Logique Épistémique S5 avec DEMO-S5

Yuxiang ZHANG
Kenan ALSAFADI

Decembre 2025

Introduction

Ce rapport présente les travaux réalisés lors du TME 10 de logique épistémique S5. L'objectif était de manipuler les notions vues en cours à l'aide du logiciel DEMO-S5, qui permet de construire et de faire évoluer des structures de Kripke suite à des annonces publiques. Les exercices portent sur le modèle Hexa (jeu de cartes) et sur le problème de l'anniversaire de Cheryl.

1 Exercice 1 : Le modèle Hexa

1.1 Structure de Kripke initiale

Après avoir chargé le fichier `jeu-hexa.hs` dans l'interpréteur Haskell avec la commande `ghci jeu-hexa.hs`, la structure de Kripke initiale `model0` a été affichée. L'affichage obtenu est le suivant :

```
ghci> model0
Mo [('r','v','j'),('r','j','v'),('j','r','v'),('j','v','r'),('v','r','j'),('v','j','r
    [a,b,c]
    []
    [(a,[[('r','v','j'),('r','j','v')],
         [('j','r','v'),('j','v','r')],
         [('v','r','j'),('v','j','r')]]),
     (b,[[('r','v','j'),('j','v','r')],
         [('r','j','v'),('v','j','r')],
         [('v','r','j'),('j','r','v')]]),
     (c,[[('r','v','j'),('v','r','j')],
         [('j','v','r'),('v','j','r')],
         [('j','r','v'),('r','j','v')]])]
    []
```

Cette structure représente tous les états possibles de distribution des cartes (rouge, jaune, verte) entre les trois joueurs a, b et c. Chaque monde possible correspond à une distribution spécifique, représentée par un tuple (x, y, z) où x est la carte de a, y celle de b, et z celle de c.

Les mondes possibles sont au nombre de 6, correspondant aux permutations des trois cartes entre les trois joueurs.

Les relations d'accessibilité pour chaque agent modélisent l'incertitude de chacun concernant les cartes des autres joueurs. Par exemple, pour l'agent a, il y a trois classes d'équivalence :

— $\{('r','v','j'),('r','j','v')\}$: les mondes où a a la carte rouge
— $\{('j','r','v'),('j','v','r')\}$: les mondes où a a la carte jaune
— $\{('v','r','j'),('v','j','r')\}$: les mondes où a a la carte verte

Cela signifie que l'agent a, connaissant sa propre carte, ne peut distinguer entre les mondes où sa carte est la même mais la distribution entre b et c diffère.

1.2 Effets des annonces publiques simples

1.2.1 Première annonce : "a sait que b possède la carte verte"

L'annonce publique de l'agent a : `"je sais que b possède la carte verte"` se traduit par la commande :

```
upd_pa model0 (Kn a holds_b_verte)
```

Le résultat obtenu est :

```
ghci> upd_pa model0 (Kn a holds_b_verte)
Mo [] [a,b,c] [] [(a,[]),(b,[]),(c,[])] []
```

Cette annonce élimine tous les mondes où la formule `Kn a holds_b_verte` n'est pas vraie. Dans la structure initiale, aucun monde ne satisfait cette condition : l'agent a ne sait jamais avec certitude que b a la carte verte (car a ne voit pas la carte de b). Ainsi, tous les mondes sont éliminés et la structure devient vide.

1.2.2 Deuxième annonce : "a ne sait pas que b possède la carte verte"

L'annonce publique de l'agent a : `"je ne sais pas que b possède la carte verte"` se traduit par :

```
upd_pa model0 (Ng (Kn a holds_b_verte))
```

Le résultat obtenu est :

```
ghci> upd_pa model0 (Ng (Kn a holds_b_verte))
Mo [('r','v','j'),('r','j','v'),('j','r','v'),('j','v','r'),
    ('v','r','j'),('v','j','r')]
    [a,b,c]
    []
    [(a,[[('r','v','j'),('r','j','v')],
         [('j','r','v'),('j','v','r')],
         [('v','r','j'),('v','j','r')]]),
     (b,[[('r','v','j'),('j','v','r')],
         [('r','j','v'),('v','j','r')],
         [('v','r','j'),('j','r','v')]]),
     (c,[[('r','v','j'),('v','r','j')],
         [('j','v','r'),('v','j','r')],
         [('j','r','v'),('r','j','v')]])]
    []
```

Cette annonce élimine les mondes où l'agent a sait que b a la verte. Comme nous l'avons vu précédemment, aucun monde ne satisfait cette condition, donc aucun monde n'est éliminé. La structure reste donc inchangée.

1.3 Séquences d'annonces

1.3.1 Première séquence

La première séquence d'annonces est :

1. **b annonce** : "je sais que je ne possède pas la carte jaune"

   ```
   Kn b (Ng holds_b_jaune)
   ```

2. **a annonce** : "je ne sais pas que b possède la carte rouge"

   ```
   Ng (Kn a holds_b_rouge)
   ```

3. **a annonce** : "je sais que je ne possède pas la carte jaune"

   ```
   Kn a (Ng holds_a_jaune)
   ```

Étape 1 : Annonce de b

```
ghci> model1 = upd_pa model0 (Kn b (Ng holds_b_jaune))
ghci> model1
Mo [('r','v','j'),('j','r','v'),('j','v','r'),('v','r','j')]
   [a,b,c]
   []
   [(a,[[('r','v','j')],[('j','r','v'),('j','v','r')],[('v','r','j')]]),
    (b,[[('r','v','j'),('j','v','r')],[('v','r','j'),('j','r','v')]]),
    (c,[[('r','v','j'),('v','r','j')],[('j','v','r')],[('j','r','v')]])]
   []
```

Après la première annonce, 4 mondes restent sur les 6 initiaux. Les mondes éliminés sont ceux où b a la carte jaune : $('r','j','v')$ et $('v','j','r')$. L'agent b sait qu'il n'a pas la jaune, donc ces mondes sont incompatibles avec sa connaissance.

