Lambda-calcul et programmation fonctionnelle : Projet — Épisode 2

Due : le dimanche 19 mars 2023 23:59 CET

1 L'histoire

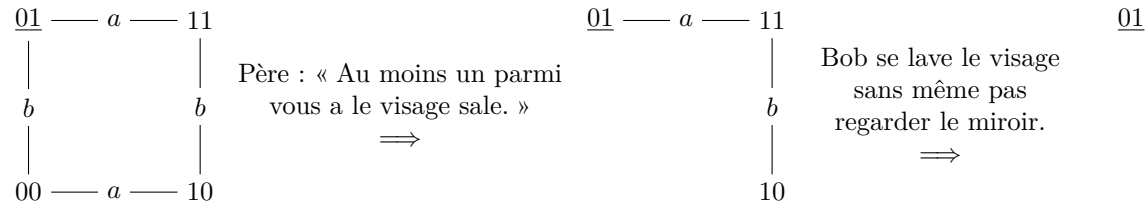
> *Un groupe d'enfants qui joue à l'extérieur est appelé à rentrer à la maison par leur père. Les enfants se réunissent au tour de lui. Comme souvent, quelques enfants se sont salis. Certaines ont même de la boue sur leurs visages. Chaque enfant peut voir le visage des autres enfants, mais ne peut pas voir son propre visage. Tout cela est connaissance commune, et les enfants sont, évidement, des logiciens parfaits. Le père maintenant leur dit : « Au moins un parmi vous a le visage sale. » Et ensuite il dit, « Celui qui sait si son propre visage est sale, fait un pas en avant. » Si personne ne fait un pas en avant, le père répète la réquisition. Après la répétition de cette réquisition plusieurs fois, tous le enfants sales font un pas en avant.*

Ceci est un scénario intrigant. Pourquoi les enfants ne font pas un pas en avant tout de suite ? Pourquoi ils le font après un certain nombre de réquisitions du père ? Ce comportement vient du fait que les enfants, qui sont des logiciens parfaits, ont raisonné logiquement sur les informations reçues. Pour mieux comprendre comment ils ont raisonné, nous allons analyser un problème plus simple.

> **Problème 1.** *Alice et Bob viennent de jouer à l'extérieur. Bob a de la boue sur son visage. Alice a le visage propre. Chacun d'eux peut voir le visage de l'autre, mais ne peut pas voir son propre visage. Leur père maintenant leur dit : « Au moins un parmi vous a le visage sale. » Bob part laver son visage sans même pas regarder le miroir, car il sait que son visage est sale. Comment a-t-il conclu cela ?*

Ici, la réponse est plus facile. Le père a dit à Alice et Bob qu'un des deux était sale. Bob peut voir le visage d'Alice, qui est propre. Donc, il a conclu que son propre visage est sale. Bien sûr, nous faisons l'hypothèse que les deux enfants sont des logiciens parfaits et que leur père dit toujours la vérité.

Nous pouvons représenter le problème 1 avec un graphe, comme dans les figures ci-dessous. (Les prochains paragraphes expliquent les figures.)



Concentrons nous d'abord sur le graphe à gauche. Les sommets, libellés par des nombres binaires, représentent les mondes possibles. Les nombres binaires 00, 01, 10 et 11 représentent la valeur de vérité des deux propositions qui nous intéressent. Dans chaque nombre binaire :

— le premier chiffre représente la valeur de vérité de la proposition « Le visage d'Alice est sale. » ;

— le deuxième chiffre représente la valeur de vérité de la proposition « Le visage de Bob est sale. ».

Donc, dans le monde possible 00, les deux enfants sont propres, et dans le monde possible 11 les deux enfants sont sales. Le monde possible dont le label est souligné est le monde réel, c.-à-d., le monde où le visage d'Alice est propre et celui de Bob est sale.

Les arêtes représentent l'incertitude des enfants. Par exemple, les deux sommets 01 et 11 sont liées par une arête libellé a. Cela veut dire qu'Alice ne peut pas distinguer entre les mondes 01 et 11. Alice peut voir que le visage de Bob est sale, mais elle ne peut pas voir son propre visage. Donc, elle ne sais pas si son visage est propre ou sale. Pour Bob, la situation est analogue. Il ne peut pas voir son visage, mais il peut voir celle d'Alice. Donc, il ne peut pas distinguer entre les mondes 00 et 01.

