Prolog contre le Minotaure



4 mars 2019
Damien Morard

Final date: 24 mars 2019 à 23h59

Files to include: Dans votre dossier privé sur gitlab (où vous aurez déjà bien pull le cours, cf.: semantique/README.md), vous irez dans le dossier semantique/Exercices/TP1. Dans ce dossier vous devrez inclure:

- Un rapport au format **PDF**, de la forme TP1_nom_prenom.pdf
- Un fichier contenant le code Prolog, de la forme TP1_nom prenom.pl

Les TPs sont des travaux *personnels*, si deux solutions ont des similarités flagrantes les deux personnes auront 0.

La date et l'heure de rendu sont *strictes*, passer le délai d'une minute utilisera un joker (pour une journée supplémentaire). Une fois les deux jokers du semestres consommés et le délai dépassé, vous recevrez une note de 0. Bien entendu la date et l'heure de rendu sont toujours considérées au fuseau horaire de Genève.

Votre rapport contiendra des exemples d'applications de vos règles et des réponses aux questions posées. Photoshop Lightroom CC 2015.5. Adobe.

Attention à la présentation de votre travail, le rapport comptera pour une partie de la note. De même, les formats de vos fichiers (pdf pour le rapport ...) et la présentation du code (indentation, commentaire ...) sont très importants! Ces critères rentreront en compte dans l'évaluation de vos TPs et pourront vous faire perdre des points.

Au cours de ce TP, vous créerez des programmes Prolog. Vous devrez maitriser l'écriture de règles récursives, et la manipulation de listes. Vous utiliserez aussi des opérations de calcul sur les entiers.

Dans les sous-sols labyrinthiques de Battelle, le Minotaure demande tous les neuf ans sept étudiants en informatique. Il a depuis longtemps abandonné l'espoir d'obtenir aussi les sept jeunes informaticiennes commandées.

Cependant, la raréfaction d'étudiants ne permet pas de conserver cette tradition. Les différentes étapes de cette séance d'exercices nous permettront de







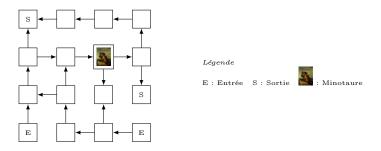


FIGURE 1 – Plan du labyrinthe

trouver le Minotaure, si possible le vaincre, et peut-être trouver la sortie des sous-sols.

Exercise 1 : Description du labyrinthe (1 points)



Le labyrinthe du sous-sol est une succession de pièces carrées, reliées entre elles par des portes. Une pièce a au maximum une porte par mur. Les portes ne s'ouvrent que dans un seul sens; tout retour en arrière est ainsi interdit. De plus, le labyrinthe ne contient pas de cycles; il est impossible de revenir dans une pièce après en être sorti.

- 1. Représentez le labyrinthe de la figure 1 dans une base de faits Prolog. Pour cela, vous remplirez la relation porte(De, Vers), qui indique les portes qui peuvent être ouvertes.
- 2. Le labyrinthe a plusieurs entrées, plusieurs sorties et un Minotaure. Représentez-les avec les relations entree(Piece), sortie(Piece) et minotaure(Piece).







Exercise 2: Chemins possibles (2.5 points)



1. Nous souhaitons tout d'abord déterminer s'il est possible d'aller d'une entrée au Minotaure, puis du Minotaure (vaincu) à une sortie. Écrivez une règle chemin(De, Vers) qui retourne vrai lorsqu'il existe un chemin de la pièce De à la pièce Vers. Cette règle doit être récursive, c'est-à-dire s'appeler elle-même, mais ne doit pas utiliser de listes.

Une règle peut s'écrire en plusieurs parties. Par exemple, le code ci-dessous retourne dans R la valeur de N!.

Plusieurs remarques sur ce code:

- $\mathbb{N} \, > \, \mathbb{0}$ permet de ne pas exécuter la règle pour les \mathbb{N} négatifs ou nuls.
- Le mot-clé is permet d'effectuer des calculs. Ainsi, X = 5 3 retourne X = 5 3 (l'expression telle quelle), tandis que l'expression X is 5 3 retourne X = 2 (l'expression calculée).
- 2. Écrivez la règle itineraire(De, Vers, Pieces) qui retourne dans Pieces une liste des pièces à parcourir pour aller de De à Vers.

