Université de Caen Normandie
L2-INFO & L2-MATH
Autre Paradigme
DM à rendre pour le 27 avril 2020

Application : les sorites de Lewis Carroll

Définitions

Un syllogisme est un raisonnement déductif formé de trois propositions, deux prémisses (la majeure et la mineure) et une conclusion, tel que la conclusion est déduite du rapprochement de la majeure et de la mineure.

Exemple de syllogisme :
- majeure : *Tous les hommes sont mortels.*
- mineure : *Socrate est un homme.*
- conclusion : *Socrate est mortel.*

Un sorite est un raisonnement déductif composé de plusieurs syllogismes, enchaînés entre eux de telle sorte que l'attribut de la majeure devienne le sujet de la mineure, l'attribut de la mineure le sujet de la proposition suivante, et ainsi jusquà la dernière proposition, dont l'attribut doit être combiné avec le sujet de la première.

Exemple 1. C'est à l'aide d'un sorite que les stoiciens démontraient que le sage est heureux (et ainsi que la sagesse suffit au bonheur). Voici ce raisonnement :

1. *qui est sage est tempérant*
2. *qui est tempérant est constant*
3. *qui est constant est sans trouble*
4. *qui est sans trouble est sans tristesse*
5. *qui est sans tristesse est heureux*

Donc *tout sage est heureux*, (i.e. la sagesse suffit au bonheur).

Considérons les propositions suivantes :
- `A` : être sage
- `B` : être tempérant
- `C` : être constant
- `D` : avoir des troubles
- `E` : être triste
- `F` : être heureux

et modélisons ce sorite sous forme d'une formule booléenne :

```
1 --  A => B
2 --  B => C
3 --  C => (non D)
4 --  (non D) => (non E)
5 --  (non E) => F
```

En prenant les formules dans l'ordre 1, 2, 3, 4 et 5, on obtient `A => F`, i.e. tout sage est heureux.

Raisonnement par contraposée. Soit P et Q deux propositions. La contraposée de la formule $(P \Rightarrow Q)$ est la formule $(\neg Q \Rightarrow \neg P)$. Ces deux formules sont équivalentes. Ainsi, pour prouver $(P \Rightarrow Q)$, on pourra prouver $(\neg Q \Rightarrow \neg P)$. C'est le raisonnement par contraposée.

Exemple : considérons P : *"il pleut"*, et Q : *"je prends mon parapluie"*. Alors la proposition "il pleut donc je prends mon parapluie" est équivalente à la proposition "je ne prends pas mon parapluie donc il ne pleut pas".

Le révérend Dodgson, plus connu sous le pseudonyme de Lewis Carroll, écrivain et logicien britannique (1832 – 1898) a proposé des énigmes dont il faut trouver la conclusion. Ces énigmes sont en fait des sorites dont il faut trouver la conclusion.

"Il ne faut pas se dépêcher de penser que la leçon ironique de Lewis Carroll logicien est une plaisanterie aimable et gratuite, et que des exemples de logique comme les siens, où la relation entre les termes est cocassement privilégiée par rapport au sens et au contenu des termes eux-mêmes, n'existent que dans le caprice de l'auteur d'Alice." (cf le document `sorites.pdf` sous eCampus.)

Exemple 2.

1. Quand je résous un exercice de logique sans ronchonner, vous pouvez être sûr que c'en est un que je comprends.
2. Ces sorites ne sont pas disposés régulièrement, comme les exercices auxquels je suis habitué.
3. Aucun exercice facile ne me donne jamais mal à la tête.
4. Je ne comprends pas les exercices qui ne sont pas disposés régulièrement, comme ceux auxquels je suis habitué.
5. Je ne ronchonne jamais devant un exercice, à moins qu'il ne me donne mal à la tête.

   Donc ...

Considérons les propositions suivantes :

— `A` : être un exercice qui me fait ronchonner
— `B` : être un exercice que je comprends
— `C` : être parmi ces sorites
— `D` : être disposé régulièrement, comme les exercices auxquels je suis habitué
— `E` : être un exercice facile
— `F` : être un exercice qui me donne mal à la tête.

et modélisons le sorite sous forme d'une formule :

```
1 --> (non A) => B      contraposee : (non B) => A
2 --> C => (non D)
3 --> E => (non F)      contraposee : F => (non E)
4 --> (non D) => (non B)
5 --> A => F
```

En prenant dans l'ordre 2, 4, (contraposée de 1), 5, (contraposée de 3), on obtient `C => (non E)`

```
C => non D => non B => A => F => non E
```

i.e. ces sorites sont des exercices difficiles (non faciles).

Architecture du DM

Nous procéderons en trois étapes :

1. Mise sous forme clausale de formules du calcul propositionnel (Partie 1 : énoncé à part)
2. Résolvante et principe de résolution (Partie 2 : énoncé à part)
3. Application des 2 premières étapes à la résolution de sorites

Dans cette dernière étape, nous étudions en détail 3 exemples différents. Puis, nous nous intéressons au calcul de la réponse (conclusion) d'un sorite en utilisant le principe de résolution.

