Avrile Floro (n° étudiant : 22000086)

Algorithmique et structures de données 1

Le 27 décembre 2023

Table des matières

Algorithmique et structures de données 1

SE 12
Avrile Floro
Étudiante n°22000086

1 SE 12

I. Exercice

Programmer un système expert d'ordre zéro en Lisp, sur une assistance à l'utilisateur dans le domaine de votre choix à partir de données trouvées sur internet ; par exemple, le site https://www.mushroom.world/ vous permet de reconnaître un champignon (mais il y en a plein d'autres, en particulier sur les fleurs).

Envoyez-moi le système expert par mail avec le sujet « Exo SE 12 » (code et données compressées + PDF de commentaires sur vos difficultés ; citez vos sources sur internet).

Pour cet exercice, nous avons décidé de créer un système expert permettant à l'utilisateur d'identifier la race d'un chat.
Nous avons choisi ce sujet car nous aimons les chats, en ayant deux.

Nous avons utilisé des documents permettant de différencier les chats (source : https://www.rover.com/uk/blog/what-breed-is-my-cat/) et nous avons ensuite complété les informations manquantes grâce à des recherches sur internet pour obtenir le poids, le type de poils ou la couleur d'une race en particulier.

Nous avons préparé deux documents distincts afin de suivre les recommendations du cours. Le fichier `SE_12.lisp` contient le code alors que le fichier `SE_12_data.lisp` contient les variables globales et en particulier les propositions basiques et les règles. Nous allons présenter ces deux documents en détail.

1.1 Le fichier SE_12_data.lisp

Le fichier `SE_12_data.lisp` contient la liste des Individus. En l'espèce, puisqu'il s'agit d'un système expert de niveau 0, il n'y a qu'un seul individu dans notre liste, qui correspond à l'animal dont nous tentons d'identifier la race.

```
1  ; Nom ..... : SE_12_data.lisp
2  ; Role ..... : données pour le système expert
3  ; Auteur ..... : Avrile Floro
4  ; Version ..... : V0.1 du 22/12/23
5  ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6  ; Usage : lisp puis (load "SE_12.lisp") puis (chat)
7
8  ; NAME : Individus
9  ; USAGES : liste des individus
10 ; USER : initialisation_prop, init_conclu, put, get-basic-value, et,
11 ;   apply-conclusion, donne_resultats
12
13 (setq Individus '(Animal))
```

Nous avons utilisé 23 propositions élémentaires dans notre programme.
Les questions, auxquelles l'utilisateur peut répondre par `t`, `nil` ou `inconnu`, sont en lien avec le poids du chat, son pelage, sa queue, ses oreilles etc.

```
15 ; NAME : prop-basique
16 ; USAGES : liste des propriétés de base
17 ; USER : chat, initialisaiton_prop, pose_questions, get-basic-value
18
19 (setq prop-basique
20    '((est-vivant . "L'animal est-il vivant?")
21      (est-chat . "L'animal est-il un chat?")
```

```
22      (pese<4 . "Le chat pèse-t-il moins de 4 kilos?")
23      (pese-4-7 . "Le chat pèse-t-il entre 4 et 7 kilos?")
24      (pese>7 . "Le chat pèse-t-il plus de 7 kilos?")
25      (est-couleur-solide . "Le chat est-il de couleur solide")
26      (est-bicolore . "Le chat est-il bicolore?")
27      (est-ecaille-tortue . "Le chat est-il de couleur écaille de tortue?")
28      (est-calico . "Le chat est-il de couleur calico?")
29      (est-colorpoint . "Le chat est-il de couleur colorpoint?")
30      (est-tabby . "Le chat est-il de couleur tabby?")
31      (a-poil-court . "Le chat a-t-il le poil court?")
32      (a-poil-mi-long . "Le chat a-t-il le poil mi-long?")
33      (a-poil-long . "Le chat a-t-il le poil long?")
34      (a-poil-frisé . "Le chat a-t-il le poil frisé?")
35      (a-pas-de-poil . "Le chat est-il sans poil?")
36      (a-courtes-pattes . "Le chat a-t-il de courtes pattes?")
37      (a-tête-plate . "Le chat a-t-il la tête plate?")
38      (a-tête-fine . "Le chat a-t-il la tête fine?")
39      (a-queue-courte . "Le chat a-t-il une queue courte?")
40      (a-grandes-oreilles . "Le chat a-t-il de grandes oreilles?")
41      (a-oreille-courbée . "Le chat a-t-il les oreilles courbées?")
42      (a-oreille-pliée . "Le chat a-t-il les oreilles pliées?")
43      (a-oreille-de-lynx . "Le chat a-t-il des oreilles de lynx?") ) )
```