Quelques exemples de manipulation de listes :

- Le code r1([]). unifie le paramètre de r1 avec la liste vide.
- Le code r1([1,2,3]). unifie le paramètre de r1 avec la liste commençant par 1, puis 2, puis 3.
- Le code r1([X|2,3]). unifie le paramètre de r1 avec la liste commençant par la valeur de X, suivie de [2,3].

Vous pouvez utiliser des expressions sur les listes pour en créer, mais aussi pour décomposer une liste en son premier élément, et le reste, comme dans [H|T] (Head, Tail).







Exercise 3: Survie dans le labyrinthe (2.5 points)



Nous envoyons le Prof. Buchs vaincre le Minotaure. Pour cela, il se repère dans le labyrinthe au moyen d'un smartphone à la pomme. Sa batterie s'épuise rapidement, à cause de la mauvaise qualité de l'appareil. Lorsque la batterie est vide, votre professeur est perdu.

 À chaque pièce parcourue, la batterie s'épuise d'une unité. Initialement, celle-ci est chargée de 15 unités.

Écrivez la règle batterie (Pieces, Batterie, Reste) qui retourne dans Reste la charge restante de la batterie après avoir parcouru les Pieces.

Testez votre programme en utilisant la règle ci-dessous, qui calcule les pièces franchies pour aller de De à Vers, vérifie que la Batterie est suffisamment chargée, et retourne dans Reste sa charge restante.

```
test_batterie(De, Vers, Batterie, Reste) :-
itineraire(De, Vers, Pieces),
batterie(Pieces, Batterie, Reste).
```

2. Écrivez la règle chemin_batterie(De, Vers, Batterie, Pieces, Reste) qui retourne dans Pieces un chemin allant de De à Vers, si la Batterie le permet, et retourne dans Reste sa charge restante. Pour l'instant, ne vérifiez pas que le chemin passe par le Minotaure, ni que De est une entrée et Vers une sortie.

Pour concaténer deux listes, utilisez l'opération suivante : append(Liste1, Liste2, ListeResultat).

3. Écrivez la règle chemin_reussite(Batterie, Pieces) qui retourne dans Pieces un chemin permettant de vaincre le Minotaure puis sortir. Attention à vérifier le chemin part d'une entrée et arrive à une sortie, et que la batterie est suffisante. Utilisez la règle chemin_batterie!

Testez votre règle avec différentes charges initiales de Batterie.







Exercise 4: Guerrier technophile (Bonus)



Le smartphone servira aussi à éclairer la pièce lors du combat contre le minotaure, ainsi qu'à tweeter le résultat du combat. L'éclairage coûte 5 unités d'énergie, et le tweet 2 unités.

- 1. Écrivez la règle reussite_complete (Batterie, Pieces) évaluée à vraie pour les chemins permettant de vaincre le Minotaure, tweeter et sortir. Batterie contiendra la charge initiale de la batterie.
 - Est-il possible pour le Prof. Buchs de vaincre le Minotaure, tweeter et sortir du labyrinthe avec une batterie chargée à 15?
 - Est-il possible d'obtenir à partir de cette règle toutes les charges initiales de batterie valides?
- 2. Écrivez la règle reussite_tweet(Batterie, Pieces) retournant vrai pour les chemins permettant de vaincre le Minotaure, tweeter le résultat de la bataille, mais pas de sortir.
 - Quels chemins permettent de vaincre le Minotaure, tweeter, mais pas de sortir avec une batterie de 15?

Écrivez vos faits et règles dans un fichier (par exemple tp1.pl), puis chargez-le dans l'interprète (par exemple en tapant [tp1]. (sans oublier le « . »). Vous pouvez ensuite tester vos règles en les appelant.

Afin d'observer l'exécution de vos règles, vous pouvez taper la commande trace. dans l'interprète Prolog, puis exécuter une règle.





