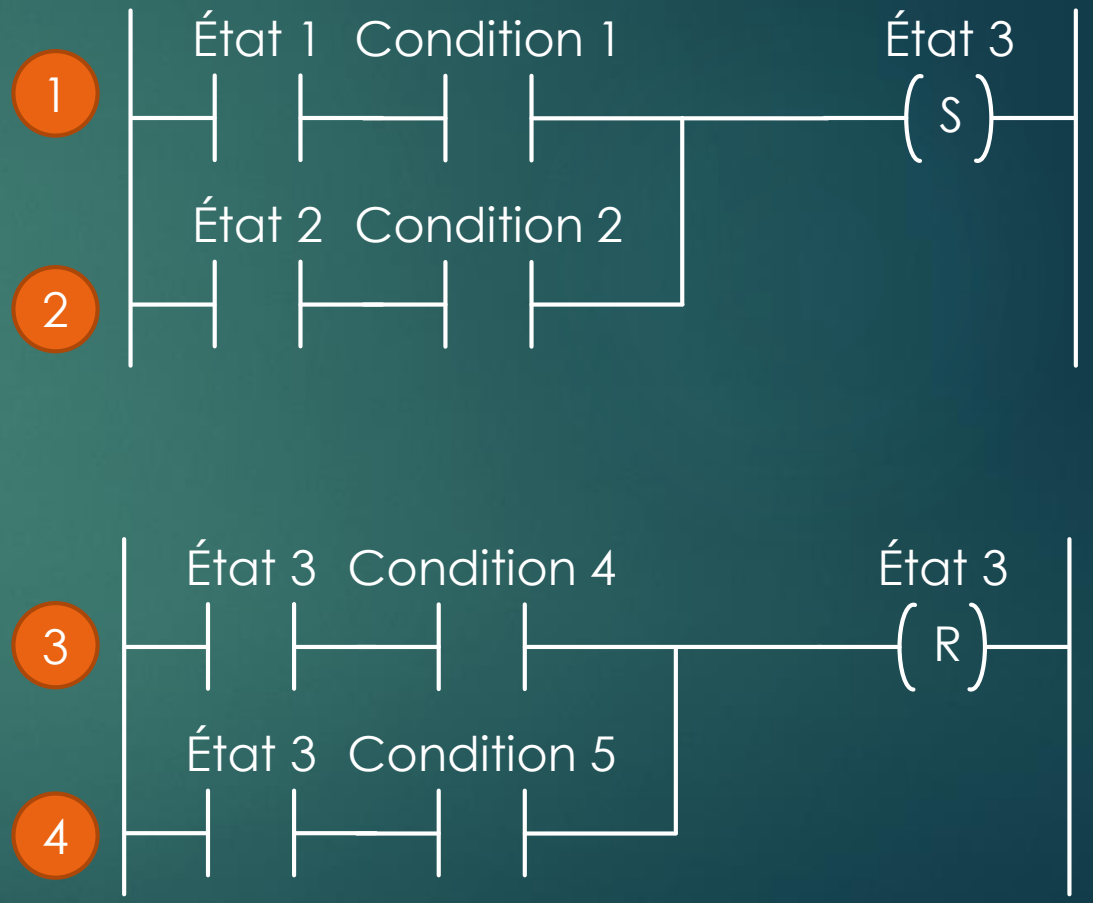
Séparation du calcul et de l'évolution de l'état

- Pour fonctionner correctement, le calcul du changement d'état (calcul de la transition) et son application (sur la bascule RS) doivent être séparés
- En effet, pour calculer l'évolution de l'état, l'état ne doit pas encore avoir changé
- De plus, cela permet de mutualiser les calculs, car une transition est toujours utilisée à deux endroits : pour activer un état et en désactiver un autre