Nous avons établi 25 règles différentes. La première règle consiste en l'identification de l'animal comme un chat. Les 24 règles suivantes permettent l'identification de la race du chat.
Nous avons ajouté la première règle `chat` et nous l'avons introduit dans les règles suivantes pour illustrer la capacité de notre programme à fonctionner avec des règles en tiroir. C'est la seule instance de ce fonctionnement dans notre programme puisque les règles suivantes sont indépendantes les unes des autres car elles permettent d'identifier différentes races de chats.

Initialement, nous avions certaines races de chats qui partageaient les mêmes caractéristiques, néanmoins avec un peu d'approfondissement, nous avons été en mesure d'ajouter des caractéristiques supplémentaires permettant une identification unique de l'ensemble des chats.

```
46 ; NAME : Règles
47 ; USAGES : liste des règles
48 ; USER : chat, init_conclu, parcourt_regles, donne_resultats, put,
49 ;   test_realisation_regle, et, apply-conclusion, donne_resultats
50
51 (setq Règles
52  '((R_chat
53       (et (est-vivant X) (est-chat X))
54       (chat X) )
55    (R_Manx
56       (et (chat X) (pese-4-7 X) (a-poil-court X) (a-queue-courte X))
57       ; toutes les couleurs sont aceptées
58       (Manx X) )
59    (R_Cornish-Rex
60       (et (chat X) (pese<4 X) (a-poil-frisé X) (a-tête-fine X) (a-grandes-oreilles X))
61       (Cornish-Rex X) )
62    (R_Bobtail-Japonais
63       (et (chat X) (pese<4 X) (a-queue-courte X))
64       ; presque toutes les couleurs, poils longs ou courts
65       (Bobtail-Japonais X) )
66    (R_Siamois
67       (et (chat X) (pese-4-7 X) (est-colorpoint X) (a-poil-court X) (a-tête-fine X))
68       (Siamois X) )
69    (R_Maine-Coon
70       (et (chat X) (pese>7 X) (est-couleur-solide X) (a-poil-long X) (a-oreille-de-lynx X))
71       (Maine-Coon X) )
```

```
(R_Siberien
   (et (chat X) (pese-4-7 X) (a-poil-mi-long X) (a-oreille-de-lynx X))
   ; toutes les couleurs sont admises
   (Sibérien X) )
(R_Scottish-Fold
   (et (chat X) (pese-4-7 X) (est-couleur-solide X) (a-tête-plate X) (a-oreille-pliée X))
   (Scottish-Fold X) )
(R_Persan
   (et (chat X) (pese-4-7 X) (a-poil-long X) (a-tête-plate X))
   ; toutes les couleurs sont admises
   (Persan X) )
(R_Abyssin
   (et (chat X) (pese-4-7 X) (est-tabby X) (a-poil-court X) (a-tête-fine X))
   (Abyssin X) )
(R_Sphynx
   (et (chat X) (pese-4-7 X) (a-pas-de-poil X) (a-tête-fine X))
   (Sphynx X) )
(R_Himalayen
   (et (chat X) (pese>7 X) (est-colorpoint X) (a-poil-long X))
   (Himalayen X) )
(R_Birman
   (et (chat X) (pese-4-7 X) (est-colorpoint X) (a-poil-long X))
   (Birman X) )
(R_Highlander
   (et (chat X) (a-queue-courte X) (a-oreille-courbée X))
   ; toutes les couleurs sont autorisées et poils courts ou longs
   (Highlander X) )
(R_Elf
   (et (chat X) (pese-4-7 X) (a-pas-de-poil X) (a-oreille-courbée X))
   (Elf X) )
(R_Selkirk-Rex
   (et (chat X) (pese-4-7 X) (a-poil-frisé X) (a-tête-plate X))
   (Selkirk-Rex X) )
(R_Norvégien
   (et (chat X) (pese>7 X) (a-poil-mi-long X) (a-oreille-de-lynx X))
   (Norvégien X) )
(R_Balinais
   (et (chat X) (pese<4 X) (est-colorpoint X) (a-poil-mi-long X))
   (Balinais X) )
(R_Turkish-Van
   (et (chat X) (est-bicolore X) (a-poil-mi-long X) (a-oreille-de-lynx X))
   (Turkish-Van X) )
(R_Munchkin
   (et (chat X) (pese<4 X) (a-courtes-pattes X) (a-tête-plate X))
   (Munchkin X) )
(R_LaPerm
   (et (chat X) (pese<4 X) (a-poil-frisé X) (a-oreille-de-lynx X))
   (LaPerm X) )
(R_Devon-Rex
   (et (chat X) (pese-4-7 X) (a-poil-court X) (a-poil-frisé X) (a-grandes-oreilles X))
   ; toutes les couleurs sont autorisées
   (Devon-Rex X) )
(R_Chartreux
   (et (chat X) (pese-4-7 X) (est-couleur-solide X) (a-poil-court X))
   (Chartreux X) )
(R_Bambino
   (et (chat X) (pese<4 X) (a-pas-de-poil X) (a-courtes-pattes X) (a-grandes-oreilles X))
   (Bambino X) ) ) )
```

