Université PARIS-SACLAY
Université d'Evry Val d'Essonne
IBGBI

Minimisation de systèmes de transitions par rapport à la bisimulation

22 septembre 2022

Encadrantes :

Serenella Cerrito
Sophie Paillocher

Soutenu par :

Moussa Adamou Idrissa, 20215530@etud.univ-evry.fr
Soro Yaya Sopegue, 20216016@etud.univ-evry.fr
Sidibe Abdoulaye, 20215834@etud.univ-evry.fr

Table des matières

Table des figures

1 INTRODUCTION

Les systèmes informatiques se démocratisent et trouvent des domaines d'application de plus en plus variés. Leur coût toujours plus réduit et leur miniaturisation permettent de les intégrer dans un nombre croissant d'appareils de la vie courante. Parmi ceux-ci, nous pouvons citer les cartes à puce à microprocesseur intégré, évolution des cartes à piste, qui sont de véritables ordinateurs portatifs. Celles-ci sont notamment utilisées dans le domaine bancaire comme cartes de paiement et dans la téléphonie mobile comme certificats d'identification du compte à débiter (cartes SIM des téléphones portables).

Les algorithmes de minimisation sont utilisés pour simplifier un problème ou optimiser un système, ou, encore, simplifier la tâche de la vérification automatique des propriétés souhaitées - dite aussi *model checking* - en réduisant la taille de l'espace des états d'un modèle abstrait du système. Tous ces algorithmes de minimisation sont basés sur le même principe : fusionner des états qu'on peut considérer équivalents. Dans notre cas, la minimisation se basera sur l'idée intuitive de la bisimulation entre deux systèmes de transition.

Ainsi, notre projet qui s'intitule : "Minimisation de systèmes de transition par rapport à la bisimulation" s'inscrit dans cette optique.

Objectif du Travail d'étude et de recherche

L'objectif final de ce TER est d'étudier la minimisation, par rapport à la bisimulation, d'un système de transitions, ayant des étiquettes aux états, vu comme constituant une interprétation de la logique temporelle CTL(voir définition 2.4) ou comme décrivant un ensemble d'interprétations de de la logique temporelle LTL qui partagent le même état de départ (voir [1] pour une description complète des logiques CTL et LTL).

Pour atteindre ce but, on nous a demandé de :

1. Tout d'abord, étudier et implémenter l'algorithme de minimisation par rapport à la bisimulation des systèmes de transitions qui n'ont pas d'étiquettes aux états mais sur les arêtes proposées par Kannelakis et Smolka. Cet algorithme est décrit dans [2].
2. Ensuite réfléchir à comment adapter cet algorithme afin de réaliser l'objectif de la minimisation d'une interprétation de CTL.

2 Préliminaires

2.1 Systèmes de transitions avec étiquettes sur les arêtes

Definition 2.1. Un système de transitions avec des étiquettes d'actions sur les arêtes est un triplet $\langle S, Act, \rightarrow \rangle$ tel que :
– S est un ensemble d'états (pas forcément fini)
– Act est un ensemble de symboles dits *actions*
– $\rightarrow$ est une relation de *transition* : $\rightarrow \subseteq S \times Act \times S$

Intuitivement, on passe d'un état s à un état s' du système, ce qu'on note $s \xrightarrow{a} s'$, grâce à l'action a, pour $s, s' \in S$ et $a \in Act$

2.2 Système de transitions avec étiquettes aux états

Soit P un ensemble de variables booléennes

Definition 2.2. Un système de transitions avec des étiquettes d'actions aux états est un triplet $\langle S, R, L \rangle$ tel que :
– S est un ensemble non vide d'états (pas forcement fini)
– $R \subseteq S \times S$ est une relation sur S, qui associe à un sommets $s \in S$ ses successeurs.
– L: $S \rightarrow 2^P$ est une fonction qui étiquette chaque sommet s avec un ensemble de variables booléennes (les variables déclarées vraies à s).

Ce type de structure est aussi connue comme structure de Kripke[3].

De facto, dans notre travail nous alons considérer toujours des systèmes où l'ensemble S des états est fini.

2.3 Grammaire des formules CTL

Soit P un ensemble de variables booléennes.

Definition 2.3. La grammaire des formules (F) de CTL est :
$F :: = p \mid True \mid (\neg F) \mid (F \vee F) \mid (F \wedge F) \mid E(F \cup F) \mid A(F \cup F) \mid E\,op\,F \mid A\,op\,F$
où $p \in P$ et $op \in \{\bigcirc, \Box, \Diamond\}$.
Tous $p \in P$ et $True$ sont les formules atomiques.
E et A : quantificateurs de chemin.
Lecture intuitive :
Si F est une formule, exprimant une propriété temporelle d'un chemin :

- $\bigcirc(F)$: F est vraie à l'instant suivant.
- $\Box(F)$: F est vraie maintenant et toujours dans le future.
- $\Diamond(F)$: F est vraie maintenant ou bien à quelque instant du future.
- $F_1 \cup F_2$: F_1 until F_2.
- $E\,F$: il existe au moins un chemin (exécution) qui rend vraie la propriété F.
- $A\,F$: Quelque soit le chemin (exécution), il rend vraie la propriété F.

2.4 Interprétation de CTL

Definition 2.4. Une interprétation de CTL est un système de transitions sans étiquettes aux arcs, mais où chaque état a comme étiquette un ensemble de variables booléennes vraies à cet état.C'est donc le même objet mathématique déjà donné dans la définition 2.2 ci-dessus.

De plus, on suppose que chaque état est le point de départ d'au moins une transition, c'est à dire que la relation R est totale

En outre, avec CTL, une interprétation est un graphe qui, déplié, donne lieu à une forêt d'arbres ayant comme racines les états initiaux, où chaque branche est de profondeur infinie. Chaque branche représente une éxecution possible du sysème modélisé.

2.5 Relation de bisimulation entre deux systèmes de transitions ayant des étiquettes exclusivement sur les arêtes

Definition 2.5. Etant donné deux système de transition avec des étiquettes sur les arêtes(définition 2.1) $\langle Sa, Act, \rightarrow\rangle$ et $\langle Sb, Act, \rightarrow\rangle$, une relation de *bisimulation* [1] sur leurs états est un ensemble de couples $\beta \subseteq Sa \times Sb$ tel que, pour tout $s_1 \in Sa$ et $s_2 \in Sb$, si $s_1 \beta s_2$ alors quelque soit $a \in Act$:
– si, pour quelque $s' \in Sa$ on a $s_1 \xrightarrow{a} s'$ alors il existe un $s" \in Sb$ tel que $s' \beta s"$ et $s_2 \xrightarrow{a} s"$
– Réciproquement, si, pour quelque $s" \in Sb$ on a $s_2 \xrightarrow{a} s"$ alors il existe un $s' \in Sa$ tel que $s' \beta s"$ et $s_1 \xrightarrow{a} s'$.