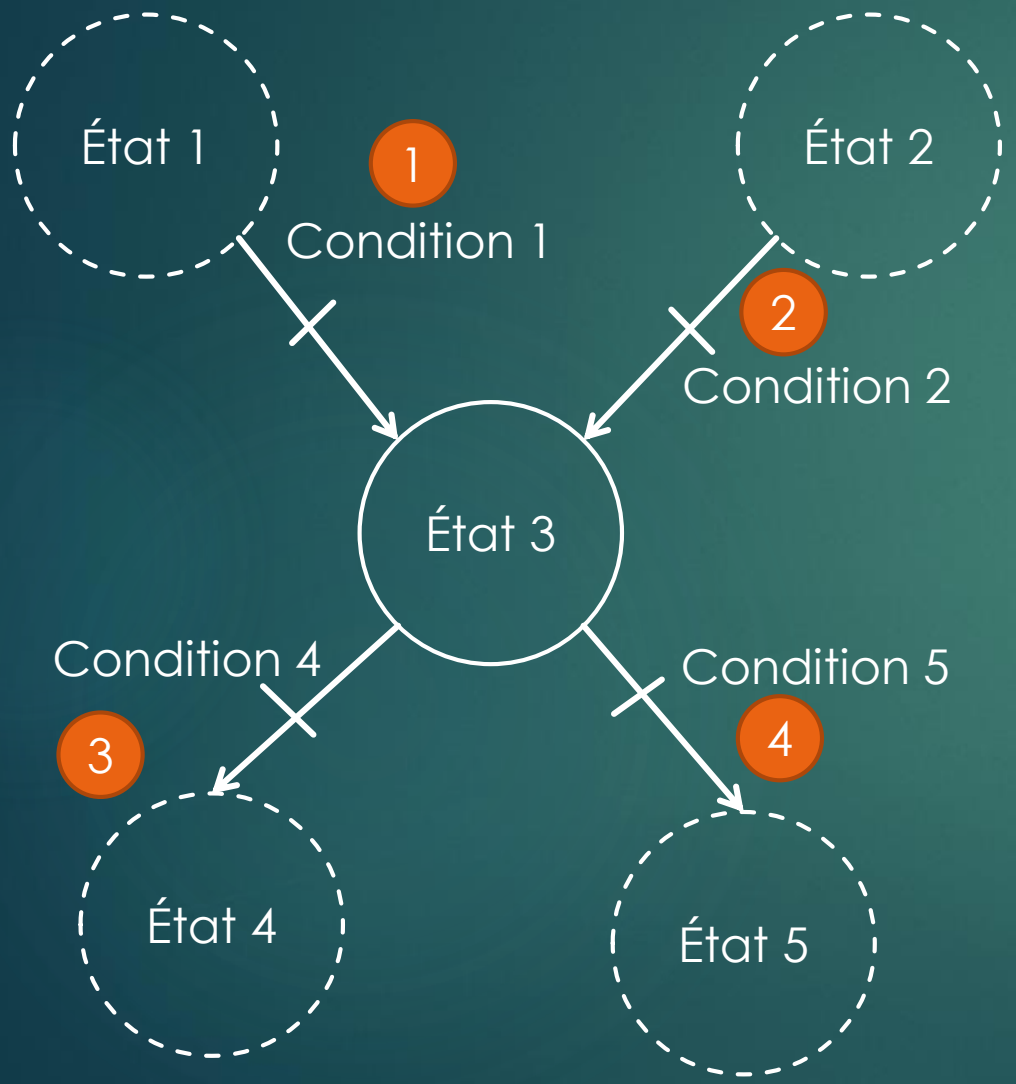


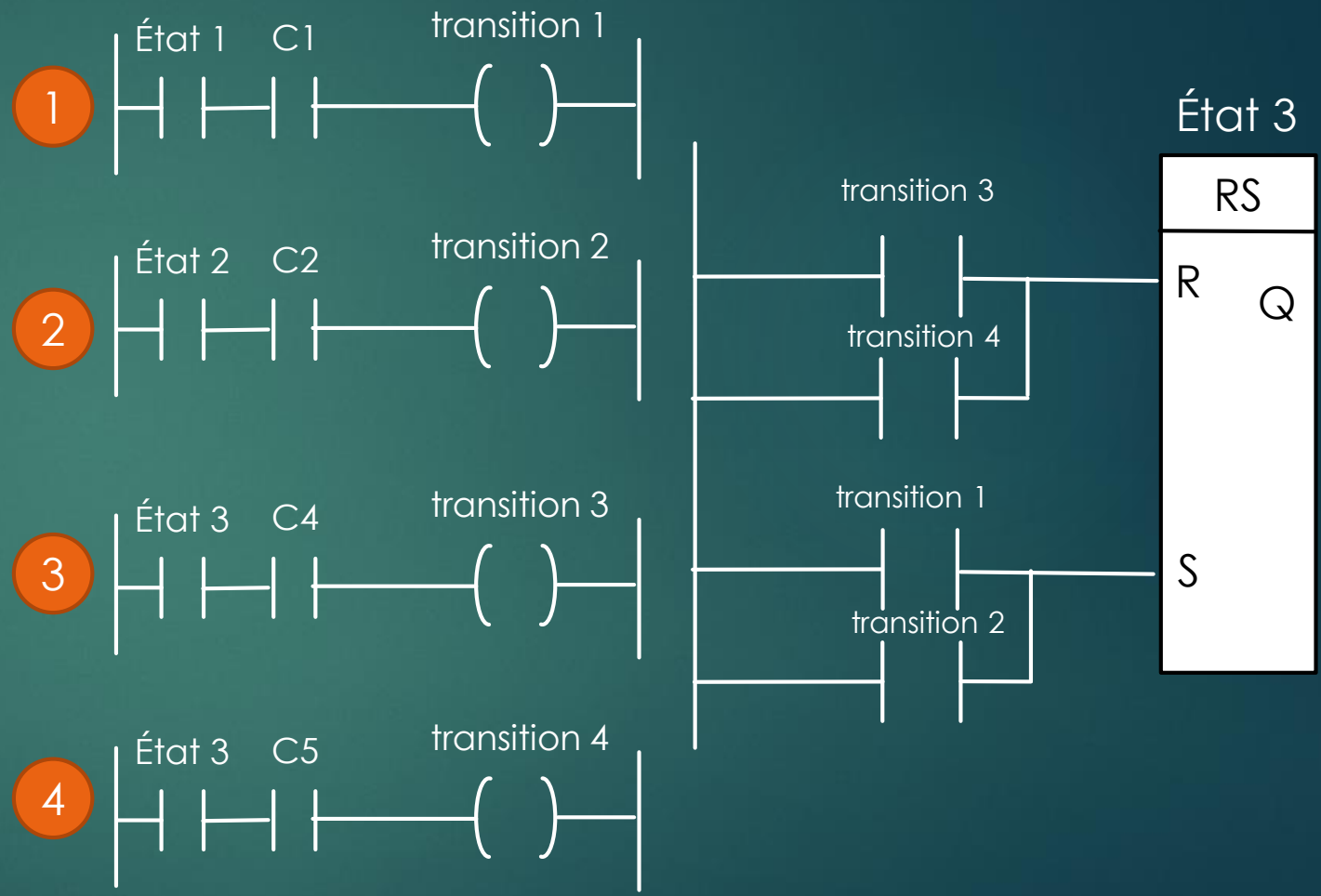
De la théorie à la pratique…





De la théorie à la pratique…





Et la réinitialisation dans tout ça ?

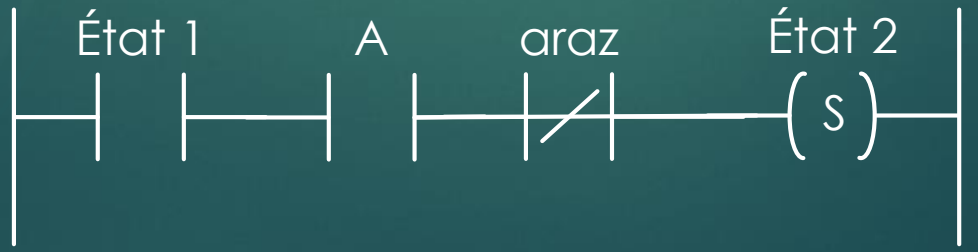
- La réinitialisation de l'état/étape doit être prise en compte
  - En effet, un bloc fonctionnel (bloc avec mémoire) doit obligatoirement comporter une entrée permettant sa remise à zéro (exemple : signal *araz*/*arazb*)
- Ceci doit être réalisé différemment selon si l'état/étape est initial(e) ou pas

Et la réinitialisation dans tout ça ?

- État/étape initial(e)
  - La remise à zéro doit *activer* l'état
  - On place donc le signal de remise à zéro en parallèle sur le Set de l'état
- État/étape non initial(e)
  - La remise à zéro doit *désactiver* l'état
  - On place donc le signal de remise à zéro en parallèle sur le Reset de l'état

Et la réinitialisation dans tout ça ?