1.2 Le fichier SE_12.lisp

Le fichier `SE_12.lisp` est le fichier principal de notre programme expert. Il contient le code permettant son exécution.

Nous commençons, au sein du fichier `SE_12.lisp`, par charger le fichier `SE_12_data.lisp`.

```
1 ; Nom ..... : SE_12.lisp
2 ; Rôle ..... : Système expert pour id un chat
3 ; Auteur ..... : Avrile Floro
4 ; Version ..... : V0.1 du 22/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "SE_12.lisp") puis (chat)
7
8 (load "SE_12_data.lisp")
```

1.2.1 La fonction chat

La fonction `chat` est la fonction principale de notre programme. C'est la fonction que l'utilisateur devra appeler au top-level afin d'exécuter notre système expert. La fonction est également lancée automatiquement lors du chargement de notre fichier `SE_12.lisp`.

Nous avons réalisé très rapidement qu'il conviendrait, lors de chaque appel de la fonction `chat`, de réinitialiser l'ensemble des propositions basiques à `inconnu` et des conclusions à `nil`. Il s'agit de l'une des dernières choses que nous avons rajoutée à notre programme.
Cette partie n'a pas posé de problèmes particuliers, si ce n'est quelques essais avant de réussir à traiter la sous-liste désirée.
La réinitialisaiton des données du programme permet à l'utilisateur de pouvoir exécuter plusieurs fois à la suite la fonction `chat` sans avoir à recharger le fichier.

Puisque la fonction `chat` est la fonction principale du programme, elle permet également au programme de mettre à jour les `prop-basiques` d'après les réponses de l'utilisateur. Elle appelle à cette fin la fonction `pose_questions`.

La fonction `parcourt_regles` parcourt les règles afin de mettre à jour les conclusions, lorsque cela est nécessaire.

Enfin, la fonction `donne_resultats` donne le résultat du programme, c'est-à-dire affiche les conclusions dont le résultat est `t`.

```
10 ; NAME : chat, lance le programme d'id du chat
11 ; ARGS : none
12 ; USAGES : (chat)
13 ; GLOBALS : prop-basique, Règles
14 ; CALL : initialisation_prop, init_conclu, pose_questions, parcourt_regles, donne_resultats
15 ; USER : top level
16
17 (defun chat ()
18   (initialisation_prop prop-basique) ; initialise les propriétés à inconnu
19   (init_conclu Règles) ; initialise les conclusions à nil
20   (princ "Bonjour, je suis un système expert permettant d'identifier un chat.")
21   (terpri)
22   (pose_questions prop-basique)  ; pose les ? pour les prop basiques
23   (terpri)
24   (parcourt_regles Règles); on parcourt les règles pour modifier les conclusions
25   (terpri)
26   (princ "Résultat: ")
```

```
27   (donne_resultats Règles) )
```

1.2.2 La fonction initialisation_prop

La fonction `initialisation_prop` parcourt les `prop-basique`. Ces `prop-basique` sont des propriétés qui sont définies à `inconnu` pour l'unique `individu` de notre programme grâce à l'utilisation de la fonction `put`. Cette initialisation permet, lorsque le programme est lancé plusieurs fois d'affilée, de réinitialiser les données pour chaque appel.

```
30 ; NAME : initialisation_prop ; initialise les propriétés à inconnu
31 ; ARGS : prop-basique
32 ; USAGES : (initialisation_prop prop-basique)
33 ; GLOBALS : prop-basique, Animal
34 ; CALL : put
35 ; USER : chat
36
37 (defun initialisation_prop (prop-basique)
38   (cond
39     ((not prop-basique)) ; si on a parcouru toutes les prop, on s'arrête
40     (t (put 'Animal (caar prop-basique) 'inconnu)
41       ; on met la propriété à inconnu et on continue
42       (initialisation_prop (cdr prop-basique)) ) ) )
```

