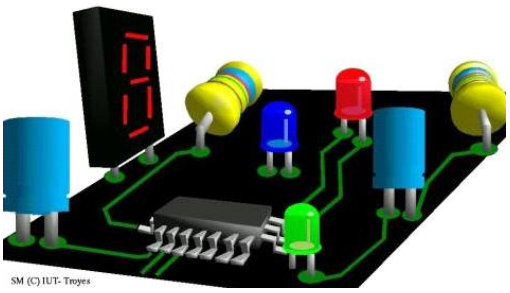
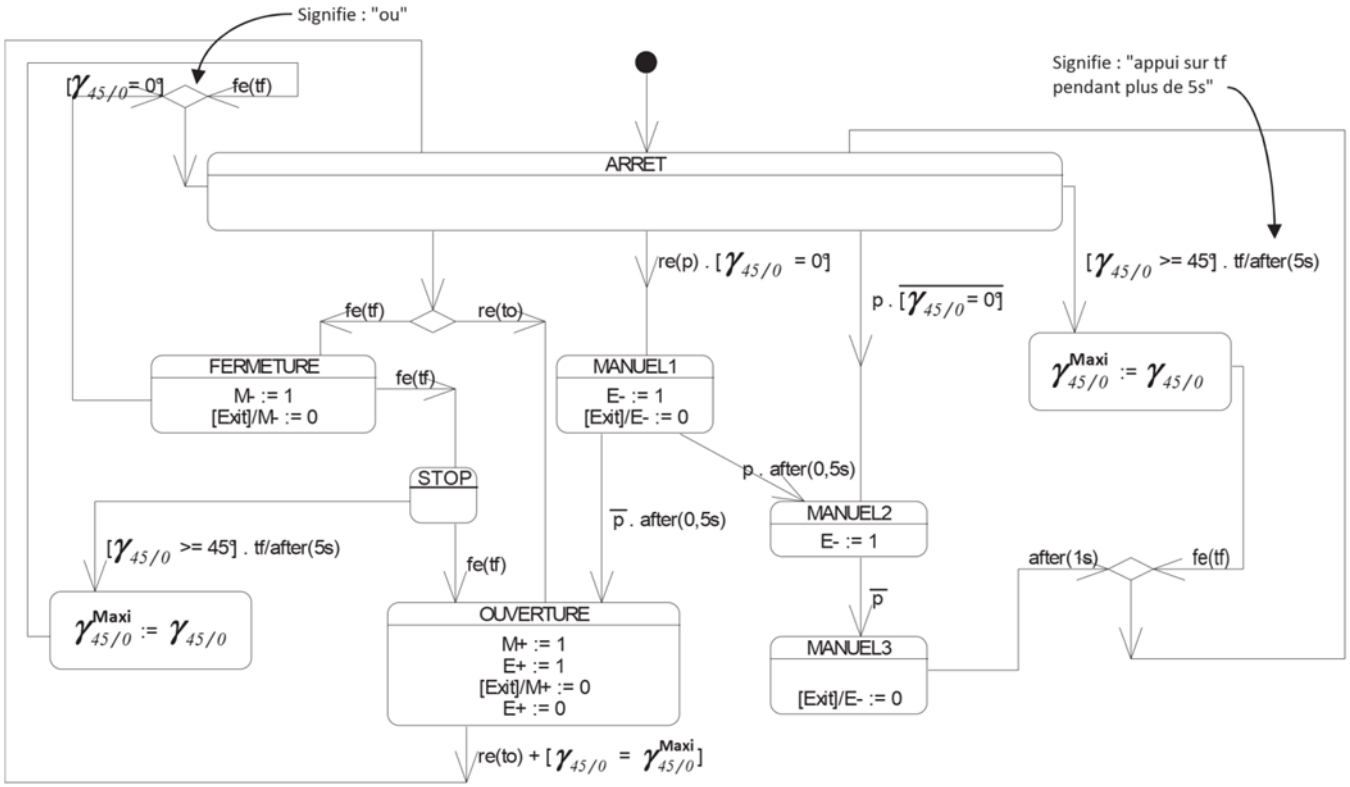
SYSTÈMES SÉQUENTIELS

GRAPHES D'ÉTATS

(COURS)





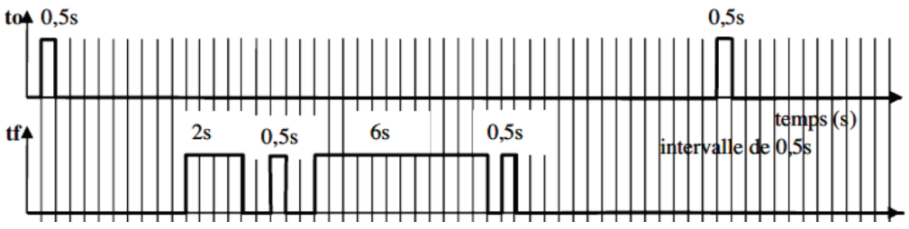


Table des matières

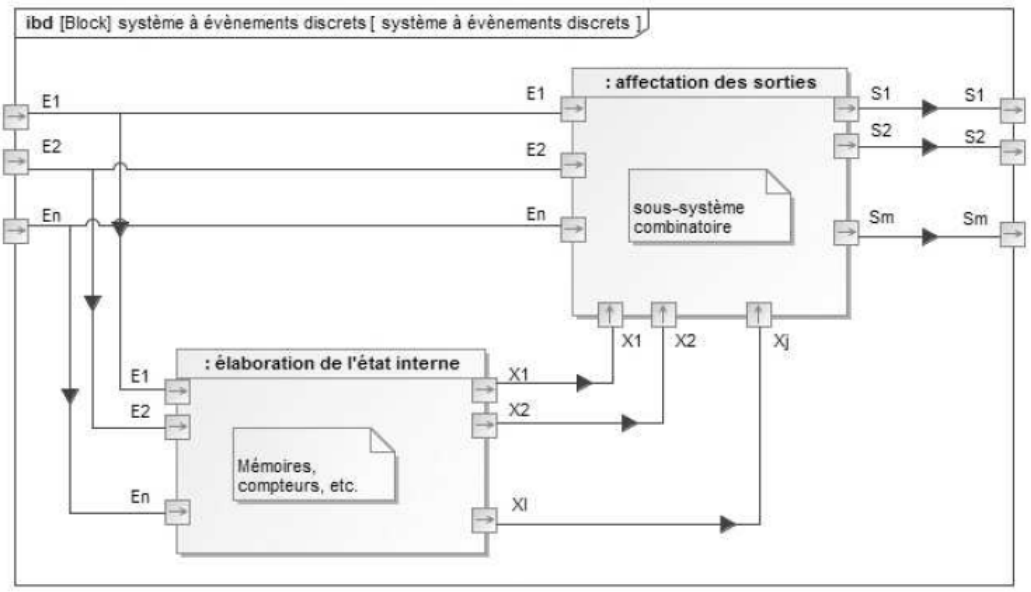
1) Définitions

1.1) Systèmes combinatoires et séquentiels

Nous avons déjà introduit les notions de systèmes automatiques, en séparant les systèmes linéaires continus et invariants (asservis ou non) des systèmes « à évènements discrets » (logiques ou numériques), eux-mêmes sous-divisés en systèmes combinatoires ou séquentiels (cf. chapitre 1 du cours sur les SLCI) :

