Institut Galilée

Rapport système complexe

10 janvier 2021

David HONG

Table des matières

Introduction

On va implémenter les algorithmes de model checking vu en cours des formules CTL. Il existe 6 cas.

1. $\phi = p$
2. $\phi = \neg\psi$
3. $\phi = \psi 1 \wedge \psi 2$
4. $\phi = EX\psi$
5. $\phi = E\ \psi 1\ U\ \psi 2$
6. $\phi = A\ \psi 1\ U\ \psi 2$

Voici tous les fichiers :

— Un rapport expliquant les structures de données utilisées, choix d'implémentations, perspectives d'amélioration et application sur quelques exemples.
— Un fichier `README.MD` contenant les instructions.
— Un dossier `src` contenant les codes en python.
— Un dossier `images` contenant des captures d'écran.
— Un fichier `in.txt` contenant la description textuelle d'une structure de Kripke et d'une formule CTL. Et aussi un fichier `in.txt.bak` pourla structure de Kripke et la formule CTL par défaut.

Structure de Kripke

Une structure de Kripke peut être représentée par
- Un ensemble d'états.
- Un ensemble de transitions.

On a plusieurs méthodes pour les structures de Kripke :
- `get_etats` elle renvoie l'ensemble des états.
- `get_transitions` elle renvoie l'ensemble des transitions.
- `add_etat` elle ajoute un état à la structure de Kripke.
- `add_transition` elle ajoute une transition à la structure de Kripke.
- `nb_etats` elle renvoie le nombre d'états de la structure de Kripke.
- `nb_transitions` elle renvoie le nombre de transitions de la structure de Kripke.
- `etat_in` test si un état est dans la structure de Kripke.
- `transition_in` test si une transition est dans la structure de Kripke.
- `print_transition` affiche la structure de Kripke en affichant les états et les transitions.

Il y a aussi les méthodes pour le model checker.

Nous allons donc faire deux autres structures une qui représente les états et une autre les transitions.

Etat

Un état est un couple qui contient un entier et un ensemble de propositions. On suppose que l'entier est positif ou nul.

Exemple : `Etat(0, ['req1'])` est un état où `req1` est vrai dans cet état.

On a plusieurs méthodes pour les états :
- `get_etat` elle renvoie l'état.
- `get_props` elle renvoie les propositions qui sont vérifiées.
- `print_etat` elle affiche l'état.

Transition

Une transition est un couple d'entier source et destination. On suppose que les entiers sont positifs ou nuls.

Exemple : `Transition(0, 1)` est une transition qui part de 0 jusqu'à 1.

On a plusieurs méthodes pour les transitions :
- `get_source` elle renvoie l'état source de la transition.
- `get_destination` elle renvoie l'état destination de la transition.
- `print_transition` elle affiche la transition.

Le model checker

Il faut implémenter les différents cas. Il suffit de suivre les algorithmes vue en cours pour l'implémentation.

Cas 1 : $\phi = p$

```python
def check_prop(self, prop):
    """
    Cas 1
    @param prop: la proposition à vérifier.
    @return res: un dict (etat: bool)
    """
    res = {}
    # Pour tous les états
    for e in self.etats:
        # prop est dans la liste des propositions atomiques donc True
        if prop in e.get_props():
            res[e.get_etat()] = True
        # prop n'est pas dans la liste des propositions atomiques donc
        ↪ False
        else:
            res[e.get_etat()] = False
    return res
```

Cas 2 : $\phi = \neg\psi$

```python
def check_not(self, prop):
    """
    Cas 2
    @param prop: la proposition à vérifier ou un dict
    @return res: un dict (etat: bool)
    """
    res = {}
    # prop est une proposition
    if isinstance(prop, str):
        # On transforme prop en un dict (marking(phi))
        prop = self.check_prop(prop)
    # On parcours tous les propositions de prop
    for i, j in prop.items():
        # On a un True donc on aura False
        if j == True:
            res[i] = False
        # On a False donc on aura True
        else:
            res[i] = True
    return res
```