Maintenant, nous pouvons nous demander pourquoi le monde 10, qui représente le cas où le visage d'Alice est sale et celui de Bob est propre, apparaît dans le graphe. Pour comprendre cela, il faut pousser la réflexion un peu plus loin et se demander ce que pense Alice à propos de ce que pense Bob, et vice-versa. Notons que, par exemple, Bob n'est pas sûr dans quelle monde possible il est vraiment. De son point de vue, il peut être

dans 01 ou bien dans 00. Dans le cas où il serait dans 00, Alice ne serait pas capable de distinguer entre 00 et 10. Donc, pour Bob, Alice peut imaginer qu'elle est dans 10. Donc, ce monde possible doit apparaître dans le graphe. De même manière, comme Alice ne sait pas si elle est dans 01 ou bien dans 11, il faut y avoir une arrête libellé b entre 11 et 10.

Maintenant, quand le père des enfants annonce « Au moins un parmi vous a le visage sale. », les deux enfants découvrent que le monde 00 n'est pas possible. Plus que cela, chacun sait que l'autre enfant le sait aussi. En effet, ceci devient connaissance commune. C'est la raison pour laquelle, après cet annonce, nous devons supprimer le monde 00 du graphe. Ce que nous amène au graphe du milieu. Dans ce graphe, Alice ne peut toujours pas distinguer entre 00 et 11, et Bob ne peut pas distinguer entre 10 et 11.

La dernière chose qui se passe dans l'histoire est que Bob lave son visage sans même pas regarder le miroir. Cela veut dire qu'il sait qu'il a le visage sale. Donc, Alice apprend qu'elle n'a pas le visage sale, car elle apprend dans quel monde possible ils sont dès le départ. S'ils était dans le monde 10, Bob n'aurait pas pu savoir que son visage est sale car, pour lui, le monde 10 serait toujours possible. La même chose pour le monde 11. Le seul cas où il sait que son visage est salle, est le monde possible 01.

2 Objectif

L'objectif de cet épisode est d'écrire un programme en Haskell capable représenter les problèmes proposés, ainsi que de réaliser le raisonnement logique nécessaire pour les analyser. Le programme sera composé d'un module (`EL.hs`) qui implémentera un raisonneur pour la *logique épistémique* (plus de détails par la suite) et un module (`ProblemN.hs`) pour chaque problème logique n que nous allons analyser.

Question 1. Les graphes que nous avons vu précédemment sont des exemples de ce que nous appelons *états épistémiques*. Une tel structure sert à représenter la connaissance d'un groupe d'agents. Dans le problème 1, les agents sont Alice et Bob.

Dans le module `EL` :

1. Définissez les trois types de base pour la construction des états épistémiques :

   `Prop` – chaîne de caractères
   Représente une proposition. Vous pouvez les écrire en toutes les lettres ou pas (par exemple : `"Le visage d'Alice est sale"`, `"Alice-sale"`, `"a-sale"`, `"as"`).

   `Agent` – chaîne de caractères
   Représente un agent (ex. : `"alice"`, `"A"`, `"a"`).

   `World` – entier (`Int`)
   Représente un monde possible (ex. : `00`, `01`).
   Note : Différemment de ce que nous avons fait dans l'épisode 1, ici les mondes possibles seront représentés par des entiers.

2. Définissez le type des états épistémiques `EpiState`. Ceci est un tuple de trois éléments :
   — Une fonction d'interprétation de type `Prop -> [World]`.
   — Une fonction d'indiscernabilité de type `Agent -> World -> [World]`.
   — Un monde possible qui représente le monde réel.
   La fonction d'interprétation doit être définie pour chaque problème. Elle prend une proposition en argument et renvoie la liste de mondes possibles où la proposition est vraie. Par exemple, dans le problème 1, nous pouvons utiliser la fonction suivante :

   ```
   interp :: Prop -> [World]
   interp "as" = [10,11]     -- Alice est sale dans les mondes 10 et 11.
   interp "bs" = [01,11]     -- Bob est sale dans les mondes 01 et 11.
   interp _    = []          -- Toutes les autres propositions sont fausses.
   ```

   Note : Nous avons utilisé les entiers 00, 01, 10 et 11 uniquement pour simplifier la visualisation de l'exemple. En effet, il est tout à fait possible de les remplacer par 0, 1, 2 et 3. Cela ne changerais rien à la modélisation.