- Systèmes **combinatoires** : les grandeurs de sortie s'expriment comme une combinaison des grandeurs d'entrée à chaque instant (sans mémorisation des valeurs précédentes) ;
- Systèmes **séquentiels** : les grandeurs de sortie s'expriment comme une combinaison des grandeurs d'entrée ET de l'état précédent des grandeurs d'entrée et de sortie. Les systèmes séquentiels ont donc besoin de mémoires afin de retenir la valeur de certaines variables (d'entrées, de sorties ou internes).

Diagramme de blocs internes d'un système séquentiel :

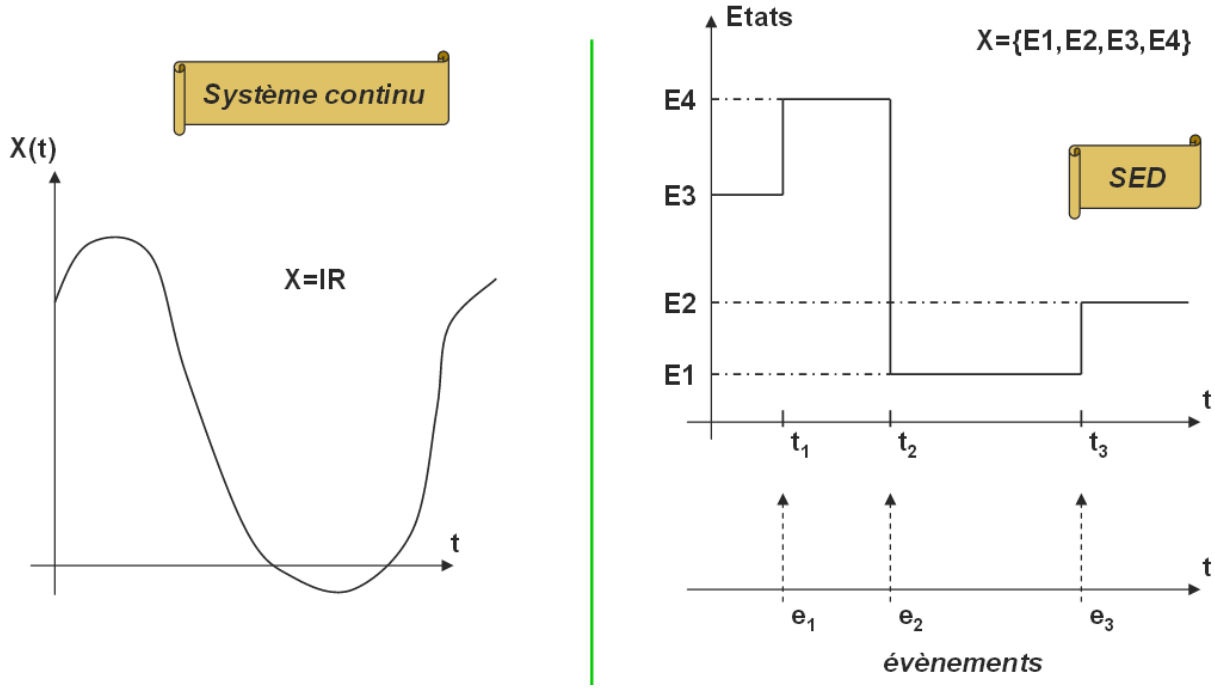


Le vecteur d'état des variables de sortie est $S = (S_1, S_2, \ldots, S_m)$, celui des entrées $E = (E_1, E_2, \ldots, E_n)$ et celui des variables internes est $X = (X_1, X_2, \ldots, X_l)$. Pour exprimer l'état des sorties, il faut disposer d'informations sur l'historique de l'état des variables d'entrées E_i.

Il faut donc connaître l'enchaînement des séquences : $S = f(E, X)$.

1.2) Systèmes discrets

Les **systèmes à évènements discrets (SED)** sont des systèmes dont l'état change seulement à certains instants, lors de l'occurrence d'évènements particuliers, contrairement aux systèmes continus dont l'état change en permanence avec l'évolution du temps.



Les **variables** des SED sont **discrètes**. Si elles sont issues d'un capteur analogique, ces variables continues seront discrétisées (numérisées), via par exemple un convertisseur analogique-numérique (CAN).

Les carte de commande et les ordinateurs (basés tous deux sur des microprocesseurs) sont des systèmes numériques à évènements discrets, et sont aujourd'hui très répandus pour commander les systèmes pluritechnologiques.

1.3) Machine à états

La **machine à nombre fini d'états** est aussi appelée **machine à états**. Elle peut être une entité matérielle (un microprocesseur par exemple), mais aussi une entité conceptuelle comme un algorithme. Elle comporte un nombre fini d'états possibles.
La machine à états est un système à évènement discrets, capable de mémoriser des données, de les traiter et de les restituer selon des scénarios définis au préalable.

Une machine à états possède un **ensemble d'états possibles fini**. L'**état** d'un système est donné par l'ensemble des valeurs de ses variables d'entrées, de sorties et internes.

Dans un état, un système peut avoir une **activité** ou être en attente. Les états d'un système se succèdent en fonction d'évènements.

Un **évènement** est une description d'occurrence qui conduit à une évolution du comportement du système. On l'appelle aussi un déclencheur (trigger).

Par exemple, pour une machine à café, les états peuvent être :
1 – Attente de pièce ;
2 – Descente du gobelet ;
3 – Versement de la poudre à café ;
4 – Versement de l'eau chaude ;
5 – Affichage "café est prêt"…
Et les évènements :
Pièce insérée ;
Gobelet présent ;
Gobelet absent ;
Timer terminé (compte-à-rebours)…
Le comportement (succession d'états) de la machine se déroule selon une séquence. Un tel système est donc bien séquentiel.

