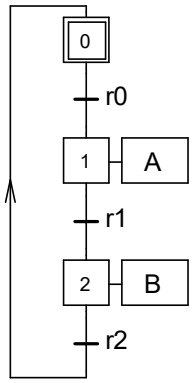
Transcription d'un grafcet en équations logiques et en programmation (langage ST)

Prérequis : connaissance de base du grafcet (syntaxe et règles d'évolution)



cas général :

Une étape **i** est une mémoire $\mathbf{X_i}$ qui peut être active ou inactive. La priorité est donné à l'activation de l'étape (règle d'évolution N°5)

$$X_i = (condition\ d'activation) + \overline{condition\ de\ désactivation}\ .\ X_i$$

Au début du fonctionnement, la partie commande démarre en situation initiale (étape initiale active et les autres étapes inactives) conformément à la règle d'évolution N°1.

Une variable d'initialisation « **init** » doit être introduite dans les équations pour cela.

D'où :

L'étape initiale X_0 doit s'activer à l'initialisation :

$$X_0 = (condition\ d'activation + init) + \overline{condition\ de\ désactivation}\ .\ X_0$$

Les autres étapes Xi (i≠0) doivent se désactiver à l'initialisation :

$$X_i = (condition\ d'activation) + \overline{condition\ de\ désactivation + init}\ .\ X_i$$

Ce qui donne les équations suivantes pour le grafcet ci-dessus :

- $X_0 = (X_2 . r_2 + init) + \overline{X_0 . r_0} . X_0$

$[X_2 . r_2]$: condition d'activation

$[X_0 . r_0]$: condition de désactivation

- $X_1 = (X_0 . r_0) + \overline{X_1 . r_1 + init}\ .\ X_1$
- $X_2 = (X_1 . r_1) + \overline{X_2 . r_2 + init}\ .\ X_2$
- $A = X_1$
- $B = X_2$

Le terme de type $\overline{X_1.r_1+init}.X_1$ peut être simplifié pour faciliter le câblage ou la programmation.

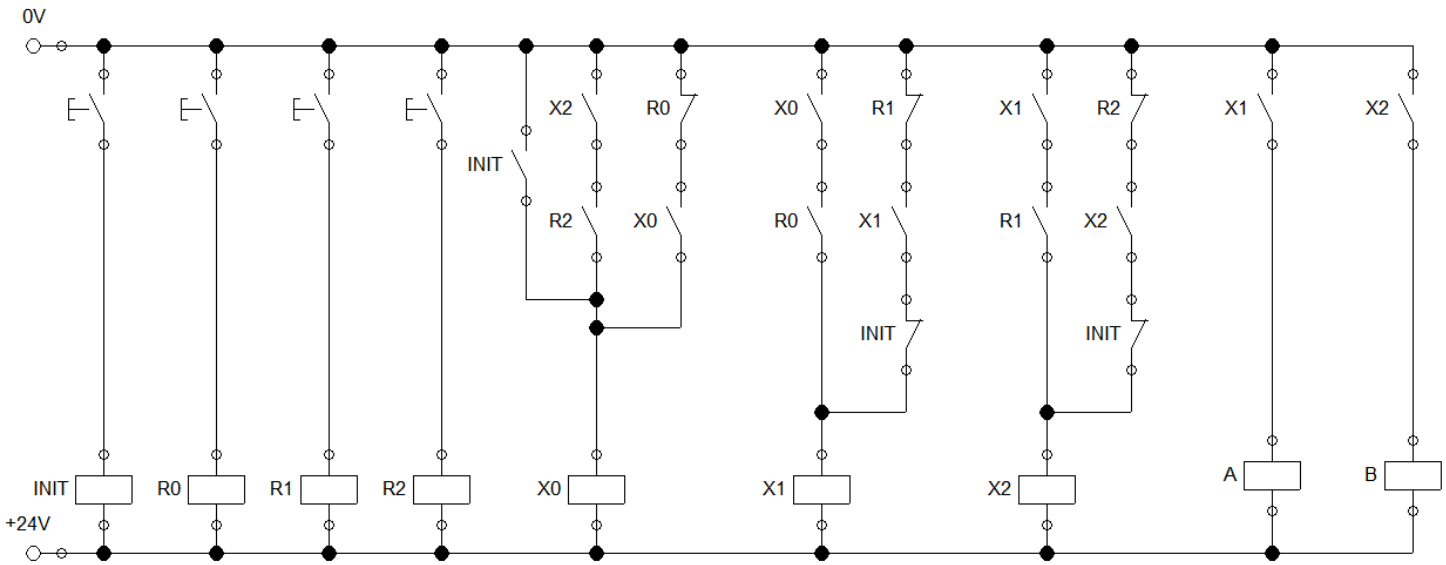
$\overline{X_1.r_1+init}.X_1$
= $\overline{X_1.r_1}.\overline{init}.X_1$ (théorème de Morgan $\overline{a.b}=\bar{a}+\bar{b}$)
= $(\overline{X_1}+\overline{r1}).\overline{init}.X_1$ (théorème de Morgan $\overline{a.b}=\bar{a}+\bar{b}$)
= $\overline{X_1}.\overline{init}.X_1+\overline{r1}.\overline{init}.X_1$ $(a+b).c=ac+bc$
= $\overline{r1}.\overline{init}.X_1$ $a.\bar{a}=0$