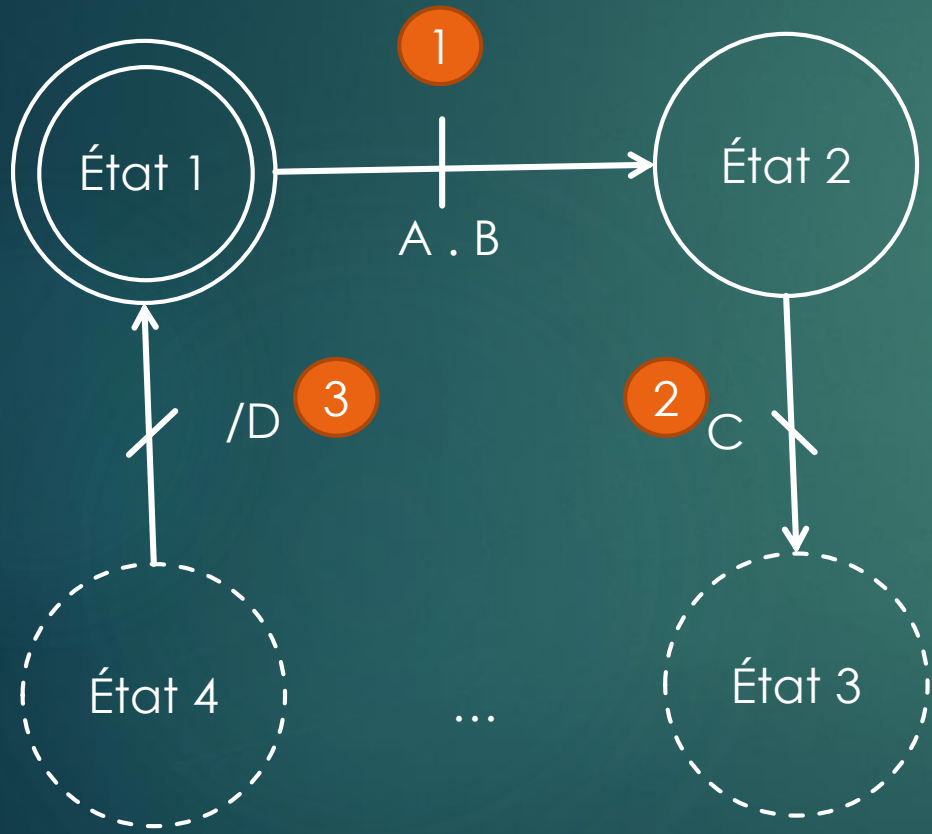
- Oui, mais que se passe-t-il si, au cours d'un même cycle, j'active un état non initial et je fais une remise à zéro
  - Le Set étant prioritaire (bascule RS), alors l'état *restera actif*
  - C'est un problème !
- Pour éviter cela, le signal de remise à zéro doit inhiber le Set des états non initiaux
  - Comment faire ça ?
- Il suffit de rajouter le signal *araz* inversé en série