Intuitivement, deux états s_1 et s_2 sont dans la relation β, et on dit qu'ils sont *bisimilaires*, quand ils exhibent le même comportement par rapport aux transitions.

La relation de bisimulation β est réfléxive, symétrique et transitive : c'est une relation d'équivalence.

Finalement, deux systèmes de transition $\langle Sa, Act, \rightarrow\rangle$ et $\langle Sb, Act, \rightarrow\rangle$, sont bisimilaires ssi il existe une relation de bisimulation β telle que : pour tout état $s_1 \in Sa$ il existe au moins un état $s_2 \in Sb$ tel que $s_1 \beta s_2$, et symétriquement pour tout $s_2 \in Sb$, il existe au moins un état $s_1 \in Sa$ tel que $s_1 \beta s_2$.

Toutefois, dans le contexte de notre TER, nous travaillons avec un seul système de transition input, disons M, l'objectif étant d'en construire un autre, disons M', qui, en un certain sens, soit inclus dans M et bisimilaire à M.

L'algorithme étudié dans ce TER construit par étapes la plus grande relation de bisimulation R pour un même système de transitions donné, au sens que le plus grand nombre possible d'états seront mis dans une même classe d'équivalence. Cela permet de partitionner l'ensemble S de ses états dans le plus petit nombre de classes d'équivalence possible, et revient à *minimiser le nombre d'états du système.*

2.6 Relation de bisimulation entre deux interprétations de CTL (structures de Kripke).

Definition 2.6. Etant donné deux structures de Kripke(définition 2.2) $\langle Sa, R, L\rangle$ et $\langle Sb, R, L\rangle$, une relation de *bisimulation* sur leurs états est une ensemble de couples $\beta \subseteq Sa \times Sb$ tel que, pour tout $s_1 \in Sa$ et $s_2 \in Sb$, si $s_1 \beta s_2$ alors :
– si, pour quelque $s' \in Sa$ on a $s_1 R s'$ alors il existe un $s" \in Sb$ tel que $s' \beta s"$ et $s_2 R s"$.
– Réciproquement, si, pour quelque $s" \in Sb$ on a $s_2 R s"$ alors il existe un $s' \in Sa$ tel que $s' \beta s"$ et $s_1 R s'$.
– s_1 et s_2 rendent vraies exactement les mêmes formules atomiques de CTL (c'est-à-dire les mêmes variables booléennes) : $L(s_1) = L(s_2)$.

Finalement, deux structures de Kripke $\langle Sa, R, L\rangle$ et $\langle Sb, R, L\rangle$, sont bisimilaires ssi il existe une relation de bisimulation β telle que : pour tout état $s_1 \in Sa$ il existe au moins un état $s_2 \in Sb$ tel que $s_1 \beta s_2$, et symétriquement pour tout $s_2 \in Sb$, il existe au moins un état $s_1 \in Sa$ tel que $s_1 \beta s_2$.

Le résultat suivant est bien évoqué au niveau de [1] :
Si s_1 et s_2 sont deux états bisimilaires de deux structures de Kripke bisimilaires alors ils rendent vraies exactement les mêmes formules de CTL.

L'algorithme à penser et à proposer une solution dans ce TER construit par étapes la plus grande relation de bisimulation R pour une même structure de Kripke , au sens que le plus grand nombre possible d'états seront mis dans une même classe d'équivalence. Cela permet de partitionner l'ensemble S des ses états dans le plus petit nombre de classes d'équivalence possible, et revient à *minimiser le nombre d'états du système.*

3 Corps du rapport

3.1 Algorithme de Kannelakis et Smolka

Cet algorithme est presenté dans [2].

3.1.1 Préambule

Soient $\langle S, Act, \rightarrow \rangle$ un Système de transition avec étiquettes sur les arêtes. Une relation d'équivalence sur l'ensemble des états S peut être représentée comme une partition de S c'est-à-dire comme un ensemble $\{B_0...B_k\}$, $k \geq 0$, de sous-ensembles non vides de S tels que :

1. $B_i \cap B_j = \emptyset$ pour tout $0 \leq i < j \leq k$ et
2. $S = B_0 \cup B_1 \cup ... \cup B_k$

Les ensembles Bi dans une partition sont généralement appelés blocs. Dans ce qui suit, en générale nous ne ferons pas de distinction entre les partitions et leurs relations d'équivalence associées.

Soit π et π' deux partitions de S. Nous disons que π' est un raffinement de π si pour chaque bloc $B' \in \pi'$ il existe un bloc $B \in \pi$ tel que $B' \subseteq B$. Observez que si π' est un raffinement de π, alors la relation d'équivalence associée à π' est incluse dans celle associée à π.

L'algorithmes de calcul de la bisimilarité due à Kanellakis et Smolka calcule des raffinements successifs d'une partition initiale π_{init} et convergent vers la plus grande bisimulation sur le système de transition étiqueté.

3.1.2 Algorithme

Soit $\pi = \{B_0, ..., B_k\}$, $k \geq 0$, une partition de l'ensemble des états S. L'algorithme de Kanellakis et Smolka est basé sur la notion de Split.

Definition 3.1. Un Split pour un bloc $B_i \in \pi$ est un bloc $B_j \in \pi$ tel que, pour une action $a \in Act$, certains états de B_i ont des transitions étiquetées a dont la cible est un état de B_j et d'autres non.

Intuitivement, en pensant aux blocs comme représentant des approximations de classes d'équivalence de processus,l'existence d'un Split B_j pour un bloc B_i dans la partition actuelle indique que nous avons une raison pour distinguer deux groupes d'ensembles d'états dans B_i à savoir ceux qui permettent une transition étiquetée a conduisant à un état dans B_j et ceux qui ne le font pas.

Par conséquent, B_i peut être divisé par B_j par rapport à l'action a en deux nouveaux blocs :

- $B_i^1 = \{s \mid s \in B_i et s_1 \xrightarrow{a} s'$, pour certains $s' \in B_j\}$ et
- $B_i^2 = B_i \setminus B_i^1$

Cette division donne lieu à la nouvelle partition :
$\pi' = \{B_0, ..., B_{i-1}, B_i^1, B_i^2, B_{i+1}, ..., B_k\}$ qui est un raffinement de π.