En généralisant cette simplification nous obtenons :

$X_0=(X_2.r_2+init)+\overline{r_0}.X_0$
$X_1=(X_0.r_0)+\overline{r_1}.\overline{init}.X_1$
$X_2=(X_1.r_1)+\overline{r_2}.\overline{init}.X_2$
$A=X_1$
$B=X_2$

Exemple de réalisation technologique : câblage électrique
La logique obtenue est réalisée en câblage électrique. La logique est bien respectée.
Rq : On remarque une instabilité si X0 est activée et r0.r1.r2=1

☞voir sur moodle le circuit de simulation fluidsim : **algo_G7_1**



Risque de perte d'activation des mémoires Xi :
Une hypothèse forte est pris en compte dans la norme GRAFCET : la **causalité à temps nul** en temps externe.
Cela signifie que, (à l'échelle de temps des sorties A et B), un changement d'état des entrées induit **instantanément** la modification des sorties correspondante à la logique du grafcet.
Cette hypothèse exclue la prise en compte des retards inhérents à tous composants physique. (cette hypothèse ne peut donc pas être respectée en réalité)

Conséquence : un câblage peut ne pas donner les résultats attendus même si la logique de commande « théorique » est respectée.

Exemple : En rajoutant ici la variable KA_r1, la logique de commande reste inchangée.

$$X_0=(X_2.r_2+init)+\overline{r_0}.X_0$$
$$X_1=(X_0.r_0)+\overline{r_1}.\overline{init}.X_1$$

$$KA_{r1}=r_1$$

$$X_2=(X_1.KA_{r1})+\overline{r_2}.\overline{init}.X_2$$
$$A=X_1$$
$$B=X_2$$

Pourtant en rajoutant ce composant (relais intermédiaire de r1), nous constatons **une perte d'activation des mémoires Xi** au moment du passage de l'étape 1 à l'étape 2.
=> L'activation de X2 et la désactivation de X1 qui « devrait » se réaliser simultanément, se fait avec un léger retard correspondant au temps de réponse du relais KA_r1 (quelques dixièmes de seconde entre l'activation électrique de la bobine et la fermeture mécanique des contacts électriques associés) : dans l'ordre :
- désactivation de la bobine X1 par $\overline{R_1}$ et activation de la bobine KA_r1 (au même instant)
puis avec un retard :
- fermeture des contacts KA_r1

