TME1 : Automates finis

(définitions, exemples, programmation en Python)

– version 2.0 (Janvier 2022) –

Mathieu Jaume

Remarque. Dans tous les TME Python, la plupart des définitions et fonctions implantées sont paramétrées par une fonction d'égalité permettant de comparer certains éléments (états, transitions, etc.). Ceci est utile lorsque l'égalité atomique de Python (==) ne permet pas de tester l'égalité de deux entités : c'est par exemple le cas lorsque l'on souhaite comparer des états représentés par des ensembles (eux même représentés par des listes sans doublons) ou encore par des couples (dont chaque composante n'est pas nécessairement d'un type atomique). Le code ainsi obtenu est générique et peut être utilisé dans différents contextes quelles que soient les représentations utilisées pour les états et les transitions (à charge pour l'utilisateur de fournir la fonction d'égalité adéquate). L'annexe sur la représentation des ensembles détaille l'implantation des ensembles finis (représentés par des listes sans doublons) et des principales fonctions sur les ensembles. Le code de cette implantation est fourni dans le fichier `ensembles.py` qui contient des fonctions dont la plupart sont paramétrées par une fonction d'égalité sur les élément des ensembles manipulés. Les types prédéfinis (`set`, `frozenset`) permettant de représenter les ensembles en Python n'offrent pas une telle souplesse (type des éléments nécessairement atomique ou bien objets non mutables). Dans tout ce qui suit, seules les étiquettes des transitions des automates ou machines de Turing (c-à-d les symboles de l'alphabet du langage considéré) sont supposées être représentées par des valeurs d'un type atomique.

1 Automates finis

Le code présenté dans cette section se trouve dans le fichier `automates_finis.py`.

1.1 Représentation

En Python, on représente l'automate fini $\mathcal{A} = (S, T, I, F)$ par un quintuplet `A = (S,T,I,F,eqS)` où `S`, `T`, `I` et `F` sont les listes représentant respectivement les ensembles d'états S, de transitions T, des états initiaux I et des états acceptants F et où `eqS` est une fonction permettant de tester l'égalité entre deux états de `S`. On choisit ici de représenter une transtion (s_1, ε, s_2) par le triplet `(s1,None,s2)` : dans une transition, `None` sert donc à indiquer que la transition est une ε-transition.

Exemple 1.1 L'automate fini $\mathcal{A}_0$ de la figure 1 peut être implanté comme suit :

exemple : automate fini $\mathcal{A}_0$ (figure 1)

```
ex_A = ([0,1,2,3,4],\
        [(0,None,2),(0,None,3),(1,"b",1),(1,"a",2),(1,"b",3), \
         (1,"b",4),(3,"a",1),(3,"b",1),(3,None,2),(4,"a",0),(4,None,0)],\
```

```
        [0],[2],eq_atom)
(ex_S,ex_T,ex_I,ex_F,ex_eqS) = ex_A
```

Cet automate sera utilisé dans la suite pour illustrer les fonctions définies : `ex_S`, `ex_T`, `ex_I` et `ex_F` désignent les quatre listes représentant les ensembles S, T, I et F de $\mathcal{A}_0$ et `ex_eqS` désigne l'égalité atomique utilisée pour tester l'égalité entre deux états de cet automate.

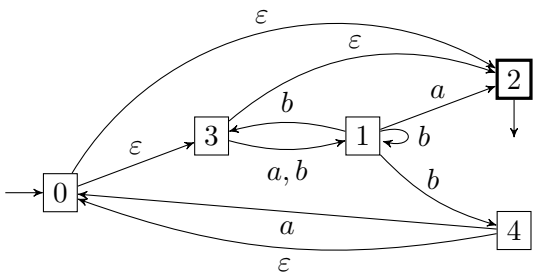


FIGURE 1 – Automate fini $\mathcal{A}_0$

Sur cet exemple, comme sur tous les exemples qui manipulent des automates dont les états sont représentés par des entiers, on utilise la fonction d'égalité `eq_atom` (qui est définie dans le fichier sur les ensembles, à partir de l'égalité atomique `==` de Python) pour tester l'égalité entre états.