L'idée de base de l'algorithme de Kanellakis et Smolka est d'itérer la division d'un bloc B_i par un bloc B_j par rapport à une action a jusqu'à ce qu'aucun raffinement supplémentaire de la partition actuelle ne soit possible. La partition résultante est souvent appelée la plus grande *partition stable* et coïncide avec la bisimilarité sur le système de transition étiqueté en entrée lorsque la partition initiale π_{init} choisie est S.

```
function split(B, a, π)
choose some state s ∈ B
B1, B2 := ∅
for each state t ∈ B do
        if s and t can reach the same set of blocks in π via a-labelled transitions
        then B1 := B1 ∪ {t}
        else B2 := B2 ∪ {t}
if B2 is empty then return {B1}
else return {B1, B2}
```

FIGURE 1 – The function split

```
π := {Pr}
changed := true
while changed do
        changed := false
        for each block B ∈ π do
                for each action a do
                        sort the a-labelled transitions from states in B
                        if split(B, a, π) = {B1, B2} ≠ {B}
                        then refine π by replacing B with B1 and B2, and set changed
                                to true
```

FIGURE 2 – The algorithm by Kanellakis and Smolka

Example 3.1. Considérons maintenant le système de transition étiqueté suivant :

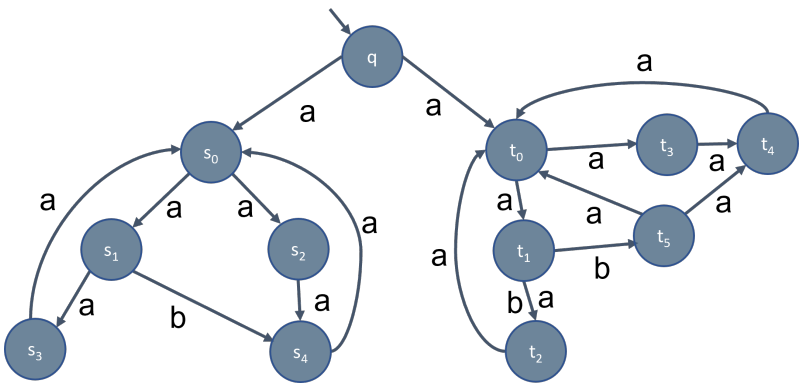

FIGURE 3 – Système de transition étudier

Soit $\{S\}$ la partition initiale associée à ce système de transition étiqueté, où

$$S = \{q, s_i, t_j \mid 0 \leq i \leq 4, 0 \leq j \leq 5\}$$

Le bloc S est un séparateur pour lui-même. En effet, certains états de S permettent des transitions étiquetées b tandis que d'autres ne le font pas. Si nous divisons S par S par rapport à l'action b, nous obtenons une nouvelle partition constituée des blocs

$$\{s_1, t_1\} \text{ et } \{q, s_i, t_j \mid 0 \leq i \leq 4, 0 \leq j \leq 5 \text{ avec } i, j \neq 1\}$$

Une visualisation possible de ce split est la suivante :

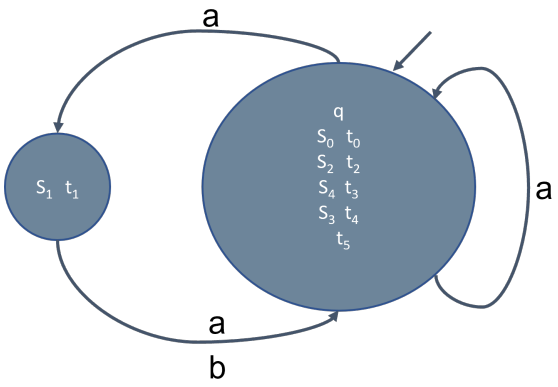

FIGURE 4 – Système de transition 2

Notez maintenant que le premier bloc est un séparateur pour le second par rapport à l'action a. En effet seuls les états s_0 et t_0 dans ce bloc permettent des transitions étiquetées a qui conduisent à un état dans le bloc $\{s_1, t_1\}$. Le fractionnement qui en résulte donne la partition.

$$\{\{s_0, t_0\}, \{s_1, t_1\}, \{q, s_i, t_j \mid 2 \leq i \leq 4, 2 \leq j \leq 5\}\}$$

Une visualisation possible de ce split est la suivante :

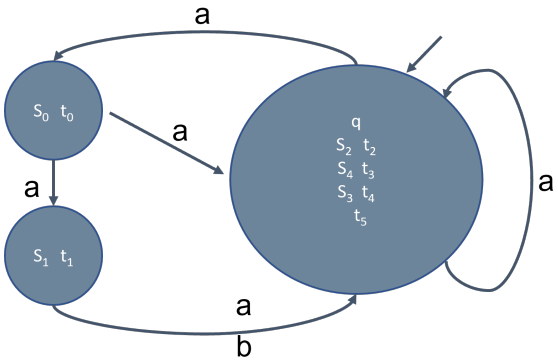

FIGURE 5 – Système de transition 3

La partition ci-dessus peut être affinée davantage. En effet, certains états du troisième bloc ont des transitions étiquetées a menant aux états du premier bloc, mais d'autres non. Par conséquent, le premier bloc est un séparateur pour le troisième par rapport à l'action a. Le fractionnement qui en résulte donne la partition

$$\{\{s_0, t_0\}, \{s_1, t_1\}, \{q, s_3, s_4, t_2, t_4, t_5\}, \{s_2, t_3\}\}$$

Une visualisation possible de ce split est la suivante :

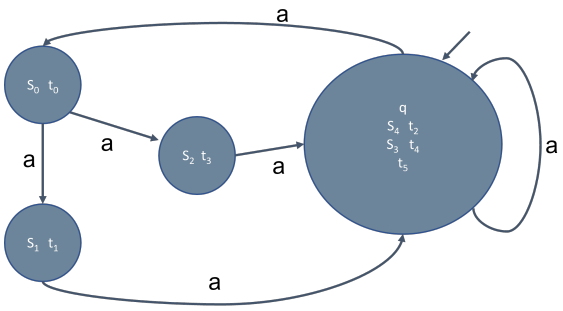

FIGURE 6 – Système de transition 4

On continue en observant que le bloc $\{q, s_3, s_4, t_2, t_4, t_5\}$ est un séparateur pour lui-même par rapport à l'action a. Par exemple $t_5 \xrightarrow{a} t_4$, mais la seule transition étiquetée a depuis s_4 est $s_4 \xrightarrow{a} s_0$. Le fractionnement qui en résulte donne la partition :

$$\{\{s_0, t_0\}, \{s_1, t_1\}, \{t_5\}, \{q, s_3, s_4, t_2, t_4\}, \{s_2, t_3\}\}$$

Une visualisation possible de ce split est la suivante :

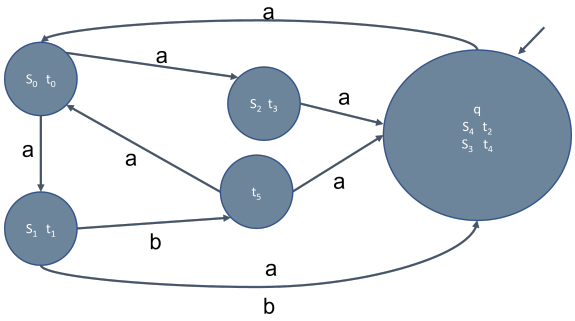

FIGURE 7 – Système de transition 5

Nous vous encourageons à continuer à affiner la partition ci-dessus jusqu'à ce que vous atteigniez la partition stable la plus grossière.