Il existe plusieurs outils afin de représenter le fonctionnement d'une machine d'états :
- Diagramme d'états(-transitions) issu du SysML ;
- Diagramme d'activités issu du SysML (pas au programme) ;
- Algorigramme ; …

Il s'agit essentiellement d'outils graphiques permettant de modéliser le comportement séquentiel en termes de déroulement d'actions temporelles.
Ces outils sont, à la base, des outils de modélisation du comportement séquentiel, mais peuvent aussi servir à la programmation des composants réalisant la fonction *Traiter* de la chaîne d'information (microcontrôleur, microprocesseur, automate programmable, …).

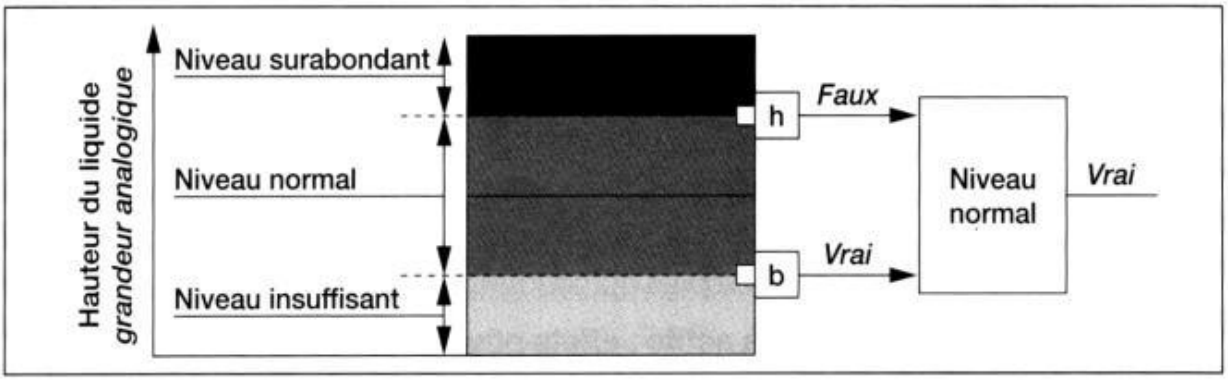
Quel que soit l'outil adopté pour modéliser le comportement séquentiel d'un système, il existe souvent plusieurs « solutions ». « La solution » la plus simple et qui respecte l'ensemble des contraintes est donc à privilégier. C'est le rôle de l'ingénieur de choisir le « bon » outil, et la « meilleure solution ».

2) Opérations logiques

2.1) Fonctions logiques

Un système combinatoire ou séquentiel est décrit par une fonction logique qui permet de définir de manière unique la sortie pour une combinaison des entrées et éventuellement des valeurs précédentes et autres variables internes en mémoire.
Une fonction logique est une combinaison de variables logiques, reliées par des opérateurs logiques.
Il existe 3 **opérateurs logiques** élémentaires : **NON ($\bar{}$)** ; **ET (.)** ; **OU (+)**.

Exemple : Soit la fonction logique *NN* traduisant un niveau normal de liquide dans une cuve.



Le capteur *b* de niveau bas est à 1 si le niveau est au-dessus de lui. De même, le capteur *h* de niveau haut est à 1 si le niveau est surabondant.
Alors, le niveau est normal (*NN* = 1) si *b* est à 1 ET si *h* est à 0. On écrit donc l'équation de la fonction logique :

NN =

On dit que *b* et *h* sont les variables d'entrée de la fonction logique *NN*, qui est appelée elle-même la variable de sortie.

2.2) Table de vérité

La table de vérité d'une fonction logique est l'inventaire des valeurs (binaires) de la variable de sortie, pour toutes les combinaisons possibles des variables d'entrée.
Pour *n* variables binaires d'entrée, il y a **2^n combinaisons possibles**, donc 2^n lignes à la table de vérité.

L'équation de la fonction logique est obtenue en écrivant les combinaisons des entrées pour lesquelles la sortie est à 1.

Exemple ci-contre : F =

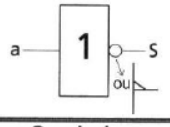
Entrées			***Sortie***
a	**b**	**c**	**F**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

2.3) Opérateurs logiques usuels

1/ L'opérateur NON ou Complément ou Inverseur

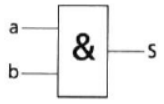
L'opérateur complément transforme une variable "vrai" en une variable "faux", ou inversement.

On le note "¯" ("barre").

Equation	Table de vérité	Symbole Norme IEEE / AFNOR
$S = \overline{a}$	a S 0 1	

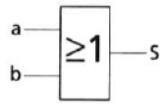
2/ L'opérateur ET

L'opérateur ET donne une sortie "vrai" **si et seulement si toutes les entrées sont "vrai".**
On le note ".".

Equation	Table de vérité	Symbole Norme IEEE / AFNOR
$S = ab$	a b S 0 0 0 1 1 0 1 1	

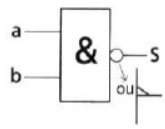
3/ L'opérateur OU

L'opérateur OU donne une sortie "vrai" **si au moins une des entrées est "vrai".**
On le note "**+**".

Equation	Table de vérité	Symbole Norme IEEE / AFNOR
$S = a + b$	a b S 0 0 0 1 1 0 1 1	

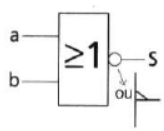
4/ L'opérateur NON ET (NAND)

Il s'agit du complément du ET.

Equation	Table de vérité	Symbole Norme IEEE / AFNOR
$S = \overline{a.b}$ $= \overline{a} + \overline{b}$	a b S 0 0 0 1 1 0 1 1	

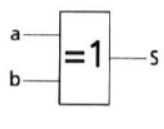
5/ L'opérateur NON OU (NOR)

Il s'agit du complément du OU.

Equation	Table de vérité	Symbole Norme IEEE / AFNOR
$S = \overline{a + b}$ $= \overline{a}.\overline{b}$	a b S 0 0 0 1 1 0 1 1	

6/ L'opérateur OU EXCLUSIF

Il donne une sortie "vrai" **si une des entrées est "vrai", mais pas les deux**.
S'il y a plus de 2 entrées, la sortie est "vrai" si un nombre impair d'entrées est "vrai". On le note "⊕".