1.2 ε-fermeture d'un état

Implantation L'ε-fermeture d'un état s, notée $\mathcal{E}_{\mathcal{A}}(s)$, peut s'obtenir en construisant un ensemble `done` d'états, initialisé à l'ensemble vide $\texttt{done}_0 = \emptyset$, et un ensemble `to_do` d'états, initialisé à $\texttt{to\_do}_0 = \{s\}$. A chaque étape i, l'ensemble $\texttt{done}_i$ est enrichi avec les états de $\texttt{to\_do}_i$ tandis que le nouvel ensemble $\texttt{to\_do}_{i+1}$ est obtenu en considérant les états accessibles avec une ε-transition à partir des états présents dans $\texttt{to\_do}_i$ qui n'apparaissent pas déjà dans $\texttt{done}_i$. Lorsque l'ensemble $\texttt{to\_do}_i$ est vide, la ε-fermeture est l'ensemble $\texttt{done}_i$.

à compléter : ε-fermeture d'un état

```
def eps_cl(eqS,s,T):
  # eqS : fonction d'egalite sur les etats
  # s : etat
  # T : liste de transitions
```

exemples : $\mathcal{E}_{\mathcal{A}_0}(0)$, $\mathcal{E}_{\mathcal{A}_0}(1)$, $\mathcal{E}_{\mathcal{A}_0}(4)$

```
>>> eps_cl(ex_eqS,0,ex_T)
[2, 3, 0]
>>> eps_cl(ex_eqS,1,ex_T)
[1]
>>> eps_cl(ex_eqS,4,ex_T)
[2, 3, 0, 4]
```

Cette fonction s'étend facilement aux ensembles d'états.

à compléter : ε-fermeture d'un ensemble d'états

```
def eps_cl_set(eqS,S,T):
  # eqS : fonction d'egalite sur les etats
  # S : liste d'etats
  # T : liste de transitions
```

exemple : $\mathcal{E}_{\mathcal{A}_0}(\{0,1\})$

```
>>> eps_cl_set(ex_eqS,[0,1],ex_T)
[1, 0, 3, 2]
```

1.3 Etats accessibles, états co-accessibles

Etats accessibles à partir d'un état avec un symbole de l'alphabet L'ensemble $\tau_{\mathcal{A}}(s,\ell)$ des états accessibles à partir d'un état $s \in S$ et d'un symbole de l'alphabet $\ell \in \Sigma$ est l'ensemble des états $s' \in S$ tels que il existe une séquence (de longueur éventuellement nulle) de ε-transitions de s vers un état s_1 et il existe une transition étiquetée par ℓ de s_1 à un état s_2 et il existe une séquence (de longueur éventuellement nulle) de ε-transitions de s_2 vers s'.

$$s \xrightarrow{\varepsilon}{}^{\star}_{T} s_1 \xrightarrow{\ell}_{T} s_2 \xrightarrow{\varepsilon}{}^{\star}_{T} s'$$

Formellement $\tau_{\mathcal{A}}(s,\ell)$ est défini par :

$$\tau_{\mathcal{A}}(s,\ell) = \left\{ \; s' \in S \mid \quad \exists s_1, s_2 \in S \; s_1 \in \mathcal{E}_{\mathcal{A}}(s) \text{ et } (s_1,\ell,s_2) \in T \text{ et } s' \in \mathcal{E}_{\mathcal{A}}(s_2) \; \right\}$$

Exemple $\tau_{\mathcal{A}_0}(1,b) = \{0,1,2,3,4\}$ et $\tau_{\mathcal{A}_0}(4,a) = \{0,1,2,3\}$ pour l'automate $\mathcal{A}_0$ de l'exemple 1.1 (représenté sur la figure 1).