   La fonction d'indiscernabilité doit être définie pour chaque problème. Elle prend un agent `i` et un monde possible `w` en arguments et renvoie la liste de mondes possibles qui sont indiscernables du monde `w` pour l'agent `i`. Par exemple, dans le problème 1, nous pouvons utiliser la fonction suivante :

```
indis :: Agent -> World -> [World]
indis "a" 00 = [00,10]      -- Alice ne peut pas distinguer 00 de 10.
indis "b" 00 = [00,01]      -- Bob ne peut pas distinguer 00 de 01.
...
```

Note : Alice ne peut pas distinguer 00 de 00, car il s'agit du même monde. Donc, 00 doit apparaître dans la liste des mondes indiscernables de 00. Un monde possible est indiscernable de lui même. Ceci est vrai pour tous les mondes possibles.

Finalement, l'état épistémique du problème 1 est donné par `(interp, indis, 01)`.

Question 2. Créez un fichier appelé `Problem1.hs`, définissez un module du même nom. Dans ce module, écrivez la définition complète de l'état épistémique initial du problème 1 (le graphe de gauche dans la figure), c.-à-d., l'expression :

```
s0 :: EpiState
```

Vous devez notamment compléter la définition de la fonction d'indiscernabilité.

Question 3. Dans le module `EL`, définissez le type `EpiFormula` des formules du langage de la *logique épistémique*. C'est-à-dire, le langage logique avec tous les opérateurs de la logique classique propositionnelle plus les deux opérateurs `Knows` et `After`, comme expliqué ci-dessous.

Pour pouvoir raisonner sur les états épistémiques, nous avons besoin d'un langage logique capable d'exprimer des phrases du type :

Alice sait que son propre visage est sale.
Alice ne sait pas que son propre visage est sale.
Alice sait que son propre visage est sale et que le visage de Bob est sale.
Alice sait que Bob ne sait pas que le visage d'Alice est sale.
Alice sait que Bob sait qu'Alice ne sait pas que son propre visage est sale.

Nous allons donc définir un langage qui contient tous les opérateurs logiques de la logique propositionnelle classique (c.-à-d., `T`, `F`, `Not`, `And`, `Or`, `Imp`, et `Eqv`), ainsi qu'un *opérateur de connaissance* `Knows` et un *opérateur dynamique* `After`. Les prochains paragraphes, expliquent ce deux derniers.

Soit `i` un agent et `phi` une formule, la construction :

```
Knows i phi
```

est une formule qui signifie « L'agent `i` sait que `phi`. » Par exemple, nous pouvons écrire :

```
Knows "a" (Var "as")
```

pour exprimer « Alice sait que le visage d'Alice est sale. » Voici comment écrire les autres phrases dans ce langage :

```
Not (Knows "a" (Var "as"))
Knows "a" (And (Var "as") (Var "bs"))
Knows "a" (Not (Knows "b" (Var "as")))
Knows "a" (Knows "b" (Not (Knows "a" (Var "as"))))
```

Mais notre langage logique n'est pas encore complet. Nous avons besoin d'un autre opérateur pour pouvoir exprimer des phrases du type :

Après l'annonce « Au moins un parmi vous a le visage sale. », Bob sait que le visage de Bob est sale.

Soit `phi` et `psi` deux formules, la construction :

```
After phi psi
```

est une formule qui signifie « Après l'annonce `phi`, `psi`. » Par exemple, la phase précédemment mentionnée peut être exprimée avec :

```
After (Or (Var "as") (Var "bs")) (Knows "b" (Var "bs"))
```

Obs. : Le type `EpiFormula` doit dériver de la classe `Show`, pour pouvoir s'afficher.

Question 4. Ajoutez dans le module `Problem1` les définitions des formules qui représentent ce problème, comme suit :

`fatherAnn :: EpiFormula`
Exprime l'annonce du père, c.-à-d., « Un parmi vous a le visage sale. »

`aliceIgn :: EpiFormula`
Exprime l'ignorance d'Alice sur son état, c.-à-d., « Alice ne sait pas si son propre visage est sale. »
Indice : Ceci est équivalent à : « Alice ne sait pas que le visage d'Alice est sale, et
Alice ne sait pas que le visage d'Alice n'est pas sale. »

`bobIgn :: EpiFormula`
Exprime l'ignorance de Bob sur son état.

`problem1 :: EpiFormula`
Exprime le problème 1 dans sa totalité, c.-à-d. :
« Les deux agents sont ignorants, et
Après l'annonce du père, Alice est ignorante, et
après l'annonce que Bob sait que son propre visage est sale, Alice et Bob ne sont pas ignorants. »
Note : L'annonce que Bob sait que son propre visage n'est pas sale est fait implicitement le moment où il part laver son visage sans même pas regarder le miroir.