Alors, quelle est la partition qui en résulte ?

$$\{\{q\}, \{s_0\}, \{t_0\}, \{s_1\}, \{t_1\}, \{s_2\}, \{t_3\}, \{t_5\}, \{s_3, s_4\}, \{t_2, t_4\}\}$$

Finalemment, le système de transition final qu'on obtient de cette partition est le suivant :

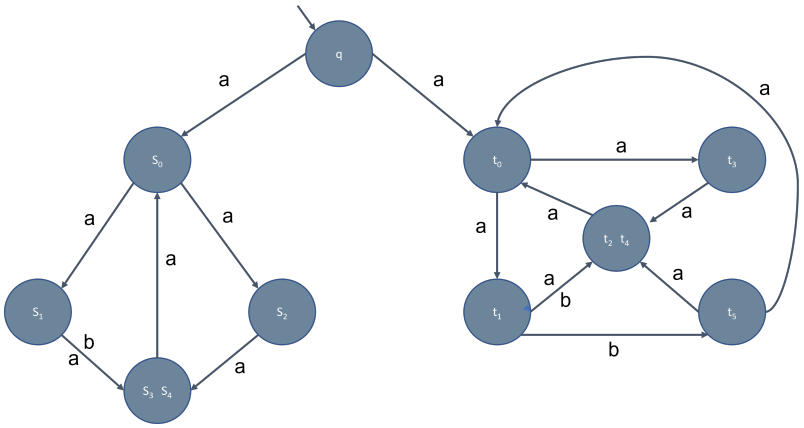

FIGURE 8 – Système de transition finale

Ainsi, on passe d'un système à 12 états à un nouveau système à 10 états qui lui est bisimilaire.

3.2 Implémentation de l'algorithme Kanellakis et Smolka et Tests

3.2.1 Implémentation

3.2.1.1 Analyse des besoins

Dans cette partie, il s'agira de spécifier les différentes structures de données que nous avons utilisés pour l'implémentation de l'algorithme de Kannelakis et Smolka [2]. Dans ce projet, nous avons fait le choix d'implémenter l'algorithme de Kannelakis et Smolka avec le langage Java qui est un langage orienté objet. Les structures de données utilisées ainsi sont les classes suivantes :

Node : Cette classe représente un noeud. Elle contient les informations sur un noeud à savoir le nom du noeud, les transitions sortantes et les noeuds fils.

Transition : C'est classe représente la liste de toutes les transitions sortantes de ce noeud d'un noeud vers un autre noeud.

Tree : Cette classe représente un arbre formé de noeuds.

B : Cette classe représente un bloc de noeud.

Pi : Cette classe contient les attributs et opérations qui permettront d'obtenir le split final à partir des classes *Node, Transition, Tree* et *B*.

3.2.1.2 Conception

Pour la conception, nous montrerons les relations entre les différentes classes. Pour cela, nous avons utilisé le langage de modélisation UML, qui permet de modéliser un système ainsi que les relations entres les différents composants de ce sysème. Nous obtenons donc le diagramme de classe suivant qui modélise les relations entre nos différentes classes :

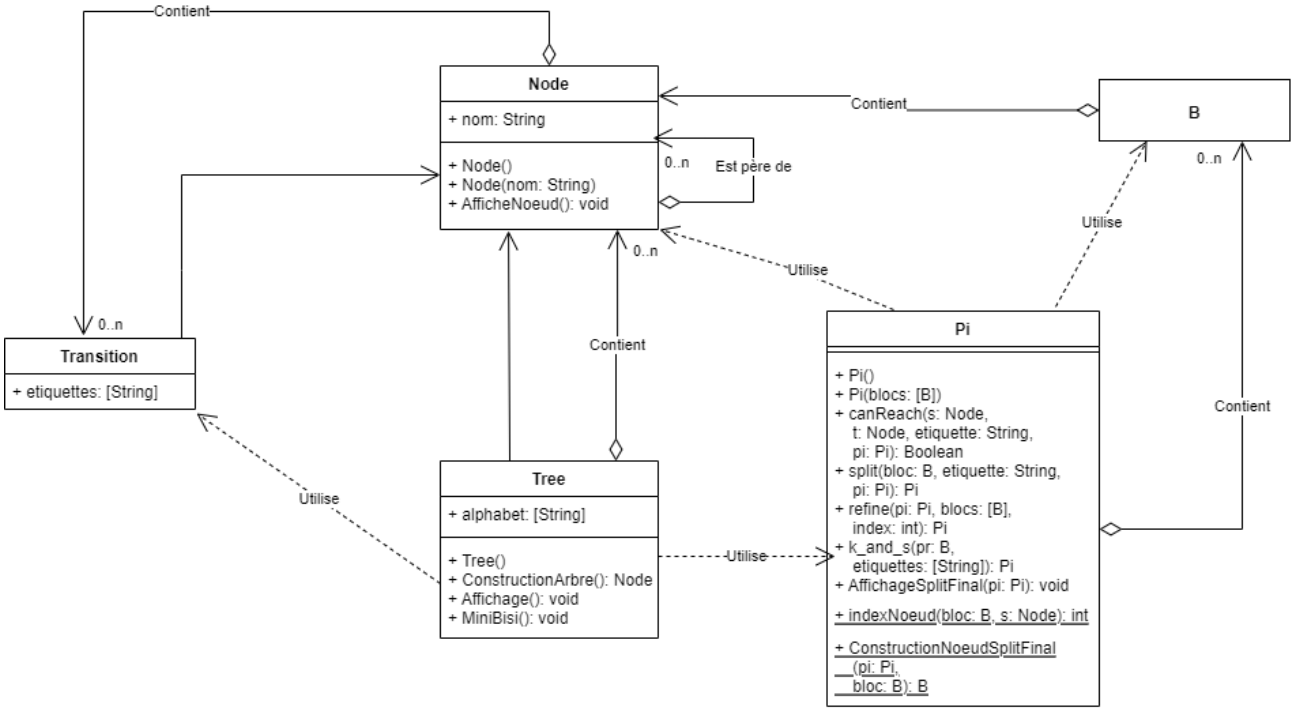

FIGURE 9 – Diagramme de classe

- Les structures de donnéés *Tree, Node, Transition* nous permettent de générer des systèmes de transitions avec étiquettes sur les arêtes que l'utilisateur saisies à partir du terminales et de les visualiser.
- Les structures de donnéés *B* et *Pi* nous permettent d'implementer l'algorithe de Kannelakis et Smolka [2], de l'appliquer sur le système de transition généré précédemment pour le minimiser et ainsi voir le nouveau systèmes obtenue bisimulaire au premier(définition 2.5)

3.2.1.3 Réalisattion

Concernant la réalisation, nous évoquerons l'environnement de travail matériel et logiciel utilisé pour l'implémentation de l'algorithme de Kannelakis et Smolka.