$\tau_{\mathcal{A}}(s,\ell)$

```
def reach_from(eqS,s,x,T):
    s_sl = eps_cl(eqS,s,T)
    trans_x = []
    for (s1,l,s2) in T:
        if l==x and is_in(eqS,s1,s_sl):
            trans_x = ajout(eqS,s2,trans_x)
    return eps_cl_set(eqS,trans_x,T)
```

exemples : $\tau_{\mathcal{A}_0}(1,b)$, $\tau_{\mathcal{A}_0}(4,a)$

```
>>> reach_from(ex_eqS,1,'b',ex_T)
[1, 4, 0, 3, 2]
>>> reach_from(ex_eqS,4,'a',ex_T)
[1, 0, 3, 2]
```

Successeurs d'un état Etant donné un automate fini $\mathcal{A} = (S,T,I,F)$, l'ensemble $\mathrm{succ}_{\mathcal{A}}(s)$ des successeurs de $s \in S$ est l'ensemble des états s' tels qu'il existe une transition de s vers s'.

$$\mathrm{succ}_{\mathcal{A}}(s) = \left\{ s' \in S \mid s \xrightarrow{x}_{T} s' \right\}$$

Exemple $\mathrm{succ}_{\mathcal{A}_0}(3) = \{1,2\}$ et $\mathrm{succ}_{\mathcal{A}_0}(1) = \{1,2,3,4\}$ pour l'automate $\mathcal{A}_0$ de l'exemple 1.1 (représenté sur la figure 1).

$\mathrm{succ}_{\mathcal{A}}(s)$

```
def succ_s(eqS,s,T):
    r = []
    for (s1,_,s2) in T:
        if eqS(s,s1):
            r = ajout(eqS,s2,r)
    return r
```

exemples : $\mathrm{succ}_{\mathcal{A}_0}(3)$, $\mathrm{succ}_{\mathcal{A}_0}(1)$

```
>>> succ_s(eq_atom,3,ex_T)
[2, 1]
>>> succ_s(eq_atom,1,ex_T)
[4, 3, 2, 1]
```

Etats accessibles à partir d'un ensemble d'états Etant donné un automate fini $\mathcal{A} = (S, T, I, F)$, l'ensemble $\tau^\star_\mathcal{A}(E)$ des états accessibles à partir d'un ensemble d'états $E \subseteq S$ est l'ensemble des états $s' \in S$ tels qu'il existe une séquence de transitions (étiquetée par un mot quelconque) depuis un état $s \in E$ jusqu'à l'état s' :

$$\tau^\star_\mathcal{A}(E) = \bigcup_{s \in E} \left\{ s' \in S \mid \exists w \in \Sigma^\star \ s \xrightarrow{w}{}^\star_T \ s' \right\}$$

Exemple $\tau^\star_{\mathcal{A}_0}(\{1,2\}) = \{0,1,2,3,4\}$ et $\tau^\star_{\mathcal{A}_0}(\{2\}) = \{2\}$ pour l'automate $\mathcal{A}_0$ de l'exemple 1.1 (représenté sur la figure 1).

Implantation L'ensemble $\tau^\star_\mathcal{A}(E)$ correspond à un point fixe qui peut être construit incémentalement avec la fonction $\text{succ}_\mathcal{A}$:

$$\tau^n_\mathcal{A}(E) = \begin{cases} E & \text{si } n = 0 \\ \tau^k_\mathcal{A}(E) \cup \bigcup\limits_{s \in \tau^k_\mathcal{A}(E)} \text{succ}_\mathcal{A}(s) & \text{si } n = k+1 \end{cases} \qquad \tau^\star_\mathcal{A}(E) = \bigcup_{n \geq 0} \tau^n_\mathcal{A}(E)$$

Ce procédé termine puisque le nombre d'états d'un automate fini est fini.

$\tau^\star_\mathcal{A}(E)$

```
def reachable(eqS,Es,T):
    _eqES = make_eq_set(eqS)
    def _add_reach(r):
        ar = []
        for qr in r:
            ar = union(eqS,ar,succ_s(eqS,qr,T))
        return union(eqS,r,ar)
    return fixpoint_from(_eqES,_add_reach,Es)
```

exemples : $\tau^\star_{\mathcal{A}_0}(\{1,2\})$, $\tau^\star_{\mathcal{A}_0}(\{2\})$

```
>>> reachable(eq_atom,[1,2],ex_T)
[4, 3, 2, 1, 0]
>>> reachable(eq_atom,[2],ex_T)
[2]
```