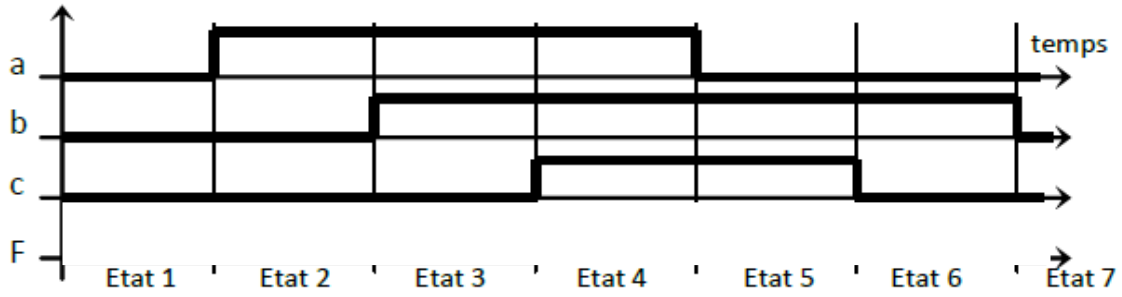
Equation	Table de vérité	Symbole Norme IEEE / AFNOR
$S = a \oplus b$ $= \bar{a}b + a\bar{b}$	a b S 0 0 0 1 1 0 1 1	

2.4) Chronogramme

Le chronogramme est un graphe qui permet de définir l'état d'une variable de sortie à partir de l'état des entrées définies chronologiquement.

Exemple ci-dessous de la fonction $F = a + \bar{a}.\bar{b}.c + \bar{a}.b.\bar{c} + \bar{a}.b.c$ *avec une évolution des entrées au cours du temps comme définie dans le tableau ci-contre.*

Variables	Etat 1	Etat 2	Etat 3	Etat 4	Etat 5	Etat 6	Etat 7
a	0	1	1	1	0	0	0
b	0	0	1	1	1	1	0
c	0	0	0	1	1	0	0

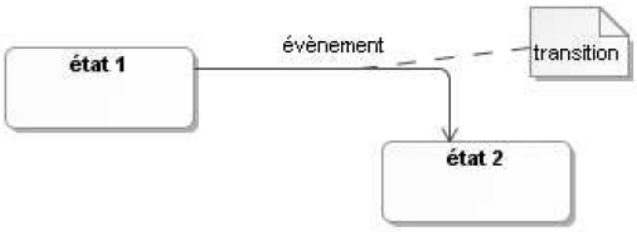


3) Diagramme d'états

3.1) Présentation et formalisme

Le diagramme d'états (*state machine diagram* ou *stm*) est un diagramme normalisé SysML, il est utilisé pour décrire le comportement d'une machine d'états :

- Le nœud (rectangle) symbolise un état interne. Il montre ce qui se passe.
- L'arc qui relie deux noeuds est appelé transition. Il indique la possibilité de passer d'un état à un autre lorsqu'un évènement se produit.



Un même état représenté par un noeud. Au cas où aucune activité n'est souhaitée, c'est un état d'attente.

Les cas d'évolution du système entre les états sont identifiés par les transitions. On trace alors un lien entre les noeuds correspondants. Chaque transition peut être qualifiée par un évènement, une condition de garde et un effet.

3.2) Règle d'évolution

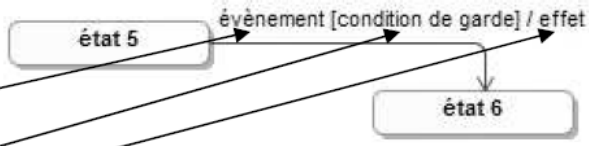
Lorsqu'une transition est active (c'est-à-dire lorsque son état source est actif) et franchissable (son évènement et sa condition de garde sont vrais), alors « instantanément » l'état source est inactivé, éventuellement l'effet de la transition est effectué, et l'état destination est activé.

3.3) Transition

Une transition représente le passage instantané d'un état vers un autre. Une transition ne peut donc pas avoir de durée. On appelle état source l'état de départ d'une transition et état destination l'état d'arrivée.
Une transition est déclenchée par un évènement. En d'autres termes : c'est l'arrivée d'un évènement qui conditionne le franchissement de la transition. Une transition peut aussi être automatique, lorsqu'on ne spécifie pas l'évènement : elle s'effectuera alors lorsque l'activit de l'état source sera terminée (et lorsqe la condition de garde sera vraie).

A une transition peuvent être associés :

un évènement,
et/ou une condition de garde,
et/ou un effet (une action).



Dans une transition,

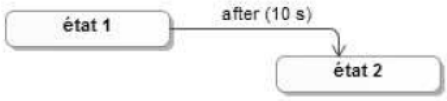
- le ET logique s'écrit **&**
- le OU logique s'écrit ||
- la fonction NON s'écrit **!**

✷ Evènements

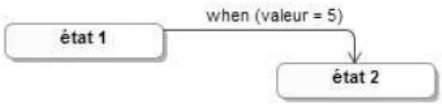
Il existe quatre types d'évènements associés à une transition :

- le **message** (signal event) : un message asynchrone est arrivé,

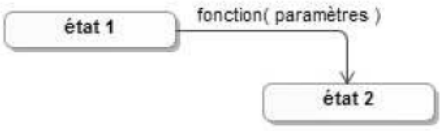
- l'**événement temporel** (time event) : un intervalle de temps s'est écoulé depuis l'entrée dans un état (mot clé « after ») ou un temps absolu a été atteint (mot clé « at »),



- l'**évènement de changement** (change event) : une valeur a changé de telle sorte que la transition est franchie (mot clé « when »),



- l'**évènement d'appel** (call event) : une requête de fonction (operation) du bloc a été effectuée. Un retour est attendu. Des arguments (paramètres) de fonction peuvent être nécessaires.



✷ Condition de garde

La condition de garde est une expression booléenne (ou fonction logique) faisant intervenir des entrées et / ou des variables internes. Elle se note entre crochets.
Elle autorise le passage d'un état à un autre.
Il est possible d'utiliser les notations non booléennes de front montant (↑) et front descendant (↓).

Remarque : Différence entre évènement et condition de garde :
Un événement est parfaitement daté dans le temps, il correspond par exemple à un passage d'une variable de 0 à 1 (front montant). Une condition de garde n'est pas datée, elle doit être vraie (niveau logique 1) à l'instant où l'événement survient pour que la transition soit franchie.

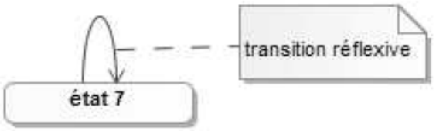
Exemple d'évènement : appui sur un bouton-poussoir, capteur fin de course atteint, etc.
Exemple de condition de garde : vitesse du véhicule non nulle, température > 20°C, etc.