1.2.3 La fonction init_conclu

La fonction `init_conclu` parcourt la base de règles. Pour chaque conclusion, la fonction réinitialise (pour `Animal`) la propriété en lien avec la conclusion à `nil`.

```
45 ; NAME : init_conclu ; initialise les conclusions à nil
46 ; ARGS : Règles
47 ; USAGES : (init_conclu Règles)
48 ; GLOBALS : Règles, Animal
49 ; CALL : put
50 ; USER : chat
51
52 (defun init_conclu (regles)
53   (cond
54     ((not regles)) ; si on a parcouru toutes les conclusions, on s'arrête
55     (t (put 'Animal (caaddr (car regles)) 'nil)
56       ; on met la conclusion à nil et on continue
57       (init_conclu (cdr regles)) ) ) )
```

1.2.4 La fonction pose_question

La fonction `pose_question` parcourt les différentes propriétés élémentaires de `prop-basique` et appelle la fonction `get-basic-value` pour chacune de ces propriétés.

```
61 ; NAME : pose_questions ; parcourt les prop basiques appelle get-basic-value poiur les ?
62 ; ARGS :  prop-basique
63 ; USAGES : (pose_questions prop-basique)
64 ; GLOBALS : prop-basique
65 ; CALL : get-basic-value
66 ; USER : test_realisation_regle, pose_questions
67
68 (defun pose_questions (prop-basique)
69   (cond
70       ((not prop-basique)) ; si on a parcouru toutes les prop, on s'arrête
71       ((get-basic-value (caar prop-basique)) ; on pose la question
72         (pose_questions (cdr prop-basique)) ) ) ) ; on continue de parcourir les prop
```

1.2.5 La fonction put

La fonction `put` est tirée du cours et a été réutilisée à l'identique.

```
82 ; NAME : put ; met des valeurs dans les propriétés
83 ; ARGS :  symbole, attribut, valeur
84 ; USAGES : (put 'Animal conclusion t)
85 ; GLOBALS : Animal, Règles
86 ; CALL : none
87 ; USER : initialisation_prop, init_conclu, apply-conclusion, get-basic-value
88
89 (defun put (symbole attribut valeur) (setf (get symbole attribut) valeur) valeur)
```

1.2.6 La fonction get-basic-value

La fonction `get-basic-value` est tirée du cours, néanmoins nous l'avons largement modifiée.

Nous avons gardé le squelette de son fonctionnement, c'est-à-dire que si la valeur de la propriété élémentaire de notre individu n'est pas `inconnu`, alors on retourne sa valeur. En revanche, si la valeur de la propriété élémentaire n'a pas été définie, alors on affiche la question qui lui est associée.

La fonction garde trace de l'entrée utilisateur. Si l'entrée correspond à `t`, `nil` ou `inconnu`, alors la valeur de la propriété est modifiée grâce à la fonction `put`. Par ailleurs, nous avons prévu le traitement des cas lorsque l'entrée utilisateur n'est pas conforme aux attentes. Dans ce cas, l'utilisateur est prié de recommencer.

Par ailleurs, nous avons également modifié la fonction afin d'utiliser des `princ` et `terpri` permettant un affichage plus esthétique.

```
85 ; NAME : get-basic-value ; màj prop basiques d'après les réponse de l'utilisateur
86 ; ARGS :  prop &aux base read
87 ; USAGES : (get-basic-value (caar prop-basique))
88 ; GLOBALS : prop-basique, Animal
89 ; CALL : put
90 ; USER : test_realisation_regle, pose_questions
91
92 (defun get-basic-value (prop &aux base read)
93   (cond
94       ((not (eq (get 'Animal prop 'inconnu) 'inconnu)) (get 'Animal prop))
95         ; si la propriété n'est pas inconnue, on retourne sa valeur
```

```
96      ((setq base (assoc prop prop-basique))
97        ; sinon on affiche la question et on lit la réponse
98        (princ (cdr base))
99        (terpri)
100       (princ "(répondez par nil/t/inconnu)")
101       (terpri)
102       (setq read (read))
103       (cond
104             ; on met à jour la propriété en fonction de la réponse
105           ((eq read 't) (put 'Animal prop t))
106           ((eq read 'nil) (put 'Animal prop nil) t)
107           ((eq read 'inconnu) (put 'Animal prop 'inconnu))
108           (t (princ "Réponse invalide, veuillez recommencer.")
109             ; si la réponse est invalide, on recommence
110             (terpri) (get-basic-value prop)) ) ) ) )
```

1.2.7 La fonction parcourt_regle

La fonction `parcourt_regle` est appelée par la fonction `chat` après la fonction `pose_questions`. C'est-à-dire que lors de l'appel de la fonction `parcourt_regle`, toutes les propriétés élémentaires de notre individu ont une valeur (`t`, `nil` ou `inconnu`). Dès lors, nous pouvons tester chacune des règles afin de voir si les conditions de réalisation sont remplies.