Etats accessibles/co-accessibles d'un automate Soit $\mathcal{A} = (S, T, I, F)$ un automate fini :

- l'ensemble des états accessibles de $\mathcal{A}$ est l'ensemble des états qui sont accessibles à partir d'un état initial de I,
- l'ensemble des états co-accessibles de $\mathcal{A}$ est l'ensemble des états à partir desquels au moins un état acceptant de F est accessible.

$$\text{Reach}(\mathcal{A}) = \tau^\star_\mathcal{A}(I) \qquad \text{CoReach}(\mathcal{A}) = \{s \in S \mid \tau^\star_\mathcal{A}(\{s\}) \cap F \neq \emptyset\}$$

à compléter : Reach($\mathcal{A}$)

```
def reach_A(A):
 # A : automate fini
```

à compléter : CoReach($\mathcal{A}$)

```
def co_reach_A(A):
 # A : automate fini
```

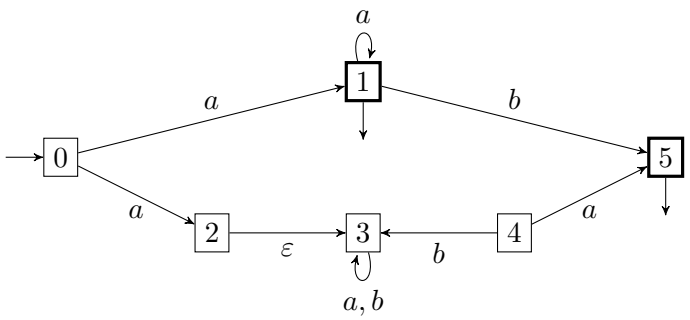


FIGURE 2 – Automate fini $\mathcal{A}_1$

Exemple 1.2 Reach($\mathcal{A}_1$) = $\{0, 1, 2, 3, 5\}$ et CoReach($\mathcal{A}_1$) = $\{0, 1, 4, 5\}$ pour l'automate $\mathcal{A}_1$ de la figure 2.

états exemple 1.2 : Reach($\mathcal{A}_1$), CoReach($\mathcal{A}_1$)

```
ex_RcoR = ([0,1,2,3,4,5],[(0,"a",1), (0,"a",2), (1,"a",1), (1,"b",5),\
                          (2,None,3), (3,"a",3), (3,"b",3), (4,"b",3),\
                          (4,"a",5)],\
           [0],[1,5],eq_atom)
>>> reach_A(ex_RcoR)
[0, 3, 2, 5, 1]
>>> co_reach_A(ex_RcoR)
[0, 1, 4, 5]
```

1.4 Langage reconnu par un automate

Un automate $\mathcal{A} = (S, T, I, F)$ permet de caractériser un ensemble de mots : un mot w est reconnu par un automate lorsqu'il existe un chemin dans l'automate à partir d'un état initial, qui termine sur un état final, et tel que la concaténation des étiquettes rencontrées sur ce chemin correspond exactement au mot w. Un automate peut donc être associé à un langage, le langage des mots qu'il reconnaît. Formellement, le langage $\mathcal{L}(\mathcal{A})$ reconnu par un automate $\mathcal{A} = (S, T, I, F)$ est défini par :

$$\mathcal{L}(\mathcal{A}) = \left\{ w \in \Sigma^\star \mid \exists s_0 \in I \ \exists s_f \in F \ s_0 \xrightarrow{w}{}^\star_T \ s_f \right\}$$

Un *langage reconnaissable* est un langage L pour lequel il existe un automate $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = L$.

Exemple Avec l'automate $\mathcal{A}_0$ de l'exemple 1.1 (représenté sur la figure 1), les mots ε et abb appartiennent au langage reconnu par $\mathcal{A}_0$, alors que les mots a et aab n'appartiennent pas à ce langage.