Cas 3 : $\phi = \psi_1 \wedge \psi_2$

```python
def check_and(self, prop1, prop2):
    """
    Cas 3
    @param prop1: la proposition à vérifier ou un dict.
    @param prop2: la proposition à vérifier ou un dict.
    @return res: un dict (etat: bool)
    """
    res = {}
    # prop est un string
    if isinstance(prop1, str) and isinstance(prop2, str):
        mark_prop1 = self.check_prop(prop1)
        mark_prop2 = self.check_prop(prop2)
        for i in self.etats:
```

```
14              res[i.get_etat()] = mark_prop1[i.get_etat()] and
                ↪  mark_prop2[i.get_etat()]
15      # prop est un dict
16      else:
17          for i, j in prop1.items():
18              res[i] = prop1[i] and prop2[i]
19      return res
```

Cas 4 : $\phi = EX\psi$

```
1   def check_next(self, prop):
2       """
3       Cas 4
4       @param prop: la proposition à vérifier ou un dict.
5       @return res: un dict (etat: bool)
6       """
7       res = {}
8       # On initialise tous à False
9       for i in self.etats:
10          res[i.get_etat()] = False
11      # prop est une proposition
12      if isinstance(prop, str):
13          # On transforme prop en un dict (marking(phi))
14          prop = self.check_prop(prop)
15      # On parcours les transitions
16      for t in self.transitions:
17          # On a trouvé une transition qui à l'état destination à vrai
18          if prop[t.get_destination()] == True:
19              res[t.get_source()] = True
20      return res
```

Cas 5 : $\phi = E\psi_1 U \psi_2$

```
1   def check_euntil(self, prop1, prop2):
2       """
```

```
        Cas 5
        @param prop1: la proposition à vérifier ou un dict.
        @param prop2: la proposition à vérifier ou un dict.
        @return res: un dict (etat: bool)
        """
        res = {}
        L = []
        seenbefore = {}
        # Initialise tout à False et seenbefore à False
        for q in self.etats:
            res[q.get_etat()] = False
            seenbefore[q.get_etat()] = False
        # prop1 est une proposition
        if isinstance(prop1, str):
            # On transforme prop1 en un dict (marking(phi1))
            prop1 = self.check_prop(prop1)
        # prop2 est une proposition
        if isinstance(prop2, str):
            # On transforme prop2 en un dict (marking(phi2))
            prop2 = self.check_prop(prop2)
        # On parcours tous les états
        for q in self.etats:
            # L'état q est vrai dans prop2 (phi2)
            if prop2[q.get_etat()] == True:
                # On ajout l'état q dans la liste L
                L.append(q.get_etat())
        # On parcours la liste l tant qu'elle n'est pas vide
        while len(L) != 0:
            # On prend un état q dans la liste L
            for q in L:
                # On met à True
                res[q] = True
                # On parcours tous les transitions
                for t in self.transitions:
                    # On a trouve une transition entrant vers q
                    if t.get_destination() == q:
                        if seenbefore[t.get_source()] == False:
                            seenbefore[t.get_source()] = True
                            if prop1[t.get_source()] == True:
                                L.append(t.get_source())
```

```
43          # On enlève q de la liste L
44          L.pop(-1)
45      return res
```

Cas 6 : $\phi = A\psi_1 U \psi_2$

```
1   def check_auntil(self, prop1, prop2):
2       """
3       Cas 6
4       @param prop1: la proposition à vérifier ou un dict.
5       @param prop2: la proposition à vérifier ou un dict.
6       @return res: un dict (etat: bool)
7       """
8       res = {}
9       L = []
10      degree = {}
11      # Initialise tout à False et les degrés de tous les états
12      for q in self.etats:
13          res[q.get_etat()] = False
14          degree[q.get_etat()] = self.get_degree(q.get_etat())
15      # prop1 est une proposition
16      if isinstance(prop1, str):
17          # On transforme prop1 en un dict (marking(phi1))
18          prop1 = self.check_prop(prop1)
19      # prop2 est une proposition
20      if isinstance(prop2, str):
21          # On transforme prop2 en un dict (marking(phi2))
22          prop2 = self.check_prop(prop2)
23      # On parcours tous les états
24      for q in self.etats:
25          # L'état q est vrai dans prop2 (phi2)
26          if prop2[q.get_etat()] == True:
27              # On ajout l'état q dans la liste L
28              L.append(q.get_etat())
29      # On parcours la liste l tant qu'elle n'est pas vide
30      while len(L) != 0:
31          # On prend un état q dans la liste L
32          for q in L:
```

```
33              # On met à True
34              res[q] = True
35              # On parcours tous les transitions
36              for t in self.transitions:
37                  # On a trouve une transition entrant vers q
38                  if t.get_destination() == q:
39                      # On décrémente son dégré
40                      degree[t.get_source()] -= 1
41                      if degree[t.get_source()] == 0 and
                        ↪  prop1[t.get_source()] == True and
                        ↪  res[t.get_source()] == False:
42                          L.append(t.get_source())
43              # On enlève q de la liste L
44              L.pop(0)
45      return res
46
47  def get_degree(self, n):
48      """
49      Calcul le degrée d'un état
50      @param n: un entier correspondant à l'état
51      @return int: le degré sortant
52      """
53      res = 0
54      if isinstance(n, int):
55          for i in self.transitions:
56              if i.get_source() == n:
57                  res += 1
58      else:
59          raise TypeError("L'argument n'est pas un entier !")
60      return res
```