Étape 2 : Annonce de a

```
ghci> model2 = upd_pa model1 (Ng (Kn a holds_b_rouge))
ghci> model2
Mo [('r','v','j'),('j','r','v'),('j','v','r')]
   [a,b,c]
   []
   [(a,[[('r','v','j')],[('j','r','v'),('j','v','r')]]),
    (b,[[('r','v','j'),('j','v','r')],[('j','r','v')]]),
    (c,[[('r','v','j')],[('j','v','r')],[('j','r','v')]])]
   []
```

Après la deuxième annonce, le monde $('v','r','j')$ est éliminé. Dans ce monde, l'agent a sait que b a la carte rouge (car a a la verte et voit que c a la jaune). L'annonce de a dit qu'il ne sait pas que b a la rouge, donc ce monde est incompatible.

Étape 3 : Annonce de a

```
ghci> model3 = upd_pa model2 (Kn a (Ng holds_a_jaune))
ghci> model3
Mo [('r','v','j')]
   [a,b,c]
   []
   [(a,[[('r','v','j')]]),
    (b,[[('r','v','j')]]),
    (c,[[('r','v','j')]])]
   []
```

Après la troisième annonce, les mondes où a a la carte jaune sont éliminés : $('j','r','v')$ et $('j','v','r')$. Il ne reste donc qu'un seul monde : $('r','v','j')$. Dans ce monde final :

— a a la carte rouge
— b a la carte verte
— c a la carte jaune

Tous les agents connaissent maintenant parfaitement la distribution des cartes, comme en témoigne le fait que chaque agent n'a qu'une seule classe d'équivalence contenant un seul monde.

1.3.2 Deuxième séquence

La deuxième séquence d'annonces est :

1. **c annonce** : "je sais que je ne possède pas la carte jaune"

   ```
   Kn c (Ng holds_c_jaune)
   ```

2. **a annonce** : "je ne sais pas que b sait que b possède la carte jaune"

   ```
   Ng (Kn a (Kn b holds_b_jaune))
   ```

3. **c annonce** : "je sais que b ne possède pas la carte verte"

   ```
   Kn c (Ng holds_b_verte)
   ```

Étape 1 : Annonce de c

```
ghci> model1 = upd_pa model0 (Kn c (Ng holds_c_jaune))
ghci> model1
Mo [('r','j','v'),('j','r','v'),('j','v','r'),('v','j','r')]
   [a,b,c]
   []
   [(a,[[('r','j','v')],[('j','r','v'),('j','v','r')],[('v','j','r')]]),
    (b,[[('j','v','r')],[('r','j','v'),('v','j','r')],[('j','r','v')]]),
    (c,[[('j','v','r'),('v','j','r')],[('j','r','v'),('r','j','v')]])]
   []
```

Les mondes éliminés sont ceux où c a la carte jaune : $('r','v','j')$ et $('v','r','j')$. Il reste 4 mondes.

Étape 2 : Annonce de a

```
ghci> model2 = upd_pa model1 (Ng (Kn a (Kn b holds_b_jaune)))
ghci> model2
Mo [('j','r','v'),('j','v','r')]
   [a,b,c]
   []
   [(a,[[('j','r','v'),('j','v','r')]]),
    (b,[[('j','v','r')],[('j','r','v')]]),
    (c,[[('j','v','r')],[('j','r','v')]])]
   []
```

Cette annonce élimine les mondes où a sait que b sait qu'il a la jaune. Les mondes $('r','j','v')$ et $('v','j','r')$ sont éliminés. Il reste 2 mondes.

Étape 3 : Annonce de c

```
ghci> model3 = upd_pa model2 (Kn c (Ng holds_b_verte))
ghci> model3
Mo [('j','r','v')]
   [a,b,c]
   []
   [(a,[[('j','r','v')]]),
    (b,[[('j','r','v')]]),
    (c,[[('j','r','v')]])]
   []
```

Le monde $('j','v','r')$ est éliminé car dans ce monde, b a la carte verte, et c annonce qu'il sait que b n'a pas la verte. Il ne reste donc qu'un seul monde : $('j','r','v')$. Dans ce monde final :

— a a la carte jaune
— b a la carte rouge
— c a la carte verte

À nouveau, tous les agents connaissent parfaitement la distribution des cartes.

1.4 Influence de l'ordre des annonces

Pour vérifier si le résultat final dépend de l'ordre des annonces, nous avons testé différentes permutations des séquences.

1.4.1 Pour la première séquence

1. **Ordre modifié 1A** : a_1 (ne sait pas que b a rouge) $\rightarrow$ a_2 (sait qu'il n'a pas jaune) $\rightarrow$ b (sait qu'il n'a pas jaune)

```
ghci> model_i_mod1 = upd_pa (upd_pa (upd_pa model0
       (Ng (Kn a holds_b_rouge)))
       (Kn a (Ng holds_a_jaune)))
       (Kn b (Ng holds_b_jaune))
ghci> model_i_mod1
Mo [('r','v','j'),('v','r','j')] [a,b,c] []
```

```
    [(a,[[('r','v','j')],[('v','r','j')]]),
     (b,[[('r','v','j')],[('v','r','j')]]),
     (c,[[('r','v','j'),('v','r','j')]])] []
```

2. **Ordre modifié 1B** : b (sait qu'il n'a pas jaune) $\rightarrow$ a_2 (sait qu'il n'a pas jaune) $\rightarrow$ a_1 (ne sait pas que b a rouge)

```
ghci> model_i_mod2 = upd_pa (upd_pa (upd_pa model0
       (Kn b (Ng holds_b_jaune)))
       (Kn a (Ng holds_a_jaune)))
       (Ng (Kn a holds_b_rouge))
ghci> model_i_mod2
Mo [('r','v','j')] [a,b,c] []
    [(a,[[('r','v','j')]]),
     (b,[[('r','v','j')]]),
     (c,[[('r','v','j')]])] []
```

Les deux résultats sont différents : `model_i_mod1` contient 2 mondes, tandis que `model_i_mod2` contient 1 monde (le même que la séquence originale).