Etude détaillée de l'exemple #1

```
-- 1 --  A => B
-- 2 --  B => C
-- 3 --  C => (non D)
-- 4 --  (non D) => (non E)
-- 5 --  (non E) => F
-- Deductions successives : A => B => C => (non D) => (non E) => F
-- donc : A => F
```

Traduction en Haskell

```
exemple1 = (Et (Imp (Var "A") (Var "B"))
             (Et (Imp (Var "B") (Var "C"))
               (Et (Imp (Var "C") (Non (Var "D")))
                  (Et (Imp (Non (Var "D")) (Non (Var "E")))
                      (Imp (Non (Var "E")) (Var "F"))))))
> formeClausale exemple1
Et (Ou (Non (Var "A")) (Var "B"))
   (Et (Ou (Non (Var "B")) (Var "C"))
      (Et (Ou (Non (Var "C")) (Non (Var "D")))
         (Et (Ou (Var "D") (Non (Var "E")))
            (Ou (Var "E") (Var "F")))))

> visuFormule (formeClausale exemple1)
"((~A v B) & ((~B v C) & ((~C v ~D) & ((D v ~E) & (E v F)))))"

> formeClausaleBis (formeClausale exemple1)
[ [Non (Var "A"),Var "B"], [Non (Var "B"),Var "C"], [Non (Var "C"),Non (Var "D")],
  [Var "D", Non (Var "E")], [Var "E",Var "F"]]

> deduire exemple1
[Non (Var "A"), Var "F"]
```

Etude détaillée de l'exemple #2

```
-- 1 --> (non A) => B      contraposee : (non B) => A
-- 2 --> C => (non D)
-- 3 --> E => (non F)      contraposee : F => (non E)
-- 4 --> (non D) => (non B)
-- 5 --> A => F
-- Deductions successives : C => non D => non B => A => F => non E
-- donc : C => (non E)
```

Traduction en Haskell

```
exemple2 = (Et (Imp (Non (Var "A")) (Var "B"))
             (Et (Imp (Var "C") (Non (Var "D")))
               (Et (Imp (Var "E") (Non (Var "F")))
                  (Et (Imp (Non (Var "D")) (Non (Var "B")))
                      (Imp (Var "A") (Var "F"))))))
> formeClausale exemple2
Et (Ou (Var "A") (Var "B"))
   (Et (Ou (Non (Var "C")) (Non (Var "D")))
      (Et (Ou (Non (Var "E")) (Non (Var "F")))
         (Et (Ou (Var "D") (Non (Var "B"))) (Ou (Non (Var "A")) (Var "F")))))

> visuFormule (formeClausale exemple2)
"(A v B) & (~C v ~D) & (~E v ~F) & (D v ~B) & (~A v F)"

> formeClausaleBis (formeClausale exemple2)
[ [Var "A", Var "B"], [Non (Var "C"), Non (Var "D")], [Non (Var "E"), Non (Var "F")],
  [Var "D", Non (Var "B")], [Non (Var "A"), Var "F"] ]

> deduire exemple2
[Non (Var "C"), Non (Var "E")]
```

Etude détaillée d'un 3ème exemple

1. Les bébés sont illogiques
2. Nul n'est méprisé quand il peut tuer un crocodile
3. Les gens illogiques sont méprisés

   Donc ...

Considérons les propositions :
- A : être un bébé
- B : être logique
- C : être méprisé
- D : tuer un crocodile

```
1 --> A => (non B)
2 --> D => (non C)       contraposee : C => (non D)
3 --> (non B) => C
-- Deductions successives : A => non B => C => non D
-- donc : A => (non D)
```

Traduction en Haskell

```
bebe = Et (Imp (Var "bebe") (Non (Var "logique")))
          (Et (Imp (Var "tuer crocodile") (Non (Var "meprise")))
              (Imp (Non (Var "logique")) (Var "meprise")))

> formeClausale bebe
Et (Ou (Non (Var "bebe")) (Non (Var "logique")))
   (Et (Ou (Non (Var "tuer crocodile")) (Non (Var "meprise")))
       (Ou (Var "logique") (Var "meprise")))

> formeClausaleBis (formeClausale bebe)
[ [Non (Var "bebe"), Non (Var "logique")],
  [Non (Var "tuer crocodile"), Non (Var "meprise")],
  [Var "logique", Var "meprise"]]

> deduire bebe
[Non (Var "bebe"), Non (Var "tuer crocodile")]
```

Obtenir la conclusion d'un sorite

La particularité des sorites est qu'ils peuvent être résolus en utilisant le principe de résolution : chaque formule ne doit servir qu'une et une seule fois.

En mettant chaque formule sous forme clausale et en partant d'une quelconque de ces clauses, en appliquant le principe de résolution en combinaison avec les autres clauses, on obtient la conclusion du sorite.

```
deduire :: Formule -> Clause
deduire x = resoudre (head sorite) (tail sorite)
   where sorite = (formeClausaleBis (formeClausale x))

resoudre :: Clause -> FormuleBis -> Clause
resoudre xs [] = xs
resoudre xs (ys:yss)
   | sontLiees xs ys = resoudre (resolvante xs ys) yss
   | otherwise       = resoudre xs (yss ++ [ys])
```

Vous pouvez directement utiliser ces 2 fonctions qui figurent déjà dans le fichier `init_All_DM.hs`