Le fichier main

C'est là que nous allons tester la structure de Kripke entré dans le fichier `in.txt`

Dans un terminal, tapez `python3 main`

Dans le main, il y plusieurs fonction :

- `KS_bool` qui construit un dictionnaire d'entier, booléen.
- `charger_kripke` qui va lire le fichier texte et initialiser la structure de Kripke.
- `charger_formule` qui va lire la dernière ligne du fichier et initialiser la formule CTL à tester.
- `get_etats_verifie` qui va renvoyer un ensemble d'états qui sont vrais.

Utilisation

Créer sa propre stucture de Kripke

Pour créer sa propre structure de Kripke, il faut modifier le fichier `in.txt`, ce fichier doit contenir les etats et les transitions.

Il faut ajouter les états en premiers car lorsque l'on ajoute une transition, elle requiert les états source et destination, or si notre structure de Kripke ne possède pas d'état, on aura une exception.

Sur chaque ligne, on va commencer par mettre une lettre qui indique si la ligne est un état ou une transition (`e` pour un état, `t` pour une transition, `f` pour la formule).

La dernière ligne du fichier sera la formule CTL.

Pour les états, on va mettre un entier qui sera le numéro de l'état, puis une suite de propositions atomiques qui sont vérifiés dans cet état.

```
1 e 1 p q
2 e 2 p
3 e 3 q
```

Pour les transitions, on va mettre deux entiers, le premier pour l'état source, et la deuxième pour l'état destination.

```
t 1 2
t 2 3
t 3 3
```

Pour la formule, on mettra la séquence d'exécution de notre model checker. On utilise les mots-clés qui sont :

- `True`
- `False`
- `not` pour le non logique
- `and` pour le et logique
- `next` pour le next
- `euntil` pour le exist until
- `auntil` pour le always until

```
f and p q
```

Cette formule correspond à $p \wedge q$

Lancer le `main.py` et notre structure de Kripke est crée et notre model checker va vérifier la formule CTL dans la structure de Kripke crée.

Voici quelques exemples :