✷ Effet

L'effet associé à une transition est effectué lorsque la transition est franchie, il ne prend pas de temps et ne peut pas être interrompu. Il se note après l'évènement, précédé d'un « / ».
Son exécution peut par exemple provoquer un changement d'état ou l'émission d'un ordre.

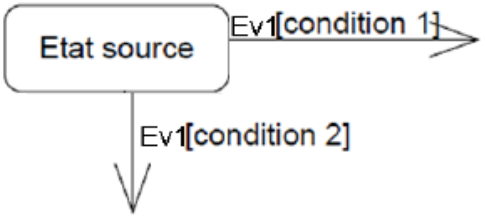
✷ Transition réflexive

Une transition réflexive entraîne une sortie d'état puis un retour dans ce même état.

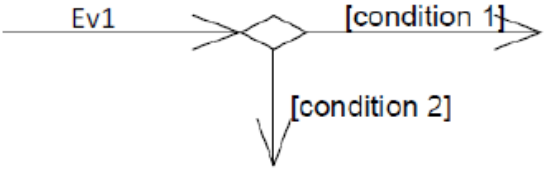


✷ Transitions conditionnelles

Plusieurs transitions peuvent quitter un même état.
Une seule d'entre elles doit être déclenchée ; les événements et / ou les conditions de garde doivent donc être **exclusives**.



Une autre représentation est possible, qui utilise le point de décision (**losange** de choix).

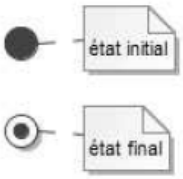


Il est possible d'utiliser une condition particulière, notée [**else**], sur un des segments en aval d'un point de décision. Ce segment n'est franchissable que si les conditions des autres segments sont toutes fausses.

3.4) États

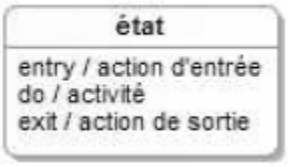
Un état représente une période de la vie du système. Pendant cette période, le système accomplit une ou plusieurs actions, ou attend un évènement. Il peut être actif ou non ; plusieurs sous-états peuvent être actifs en même temps dans le même système.
Chaque (sous-)état est dessiné sous la forme d'un rectangle aux coins arrondis (ou une ovale), contenant son nom.

✷ États initial et final



Il existe deux états particuliers :
- l'état initial : pseudo-état qui indique un point d'entrée dans un graphe ;
- l'état final : non obligatoire, il indique que le système décrit n'a plus d'état actif.

✷ Activités et actions associées à l'état



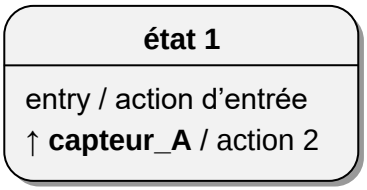
A un état, on peut principalement rattacher :
- une action d'entrée, précédée de « entry/ » ;
- une activité, précédée de « do/ » ;
- une action de sortie, précédée de « exit/ ».

Une **activité** peut être considérée comme une unité de comportement. Elle prend du temps et peut être interrompue. On la trouve à l'intérieur des noeuds du diagramme (mot-clé « do »).

A contrario, une **action** ne prend pas de temps et ne peut pas être interrompue. Son exécution peut par exemple provoquer un changement d'état, l'émission d'un ordre pour un pré-actionneur ou un retour de valeur. On peut les trouver dans les transitions (effet) ou dans les états (mot-clé « entry » ou « exit »). Une action peut être :
- un ordre de mise à 1 d'une sortie ; la syntaxe est alors SORTIE := 1 ;
- un ordre de mise à 0 d'une sortie ; la syntaxe est alors SORTIE := 0 ;
- une modification d'une variable numérique interne, par exemple un compteur C : la syntaxe est alors C := C + 1 ; pour incrémenter la valeur du compteur C de 1.

Pendant que l'état est actif, un évènement peut lancer une action avec la syntaxe : **évènement/** suivi de l'action. Cette action est lancée chaque fois que l'évènement survient, tant que l'état est actif.

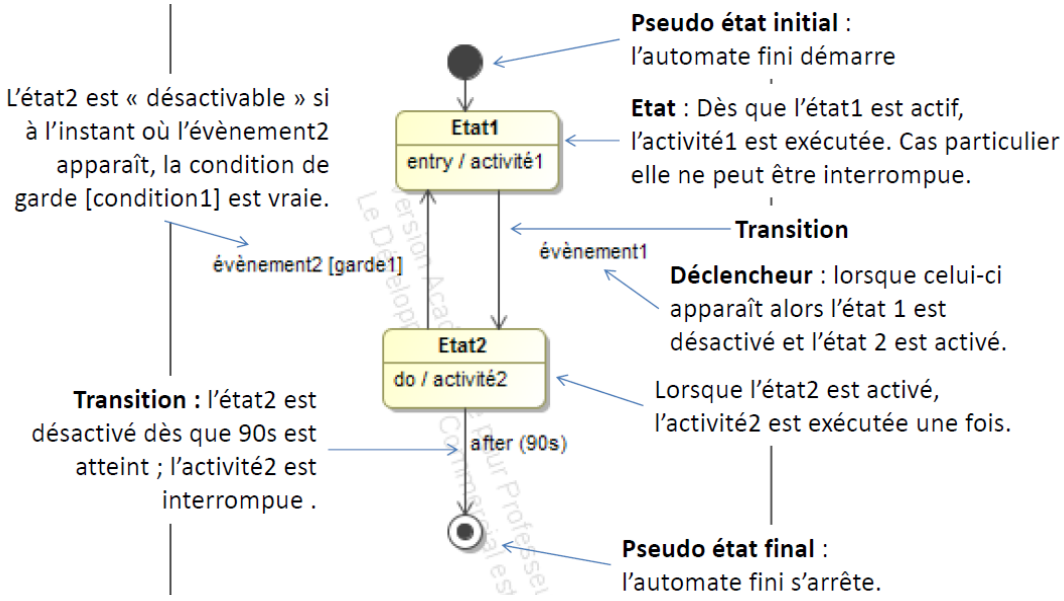
Exemple d'un évènement « front montant du capteur_A » déclanchant une action 2 :



Remarque : Un état peut ne pas contenir d'action. Il sert alors à attendre le déclenchement de la transition suivante. On parle d'**état d'attente**.

Exemple d'un état vide, d'attente.