L'appartenance d'un mot w au langage $\mathcal{L}(\mathcal{A})$ reconnu par un automate fini $\mathcal{A}$ est un problème décidable. Il suffit pour cela de déterminer pour chaque état initial q s'il existe un chemin acceptant à partir de q étiqueté par les symboles de w. Pour chaque symbole $q \in I$, on peut construire une suite d'ensembles $\mathtt{wa}_i$ d'états tel qu'à chaque étape i, $\mathtt{wa}_i$ contient les états accessibles à partir de q par un chemin étiqueté par les i premiers symboles symboles de w. Initialemement $\mathtt{wa}_0 = \{q\}$ et le mot w est accepté par $\mathcal{A}$ si et seulement si il existe un état final dans $\mathtt{wa}_n$ où n est la longueur du mot w ($\mathtt{wa}_n \cap F \neq \emptyset$).

appartenance d'un mot à $\mathcal{L}(\mathcal{A})$

```
def accept_word_finite_aut(A,w):
    (S,T,I,F,eqS)=A
    def _aux(sa,wa):
        if wa==[]:
            return intersection(eqS,eps_cl(eqS,sa,T),F) !=[]
        else:
            trans = reach_from(eqS,sa,wa[0],T)
            for st in trans:
                if _aux(st,wa[1:]):
                    return True
            return False
    for si in I:
        if _aux(si,w):
            return True
    return False
```

exemples : $\varepsilon \in \mathcal{L}(\mathcal{A}_0)$? $abb \in \mathcal{L}(\mathcal{A}_0)$? $aab \in \mathcal{L}(\mathcal{A}_0)$?

```
>>> accept_word_finite_aut(ex_A,[])
True
>>> accept_word_finite_aut(ex_A,['a','b','b'])
True
>>> accept_word_finite_aut(ex_A,['a','a','b'])
False
```

2 Automates finis déterministes

Le code présenté dans cette section se trouve dans le fichier `automates_finis_det.py`.

Un automate $\mathcal{A} = (S, T, I, F)$ est *déterministe* si et seulement si :
— I ne contient qu'un seul état, et
— T ne contient aucune ε-transition, et
— la relation de transition T est fonctionnelle (déterministe) :

$$\forall s_1, s_2, s_3 \in S \ \forall \ell \in \Sigma \ \left(s_1 \xrightarrow{\ell}_T s_2 \text{ et } s_1 \xrightarrow{\ell}_T s_3\right) \Rightarrow s_2 = s_3$$

Exemple L'automate $\mathcal{A}_0$ de l'exemple 1.1 (représenté sur la figure 1) n'est pas déterministe puisque sa relation de transition contient des ε-transitions. De plus à partir d'un même état il existe plusieurs transitions étiquetées par la même symbole de Σ (par exemple il y a deux transitions étiquetées par b à partir de l'état 1).

2.1 Test du déterminisme d'un automate fini

Pour déterminer si un automate est déterministe, on définit la fonction `lt_from_s_deterministic` qui permet de tester si un ensemble de transitions issues d'un même état représente une relation fonctionnelle et ne contenant aucune ε-transition. Pour cela on définit les fonctions suivantes.

Transitions issues d'un état Etant donné un état $s \in S$, on note $T_{|s} \subseteq T$ l'ensemble des transitions de $\mathcal{A} = (S, T, I, F)$ ayant pour origine l'état s :

$$T_{|s} = \left\{ s_1 \xrightarrow{\ell}_T s_2 \mid s_1 = s \right\}$$

$T_{|s}$

```
def lt_from_s(eqS,s,T):
    return [(s1,l,s2) for (s1,l,s2) in T if eqS(s1,s)]
```

exemple : $T_{|3}$ pour l'automate $\mathcal{A}_0$

```
>>> lt_from_s(eqS,3,ex_T)
[(3, 'a', 1), (3, 'b', 1), (3, None, 2)]
```

Etiquettes présentes sur les transitions issues d'un état L'ensemble des étiquettes présentes sur les transitions de T issues d'un état s est défini par :

$$\left\{\ell \mid \exists\,(s,\ell,s') \in T \text{ et } \ell \neq \varepsilon\right\}$$

étiquettes sur les transitions issues d'un état

```
def label_from(eqS,s,T):
    R = []
    for (si,l,sf) in T:
        if eqS(si,s) and l != None:
            R = ajout(eq_atom,l,R)
    return R
```

exemples : étiquettes sur les transitions de l'automate $\mathcal{A}_0$ issues des états 0, 3 et 4

```
>>> label_from(ex_eqS,0,ex_T)
[]
>>> label_from(ex_eqS,3,ex_T)
['b', 'a']
>>> label_from(ex_eqS,4,ex_T)
['a']
```