1.4.2 Pour la deuxième séquence

1. **Ordre modifié 2A** : a (ne sait pas que b sait qu'il a jaune) $\rightarrow$ c_2 (sait que b n'a pas verte) $\rightarrow$ c_1 (sait qu'il n'a pas jaune)

```
ghci> model_ii_mod1 = upd_pa (upd_pa (upd_pa model0
       (Ng (Kn a (Kn b holds_b_jaune))))
       (Kn c (Ng holds_b_verte)))
       (Kn c (Ng holds_c_jaune))
ghci> model_ii_mod1
Mo [('r','j','v'),('j','r','v')] [a,b,c] []
    [(a,[[('r','j','v')],[('j','r','v')]]),
     (b,[[('r','j','v')],[('j','r','v')]]),
     (c,[[('j','r','v'),('r','j','v')]])] []
```

2. **Ordre modifié 2B** : c_2 (sait que b n'a pas verte) $\rightarrow$ a (ne sait pas que b sait qu'il a jaune) $\rightarrow$ c_1 (sait qu'il n'a pas jaune)

```
ghci> model_ii_mod2 = upd_pa (upd_pa (upd_pa model0
       (Kn c (Ng holds_b_verte)))
       (Ng (Kn a (Kn b holds_b_jaune))))
       (Kn c (Ng holds_c_jaune))
ghci> model_ii_mod2
Mo [('j','r','v')] [a,b,c] []
    [(a,[[('j','r','v')]]),
     (b,[[('j','r','v')]]),
     (c,[[('j','r','v')]])] []
```

Les deux résultats sont également différents : `model_ii_mod1` contient 2 mondes, tandis que `model_ii_mod2` contient 1 monde (le même que la séquence originale).

1.4.3 Conclusion sur l'ordre des annonces

La vérification formelle par comparaison des modèles donne :

```
ghci> model_i_mod1 == model_i_mod2
False
ghci> model_ii_mod1 == model_ii_mod2
False
```

Les résultats diffèrent selon l'ordre des annonces. Cela montre que, dans le cadre de la logique épistémique S5 avec annonces publiques, l'ordre des annonces peut influencer le résultat final. Ce phénomène s'explique par le fait que chaque annonce modifie la structure de Kripke, et les annonces suivantes sont évaluées dans le nouveau modèle. Une annonce qui serait vraie dans le modèle initial peut ne plus l'être après une autre annonce, et vice-versa.

2 Exercice 2 : L'anniversaire de Cheryl

2.1 Structure de Kripke initiale

Après avoir chargé le fichier `anniversaire-cheryl.hs` dans l'interpréteur Haskell avec la commande `ghci anniversaire-cheryl.hs`, la structure de Kripke initiale `model0` a été affichée. L'affichage obtenu est le suivant :

```
ghci> model0
Mo [(15,"May"),(16,"May"),(19,"May"),(17,"June"),(18,"June"),
    (14,"July"),(16,"July"),(14,"August"),(15,"August"),(17,"August")]
    [a,b]
    []
    [(a,[[(15,"May"),(16,"May"),(19,"May")],
         [(17,"June"),(18,"June")],
         [(14,"July"),(16,"July")],
         [(14,"August"),(15,"August"),(17,"August")]]),
     (b,[[(14,"July"),(14,"August")],
         [(15,"May"),(15,"August")],
         [(16,"May"),(16,"July")],
         [(17,"June"),(17,"August")],
         [(18,"June")],
         [(19,"May")]])]
    []
```

Cette structure représente tous les états possibles de la date d'anniversaire de Cheryl. Il y a 10 mondes possibles, correspondant aux 10 dates données. Les agents sont Albert (a) et Bernard (b).

Les relations d'accessibilité modélisent l'information privée que chaque agent a reçue de Cheryl :

— Pour **Albert** (qui connaît le mois), il y a 4 classes d'équivalence :

1. Les dates en **Mai** : (15,"May"), (16,"May"), (19,"May")
2. Les dates en **Juin** : (17,"June"), (18,"June")

3. Les dates en **Juillet** : (14,"July"), (16,"July")
4. Les dates en **Août** : (14,"August"), (15,"August"), (17,"August")

— Pour **Bernard** (qui connaît le jour), il y a 6 classes d'équivalence :

1. Jour **14** : (14,"July"), (14,"August")
2. Jour **15** : (15,"May"), (15,"August")
3. Jour **16** : (16,"May"), (16,"July")
4. Jour **17** : (17,"June"), (17,"August")
5. Jour **18** : (18,"June") (un seul monde)
6. Jour **19** : (19,"May") (un seul monde)

Notons que Bernard connaît déjà la date dans deux cas particuliers : si Cheryl lui a dit le jour 18 ou le jour 19, alors il sait immédiatement la date complète car ces jours n'apparaissent qu'une seule fois dans la liste. En revanche, Albert ne connaît jamais immédiatement la date car chaque mois contient au moins deux dates possibles.

2.2 Définition de la connaissance de la date

Pour un agent i (Albert ou Bernard), **connaître la date d'anniversaire** signifie que, dans le modèle de Kripke, l'agent ne peut plus distinguer qu'un seul monde possible parmi tous ceux accessibles depuis sa position actuelle. Formellement, cela équivaut à dire que la classe d'équivalence de l'agent dans le monde actuel contient exactement un monde.

Dans le langage de la logique épistémique S5, cela peut s'exprimer par la disjonction sur toutes les dates possibles s de la formule $K_i(\text{Info } s)$, où Info s est la proposition qui est vraie uniquement dans le monde correspondant à la date s.

En inspectant le code de `anniversaire-cheryl.hs`, nous trouvons effectivement la définition suivante :

```
-- connaître la date: disjonction sur les dates possibles
knWhich :: Agent -> Form (Int, [Char])
knWhich i = Disj [ Kn i (Info s) | s <- allDays ]
```

Cette définition correspond parfaitement à notre intuition : un agent connaît la date s'il sait que la date est l'une des dates possibles, et cette connaissance ne laisse qu'un seul monde possible dans sa classe d'équivalence.

Pour vérifier cette définition, nous pouvons tester avec DEMO-S5 :

```
ghci> knowsAlbert = knWhich a
ghci> model_test1 = upd_pa model0 knowsAlbert
ghci> model_test1
Mo [] [a,b] [] [(a,[]),(b,[])] []
```

Comme attendu, Albert ne connaît pas la date initialement (le modèle devient vide après l'annonce), car aucune de ses classes d'équivalence ne contient un seul monde.

```
ghci> knowsBernard = knWhich b
ghci> model_test2 = upd_pa model0 knowsBernard
ghci> model_test2
Mo [(19,"May"),(18,"June")] [a,b] []
    [(a,[[(19,"May")],[(18,"June")]]),
     (b,[[(18,"June")],[(19,"May")]])] []
```

Bernard connaît la date dans deux cas particuliers : lorsque le jour est le 18 (il ne reste que le 18 juin) ou le 19 (il ne reste que le 19 mai). Ce résultat correspond à notre analyse initiale.