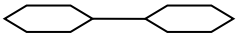
3.5) Exemple générique de base

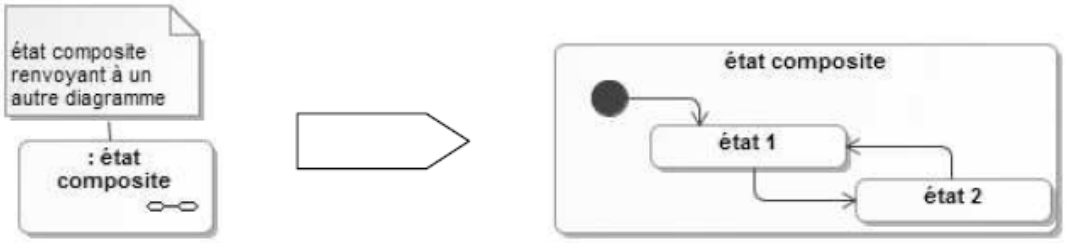


3.6) États composites et hiérarchisation

✷ État composite

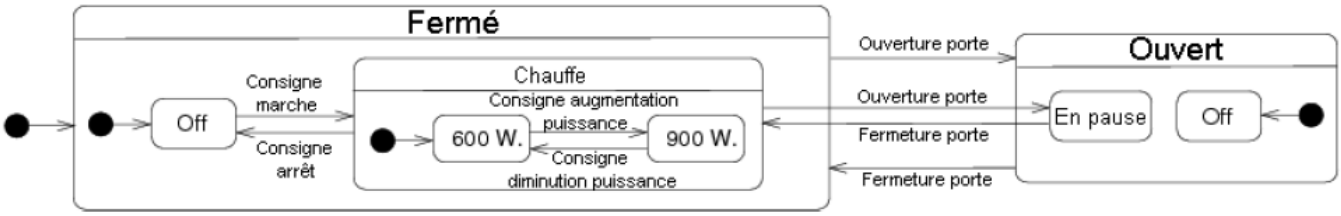
Lorsque le comportement à décrire :
- risque d'être peu lisible car trop complexe,
- fait apparaître des parties "séparables" de l'ensemble,

alors on extrait ces parties pour en faire des sous-graphes séparés, hiérarchiquement inférieurs au graphe principal. On englobe ces sous-graphes dans un état appelé « état composite » ou « super-état » ou « état sous-machine » ou « sous-machine à état ».
Un état composite est composé de plusieurs sous-états internes et possède un état initial. Il est repéré par ce symbole, en bas à droite de l'état : 



✷ Règles d'évolution avec des états composites

Prenons l'exemple d'un four à micro-ondes :

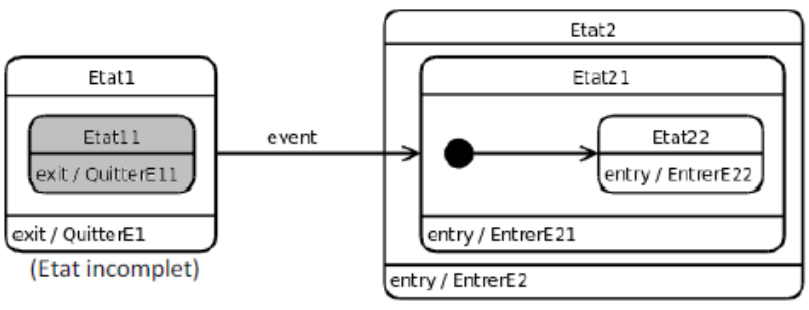


Il y a 2 états composites "Fermé" et "Ouvert". L'état composite "Fermé" contient lui-même un état composite "Chauffe". Il contient lui-même un sous-graphe formé de 2 états "600 W" et "900 W".

• Il peut y avoir des **transitions ayant pour source / cible la frontière** d'un sous-état ou la frontière d'un état composite. Si une transition de frontière est franchie, alors l'ensemble de l'état composite source est désactivé, et c'est l'état initial de l'état composite cible qui est activé. Si la transition est vide (sans événement, ou « 1 ») : l'événement implicite est la fin la dernière activité susceptible d'être réalisée dans le diagramme d'état composite, par exemple l'arrivée à un pseudo état final.

• Quand **plusieurs transitions sont possibles**, on choisit de suivre celle qui part de l'état le plus en bas de la hiérarchie des états actifs, donc ici partant de "Chauffe" et non pas de "Fermé". Autrement dit :
- si l'état actif est "Chauffe / 600W" quand on ouvre la porte, l'état "En pause" s'active ;
- si l'état actif est "Off" contenu dans "Fermé" quand on ouvre la porte, l'état "Ouvert" s'active et donc l'état "Off" qu'il contient.

• *Autre exemple de l'évolution avec des états composites :*

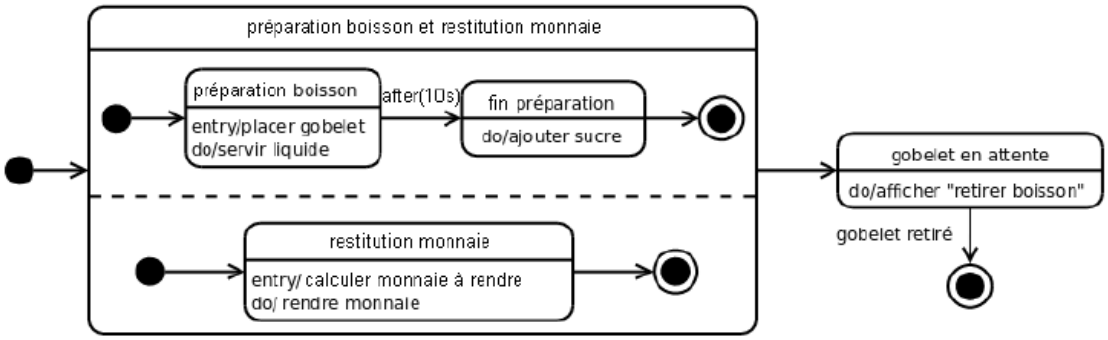


Supposons que l'état actif soit l'Etat11. Alors, lorsque l'évènement "event" arrive, l'Etat1 est désactivé et il se produit successivement les actions suivantes :
- Quitter E11
- Quitter E1
- EntrerE2
- EntrerE21
- EntrerE22,
et l'état actif est l'Etat22 (les états composites Etat 21 et Etat2 sont aussi actifs).

✷ Concurrence et synchronisation (états orthogonaux)

Dans un état composite, plusieurs graphes d'états peuvent évoluer simultanément (en parallèle). On dit qu'il y a concurrence de plusieurs états.
Par exemple pour un distributeur de boissons :

L'état composite est dit orthogonal car il comporte plus d'une région, chaque région représentant un flot d'exécution.