Question 5. Écrivez les deux fonctions mutuellement récursives :

```
epiSat :: EpiState -> EpiFormula -> Bool
update :: EpiState -> EpiFormula -> EpiState
```

La fonction `epiSat` prend un état épistémique `s` et une formule `phi` en arguments et renvoie `True` si `s` satisfait `phi`, et `False` sinon. Pour réaliser cette tâche, dans le cas des formules du type `After`, la fonction `epiSat` appele la fonction `update`. La fonction `update` prend un état épistémique `s` et une formule `phi` et renvoie un nouvel état épistémique correspondant à la *mise à jour* de `s` par `phi`. Pour réaliser cette tâche, `update` appele la fonction `epiSat`. Les paragraphes suivants expliquent le fonctionnement de ces deux fonctions.

La satisfaction d'une formule `phi` par un état épistémique `s` est définie de manière récursive, comme suit. Par la suite, nous supposons que l'état épistémique est donné par `s = (interp, indis, w)`.

`T`. Cette formule est satisfaite par tous les états épistémiques.

`F`. Cette formule n'est satisfaite par aucun état épistémique.

`(Var p)`
Un état épistémique satisfait une proposition quand celle-ci est vraie dans le monde réel de cet état épistémique. Autrement dit, `s` satisfait `(Var p)` si et seulement si `w` appartient à l'interprétation de `p`.

`Not`, `And`, `Or`, `Imp` et `Eqv`
La satisfaction de ces formules est définie de manière standard. C'est-à-dire, par exemple, l'état épistémique `s` satisfait `(Not phi)` si et seulement si `s` ne satisfait pas `phi`. L'état `s` satisfait `(And phi psi)` si et seulement si, `s` satisfait `phi` et `s` satisfait `psi`. Les cas des opérateurs `Or`, `Imp` et `Eqv` sont analogues.

`(Knows a phi)`
L'état épistémique `s` satisfait la formule `(Knows a phi)` quand `phi` est vraie dans le monde réel de cet état et l'agent `a` n'a pas de doute sur la valeur de vérité de `phi`. L'agent `a` doute de la valeur de vérité de `phi` quand il y au moins un monde possible indiscernable du monde réel pour `a` où `phi` est fausse. Autrement dit, l'état `s` satisfait la formule `(Knows a phi)` si et seulement si, pour tous les mondes possibles `w'` qui sont indiscernables de `w` pour `a`, l'état épistemique formé par `(interp, indis, w')` satisfait `phi`.

Par exemple, dans l'état épistémique initial du problème 1 (le graphe de gauche dans la figure), la formule `(Knows "b" (Var "bs"))` est fausse, parce que la formule `(Var "bs")` est fausse dans un des mondes indiscernables du monde réel pour Bob, notamment 00. Pourtant, cette même formule est vraie dans le deuxième état épistemique du problème (le graphe du milieu), parce la formule `(Var "bs")` est vraie dans tous les mondes possibles indiscernables du monde réel pour Bob (ici, il y en a qu'un seul, le monde réel lui même 01).

`(After phi psi)`
L'état épistémique `s` satisfait la formule `(After phi psi)` quand `s` satisfait `phi` et, en plus, la formule `psi` est satisfaite par la *mise à jour* de `s` par `phi`. Cette mise à jour est réalisée par la fonction `update`.

Par exemple, dans le problème 1, la formule `(After fatherAnnonce (Knows "b" (Var "bs")))` est satisfaite par l'état épistémique initial (le graphe de gauche), parce que, une fois mis à jour, l'état épistémique en question (le graphe du milieu) satisfait `(Knows "b" (Var "bs"))`.

Comme vu dans le problème 1, après l'annonce fait par le père, les enfants apprennent qu'un des mondes n'est plus possible, et il est supprimé de l'état épistémique. Ceci est le cas parce que l'annonce faite par le père n'est pas vraie dans ce monde possible. La fonction `update` doit donc supprimer les mondes possibles où la formule de mise à jour `phi` n'est pas vraie. Autrement dit, `update s phi` renvoie un nouvel état épistémique `s' = (interp', indis', w)`, où les mondes possibles où la formule `phi` est fausse ont été supprimés. Les nouvelles fonctions d'interprétation et d'indiscernabilité sont donc modifiées de façon à ne plus renvoyer les mondes possibles qui ne satisfont pas la formule de mise à jour `phi`. Pour vérifier si un monde possible satisfait une formule, nous devons nous servir de la fonction `epiSat`, d'où la récursion mutuelle.