On étend cette fonction pour obtenir l'ensemble des étiquettes de Σ présentes sur les transitions issues d'un ensemble E d'états de l'automate :

$$\bigcup_{s\in E} \left\{\ell \mid \exists\,(s,\ell,s') \in T \text{ et } \ell \neq \varepsilon\right\}$$

étiquettes sur les transitions issues d'un ensemble d'états

```
def label_from_set(eqS,S,T):
    R = []
    for s in S:
        R = union(eq_atom,label_from(eqS,s,T),R)
    return R
```

exemple : étiquettes sur les transitions de l'automate $\mathcal{A}_0$ issues de l'ensemble d'états $\{0,4\}$

```
>>> label_from_set(ex_eqS,[0,4],ex_T)
['a']
```

Test du déterminisme d'un automate La fonction `is_deterministic` qui teste si un automate est déterministe peut être simplement définie en testant le cardinal de I et en vérifiant que pour chaque état $s \in S$ l'ensemble des transitions issues de s est déterministe.

La fonction `lt_from_s_deterministic` permet de tester qu'un ensemble de transitions T ne contient pas plus d'une transition pour une même étiquette.

test du déterminisme des transitions issues d'un état

```
def lt_from_s_deterministic(T):
    def _aux(L):
        if len(L)==0:
            return True
        else:
            if L[0] == None or L[0] in L[1:]:
                return False
            else:
                return _aux(L[1:])
    return _aux([l for (_,l,_) in T])
```

exemples : déterminisme des transitions de $\mathcal{A}_0$ issues de l'état 3

```
>>> lt_from_s_deterministic(lt_from_s(ex_eqS,3,ex_T))
False
>>> lt_from_s_deterministic([(3, 'a', 1), (3, 'b', 1)])
True
```

à compléter : test du déterminisme d'un automate

```
def is_deterministic(A):
 # A : automate fini
```

exemple : test du déterminisme de l'automate $\mathcal{A}_0$

```
>>> is_deterministic(ex_A)
False
```

2.2 Déterminisation

Implantation La construction de la fonction de transition T' de l'automate $\det(\mathcal{A})$ peut s'obtenir incrémentalement (à partir de la valeur initiale $T'_0 = \emptyset$) en construisant un ensemble `done` d'états, initialisé à l'ensemble vide $\texttt{done}_0 = \emptyset$, et un ensemble `to_do` d'états, initialisé à $\texttt{to\_do}_0 = \{\mathcal{E}_{\mathcal{A}}(I)\}$. A chaque étape i, l'ensemble $\texttt{done}_i$ est enrichi avec un état S_i de $\texttt{to\_do}_i$ tandis que :

- le nouvel ensemble T'_{i+1} est obtenu en ajoutant à T'_i toutes les transitions issues de S_i
- le nouvel ensemble $\texttt{to\_do}_{i+1}$ est obtenu en supprimant S_i de $\texttt{to\_do}_i$ et en y ajoutant tous les états rencontrés dans les nouvelles transitions issues de S_i qui n'apparaissent pas déjà dans $\texttt{done}_i$

La construction est terminée (c-à-d $T' = T'_i$) lorsque l'ensemble $\texttt{to\_do}_i$ est vide. L'ensemble des états accessibles de $\det(\mathcal{A})$ est alors l'ensemble $\texttt{done}_i$.

L'ajout d'éléments dans l'ensemble `done` représenté par une liste sans doublon nécessite de pouvoir tester si un élément appartient à cet ensemble. Ici, les éléments ajoutés sont des ensembles d'états de $\mathcal{A}$, et on utilise donc la fonction d'égalité `make_eq_set(eqS)` (où `eqS` est la fonction d'égalité sur les états de $\mathcal{A}$), définie dans l'annexe sur la représentation des ensembles, pour tester l'égalité de deux ensembles d'états. On procède exactement de la même manière pour tester l'égalité de deux transitions de T' (ce qui est nécessaire pour ajouter des transitions de la forme (S_1, ℓ, S_2) à un ensemble de transitions). Si `eqS` est la fonction d'égalité entre les états sur lesquels porte la relation de transition, alors `make_eq_trans(eqS)` permet de tester l'égalité entre deux transitions (on rappelle que l'on suppose ici que l'égalité des étiquettes peut se tester avec l'égalité atomique de Python).