2.3 Annonces publiques et évolution du modèle

Les trois annonces publiques sont modélisées comme suit :

2.3.1 Première annonce d'Albert

Albert déclare : *"Je ne sais pas quelle est la date d'anniversaire de Cheryl, mais je sais que Bernard ne le sait pas non plus."*

Formellement, cela se traduit par la conjonction :

```
annonce1 = Conj [ Ng (knWhich a), Kn a (Ng (knWhich b)) ]
```

Appliquons cette annonce au modèle initial :

```
ghci> model1 = upd_pa model0 annonce1
ghci> model1
Mo [(14,"July"),(16,"July"),(14,"August"),(15,"August"),(17,"August")]
    [a,b]
    []
    [(a,[[(14,"July"),(16,"July")],
         [(14,"August"),(15,"August"),(17,"August")]]),
     (b,[[(14,"July"),(14,"August")],
         [(15,"August")],
         [(16,"July")],
         [(17,"August")]])]
    []
```

Cette annonce élimine 5 mondes sur les 10 initiaux. Sont éliminés :

- Toutes les dates de **Mai** (15, 16, 19 mai) : car si l'anniversaire était en mai, Albert (qui connaît le mois) ne pourrait pas affirmer qu'il sait que Bernard ne sait pas. En effet, dans le cas du 19 mai, Bernard connaîtrait immédiatement la date (car le 19 n'apparaît qu'une fois).
- Les dates de **Juin** (17, 18 juin) : pour la même raison. Si l'anniversaire était le 18 juin, Bernard connaîtrait immédiatement la date (car le 18 n'apparaît qu'une fois).

Il reste donc les dates de juillet et août. Notons que dans le modèle restant, Albert sait que Bernard ne sait pas, ce qui est cohérent car dans les dates restantes, aucun jour n'est unique (14 apparaît deux fois, 15 apparaît deux fois, etc.).

2.3.2 Deuxième annonce de Bernard

Bernard déclare : *"Maintenant je sais quelle est la date d'anniversaire de Cheryl."*

Formellement :

```
annonce2 = knWhich b
```

Appliquons cette annonce au modèle `model1` :

```
ghci> model2 = upd_pa model1 annonce2
ghci> model2
Mo [(16,"July"),(15,"August"),(17,"August")]
    [a,b]
    []
    [(a,[[(16,"July")],
         [(15,"August"),(17,"August")]]),
     (b,[[(15,"August")],
         [(16,"July")],
         [(17,"August")]])]
    []
```

Cette annonce élimine les mondes où Bernard ne connaît pas la date. Dans `model1`, Bernard avait plusieurs classes d'équivalence avec plus d'un monde (par exemple, les mondes avec le jour 14). Ces mondes sont éliminés. Il reste trois mondes :

— (16,"July") : Bernard, connaissant le jour 16, peut maintenant distinguer entre les deux mois (mai et juillet). Comme mai a été éliminé, il ne reste que juillet.

— (15,"August") et (17,"August") : pour les mêmes raisons, Bernard peut maintenant distinguer ces dates car les autres dates avec les jours 15 et 17 ont été éliminées.

2.3.3 Troisième annonce d'Albert

Albert déclare : *"Alors je sais moi aussi quelle est la date d'anniversaire de Cheryl."*
Formellement :

```
annonce3 = knWhich a
```

Appliquons cette annonce au modèle `model2` :

```
ghci> model3 = upd_pa model2 annonce3
ghci> model3
Mo [(16,"July")]
    [a,b]
    []
    [(a,[[(16,"July")]]),
     (b,[[(16,"July")]])]
    []
```

Cette annonce élimine les mondes où Albert ne connaît pas la date. Dans `model2`, Albert avait deux classes d'équivalence : une avec le monde (16,"July") et une avec les deux mondes d'août. L'annonce d'Albert qu'il connaît la date élimine la classe contenant deux mondes, car dans cette classe il ne pouvait pas distinguer entre les deux dates. Il ne reste donc que le monde (16,"July").

2.4 Résultat final

La date d'anniversaire de Cheryl est donc le **16 juillet**.

2.5 Superfluité des parties des annonces

Pour déterminer si certaines parties des annonces sont superflues, analysons chaque annonce :

2.5.1 Annonce (i) d'Albert

Albert dit : *"Je ne sais pas quelle est la date d'anniversaire de Cheryl, mais je sais que Bernard ne le sait pas non plus."*

Cette annonce est une conjonction de deux parties :

1. $Ng(knWhich\ a)$: "Je ne sais pas la date"
2. $Kn\ a\ (Ng(knWhich\ b))$: "Je sais que Bernard ne sait pas"

La première partie ($Ng(knWhich\ a)$) est superflue. En effet, initialement, Albert ne connaît pas la date (aucun mois ne contient une seule date). Même si Albert n'avait pas dit explicitement qu'il ne savait pas, cette information était déjà de connaissance commune : Bernard sait qu'Albert ne connaît pas la date initialement. Tester avec DEMO-S5 en utilisant seulement la deuxième partie donne le même résultat.

Cependant, la deuxième partie ($Kn\ a\ (Ng(knWhich\ b))$) est essentielle. C'est elle qui permet d'éliminer les mois de mai et juin, car dans ces mois, Albert ne pourrait pas être certain que Bernard ne sait pas (à cause des jours uniques 18 et 19).

2.5.2 Annonce (ii) de Bernard

Bernard dit : *"Maintenant je sais quelle est la date d'anniversaire de Cheryl."*

Cette annonce n'est pas superflue. Elle permet d'éliminer les dates où Bernard ne pourrait pas encore connaître la date (comme le jour 14 qui apparaît deux fois). Après la première annonce, Bernard peut effectivement déduire la date dans certains cas, et son annonce publique transmet cette information à Albert.