Environnement matériel

Pour l'implémentation de l'algorithme de Kannelakis et Smolka, nous avons utilisé un ordinateur portable avec les caractéristiques suivantes :
Modèle : Asus TUF FX505
Ram : 16 GB
Processeur : Intel I7 7è gen
Disque dur : 1 TO

Environnement logiciel

Les logiciels utilisés pour l'implémentation de l'algorithme de Kannelakis et Smolka sont :
Draw.io : C'est un logiciel de dessin graphique multiplateforme gratuit et open source développé en HTML5 et JavaScript. Son interface peut être utilisée pour créer des diagrammes tels que des organigrammes, des wireframes, des diagrammes UML, des organigrammes et des diagrammes de réseau.
Dropbox.com : C'est un service de stockage et de partage de copies de fichiers locaux en ligne.
Eclipse : C'est un environnement de développement intégré (IDE) utilisé en programmation informatique. Il contient un espace de travail de base et un système de plug-in extensible pour personnaliser l'environnement. C'est un IDE populaire pour le développement Java.

3.2.2 Test

Les figures suivantes décrivent le résultat d'un exemple d'exécution de la création d'un système de transition à sa minimisation grâce à l'application de l'agorithme de Kannelakis et Smolka implémenté :

Tout d'abord, l'utilisateur saisit son système de transition en suivant les instructions puis valide.

```
Entrez le nom du noeud root:
q

Entrez le nombre d'enfants pour le noeud q
2

Entrez le nom du noeud 0, fils du noeud q
s0

Entrez le nombre d'enfants pour le noeud s0
2

Entrez le nom du noeud 0, fils du noeud s0
s1

Entrez le nombre d'enfants pour le noeud s1
2

Entrez le nom du noeud 0, fils du noeud s1
s3

Entrez le nombre d'enfants pour le noeud s3
1

Entrez le nom du noeud 0, fils du noeud s3
s0

Donnez une etiquette pour la transition (s3->s0), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (s3->s0), sinon tapez -1 pour terminer:
-1

Entrez le nom du noeud 1, fils du noeud s1
s4

Entrez le nombre d'enfants pour le noeud s4
1

Entrez le nom du noeud 0, fils du noeud s4
s0

Donnez une etiquette pour la transition (s4->s0), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (s4->s0), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (s1->s3), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (s1->s3), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (s1->s4), sinon tapez -1 pour terminer:
b

Donnez une etiquette pour la transition (s1->s4), sinon tapez -1 pour terminer:
-1

Entrez le nom du noeud 1, fils du noeud s0
s2

Entrez le nombre d'enfants pour le noeud s2
1

Entrez le nom du noeud 0, fils du noeud s2
s4
```

FIGURE 10 – Création d'un système de transition (1)

```
Entrez le nom du noeud 0, fils du noeud s2
s4

Donnez une etiquette pour la transition (s2->s4), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (s2->s4), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (s0->s1), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (s0->s1), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (s0->s2), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (s0->s2), sinon tapez -1 pour terminer:
-1

Entrez le nom du noeud 1, fils du noeud q
t0

Entrez le nombre d'enfants pour le noeud t0
2

Entrez le nom du noeud 0, fils du noeud t0
t1

Entrez le nombre d'enfants pour le noeud t1
2

Entrez le nom du noeud 0, fils du noeud t1
t2

Entrez le nombre d'enfants pour le noeud t2
1

Entrez le nom du noeud 0, fils du noeud t2
t0

Donnez une etiquette pour la transition (t2->t0), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t2->t0), sinon tapez -1 pour terminer:
-1

Entrez le nom du noeud 1, fils du noeud t1
t5

Entrez le nombre d'enfants pour le noeud t5
2

Entrez le nom du noeud 0, fils du noeud t5
t0

Entrez le nom du noeud 1, fils du noeud t5
t4

Entrez le nombre d'enfants pour le noeud t4
1

Entrez le nom du noeud 0, fils du noeud t4
t0
```

FIGURE 11 – Création d'un système de transition (2)

```
Entrez le nom du noeud 0, fils du noeud t4
t0

Donnez une etiquette pour la transition (t4->t0), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t4->t0), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (t5->t0), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t5->t0), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (t5->t4), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t5->t4), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (t1->t2), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t1->t2), sinon tapez -1 pour terminer:
b

Donnez une etiquette pour la transition (t1->t2), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (t1->t5), sinon tapez -1 pour terminer:
b

Donnez une etiquette pour la transition (t1->t5), sinon tapez -1 pour terminer:
-1

Entrez le nom du noeud 1, fils du noeud t0
t3

Entrez le nombre d'enfants pour le noeud t3
1

Entrez le nom du noeud 0, fils du noeud t3
t4

Donnez une etiquette pour la transition (t3->t4), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t3->t4), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (t0->t1), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t0->t1), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (t0->t3), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (t0->t3), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (q->s0), sinon tapez -1 pour terminer:
a
```

FIGURE 12 – Création d'un système de transition (3)

```
Donnez une etiquette pour la transition (q->s0), sinon tapez -1 pour terminer:
-1

Donnez une etiquette pour la transition (q->t0), sinon tapez -1 pour terminer:
a

Donnez une etiquette pour la transition (q->t0), sinon tapez -1 pour terminer:
-1
```

FIGURE 13 – Création d'un système de transition (4)

Notre modèle lui affiche par la suite une visualisation du système de transition qu'il a créé.

```
----------------------------
Arborescence initiale de notre système de transition

q => s0,t0,/ q ,a, s0 --- q ,a, t0 --- /
s0 => s1,s2,/ s0 ,a, s1 --- s0 ,a, s2 --- /
s1 => s3,s4,/ s1 ,a, s3 --- s1 ,b, s4 --- /
s3 => s0,/ s3 ,a, s0 --- /
s4 => s0,/ s4 ,a, s0 --- /
s2 => s4,/ s2 ,a, s4 --- /
t0 => t1,t3,/ t0 ,a, t1 --- t0 ,a, t3 --- /
t1 => t2,t5,/ t1 ,a,b, t2 --- t1 ,b, t5 --- /
t2 => t0,/ t2 ,a, t0 --- /
t5 => t0,t4,/ t5 ,a, t0 --- t5 ,a, t4 --- /
t4 => t0,/ t4 ,a, t0 --- /
t3 => t4,/ t3 ,a, t4 --- /
```