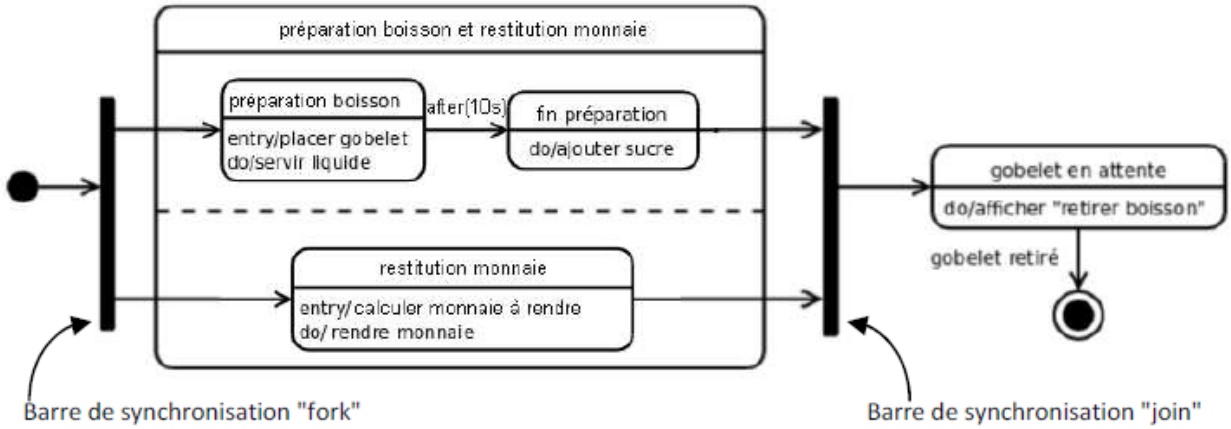


• Graphiquement, dans un état orthogonal, les différentes régions sont séparées par un trait horizontal ou vertical en pointillés allant d'un bord à l'autre de l'état composite.

• Chaque région peut posséder un état initial et final. Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes.

• Toutes les régions concurrentes d'un état composite orthogonal doivent atteindre leur état final pour que l'état composite soit considéré comme terminé. La synchronisation est alors automatique et la transition de sortie de l'état composite est déclenchée.

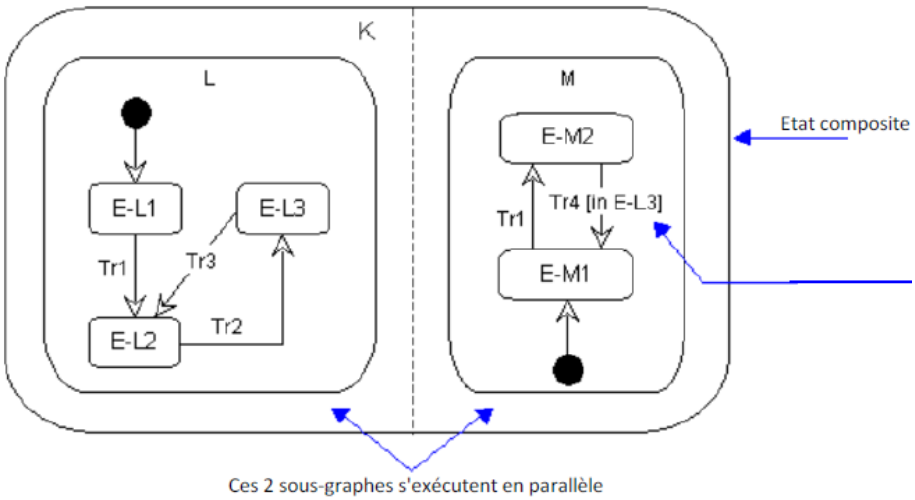
• Il est également possible de représenter ce type de comportement au moyen de transitions concurrentes constituées de **barres de synchronisation "fork" et "join"**. Le graphe ci-dessous est une représentation équivalente à la précédente :



• Les transitions automatiques (sans évènement ou condition de garde) qui partent d'une barre de synchronisation "fork" se déclenchent en même temps. On ne franchit une barre de synchronisation "join" qu'après déclenchement de toutes les transitions qui s'y rattachent.

Remarque : Sur ce diagramme, l'état orthogonal "préparation boisson et restitution monnaie" peut éventuellement ne pas apparaître (tout en gardant la représentation de ses sous-états) pour alléger la représentation, car la notion de concurrence est clairement apparente de par l'utilisation des barres.

✷ Conditionner une transition par l'activité de l'état d'un autre sous-graphe :



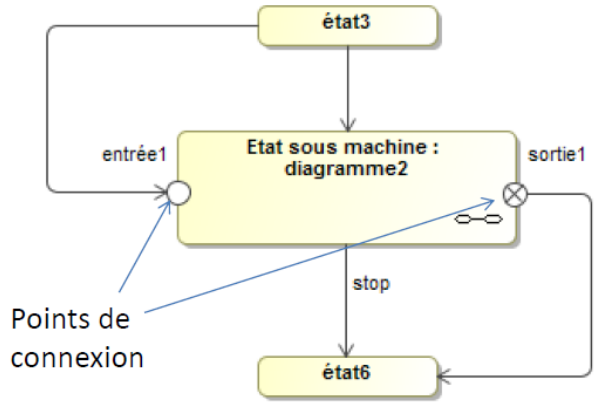
L'état composite orthogonal K est composé des sous-états L et M.
L et M s'activent simultanément, et évoluent indépendamment, en parallèle. Au départ, EL1 et E-M1 sont actifs.

Pour que M passe d'E-M2 à E-M1, il faut que l'évènement Tr4 survienne, à condition que E-L3 soit actif.
Cet évènement est conditionné par l'activité de l'état E-L3 de l'autre sous-graphe.

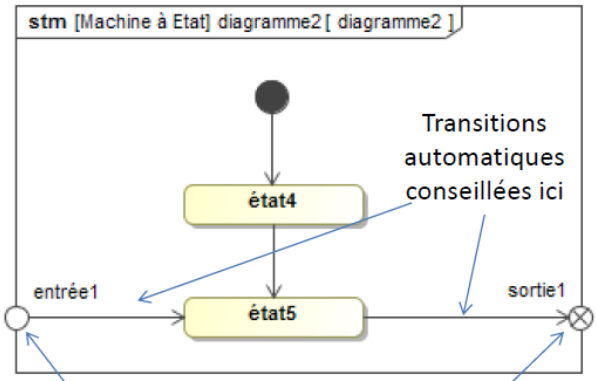
La syntaxe de la condition de garde vérifiant l'activité de ETAT est : [in ETAT].