égalité sur les transitions

```
def eq_trans(eqS,t1,t2):
    (si1,l1,sf1) = t1
    (si2,l2,sf2) = t2
    return l1==l2 and eqS(si1,si2) and eqS(sf1,sf2)

def make_eq_trans(eqS):
    def _eq_trans(t1,t2):
        return eq_trans(eqS,t1,t2)
    return _eq_trans
```

exemple : égalité sur les transitions lorsque les états sont des ensembles d'entiers

```
>>> eq_set_atom = make_eq_set(eq_atom)
>>> eq_trans_atom = make_eq_trans(eq_set_atom)
>>> eq_trans_atom(([1,2,3],'a',[4,2]),([2,1,3],'a',[2,4]))
True
```

Puisque les états sur lesquels porte la relation de transition sont ici des ensembles d'états de $\mathcal{A}$, on pourra tester l'égalité de deux transitions de T' avec la fonction `make_eq_trans(make_eq_set(eqS))` où `eqS` est la fonction d'égalité sur les états de $\mathcal{A}$.

La fonction qui permet de construire l'automate $\det(\mathcal{A})$ peut maintenant s'écrire simplement.

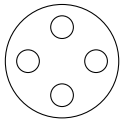
à compléter : déterminisation d'un automate

```
def make_det(A):
 # A : automate fini
```

exemple : $\det(\mathcal{A}_0)$

```
>>> make_det(ex_A)
(
[[2, 1], [1, 0, 3, 2], [1, 4, 0, 3, 2], [2], [1], [0, 3, 2]],
[([2, 1], 'a', [2]),
 ([2, 1], 'b', [1, 4, 0, 3, 2]),
 ([1, 0, 3, 2], 'a', [2, 1]),
 ([1, 0, 3, 2], 'b', [1, 4, 0, 3, 2]),
 ([1, 4, 0, 3, 2], 'a', [1, 0, 3, 2]),
 ([1, 4, 0, 3, 2], 'b', [1, 4, 0, 3, 2]),
 ([1], 'a', [2]), ([1], 'b', [1, 4, 0, 3, 2]),
 ([0, 3, 2], 'b', [1]),
 ([0, 3, 2], 'a', [1])],
[[0, 3, 2]],
[[2, 1], [1, 0, 3, 2], [1, 4, 0, 3, 2], [2], [0, 3, 2]],<...fct...> )
```

Exercice 2.1 (– Facultatif – Problème du barman aveugle) Un barman aveugle manipule avec des gants de boxe un plateau rond sur lequel se trouvent 4 verres qui peuvent être à l'endroit (sur leur pied) ou à l'envers.

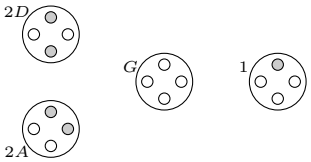


A chaque tour le barman choisit d'effectuer un des trois mouvements :

- r : il retourne un des 4 verres
- d : il retourne 2 verres en diagonale
- a : il retourne 2 verres adjacents

Etant donné un état du plateau, on cherche à déterminer comment doit procéder le barman pour que les 4 verres soient dans le même sens.

1. Représenter ce problème par un automate fini non déterministe à 4 états sur l'alphabet $\{r, d, a\}$:



2. Déterminiser cet automate en choisissant $\{G\}$ comme état final et $I = \{2D, 2A, 1\}$ comme états initiaux.
3. En déduire comment doit procéder le barman pour que les 4 verres soient dans le même sens

à compléter : problème du barman aveugle

```
ex_BA = (["2D","2A","G","1"],\
        [("2D","R","1"),("2D","D","G"),... A COMPLETER],\
        ["2D","2A","1"],["G"],eq_atom)
>>> make_det(ex_BA)
```