FIGURE 14 – Arborescence initiale de notre système de transition

Par la suite, notre modèle applique l'algorithme implémenté pour minimiser ce système et nous affiche la partition finale obtenue.

```
----------------------------
Split final obtenu pour notre système de transition

{ { t5 }, { s1 }, { t1 }, { s0 }, { t0 }, { q }, { s3, s4 }, { t2, t4 }, { s2 }, { t3 } }
```

FIGURE 15 – Split final obtenu pour notre système de transition

Et enfin, il nous affiche le système de transition minimale obtenue à partir de la partition finale.

```
----------------------------
Arborescence finale du système bisimilaire obtenu par rapport au système initiale

 t5 => t0, t2 t4,/ t5 ,a, t0 --- t5 ,a, t2 t4 --- /
 s1 => s3 s4,/ s1 ,a,b, s3 s4 ---/
 t1 => t2 t4, t5,/ t1 ,a,b, t2 t4 --- t1 ,b, t5 --- /
 s0 => s1, s2,/ s0 ,a, s1 --- s0 ,a, s2 --- /
 t0 => t1, t3,/ t0 ,a, t1 --- t0 ,a, t3 ---/
 q => s0, t0,/ q ,a, s0 --- q ,a, t0 --- /
 s3 s4 => s0,/ s3 s4 ,a, s0 --- /
 t2 t4 => t0,/ t2 t4 ,a, t0 --- /
 s2 => s3 s4,/ s2 ,a, s3 s4 --- /
 t3 => t2 t4,/ t3 ,a, t2 t4 --- /
```

FIGURE 16 – Arborescence finale du système bisimilaire obtenu par rapport au système initiale

3.3 Proposition d'une solution afin d'adapter l'algorithme pour minimiser une interprétation CTL

Pour minimiser une interprétation de CTL (définition 2.4), il suffit de modifier l'initialisation de l'algorithme de Kanellakis et Smolka : dans la partition initiale, deux états seront dans un même bloc si et seulement si leurs étiquettes sont égales (même ensemble de variables booléennes vraies sur ces état).

Puis, on procède en suivant les étapes décrit au niveau de l'algorithme de Kanellakis et Smolka (voir la définition 3.1). On observera que, *de facto*, ici les transitions ont comme étiquette une unique action a.

Example 3.2. Considérons maintenant la structures de Kripke suivante (en violet nous indiquons des formules de CTL vraies ou fausses aux états considérés) :

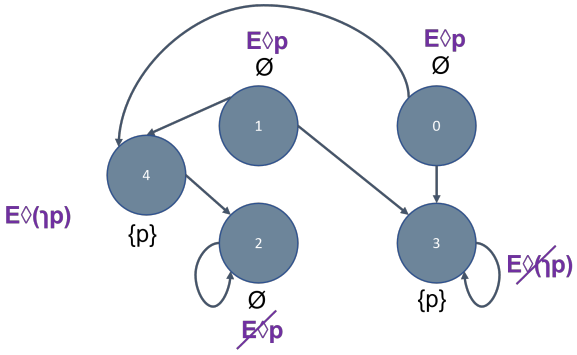

FIGURE 17 – structure de Kripke à minimiser

D'après la definition précédente, la partition initiale $\{S\}$ associée à ce système est :

$$S = \{\{0,1,2\},\{3,4\}\}$$

En déroulant l'algorithme de Kanellakis et Smolka sur ce dernier on obtient la partion finale qui suit :

$$\{\{0,1\},\{2\},\{3\},\{4\}\}$$

Une visualisation possible de ce split est la suivante :

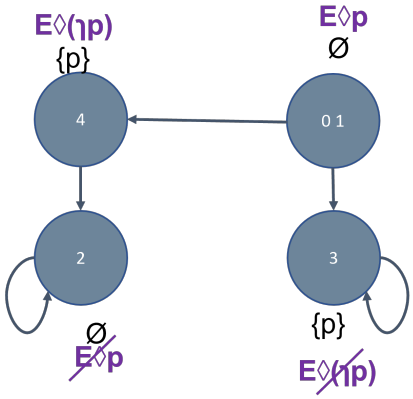

FIGURE 18 – structures de Kripke résultat

4 CONCLUSION

Lorsqu'il est appliqué à un système de transitions ayant des étiquettes exclusivement sur les arêtes avec n états et m transitions, l'algorithme de Kanellakis et Smolka [2] calcule la partition correspondant à bisimilarité en temps de $O(nm)$. Ce dernier a également une complexité spatiale de $O(n+m)$.

Une question naturelle à se poser à ce stade est de savoir si l'algorithme de Kanellakis et Smolka est optimal, ce qui ne est pas le cas. En effet, la compléxité temporelle d'un algorithme de raffinement de partition comme celui que nous venons d'évoquer peut être considérablement améliorée par une utilisation astucieuse des structures de données.

Ainsi l'algorithme de Paige et Tarjan présenté dans [2] qui est également basé sur l'idée du raffinement des partitions améliore significativement l'efficacité de l'algorithme de Kanellakis et Smolka en utilisant des structures de données plus complexes.

La faible complexité de l'algorithme de Paige et Tarjan en temps de $O(m \log n)$ est obtenue en utilisant des informations sur les fractionnements précédents pour améliorer l'efficacité des fractionnements futurs.

Ce travail nous a été considérablement instructif. En effet, la réalisation d'un tel projet nous a permis non seulement de nous initier au travail d'étude et de recherche scientifique (lire, comprendre et tirer des informations d'un article scientifique), mais aussi d'avoir une expérience professionnelle. Nous nous sommes aussi familiarisés avec l'environnement Overleaf pour la production de ce document.

Après l'adaptation et la validation de notre algorithme pour minimiser une interprétation de CTL, ne serait-il pas idéal de l'implémenter et le tester?

Références

[1] Stéphane DEMRI, Valentin GORANKO et Martin LANGE. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. DOI : 10.1017/CBO9781139236119.

[2] Jirí Srba Davide Sangiorgi LUCA ACETO Anna Ingolfsdottir et Jan RUTTEN. « The Algorithmics of Bisimilarity ». In : (2011). URL : https://www.ru.is/faculty/luca/PAPERS/algobisimchapter.pdf.

[3] « Structure de Kripke ». In : (2021). URL : https://fr.wikipedia.org/wiki/Structure_de_Kripke.