Question 6. Pour chaque problème n proposé ci-dessous, créez un fichier `ProblemN.hs` contenant un module du même nom. Dans ce module, définissez l'état épistémique initial du problème et les formules requises. L'état épistémique initial et les formules doivent modéliser correctement le problème. Notament, les formules doivent être satisfaites (ou pas) par l'état initial.

> **Problème 2.** *Comme avant, Alice et Bob jouent à l'extérieur. Les deux ont leurs visages sales. Ainsi comme avant, leur père les informe qu'au moins un des deux a le visage sale. Ensuite il dit, « Celui qui sait si son propre visage est sale, fait un pas en avant. », mais ni Alice ni Bob bougent. Il répète sa réquisition : « Celui qui sait si son propre visage est sale, fait un pas en avant. » Cette fois, Alice et Bob font un pas en avant en même temps.*

Pour le problème ci-dessus, définissez les expressions :
— `s0 :: EpiState`
— `fatherAnn :: EpiFormula`
— `aliceIgn :: EpiFormula`
— `bobIgn :: EpiFormula`
— `problem2 :: EpiFormula`

> **Problème 3.** *Alice, Bob et Caroline jouent à l'extérieur. Tous les trois ont leurs visages sales. Le père leur dit qu'au moins un des trois a le visage sale, et ensuite dit, « Celui qui sait si son propre visage est sale, fait un pas en avant. », mais rien ne se passe. Il répète cela deux fois. La troisième fois où il le dit, tous les trois font un pas en avant en même temps.*

Pour le problème ci-dessus, définissez les expressions (analogues au problème 1) :
— `s0 :: EpiState`
— `fatherAnn :: EpiFormula`
— `aliceIgn :: EpiFormula`
— `bobIgn :: EpiFormula`
— `carolineIgn :: EpiFormula`
— `problem3 :: EpiFormula`

> **Problème 4.** *Anne et Bill, écoutent le suivant : « Soit deux nombres naturels consecutifs entre* 0 *et* 4*. Je vais chuchoter un de ces deux nombres dans l'oreille de Anne et l'autre dans l'oreille de Bill. » Après cette action, Anne et Bill on la conversation suivante :*
> — *Anne : « Je ne connaît pas ton nombre. »*
> — *Bill : « Je ne connaît pas ton nombre. »*
> — *Anne : « Je connaît ton nombre. »*
> *D'abord ils ne connaissent pas les nombres, et ensuite ils les connaîsent.*

Pour le problème ci-dessus, définissez les expressions :

`s0 :: EpiState` L'état épistémique initial (après le chuchotement).

`anneIgn :: EpiFormula`
Exprime que « Anne ne connaît pas le nombre de Bill. »

`billIgnorance :: EpiFormula`
Analogue pour Bill.

`problem4 :: EpiFormula`
Exprime que le problème dans sa totalité, c.-à-d. :
« Anne ne connaît pas le nombre de Bill, et Bill ne connaît pas le nombre de Anne, et
Après que Anne annonce son ignorance,
et après que Bill annonce son ignorance,
Anne connaît le nombre de Bill. »

3 Ce que vous devez rendre

Vous devez rendre 6 fichiers :

`Readme:` Ce fichier contiendra (au moins) les noms et prénoms des intégrants de l'équipe du projet.

`EL.hs:` Contiendra le code source du module de même nom. Il s'agit du programme capable de raisonner sur des formules de la logique épistémique. Ce module doit exporter les objets listés ci-dessous. Leurs spécifications sont données dans les questions de la section 2.

- `Prop`
- `Agent`
- `World`
- `EpiState`
- `EpiFormula (..)`
- `epiSat` et `testEpiSat`
- `update` et `testUpdate`
- `testAll`

`Problem1.hs, ..., Problem4.hs:` Un fichier pour chaque problème. Chaque fichier contiendra la modélisation du n-ième problème (voir les sections suivantes). Chaque fichier doit exporter la formule `problemN` qui exprime le problème en question ainsi que l'état épistemique initial `s0`. Chacun de ces modules doivent importer le module `EL` pour pouvoir compiler.

4 Correction

La correction du projet sera faite automatiquement, de la même manière que pour l'épisode 1.

Observation : Ainsi que pour lépisode 1, les options GHC `-w` et `-fno-warn-...` (qui servent à éviter l'affichage des *warnings*) sont strictement interdites.