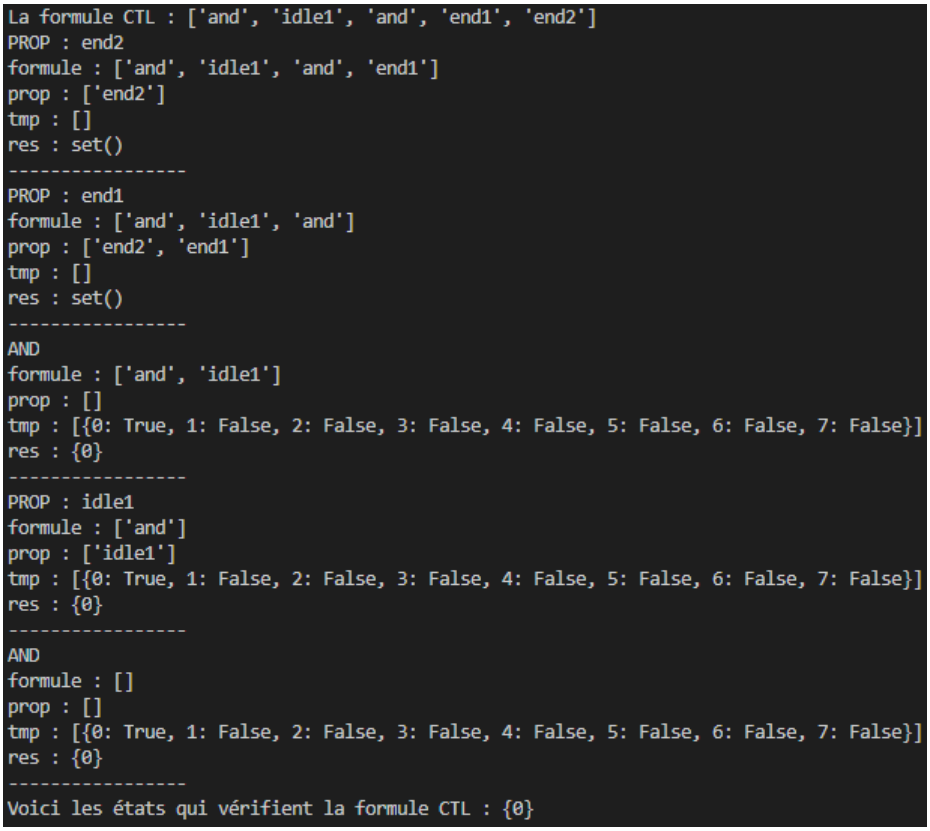

FIGURE 1 – Formule $idle_1 \wedge (end_1 \wedge end_2)$ (f and idle1 and end1 end2)

```
La formule CTL : ['and', 'False', 'not', 'True']
BOOL : True
formule : ['and', 'False', 'not']
prop : [{0: True, 1: True, 2: True, 3: True, 4: True, 5: True, 6: True, 7: True}]
tmp : []
res : set()
----------------
NOT
formule : ['and', 'False']
prop : []
tmp : [{0: False, 1: False, 2: False, 3: False, 4: False, 5: False, 6: False, 7: False}]
res : set()
----------------
BOOL : False
formule : ['and']
prop : [{0: False, 1: False, 2: False, 3: False, 4: False, 5: False, 6: False, 7: False}]
tmp : [{0: False, 1: False, 2: False, 3: False, 4: False, 5: False, 6: False, 7: False}]
res : set()
----------------
AND
formule : []
prop : []
tmp : [{0: False, 1: False, 2: False, 3: False, 4: False, 5: False, 6: False, 7: False}]
res : set()
----------------
Voici les états qui vérifient la formule CTL : set()
```

FIGURE 2 – Formule $False \wedge (\neg True)$ (f and False not True)

Exemples

Nous allons tester notre model checker sur quelques exemples vue en cours. Dans le fichier `test.py` il y a la structure de Kripke de l'exclusion mutuelle (la structure de Kripke dans cet exemple à été crée directement dans le fichier) voici son contenu :