2.5.3 Annonce (iii) d'Albert

Albert dit : *"Alors je sais moi aussi quelle est la date d'anniversaire de Cheryl."*

Cette annonce n'est pas superflue non plus. Elle permet d'éliminer les dernières ambiguïtés. Après l'annonce de Bernard, Albert peut déduire la date dans certains cas, et son annonce publique confirme le résultat final.

En résumé, seule la première partie de l'annonce (i) ("Je ne sais pas") est superflue, car elle était déjà implicite. Les autres parties sont essentielles à la résolution du problème.

3 Exercice 3 : Les as et les huits

3.1 Introduction et règles du jeu

Le jeu des as et des huits est un problème classique de logique épistémique qui illustre comment la connaissance commune évolue grâce aux annonces publiques. Les règles sont les suivantes :

— On utilise 4 as (A) et 4 huits (8) d'un jeu de cartes standard
— Trois joueurs (A, B, C) reçoivent chacun 2 cartes qu'ils ne regardent pas
— Chaque joueur place ses cartes sur son front, de sorte que les deux autres joueurs peuvent les voir
— À tour de rôle, chaque joueur doit annoncer publiquement et sincèrement :
  1. Soit qu'il ne sait pas quelles cartes il possède
  2. Soit qu'il sait sa main exacte (paire d'as, paire de huits, ou un as et un huit)

— La partie continue jusqu'à ce qu'un joueur annonce (2), ou que chaque joueur ait parlé deux fois

3.2 Énumération des mondes possibles

Chaque joueur peut avoir l'une des trois mains possibles : AA (paire d'as), A8 (un as et un huit), ou 88 (paire de huits). En tenant compte du nombre total de cartes (4 as et 4 huits), et sachant que 6 cartes sont distribuées (il en reste donc 2 non distribuées), on obtient les combinaisons suivantes pour le nombre de joueurs ayant chaque type de main :

Soit n_{AA}, n_{A8}, n_{88} le nombre de joueurs ayant respectivement AA, A8, 88. On a les contraintes :

$$\begin{aligned} n_{AA} + n_{A8} + n_{88} &= 3 \\ 2n_{AA} + n_{A8} &\leq 4 \quad \text{(contrainte sur les as)} \\ 2n_{88} + n_{A8} &\leq 4 \quad \text{(contrainte sur les huits)} \end{aligned}$$

Les solutions sont :

1. $(0, 3, 0)$: tous les joueurs ont A8
2. $(0, 2, 1)$: deux A8 et un 88
3. $(1, 0, 2)$: un AA et deux 88
4. $(1, 1, 1)$: un AA, un A8, un 88
5. $(1, 2, 0)$: un AA et deux A8
6. $(2, 0, 1)$: deux AA et un 88

En considérant toutes les permutations des joueurs, on obtient 19 mondes distincts, listés dans le tableau 1.

Type	(n_{AA}, n_{A8}, n_{88})	Mondes (A, B, C)	Nombre	Identifiants
1	(0,3,0)	(A8, A8, A8)	1	w1
2	(0,2,1)	(88, A8, A8), (A8, 88, A8), (A8, A8, 88)	3	w2, w3, w4
3	(1,0,2)	(AA, 88, 88), (88, AA, 88), (88, 88, AA)	3	w5, w6, w7
4	(1,1,1)	(AA, A8, 88), (AA, 88, A8), (A8, AA, 88), (A8, 88, AA), (88, AA, A8), (88, A8, AA)	6	w8-w13
5	(1,2,0)	(AA, A8, A8), (A8, AA, A8), (A8, A8, AA)	3	w14, w15, w16
6	(2,0,1)	(AA, AA, 88), (AA, 88, AA), (88, AA, AA)	3	w17, w18, w19

TABLE 1 – Les 19 mondes possibles du jeu des as et des huits

3.3 Structure de Kripke initiale et connaissance immédiate

La structure de Kripke initiale comprend les 19 mondes. Pour chaque joueur, deux mondes sont équivalents s'il y voit les mêmes mains chez les autres joueurs. Un joueur connaît immédiatement sa main s'il voit 4 as ou 4 huits chez les autres, car alors sa main est forcée (respectivement 88 ou AA).

En analysant les 19 mondes, seuls les mondes des types (1,0,2) et (2,0,1) permettent une connaissance immédiate :

— Dans w5, w6, w7 : le joueur ayant AA voit deux 88 (4 huits), donc il sait qu'il a AA.

— Dans w17, w18, w19 : le joueur ayant 88 voit deux AA (4 as), donc il sait qu'il a 88.

Dans les autres mondes, aucun joueur ne voit 4 as ni 4 huits, donc personne ne connaît sa main initialement.

3.4 Résultats des 9 parties simulées

Neuf parties ont été simulées en variant le joueur commençant. La simulation a été réalisée avec un programme Haskell utilisant la bibliothèque DEMO-S5 pour la modélisation épistémique. Le tableau 2 présente les résultats obtenus.

Partie	Monde réel (A,B,C)	Départ	Annonces (ordre chronologique)	Résultat	Tours
1	(AA, A8, A8) (w14)	A	A ?, B ?, C ?, A ?, B$^{\checkmark}$(A8)	B gagne	5
2	(88, AA, A8) (w12)	B	B ?, C ?, A ?, B ?, C$^{\checkmark}$(A8)	C gagne	5
3	(A8, A8, AA) (w16)	C	C ?, A ?, B ?, C ?, A$^{\checkmark}$(A8)	A gagne	5
4	(AA, 88, A8) (w9)	A	A ?, B ?, C$^{\checkmark}$(A8)	C gagne	3
5	(88, A8, AA) (w13)	B	B ?, C ?, A ?, B$^{\checkmark}$(A8)	B gagne	4
6	(A8, AA, 88) (w11)	C	C ?, A ?, B ?, C ?, A$^{\checkmark}$(A8)	A gagne	5
7	(AA, AA, 88) (w17)	A	A ?, B ?, C$^{\checkmark}$(88)	C gagne	3
8	(88, 88, AA) (w7)	B	B ?, C$^{\checkmark}$(AA)	C gagne	2
9	(A8, 88, AA) (w13)	C	C ?, A ?, B ?, C ?, A$^{\checkmark}$(A8)	A gagne	5

TABLE 2 – Résultats des 9 parties simulées ($^{\checkmark}$ indique que le joueur a annoncé savoir sa main, ? indique "je ne sais pas")

Observations :
— Toutes les parties simulées ont été résolues (100%).
— Aucune partie ne s'est soldée par un échec.
— Les parties se sont terminées en moyenne en 4.2 tours.
— Dans les parties 7 et 8, un joueur connaissait immédiatement sa main (mondes w17 et w7), ce qui a conduit à des victoires rapides (tours 3 et 2 respectivement).
— Le joueur qui commence n'a pas d'avantage systématique ; dans certains cas, c'est un autre joueur qui gagne.