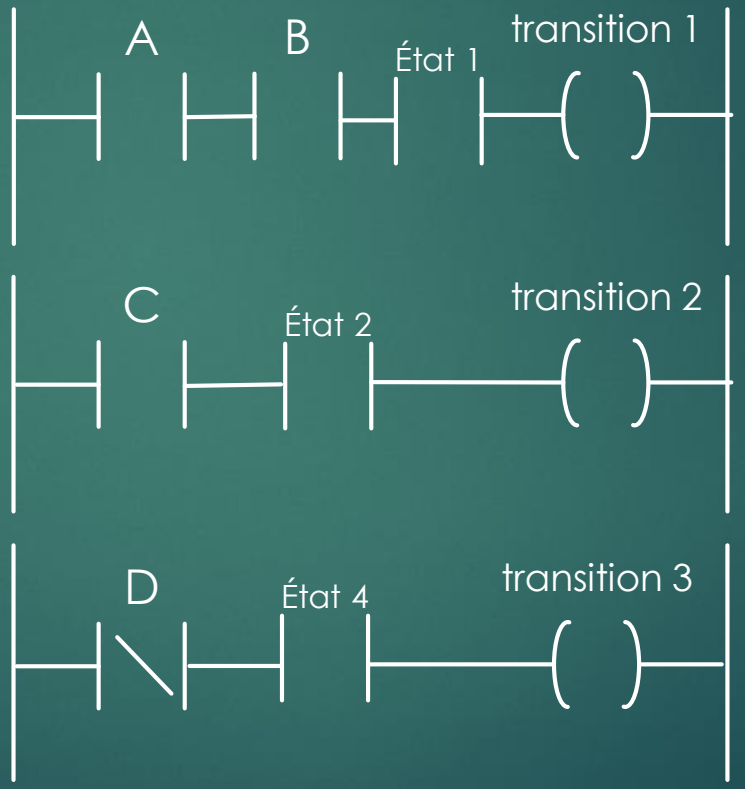


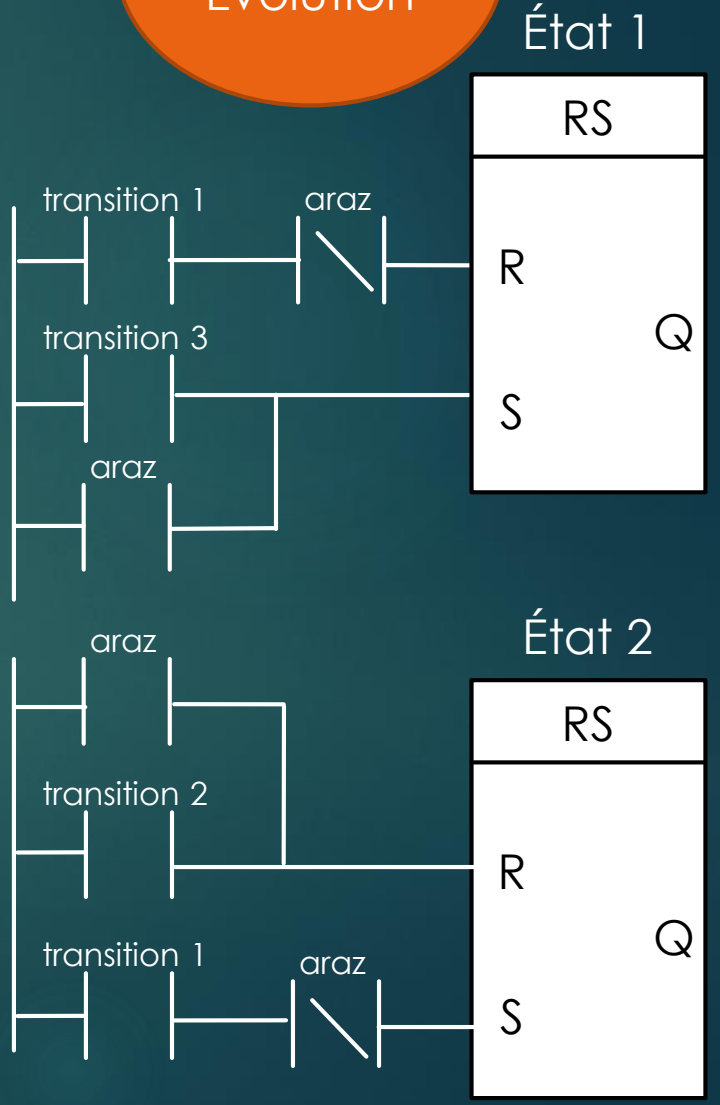
Exemple complet

1
Calcul
des
transitions

2
Évolution







1 – PRINCIPE GÉNÉRAL

2 – STOCKAGE DE L'ÉTAT

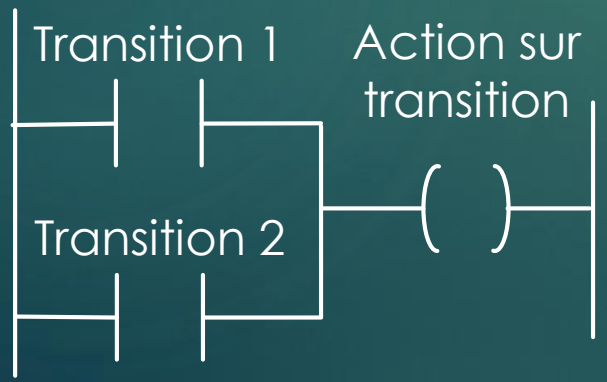
3 – ÉVOLUTION

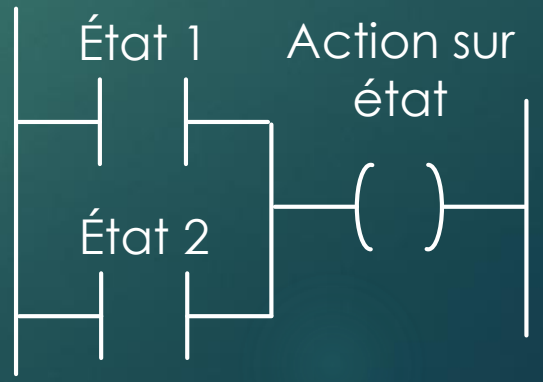
4 – Actions

5 – STRUCTURE GLOBALE DU PROGRAMME

Les actions non mémorisées

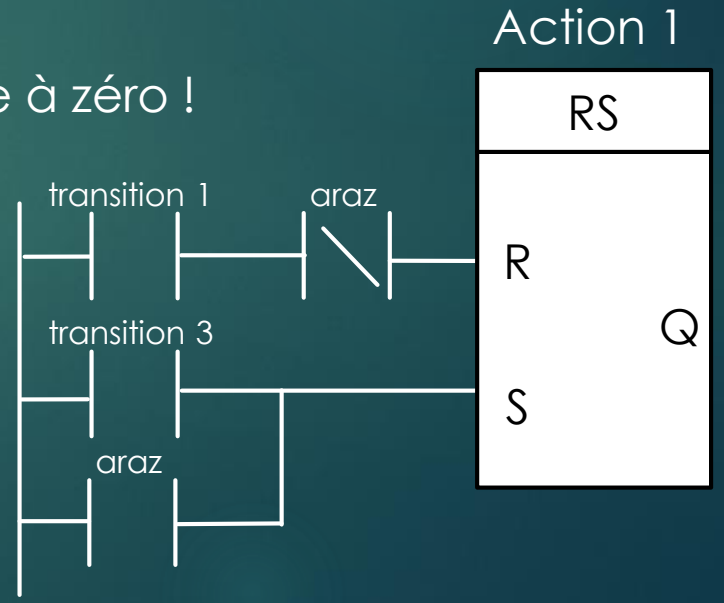
- Les actions non mémorisées sont de simples équations logiques dépendant
  - De l'état pour les actions sur état
  - De l'état et d'une condition pour les actions sur transition
  - Éventuellement, d'une condition supplémentaire pour les actions conditionnelles
- On décrit simplement la condition à l'aide d'une équation Ladder (interrupteurs en série ou en parallèle)





Les actions mémorisées

- Les actions mémorisées doivent être maintenues
  - Elles sont donc stockées dans des bascules
  - Utilisation de bascules RS pour les actions booléennes
- On génère donc une équation pour le Set de la bascule, et une équation pour le Reset
  - Et on n'oublie la valeur initiale avec l'entrée de remise à zéro !



Calcul des actions

- Actions sur transition et actions sur état conditionnelles remplaçant une actions sur transition
  - L'équation pour activer ces transitions est déjà calculée : c'est celle correspondant à la transition
- Les actions sur état
  - Celles-ci ne dépendent que de l'état, leur équation est simple
- Actions non booléennes
  - Les actions non binaires peuvent être réalisées à l'aide d'un multiplexeur, bien que l'utilisation soit plus compliquée, car il faut alors calculer les équations des voies de sélection du mux

1 – PRINCIPE GÉNÉRAL

2 – STOCKAGE DE L'ÉTAT

3 – ÉVOLUTION

4 – ACTIONS

5 – Structure globale du programme

Structure globale du programme

On définit comme variables internes :

- Les variables d'état (mémorisées)
- Les variables correspondant aux transitions (non mémorisées)
- Les variables pour les sorties mémorisées

Définition des entrée, des variables internes et des sorties

En-tête du programme

On affecte aux variables des transitions leur valeur à l'aide d'équations combinatoires