✷ Points d'entrée/sortie multiples :

Lorsque plusieurs points d'entrée et/ou sortie sont utiles dans une sous machine, il faut placer des pseudo-états correspondants :



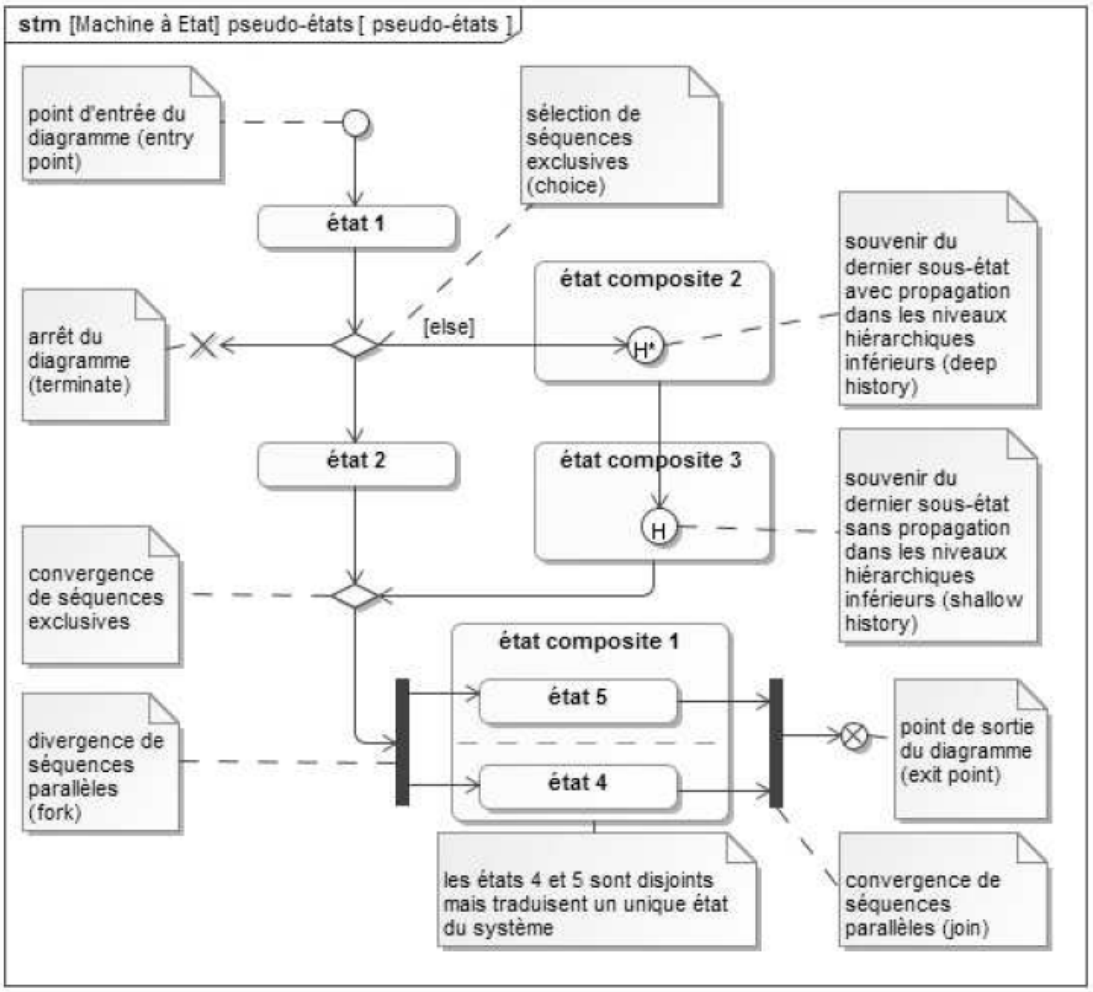
C'est le nom (ex : entrée1) du point de référence qui fait le lien avec le pseudo-état point d'entrée de la sous machine.



3.7) Les neufs pseudo-états, et l'historique H et H*

Le formalisme SysML admet neuf pseudo-états :

- « shallow history » (H) : permet à un état de niveau hiérarchique supérieur (état composite) de se souvenir du dernier sous-état, avant qu'il n'évolue vers un autre état,
- « deep history » (H*) : idem que précédemment mais avec la propagation de l'historique à tous les sous-états composites de niveaux hiérarchiques inférieurs,
- « fork » ▬ et « join » ▬ : divergence et convergence de séquences parallèles,
- « choice » ◇ : sélection et convergence de séquences exclusives. Il est nécessaire qu'une condition située en aval soit vraie pour que l'évolution du système se poursuive. Les conditions de gardes doivent être exclusives. Le mot clé « else » peut-être utilisé pour englober tout ce qui n'est pas décrit dans les autres expressions booléennes. Les conditions de garde situées en aval sont toutes évaluées une fois le pseudo-état atteint,
- « junction » ● : idem au pseudo-état « choice », à la différence que pour qu'un chemin soit emprunté, toutes les conditions de garde situées en aval et en amont, doivent être vraies. L'évaluation des conditions avales est réalisée avant que le pseudo-état soit atteint,
- « entry point » ○ et « exit point » ⊗ : permet de créer un point d'entrée du diagramme et un point de sortie vers un autre diagramme,
- « terminate » ✕ : permet de terminer une séquence sans destruction de l'instance de bloc.



3.8) Démarche de modélisation d'un système par un graphe d'états

Etape 1 : Définir la frontière du système et recenser les variables d'entrées et de sorties ;

Etape 2 : Recenser, nommer et tracer les états du système ;

Etape 3 : Tracer les transitions entre les états en fonction du comportement séquentiel souhaité ou observé ;

Etape 4 : Définir les conditions (et évènements) associées à chaque transition et les actions associées à chaque état ;

Etape 5 : Vérifier, pour le graphe d'états complet, les propriétés de :

- **complétude** : signifie que le comportement est toujours défini : à chaque évolution des variables d'entrées (conditions ou évènements), et quel que soit l'état source, il existe au moins un état destination ;
- **non contradiction** : signifie qu'à tout changement des variables d'entrées (conditions et évènements), un et un seul état destination existe ;

Exemple : commande TOR d'un moteur

2 boutons poussoir ***m*** (marche) et ***a*** (arrêt) commandent le fonctionnement d'un moteur.

Le cahier des charges est le suivant :

- le moteur doit démarrer si le bouton poussoir marche « m » est actionné,
- une fois ce dernier relâché, le moteur doit continuer à tourner jusqu'à l'appui sur « a »,
- en cas d'appui simultané sur les deux boutons, on privilégie l'arrêt du moteur.

Tracer le diagramme d'états permettant de commander ce moteur à partir des deux boutons.