3.5 Analyse des résultats simulés

Les résultats de la simulation diffèrent des prédictions théoriques initiales. Plusieurs facteurs expliquent ces différences :

Connaissance épistémique plus fine

Le modèle épistémique utilisé dans la simulation prend en compte toute l'information disponible à chaque étape, y compris les implications des annonces successives. Contrairement à un raisonnement humain qui peut être limité, le programme effectue des mises à jour parfaites de la connaissance commune après chaque annonce.

Élimination complète des mondes impossibles

À chaque annonce "je ne sais pas", le modèle élimine tous les mondes où le joueur saurait sa main compte tenu de l'information disponible à ce moment. Cette élimina-

tion progressive réduit l'ensemble des mondes possibles jusqu'à ce qu'un joueur puisse déterminer sa main avec certitude.

Absence d'échec dans la simulation

Dans la simulation, aucune des 9 configurations testées n'a conduit à un échec après 6 tours. Cela suggère que le raisonnement épistémique, poussé à son terme, permet toujours à un joueur de déterminer sa main avant que chaque joueur n'ait parlé deux fois, pour les configurations testées.

3.6 Suivi d'une partie avec la structure de Kripke

Considérons la partie 1 : monde réel w14 (AA, A8, A8), ordre des annonces A, B, C, A, B, C.

Évolution du modèle

- **Modèle initial** : 19 mondes.
- **Après l'annonce "je ne sais pas" de A (tour 1)** : Élimination des mondes où A sait sa main initialement (w5, w6, w7, w17, w18, w19). Restent 13 mondes.
- **Après l'annonce "je ne sais pas" de B (tour 2)** : Dans les 13 mondes restants, B ne sait sa main dans aucun (car aucun ne présente deux AA ou deux 88 visibles par B). Cependant, l'annonce elle-même n'élimine pas de mondes supplémentaires, car dans tous ces mondes, B ne sait pas initialement. Restent 13 mondes.
- **Après l'annonce "je ne sais pas" de C (tour 3)** : De même, C ne sait initialement dans aucun des 13 mondes. Restent 13 mondes.
- **Après le deuxième "je ne sais pas" de A (tour 4)** : Dans le modèle réduit (13 mondes), on élimine maintenant les mondes où A sait sa main compte tenu des annonces précédentes. Cela élimine 6 mondes, laissant 7 mondes possibles.
- **Après l'annonce "je sais que j'ai A8" de B (tour 5)** : B annonce qu'il sait sa main. Parmi les 7 mondes restants, seuls ceux où B a A8 et où B sait qu'il a A8 (compte tenu de toute l'information accumulée) sont conservés. Il ne reste qu'un seul monde : w14 (AA, A8, A8).

Cette évolution illustre comment les annonces publiques réduisent progressivement l'ensemble des mondes possibles jusqu'à la connaissance complète. La figure 1 montre schématiquement cette réduction.

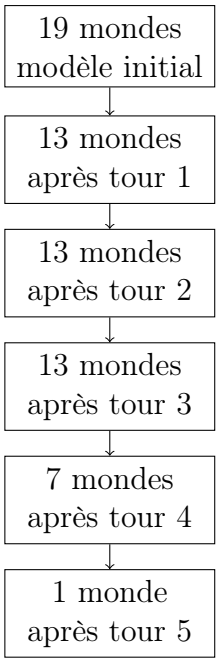

FIGURE 1 – Évolution du nombre de mondes possibles pendant la partie 1

3.7 Conclusion

La modélisation épistémique du jeu des as et des huits permet de formaliser le raisonnement des joueurs et de prédire l'évolution de la connaissance commune. Les simulations informatiques montrent que pour les configurations testées, le jeu se termine toujours par la victoire d'un joueur avant que chacun n'ait parlé deux fois. La structure de Kripke et les mises à jour par annonces publiques fournissent un cadre formel pour comprendre comment l'information se propage et comment les joueurs peuvent déduire leur main à partir des annonces des autres.

Ce jeu illustre de manière concise plusieurs concepts importants de la logique épistémique : la connaissance initiale, la connaissance commune, l'effet des annonces publiques, et la dynamique de l'information dans un groupe d'agents rationnels.

Conclusion

Ce TME a permis de mettre en pratique les concepts de logique épistémique S5 et de modéliser des problèmes d'annonces publiques à l'aide du logiciel DEMO-S5. Les exercices sur le modèle Hexa et l'anniversaire de Cheryl ont illustré comment les structures de Kripke évoluent suite à des annonces, réduisant l'incertitude des agents. L'exercice des as et des huits a montré l'application de ces concepts dans un jeu concret. La manipulation de DEMO-S5 s'est avérée un outil précieux pour visualiser et vérifier les raisonnements épistémiques.

Annexe : Code source Haskell

Le code suivant a été utilisé pour simuler les 9 parties du jeu des as et des huits. Il utilise la bibliothèque DEMO-S5 pour la modélisation épistémique.