Calcul des transitions actives

L'évolution consiste en la prise en compte des transitions (et du reset) pour stimuler les bascules RS

Évolution

Les actions sont générées après le changement d'état de manière à activer immédiatement les actions sur état

Génération des actions

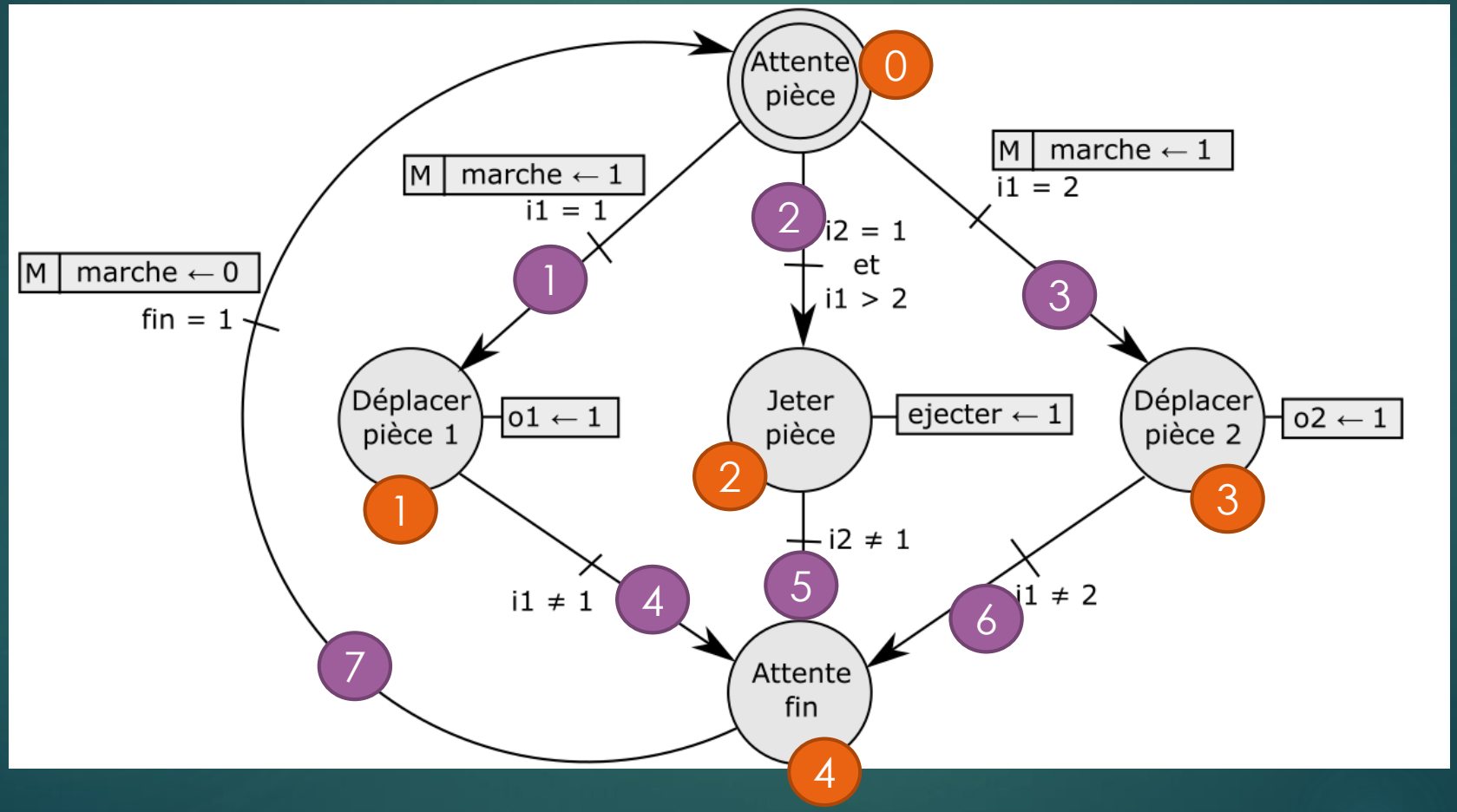
Corps du programme

Exercices

PARTIE TD

Exercice 1

Réaliser le programme LD correspondant à cette MAE



Exercice 2

Réaliser le programme LD correspondant à ce Grafcet