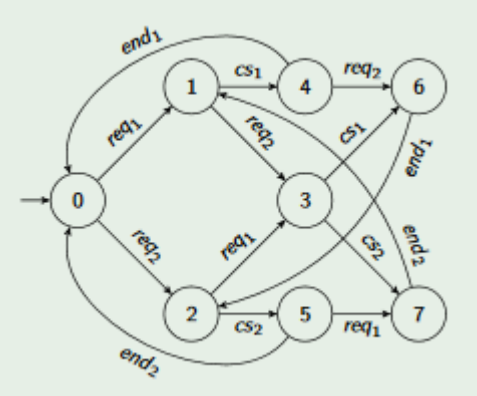

FIGURE 3 – Exemple de l'exclusion mutuelle

```python
# On créer une structure de Kripke de l'exclusion mutuelle
K = Kripke()
# On ajoute les états
K.add_etat(Etat(0, ["end1", "end2", "idle1", "idle2"]))
K.add_etat(Etat(1, ["req1", "end2", "idle2"]))
K.add_etat(Etat(2, ["req2", "end1", "idle1"]))
K.add_etat(Etat(3, ["req1", "req2"]))
K.add_etat(Etat(4, ["cs1", "idle2"]))
K.add_etat(Etat(5, ["cs2", "idle1"]))
K.add_etat(Etat(6, ["req2", "cs1"]))
K.add_etat(Etat(7, ["req1", "cs2"]))
# On ajoute les transitions
K.add_transition(Transition(0, 1))
K.add_transition(Transition(0, 2))
K.add_transition(Transition(1, 3))
K.add_transition(Transition(1, 4))
K.add_transition(Transition(2, 3))
K.add_transition(Transition(2, 5))
K.add_transition(Transition(3, 6))
K.add_transition(Transition(3, 7))
K.add_transition(Transition(4, 0))
K.add_transition(Transition(4, 6))
K.add_transition(Transition(5, 0))
K.add_transition(Transition(5, 7))
K.add_transition(Transition(6, 2))
K.add_transition(Transition(7, 1))

# Cas 1
checker_prop = K.check_prop("req1")
# Cas 2
checker_not = K.check_not("req1")
# Cas 3
checker_and = K.check_and("req1", "req2")
# Cas 4
checker_next = K.check_next("req1")
# Cas 5
checker_euntil = K.check_euntil("req1", "cs1")
# Cas 6
checker_auntil = K.check_auntil("req1", "cs1")
# Exemple sur une formule CTL
```

```
41 checker_formule1 = K.check_not(K.check_euntil(KS_true(K.nb_etats()),
   ↪ K.check_not(K.check_euntil(KS_true(K.nb_etats()),
   ↪ K.check_and(K.check_prop("idle1"), K.check_prop("idle2"))))))
42 checker_formule2 = K.check_not(K.check_and(K.check_prop("end1"),
   ↪ K.check_prop("end2")))
43 checker_formule3 = K.check_and(K.check_and(K.check_prop("req1"),
   ↪ K.check_prop("req2")), K.check_prop("req2"))
44
45 print("Cas 1 : {}".format(get_etats_verifie(checker_prop)))
46 print("Cas 2 : {}".format(get_etats_verifie(checker_not)))
47 print("Cas 3 : {}".format(get_etats_verifie(checker_and)))
48 print("Cas 4 : {}".format(get_etats_verifie(checker_next)))
49 print("Cas 5 : {}".format(get_etats_verifie(checker_euntil)))
50 print("Cas 6 : {}".format(get_etats_verifie(checker_auntil)))
51 print("Exemple sur la formule 1 CTL :
   ↪ {}".format(get_etats_verifie(checker_formule1)))
52 print("Exemple sur la formule 2 CTL :
   ↪ {}".format(get_etats_verifie(checker_formule2)))
53 print("Exemple sur la formule 3 CTL :
   ↪ {}".format(get_etats_verifie(checker_formule3)))
```

Nous allons tester quelques formules sur tous les cas.

1. $\phi = req_1$
2. $\phi = \neg req_1$
3. $\phi = req_1 \wedge req_2$
4. $\phi = EX\ req_1$
5. $\phi = E\ req_1\ U\ cs_1$
6. $\phi = A\ req_1\ U\ cs_1$
7. $\phi = \neg(E\ true\ U\ \neg(E(true\ U\ (idle_1 \wedge idle_2))))$
8. $\phi = \neg(end_1 \wedge end_2)$
9. $\phi = (req_1 \wedge req_2) \wedge req_1$

On peux tester ces formules dans le fichier `in.txt`

1. `f req1`
2. `f not req1`
3. `f and req1 req2`

4. `f next req1`
5. `f euntil req1 cs1`
6. `f auntil req1 cs1`
7. `f not euntil True not euntil True and idle1 idle2`
8. `f not and end1 end2`
9. `f and and req1 req2 req1`

Pour tester ces cas, il suffit tapez dans un terminal `python3 test.py`

Voici les résultats :

```
Cas 1 : {1, 3, 7}
Cas 2 : {0, 2, 4, 5, 6}
Cas 3 : {3}
Cas 4 : {0, 1, 2, 3, 5, 7}
Cas 5 : {1, 3, 4, 6, 7}
Cas 6 : {4, 6}
Exemple sur la formule 1 CTL : {0, 1, 2, 3, 4, 5, 6, 7}
Exemple sur la formule 2 CTL : {1, 2, 3, 4, 5, 6, 7}
Exemple sur la formule 3 CTL : {3}
```

FIGURE 4 – Résultats des différents cas

Les 6 cas on été vu en cours, voici les résultats :