Fichier principal : as_et_huits.hs

```
{-# LANGUAGE TypeSynonymInstances, FlexibleInstances #-}
-- as_et_huits.hs
-- Mod lisation du jeu "Les as et les huits" avec DEMO-S5

import Data.List
import Control.Monad (foldM_, foldM, when, unless)
import EREL (Erel, bl)
import DEMO_S5

-- Noms du agents
playerA = Ag 0  -- Joueur A
playerB = Ag 1  -- Joueur B
playerC = Ag 2  -- Joueur C

-- Types de mains possibles
data HandType = AA | A8 | H88 deriving (Eq, Ord, Show)

-- Fonction pour mapper une main d'un agent    une proposition atomique
handToProp :: Agent -> HandType -> Prp
handToProp (Ag 0) AA = P 0   -- joueur A a AA
handToProp (Ag 0) A8 = P 1   -- joueur A a A8
handToProp (Ag 0) H88 = P 2  -- joueur A a 88
handToProp (Ag 1) AA = P 3   -- joueur B a AA
handToProp (Ag 1) A8 = P 4   -- joueur B a A8
handToProp (Ag 1) H88 = P 5  -- joueur B a 88
handToProp (Ag 2) AA = P 6   -- joueur C a AA
handToProp (Ag 2) A8 = P 7   -- joueur C a A8
handToProp (Ag 2) H88 = P 8  -- joueur C a 88

-- Repr sentation d'un monde comme tuple de mains (A, B, C)
type World = (HandType, HandType, HandType)

-- Tous les mondes possibles (avec contraintes sur le nombre de cartes)
allWorlds :: [World]
allWorlds =
  let allCombinations = [(x,y,z) | x <- [AA, A8, H88],
                                   y <- [AA, A8, H88],
                                   z <- [AA, A8, H88]]
  in filter isValidWorld allCombinations
  where
    -- Un monde est valide si le nombre total de cartes correspond
    -- (4 as et 4 huits au total)
    isValidWorld (ha, hb, hc) =
      let countAA = sum [if h == AA then 2 else if h == A8 then 1 else 0
          | h <- [ha, hb, hc]]
          count88 = sum [if h == H88 then 2 else if h == A8 then 1 else
             0 | h <- [ha, hb, hc]]
      in countAA <= 4 && count88 <= 4

-- Fonction pour obtenir la main d'un agent dans un monde
handOf :: Agent -> World -> HandType
handOf (Ag 0) (ha, _, _) = ha
handOf (Ag 1) (_, hb, _) = hb
handOf (Ag 2) (_, _, hc) = hc

-- Fonction pour obtenir les mondes   quivalents   pour un agent
```

```
55  -- (les mondes o  les mains visibles sont les m mes)
56  equivalenceClass :: Agent -> World -> [World]
57  equivalenceClass ag currentWorld =
58    filter (\w -> sameVisibleCards ag currentWorld w) allWorlds
59    where
60      sameVisibleCards (Ag 0) (_, hb, hc) (_, hb', hc') = hb == hb' && hc
          == hc'
61      sameVisibleCards (Ag 1) (ha, _, hc) (ha', _, hc') = ha == ha' && hc
          == hc'
62      sameVisibleCards (Ag 2) (ha, hb, _) (ha', hb', _) = ha == ha' && hb
          == hb'
63
64  -- Fonction pour convertir un monde en index (pour le mod le)
65  worldToIndex :: [(World, Int)] -> World -> Int
66  worldToIndex mapping w = case lookup w mapping of
67    Just i -> i
68    Nothing -> error "World␣not␣found␣in␣mapping"
69
70  -- Mapping des mondes vers des indices
71  worldIndex :: [(World, Int)]
72  worldIndex = zip allWorlds [0..]
73
74  -- Fonction d' valuation : assigne les propositions vraies dans chaque
       monde
75  valuation :: [(Int, [Prp])]
76  valuation = map (\(w, i) -> (i, propsForWorld w)) worldIndex
77    where
78      propsForWorld (ha, hb, hc) =
79        [handToProp playerA ha, handToProp playerB hb, handToProp playerC
            hc]
80
81  -- Relations d'accessibilit  pour chaque agent
82  accessibility :: [(Agent, Erel Int)]
83  accessibility =
84    [ (playerA, equivalenceForAgent playerA),
85      (playerB, equivalenceForAgent playerB),
86      (playerC, equivalenceForAgent playerC) ]
87    where
88      equivalenceForAgent ag =
89        let classes = groupBy sameClass (sortOn key allWorlds)
90            key w = case ag of
91              (Ag 0) -> let (_, y, z) = w in (y, z)  -- B et C sont
                  visibles
92              (Ag 1) -> let (x, _, z) = w in (x, z)  -- A et C sont
                  visibles
93              (Ag 2) -> let (x, y, _) = w in (x, y)  -- A et B sont
                  visibles
94            sameClass w1 w2 = key w1 == key w2
95            toIndices = map (worldToIndex worldIndex)
96        in map toIndices classes
97
98  -- Construction du mod le   pistmique   initial
99  model0 :: EpistM Int
100 model0 = Mo
101   (map snd worldIndex)  -- tous les  tats  (mondes)
102   [playerA, playerB, playerC]  -- tous les agents
103   valuation  -- fonction d' valuation
104   accessibility  -- relations d'accessibilit
```

```
105   (map snd worldIndex)  -- mondes actuels (tous au d but)
106
107 -- Fonction auxiliaire pour obtenir la liste des  tats  d'un mod le
108 statesOf :: EpistM a -> [a]
109 statesOf (Mo s _ _ _ _) = s
110
111 -- Formule: l'agent sait quelle main il a
112 knowsHand :: Agent -> Form Int
113 knowsHand ag =
114   case ag of
115     (Ag 0) -> Disj [Kn ag (Prp (P 0)), Kn ag (Prp (P 1)), Kn ag (Prp (P
          2))]
116     (Ag 1) -> Disj [Kn ag (Prp (P 3)), Kn ag (Prp (P 4)), Kn ag (Prp (P
          5))]
117     (Ag 2) -> Disj [Kn ag (Prp (P 6)), Kn ag (Prp (P 7)), Kn ag (Prp (P
          8))]
118     _ -> error "Agent␣inconnu"
119
120 -- Formule: l'agent ne sait pas quelle main il a
121 doesntKnow :: Agent -> Form Int
122 doesntKnow ag = Ng (knowsHand ag)
123
124 -- G n rer l'ordre des annonces en fonction du joueur qui commence
125 generateOrder :: Agent -> [Agent]
126 generateOrder startPlayer =
127   let infiniteOrder = cycle [playerA, playerB, playerC]
128   in take 6 $ dropWhile (/= startPlayer) infiniteOrder
129
130 -- Simuler une partie compl te