Appendice

```java
package GT1;

import java.util.ArrayList;

public class Node {
  public String nom;
  public ArrayList<Node> enfants;
  public ArrayList<Transition> t;

  public Node(String nom) {
    this.nom = nom;
    this.t = new ArrayList<Transition>();
    this.enfants = new ArrayList<Node>();
  }

  public Node() {

    this.t = new ArrayList<Transition>();
    this.enfants = new ArrayList<Node>();
  }

  public void AfficheNoeud() {

    String enfants = this.nom + "=>";
    for (Node enfant : this.enfants) {
      enfants += enfant.nom + ",";

    }
    enfants += "/";
    for (Transition t : this.t) {
      String r = "";
      enfants += this.nom + " ,";
      for (String s : t.etiquettes) {
        r += s + ",";
      }
      enfants += r + t.n2.nom + "---";
    }
    enfants += "/";
    System.out.println(enfants);
  }
}
```

```java
package GT1;

import java.util.ArrayList;

public class Transition {
  /*
  Liste d'etiquette pour un noeud et une transition
   */
  public ArrayList<String> etiquettes = new ArrayList<String>();
  public Node n2;
}
```

```java
package GT1;

import java.util.ArrayList;
import java.util.Scanner;

import Splite.B;
import Splite.Pi;

public class Tree {
  /*
   * Represente le noeud root
   */
  public Node root;

  /*
   * Represente la liste de l'alphabetbet a remplir par l'utilisateur
   */
  public ArrayList<String> alphabet;

  /*
   * Represente une liste contenant tous les noeuds de l'abre genere
   */
  public ArrayList<Node> noeuds;

  public Tree() {
    /*
     * Initialisation des attributs
     */
    alphabet = new ArrayList<String>();
    noeuds = new ArrayList<Node>();
    Scanner s = new Scanner(System.in);

    /*
     * Construction de l'arbre
     */
    this.root = ConstructionArbre(s, null, 0);
  }

  public Node ConstructionArbre(Scanner s, Node parent, int i) {

    if (parent == null) {
      /*
       * Siginifie que nous sommes au niveau du noeud root
       */
      System.out.println("Entrez le nom du noeud root:");
    } else {
      /*
       * Siginifie que nous sommes au niveau d'un noeud avec parent
       */
      System.out.println("Entrez le nom du noeud " + i + ", fils du noeud " + parent.
    nom);

    }
    /*
     * Lecture du nom du noeud
     */
```

```
56      String nom = s.next();
57
58      /*
59       * Determinera si le noeud existe deja ou non
60       */
61      String exist = "";
62      int kk = -1;
63
64      for (int k = 0; k < noeuds.size(); k++) {
65
66        /*
67         * Verifie si le nom entre appartient a un noeud existant puis recuperer
68         * l'indice
69         */
70        if (noeuds.get(k).nom.equals(nom)) {
71          exist = noeuds.get(k).nom;
72          kk = k;
73          break;
74        }
75      }
76
77      /*
78       * Creation du noeud en construction
79       */
80      Node no = new Node(nom);
81      noeuds.add(no);
82
83      if (!exist.isEmpty()) {
84
85        /*
86         * Si le nom n'existe pas alors ajouter le noeud cree a la liste des noeuds
87         */
88        parent.enfants.add(noeuds.get(kk));
89        noeuds.remove(no);
90      } else {
91
92        /*
93         * Si le nom n'existe pas alors c'est un nouveau noeud a creer
94         */
95        no = new Node(nom);
96        System.out.println("Entrez le nombre d'enfants pour le noeud " + no.nom);
97        int n = s.nextInt();
98        ArrayList<Node> enfants = new ArrayList<Node>();
99        for (int j = 0; j < n; j++) {
100         /*
101          * Construire les noeuds fils recursivement
102          */
103         Node enfant = ConstructionArbre(s, no, j);
104         enfants.add(enfant);
105       }
106       no.enfants = enfants;
107       Transition tt;
108       for (int j = 0; j < no.enfants.size(); j++) {
109         tt = new Transition();
110         boolean etq = true;
111         while (etq) {
```

```
112
113           /*
114            * tant que l'utilisateur a des etiquettes a mettre alors construire
115            * l'alphabetbet
116            */
117           System.out.println("Donnez une etiquette pour la transition (" + no.nom + "
    ->"
118               + no.enfants.get(j).nom + "), sinon tapez -1 pour terminer:");
119           String res = s.next();
120           if (res.equals("-1")) {
121             etq = false;
122           } else {
123             if (!alphabet.contains(res))
124               alphabet.add(res);
125             tt.etiquettes.add(res);
126           }
127         }
128
129         /*
130          * Construire la liste des transitions pour le noeud actuel
131          */
132         if (tt.etiquettes.size() > 0) {
133           tt.n2 = no.enfants.get(j);
134           no.t.add(tt);
135         }
136       }
137       for (int k = 0; k < noeuds.size(); k++) {
138
139         if (noeuds.get(k).nom.equals(no.nom)) {
140           noeuds.set(k, no);
141           break;
142         }
143       }
144     }
145     return no;
146   }
147
148   public void Affichage() {
149     for (int j = 0; j < this.noeuds.size(); j++) {
150       this.noeuds.get(j).AfficheNoeud();
151     }
152   }
153
154   public void MiniBisi() {
155     B b0 = new B();
156     for (int i = 0; i < this.noeuds.size(); i++) {
157       b0.pi.add(noeuds.get(i));
158     }
159
160     ArrayList<B> block = new ArrayList<B>();
161     block.add(b0);
162
163     Pi pii = new Pi(block);
164     Pi pi_ = pii.k_and_s(b0, this.alphabet);
165     System.out.println("--------------------");
166     pii.AffichageSplitFinal(pi_);
```

```
167
168     B b4 = Pi.ConstructionNoeudSplitFinal(pi_, b0);
169     for (int j = 0; j < b4.pi.size(); j++) {
170       b4.pi.get(j).AfficheNoeud();
171     }
172   }
173 }
```

```
1  package Splite;
2
3  import java.util.ArrayList;
4
5  import GT1.Node;
6  import GT1.Transition;
7
8  public class Pi {
9
10   ArrayList<B> pi;
11
12   Pi() {
13     pi = new ArrayList<B>();
14   }
15
16   public Pi(ArrayList<B> blocs) {
17     pi = blocs;
18   }
19
20   public static boolean canReach(Node s, Node t, String etiquette, Pi pi) {
21     ArrayList<B> sB = new ArrayList<B>();
22     ArrayList<B> tB = new ArrayList<B>();
23     for (int l = 0; l < s.t.size(); l++) {
24       if (s.t.get(l).etiquettes.contains(etiquette)) {
25         for (int i = 0; i < pi.pi.size(); i++) {
26           for (int j = 0; j < pi.pi.get(i).pi.size(); j++) {
27             if (s.t.get(l).n2.nom.equals(pi.pi.get(i).pi.get(j).nom)) {
28               sB.add(pi.pi.get(i));
29             }
30           }
31         }
32       }
33     }