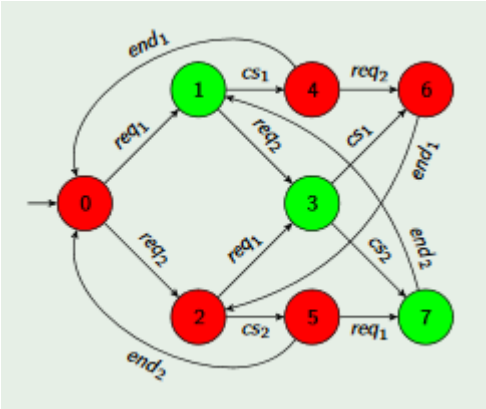

FIGURE 5 – Cas 1

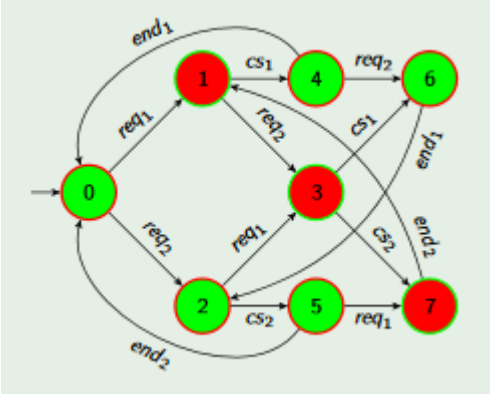

FIGURE 6 – Cas 2

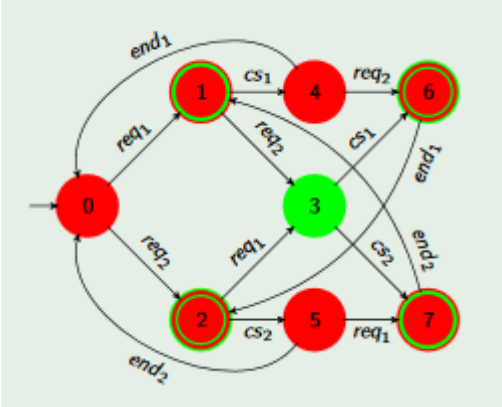

FIGURE 7 – Cas 3

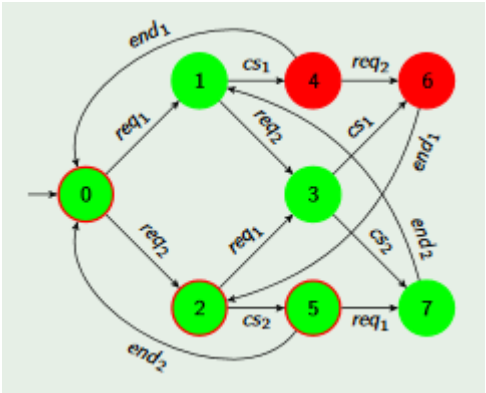

FIGURE 8 – Cas 4

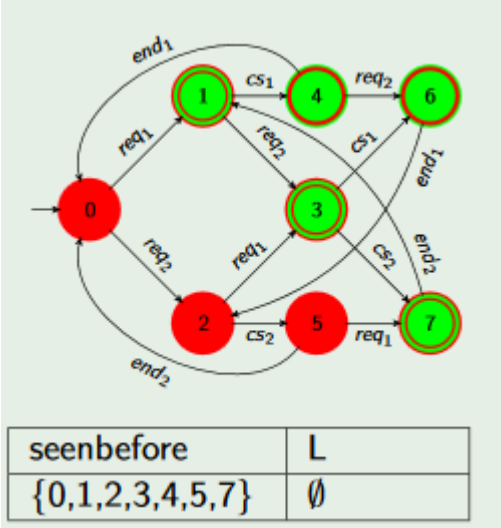

FIGURE 9 – Cas 5

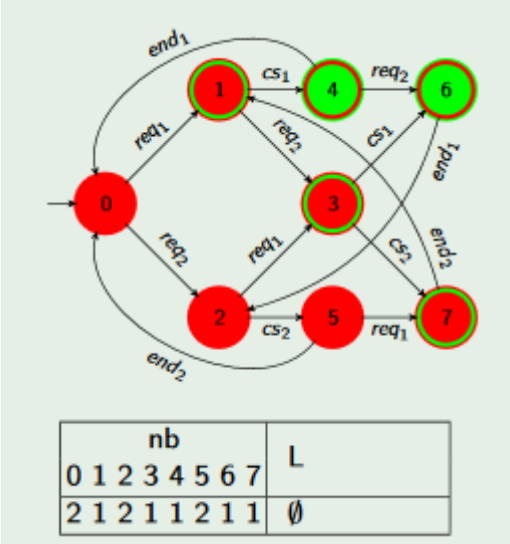

FIGURE 10 – Cas 6

Perspectives d'amélioration

— Vérifier que le fichier `in.txt` est correcte.
— Vérifier que la formule CTL entré est syntaxiquement correcte (`f not` n'est pas valide)
—

— La conversion automatique des formules CTL (transformation des $\Rightarrow$, $\vee$, etc.)
— Faire un model checker sur les formules LTL
— Essayer de faire ce projet avec des réseaux de pétri
— Améliorer du code
— Afficher la structure de Kripke avec un graphe