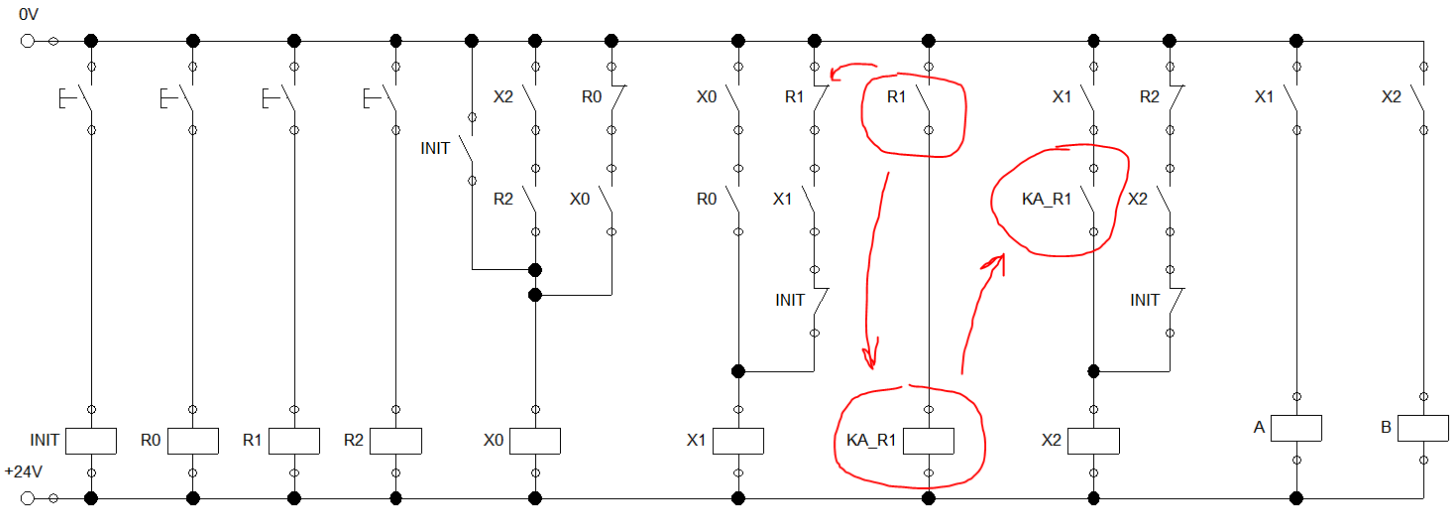
chronogramme

Au moment de la fermeture des contacts KA_r1, la mémoire X1 n'est donc plus activée, la condition d'activation de X2= $X_1.KA_{r1}$ n'est plus vraie donc la mémoire X1 ne s'active pas.
Il n'y a donc plus aucune mémoire activée, les sorties A et B ne sont plus commandées.

☞voir sur moodle le circuit de simulation fluidsim : **algo_G7_2**

Une ré-initialisation des mémoires Xi est alors nécessaires pour reprendre le fonctionnement à partir de l'étape 0.

Une variable intermédiaire KA_R1 est rajoutée pour "temporiser" la prise en compte de R1 dans l'activation de X2.
ici la temporisation correspond au temps d'activation du relais KA_R1



Ici le retard a été créé volontairement, il est donc facilement identifiable et prévisible : dans les réalisations technologiques des parties commandes, l'influences des temps de réponses des composants et des retards inhérents à leur conception est plus délicat à prévoir.

=> La transcription exacte de la logique du grafcet en câblage électrique ou électronique comporte des risques de perte d'activités des mémoires Xi.

<u>Question :</u>
Une logique programmée serait-elle plus facile à mettre en œuvre pour respecter la logique du grafcet ?

Exemple de réalisation technologique : programmation en langage littéral structuré (ST) sur API-M340.

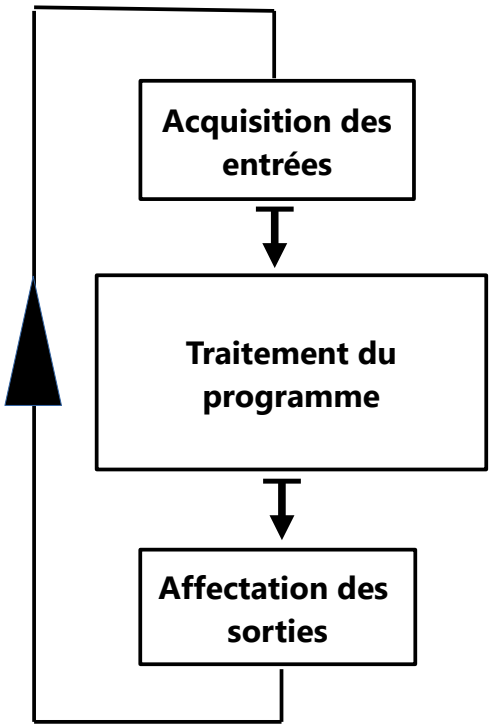
La logique booléenne de commande est écrite en langage littéral structuré (proche du langage C) à l'aide du logiciel UnityPro permettant la programmation des automates M340 Schneider Electric.

```
(*GESTION DES MEMOIRES*)
X0:=X2 and r2 or init or not r0 and X0;
X1:=X0 and r0 or not r1 and not init and X1;
X2:=X1 and r1 or not r2 and not init and X2;

(*AFFECTATION DES SORTIES*)
A:=X1;
B:=X2;
```