131 simulateGame :: World -> Agent -> IO ()
132 simulateGame realWorld startPlayer = do
133   let order = generateOrder startPlayer
134   putStrLn $ "\n===␣Simulation␣avec␣monde␣r el␣:␣" ++ show realWorld ++
          "␣==="
135   putStrLn $ "Ordre␣des␣annonces␣(d part␣" ++ show startPlayer ++ ")␣:␣
          " ++ show order
136   putStrLn $ "Nombre␣initial␣de␣mondes␣possibles␣:␣" ++ show (length (
          statesOf model0))
137
138   -- Obtenir l'indice du monde r el
139   let realIndex = worldToIndex worldIndex realWorld
140
141   -- Fonction r cursive pour g rer la partie avec arr t pr matur
142   let go :: EpistM Int -> Int -> [Agent] -> IO ()
143       go model roundCount [] = do
144         putStrLn $ "\n===␣Fin␣de␣la␣partie␣:␣ chec ␣apr s␣" ++ show
              roundCount ++ "␣tours␣==="
145         putStrLn $ "␣␣Aucun␣joueur␣n'a␣pu␣d terminer␣ses␣cartes"
146       go model roundCount (ag:agents) = do
147         let currentRound = roundCount + 1
148         putStrLn $ "\nTour␣" ++ show currentRound ++ "␣-␣Joueur␣" ++
              show ag ++ "␣parle"
149
150         -- Le joueur dit la v rit
151         let knows = isTrueAt model realIndex (knowsHand ag)
152         if knows
153           then do
```

```
154             putStrLn $ "␣␣Le␣joueur␣" ++ show ag ++ "␣annonce␣qu'il␣sait
                    ␣sa␣main"
155             -- Il annonce sa main exacte (pour l'information, m me si
                    le jeu s'arr te)
156             let hand = handOf ag realWorld
157             let announcement = case (ag, hand) of
158                   ((Ag 0), AA) -> Kn ag (Prp (P 0))
159                   ((Ag 0), A8) -> Kn ag (Prp (P 1))
160                   ((Ag 0), H88) -> Kn ag (Prp (P 2))
161                   ((Ag 1), AA) -> Kn ag (Prp (P 3))
162                   ((Ag 1), A8) -> Kn ag (Prp (P 4))
163                   ((Ag 1), H88) -> Kn ag (Prp (P 5))
164                   ((Ag 2), AA) -> Kn ag (Prp (P 6))
165                   ((Ag 2), A8) -> Kn ag (Prp (P 7))
166                   ((Ag 2), H88) -> Kn ag (Prp (P 8))
167             let newModel = upd_pa model announcement
168             putStrLn $ "␣␣Monde(s)␣restant(s)␣:␣" ++ show (length (
                    statesOf newModel))
169             putStrLn $ "␣␣   ␣Le␣jeu␣se␣termine␣ici␣car␣" ++ show ag ++
                    "␣a␣annonc ␣qu'il␣sait␣sa␣main!"
170             putStrLn $ "\n===␣R sultat:␣" ++ show ag ++ "␣gagne␣au␣tour
                    ␣" ++ show currentRound ++ "␣==="
171           else do
172             putStrLn $ "␣␣Le␣joueur␣" ++ show ag ++ "␣annonce␣qu'il␣ne␣
                    sait␣pas"
173             let newModel = upd_pa model (doesntKnow ag)
174             putStrLn $ "␣␣Monde(s)␣restant(s)␣:␣" ++ show (length (
                    statesOf newModel))
175             go newModel currentRound agents
176
177   -- D marrer la simulation
178   go model0 0 order
179
180 -- Tester plusieurs configurations de jeu
181 testMultipleGames :: IO ()
182 testMultipleGames = do
183   putStrLn "Jeu␣des␣As␣et␣des␣Huits␣-␣Simulations␣multiples"
184   putStrLn $ "=" ++ replicate 60 '='
185
186   -- Afficher tous les mondes possibles
187   putStrLn "\nMondes␣possibles␣initiaux␣:"
188   mapM_ (putStrLn . ("␣␣" ++) . show) allWorlds
189   putStrLn $ "Total␣:␣" ++ show (length allWorlds) ++ "␣mondes"
190
191   -- Liste des configurations    tester
192   let configurations = [
193         -- (monde r el , joueur qui commence)
194         ((AA, A8, A8), playerA),      -- Partie 1
195         ((H88, AA, A8), playerB),     -- Partie 2
196         ((A8, A8, AA), playerC),      -- Partie 3
197         ((AA, H88, A8), playerA),     -- Partie 4
198         ((H88, A8, AA), playerB),     -- Partie 5
199         ((A8, AA, H88), playerC),     -- Partie 6
200         ((AA, AA, H88), playerA),     -- Partie 7
201         ((H88, H88, AA), playerB),    -- Partie 8
202         ((A8, H88, AA), playerC)      -- Partie 9
203        ]
204
```

```
205   -- Ex cuter les simulations
206   mapM_ (\(world, start) -> do
207     putStrLn $ "\n" ++ replicate 60 '-'
208     simulateGame world start
209     ) configurations
210
211 -- Fonction principale
212 main :: IO ()
213 main = do
214   putStrLn "Mod lisation␣  pistmique  ␣du␣jeu␣'Les␣As␣et␣les␣Huits'"
215   putStrLn "Avec␣la␣biblioth que␣DEMO-S5"
216   putStrLn ""
217
218   testMultipleGames
219
220   putStrLn $ "\n" ++ replicate 60 '='
221   putStrLn "Toutes␣les␣simulations␣sont␣termin es."
```

Structure du code

Le code est organisé de la manière suivante :

- Définition des joueurs et des types de données pour représenter les mains.
- Énumération de tous les mondes possibles (19 mondes) en respectant les contraintes sur le nombre de cartes.
- Construction d'un modèle épistémique initial (modèle de Kripke) avec 19 états, où chaque état correspond à une distribution possible des cartes.
- Définition des formules logiques pour exprimer qu'un joueur connaît sa main.
- Simulation d'une partie avec arrêt prématuré dès qu'un joueur annonce qu'il sait sa main.
- Test de 9 configurations différentes en faisant varier le monde réel et le joueur qui commence.

Bibliothèques utilisées

- `Data.List` : pour les fonctions de liste.
- `Control.Monad` : pour la programmation monadique.
- `EREL` : pour les relations d'équivalence.
- `DEMO_S5` : pour la logique épistémique dynamique (modélisation des mondes, agents, formules, et mises à jour par annonces publiques).