34
35     for (int l = 0; l < t.t.size(); l++) {
36       if (t.t.get(l).etiquettes.contains(etiquette)) {
37         for (int i = 0; i < pi.pi.size(); i++) {
38           for (int j = 0; j < pi.pi.get(i).pi.size(); j++) {
39             if (t.t.get(l).n2.nom.equals(pi.pi.get(i).pi.get(j).nom)) {
40               tB.add(pi.pi.get(i));
41             } else {
42               // System.out.println(t.t.get(l).n2.nom);
43               // System.out.println(pi.pi.get(i).pi.get(j).nom);
44             }
45
46           }
47         }
48       }
```

```
49      }
50
51      if (sB.size() == tB.size()) {
52
53        if (sB.size() != 0) {
54          for (int i = 0; i < sB.size(); i++) {
55            if (!tB.contains(sB.get(i))) {
56              return false;
57            }
58          }
59          for (int i = 0; i < tB.size(); i++) {
60            if (!sB.contains(tB.get(i))) {
61              return false;
62            }
63          }
64          return true;
65        }
66        return true;
67      } else {
68        boolean exist = true;
69        for (int i = 0; i < sB.size(); i++) {
70          if (!tB.isEmpty()) {
71            if (!tB.contains(sB.get(i))) {
72              exist = false;
73            }
74          } else
75            exist = false;
76
77        }
78        for (int i = 0; i < tB.size(); i++) {
79          if (!sB.isEmpty()) {
80            if (!sB.contains(tB.get(i))) {
81              exist = false;
82            }
83          } else
84            exist = false;
85        }
86        return exist == true;
87      }
88    }
89
90    public Pi split(B bloc, String etiquette, Pi pi) {
91      // int index = (int) (Math.random() * bloc.pi.size());
92      // System.out.println("randex:" + index);
93      Node s = bloc.pi.get(0);
94      B b1 = new B();
95      B b2 = new B();
96
97      for (int i = 0; i < bloc.pi.size(); i++) {
98        if (canReach(s, bloc.pi.get(i), etiquette, pi)) {
99          b1.pi.add(bloc.pi.get(i));
100       } else {
101         b2.pi.add(bloc.pi.get(i));
102         // System.out.print(b2.pi);
103       }
104     }
```

```
105     ArrayList<B> pt = new ArrayList<B>();
106     if (b2.pi.isEmpty()) {
107       pt.add(b1);
108       return new Pi(pt);
109     } else {
110       if (!b1.pi.isEmpty())
111         pt.add(b1);
112       pt.add(b2);
113       return new Pi(pt);
114     }
115   }
116
117   public Pi refine(Pi pi, ArrayList<B> pip, int index) {
118
119     for (int i = 0; i < pip.size(); i++) {
120       pi.pi.add(pip.get(i));
121     }
122     pi.pi.remove(index);
123     return pi;
124
125   }
126
127   public Pi k_and_s(/* also {Pr}* initially */ B pr, ArrayList<String> etiquettes /*
        , ArrayList<Transition> t */) {
128
129     ArrayList<B> block = new ArrayList<B>();
130     block.add(pr);
131     Pi pi = new Pi(block);
132
133     boolean changed = true;
134     ArrayList<B> pip;
135     B copy = new B();
136
137     while (changed) {
138       changed = false;
139       for (int i = 0; i < pi.pi.size(); i++) {
140
141         for (int j = 0; j < etiquettes.size(); j++) {
142           copy.pi = pi.pi.get(i).pi;
143           pip = pi.split(copy, etiquettes.get(j), pi).pi;
144
145           if (pip.size() > 1) {
146             pi = refine(pi, pip, i);
147             changed = true;
148
149           }
150
151         }
152       }
153     }
154
155     return new Pi(pi.pi);
156   }
157
158   public void AffichageSplitFinal(Pi pi) {
159     System.out.println("//");
```

```
160     if (pi.pi.size() > 1)
161       System.out.print("{ ");
162     for (int i = 0; i < pi.pi.size(); i++) {
163       System.out.print("{ ");
164       for (int j = 0; j < pi.pi.get(i).pi.size(); j++) {
165         if (j != pi.pi.get(i).pi.size() - 1)
166           System.out.print(pi.pi.get(i).pi.get(j).nom + ", ");
167         else
168           System.out.print(pi.pi.get(i).pi.get(j).nom);
169       }
170       if (i != pi.pi.size() - 1)
171         System.out.print(" }, ");
172       else
173         System.out.print(" }");
174     }
175     if (pi.pi.size() > 1)
176       System.out.print(" }");
177     System.out.println(" ");
178     System.out.println("//");
179   }
180
181
182   public static int IndexNoeud(B bloc, Node s) {
183     int i = 0;
184     for (int j = 0; j < bloc.pi.size(); j++) {
185       if (bloc.pi.get(j).nom.contains(s.nom)) {
186         i = j;
187       }
188     }
189     return i;
190   }
191
192   public static B ConstructionNoeudSplitFinal(Pi pi, B bloc) {
193     B b0 = new B();
194     for (int i = 0; i < pi.pi.size(); i++) {
195       String s = "";
196       Node n = new Node();
197       for (int j = 0; j < pi.pi.get(i).pi.size(); j++) {
198         s = s + " " + pi.pi.get(i).pi.get(j).nom;
199       }
200       n.nom = s;
201       b0.pi.add(n);
202     }
203     for (int i = 0; i < bloc.pi.size(); i++) {
204       int ind = IndexNoeud(b0, bloc.pi.get(i));
205       Node l = b0.pi.get(ind);
206       for (int j = 0; j < bloc.pi.get(i).t.size(); j++) {
207         Transition t = bloc.pi.get(i).t.get(j);
208         int ind2 = IndexNoeud(b0, t.n2);
209         Node l2 = b0.pi.get(ind2);
210         if (l.t.size() == 0) {
211           t.n2 = l2;
212           l.t.add(t);
213           l.enfants.add(l2);
214         } else {
215           for (int k = 0; k < l.t.size(); k++) {
```

```
            if (!(l.t.get(k).n2.nom.equals(l2.nom))) {
              t.n2 = l2;
              l.t.add(t);
              l.enfants.add(l2);
            } else {
              for (int h = 0; h < t.etiquettes.size(); h++) {
                int exist = 0;
                for (int g = 0; g < l.t.get(k).etiquettes.size(); g++) {
                  if (l.t.get(k).etiquettes.get(g).equals(t.etiquettes.get(h))) {
                    exist++;
                  }
                }
                if (exist == 0)
                  l.t.get(k).etiquettes.add(t.etiquettes.get(h));
              }
            }
          }
        }
      }
      b0.pi.set(ind, l);
    }
    return b0;
  }

}
```

```
package Splite;

import java.util.ArrayList;

import GT1.Node;

public class B {
  public ArrayList<Node> pi = new ArrayList<Node>();
}
```

```
package GT1;

public class Main {

  public static void main(String[] args) {
    Tree tree = new Tree();
    System.out.println(tree.noeuds.size());
    System.out.println("-----------------------------");
    tree.Affichage();
    System.out.println("-----------------------------");
    tree.MiniBisi();
    // Transition t = new Transition();
  }

}
```