Ce programme ne fonctionne pas, même si les conditions écrites pour les mémoires **Xi** correspondent aux équations logiques issues du grafcet.

En voici la raison : L'automate fonctionne de manière cyclique en effectuant à chaque cycle : l'**acquisition des entrées**, puis le **traitement du programme** (jusqu'à obtention d'un état stable), puis l'**affectation des sorties**. La période du cycle est fixe ou variable suivant la configuration de l'automate. L'ordre de grandeur de la période est de 50ms.
=> en 1s le programme est donc lu et exécuté environ 20 fois.



Le traitement du programme est fait ligne après ligne. Les instructions d'affectations des variables (X1:=X0 and r0... par exemple) présentes sur une lignes sont réalisées avant passage à la ligne suivante. **L'ensemble du programme est lu et exécuté avant l'affectation des sorties.**

```
X0:=X2 and r2 or init or not r0 and X0;
X1:=X0 and r0 or not r1 and not init and X1;
X2:=X1 and r1 or not r2 and not init and X2;
```

Dans notre cas, après initialisation, X0=1 (l'état est stable).
Si la variable r0=1 est détectée dans l'acquisition des entrées, alors :
- **la première ligne du programme impose X0:=0** (`not r0`)
- la deuxième ligne (lue juste après la première) **impose X1:=0** (la condition `X0 and r0` est fausse puisque X0=0).

En d'autres termes : la désactivation de X0 est faite avant le traitement des conditions d'activation de X1, or X0 est une condition nécessaire pour l'activation de X1: il y a donc une perte d'activité des mémoires Xi.

Conclusion : L'activation d'une mémoire et la désactivation de la mémoire précédente ne sont pas réalisées simultanément ce qui provoque une perte d'activation des mémoires Xi.
Pour + de détails, voir : 3.3 Exécution monotâches p79 (manuel-de-référence UNITY PRO)

Modification du programme :

- L'activation d'une étape et la désactivation de l'étape précédente doivent se faire simultanément, ces actions doivent donc appartenir à la même instruction (ici : de If… à end_if)
- **Une instruction [if…end_if] est utilisée à chaque transition entre 2 étapes du grafcet.**
- Une instruction [if…end_if] réalise l'**initialisation grâce au bit système %S21** de l'automate (%S21=1 au premier cycle de scrutation uniquement)

```
(*AFFECTATION DES MEMOIRES*)

(*initialisation*)
(*bit %S21: bit drapeau 1er cycle de scrutation*)
if %S21 then
X0:=1;
X1:=0;
X2:=0;
end_if;

(* franchissement 0=>1 *)
if X0 and r0 then
X0:=0;
X1:=1;
end_if;

(* franchissement 1=>2 *)
if X1 and r1 then
X1:=0;
X2:=1;
end_if;

(*franchissement 2=>0*)
if X2 and r2 then
X2:=0;
X0:=1;
end_if;

(*AFFECTATION DES SORTIES*)

(*étape_1*)
A:=X1;
(*étape_2*)
B:=X2;
```

Programmation d'une temporisation en langage ST

```
Tempo_pale_sens_1 (IN := X_71, PT := t#2s);
Tempo_pale_arret (IN := X_72, PT := t#2s);
Tempo_pale_sens_2 (IN := X_73, PT := t#2s);

(* franchissement 70=>71 *)
if X_70 and _1s1 and not ckm11 and not ckm12
X_70:=0;
X_71:=1;
end_if;

(* franchissement 71=>72 *)
if X_71 and Tempo_pale_sens_1.q then
X_71:=0;
X_72:=1;
end_if;
```

Programmation d'un compteur en langage ST

Action mémorisée

convergence en OU

divergence en OU

convergence en ET

divergence en ET