1 Exercice 1 - Prise en main de CLIPS

Question 2: *Essayer les commandes de base de l'interpréteur CLIPS et expliquer leur résultat*

- `(reset)`: permet d'effacer la base de faits et d'y insérer les faits initiaux `(initial-fact)`
- `(rules)`: renvoie la liste des règles définies dans le fichier CLIPS
- `(facts)`: renvoie la liste des faits de la base au moment de l'exécution
- `(run)`: lance l'exécution de l'algorithme RETE
- `(retract i)`: avec `i` entier permet de retirer de la base le fait d'indice `i`
- `(assert (<predicat> <x> <y>))`: permet de rentrer dans la base un nouveau fait

Question 3: *Tester d'autres commandes de l'interpréteur CLIPS :* `(watch facts)`, `(watch rules)`, `(clear)`.

- `(watch facts)`: permet d'afficher pas à pas l'insertion et la suppression des faits
- `(watch rules)`: permet de regarder pas à pas l'activation des règles
- `(watch clear)`: efface la base de faits et la base de règles. Entrer la commande `(run)` ne produit alors aucun nouveau fait, même après un `(reset)`

Question 5: *Que se passe-t-il si: l'on retire le fait* 0*, juste après* `(reset)` *et avant de lancer les inférences ? Pourquoi ? Que peut-on en déduire sur le rôle du fait 0 ?*

Le fait 0 correspondand au fait initial sur lequel se déroulent les règles de la base, le supprimer juste après un `(reset)` revient à ne laisser aucun faits dans la base sur lesquels faire tourner nos règles. La commande `(run)` ne produit alors rien de nouveau dans la base.

Question 6:

Création du fichier `famille-suite.clp`. Ajout de la règle `oncle_tante` réduite: un oncle ou une tante est le frère ou la soeur d'un parent. *(cf. fichier* `famille-suite.clp`*)*

Question 7:
Ajout de la règle `cousin_cousine`. *(cf. fichier* `famille-suite.clp`*)*

Question 8:

Création du fichier `famille-unique.clp`. Ajout de la règle `enfant_unique_bas`. *(cf. fichier* `famille-unique.clp`*)*

Apres avoir lancé les inférences, nous remarquons trois faits en particulier: `(enfant_unique_bad alain)`, `(enfant_unique_bad yves)` et `(enfant_unique_bad bob)`; or seul Yves est bien enfant unique. Cela s'explique par le fait que la stratégie par défaut de ce système de règles de production est la stratégie en profondeur (`depth`). Ainsi, le moteur cherche à appliquer des règles en partant des faits se trouvant le plus haut dans la pile. Dans le cas des nouveaux faits cités plus haut, la règle `enfant_unique_bad` est appelée sur Alain avant même que le système ait produit le fait que Luc et Alain sont frères, ce qui mène à cette erreur. Même chose pour Bob.

Question 9:

Ajout de la règle `enfant_unique_priorite`. *(cf. fichier* `famille-unique.clp`*)*

Cette fois-ci, nous définissons une priorité sur les règles en utilisant `salience`. Nous déclarons une salience négative pour la règle `enfant_unique_priorite` et aucune pour les autres règles, ce qui nous assure que les appels à `enfant_unique_priorite` se feront toujours en dernier. Maintenant nous avons seulement le fait `(enfant_unique_priorite yves)`, ce qui est vrai.

Question 10: *Proposer une autre solution pour maintenir une base de faits cohérente sans priorités.*

On pourrait rétracter le fait `enfant_unique` dans un fait `naissance`.

2 Exercice 3 - Diagnostic médical

Question 1:

Implémentation du système à base de connaissances *(cf. fichier* `diagnostic-medical.clp`*)*.

Question 2:

Ajout des faits initiaux dans la base de connaissances `(initial-fact)`. La maladie diagnostiquée est la rougeole.

Question 3: *Déterminer quelle est la stratégie de CLIPS pour choisir la règle à activer quand plusieurs règles sont applicables.*

En exécutant les inférences pas à pas avec `(run 1)` et `(agenda)`, nous nous rendons compte que CLIPS choisit à chaque tour la règle activable la plus haute dans la pile des règles activables.

Les règles les plus récemment activées sont placées au-dessus des règles de même priorité (même salience). Par exemple, si un fait f_a peut activer deux règles R_1 et R_2, et si un fait f_b peut activer R_3 et R_4, alors si on `assert` f_a avant f_b, R_3 et R_4 se trouveront au dessus de R_1 et R_2 dans l'agenda. Par contre, la position de R_1 par rapport à R_2 et celle de R_3 par rapport à R_4 seront arbitraires.

Il s'agit-là de la stratégie par défaut activée par CLIPS: la stratégie en profondeur `(depth)`. La commande `(get-strategy)` nous permet de bien confirmer cela.

Question 4: *CLIPS procède-t-il par chaînage avant ou par chaînage arrière ?*

CLIPS procède par chaînage avant: à partir de sa base de faits, il cherche tous les faits qu'il peut déduire de la base de règles.

Question 5: *Saturer la base de faits* `(run)` *pour déterminer tout ce qui peut être prouvé.*

Nous saturons la base de faits et trouvons en finalité les faits suivants:

```
f-0     (initial-fact)
f-1     (taches_rouges patient)
f-2     (peu_boutons patient)
f-3     (sensation_froid patient)
f-4     (forte_fievre patient)
f-5     (yeux_douloureux patient)
f-6     (amygdales_rouges patient)
f-7     (peau_pele patient)
f-8     (peau_seche patient)
f-9     (signe_suspect patient)
f-10    (rougeole patient)
f-11    (douleur patient)
f-12    (etat_febrile patient)
f-13    (eruption_cutanee patient)
f-14    (exantheme patient)
```

Question 6: *Repérer la maladie diagnostiquée et, à la main, faire un chaînage arrière. Comparer le nombre de règles utilisées avec le nombre de règles utilisées par CLIPS.*

CLIPS utilise 9 règles (en comptant l'initialisation de la base de faits). En effectuant un chaînage arrière à la main, nous trouvons qu'il est possible de n'utiliser que 2 règles.