La fonction `parcourt_regles` parcourt l'ensemble des règles et appelle pour chaque règle `test_realisation_regle`. Le comportement de `parcourt_regles` lorsque `test_realisation_regle` est vraie influence le comportement de notre système expert.
Initialement, lorsque nous avions des chevauchements parmi les propriétés de nos races de chat, nous n'appelions pas récursivement la fonction `parcourt_regles` si la fonction `test_realisation_regle` était vraie. Cela avait pour conséquence de seulement retourner la première règle réalisée. Ce comportement n'était néanmoins pas satisfaisant car il empêchait le fonctionnement en tiroir des différentes conclusions.
Une fois les propriétés de nos chats mieux définies, nous avons pu appeler récursivement la fonction `parcourt_regles` lorsque la fonction `test_realisation_regle` était réalisée. De cette sorte, on parcourt et on teste l'ensemble des règles de notre système expert.

```
114 ; NAME : parcourt_regles ; parcourt les règles et appelle test_realisation_regle
115 ; ARGS :  Règles
116 ; USAGES : (parcourt_regles Règles)
117 ; GLOBALS : Règles
118 ; CALL : test_realisation_regle
119 ; USER : chat
120
121 (defun parcourt_regles (regles)
122   (cond
123     ((not regles)) ; si on a parcouru toutes les règles, on s'arrête
124     ((test_realisation_regle (car regles)) (parcourt_regles (cdr regles)))
125       ; si la règle est vraie, on continue de parcourir les règles
126     (t (parcourt_regles (cdr regles)) ) ) )
127       ; sinon on continue de parcourir les règles
```

1.2.8 La fonction test_realisation_regle

La fonction `test_realisation_regle` vérifie si une règle est réalisée selon la sortie de `apply`. Si la règle est réalisée, alors la fonction appelle `apply-conclusion` pour mettre à jour les propriétés de notre individu.

```
130 ; NAME : test_realisation_regle ; vérifie la réalisation d'une règle et appelle
131 ; apply-conclusion si elle est vraie
132 ; ARGS :  Règles
133 ; USAGES : (test_realisation_regle (car regles)
134 ; GLOBALS : Règles
135 ; CALL : apply-conclusion
136 ; USER : parcourt_regles
137
138 (defun test_realisation_regle (regles)
139   (cond
140      ((not regles) nil) ; si on a parcouru toutes les règles, on s'arrête
141      ((apply (caadr regles) (cdadr regles))
142        ; on applique la fonction de la règle aux arguments de la règle
143        (apply-conclusion (caaddr regles)) ) ) )
144          ; on applique la conclusion de la règle
```

1.2.9 La fonction et

Le `et` est le seul connecteur que nous utilisons en l'espèce dans notre système expert. Nous avons dû créer une fonction `et` adaptée à notre usage car l'opérateur `and` proposé par Lisp ne fonctionnait pas avec `apply`.

En particulier, nous avons dû passer à la fonction `et` un argument `(&rest args)`, sinon nous n'arrivions pas à construire convenablement la fonction. Ainsi, la fonction `et` peut prendre un nombre variables d'arguments.

La fonction `et` a les mêmes propriétés que le `and`, à savoir si tous ses arguments ont été parcourus on retourne `t`. En revanche, si un argument n'est pas `t`, alors on retourne `nil`. De même, lorsqu'un argument de la fonction est `t`, on passe à l'argument suivant.

```
147 ; NAME : et ; vérifie si toutes les propriétés sont vraies
148 ; ARGS :  &rest args
149 ; USAGES : (et (est-vivant X) (est-chat X))
150 ; GLOBALS : Animal, Règles
151 ; CALL : none
152 ; USER : test_realisation_regle (via apply)
153
154 (defun et (&rest args)
155   (cond
156      ((not args))  ; si on a parcouru tous les args, retourne vrai
157      ((equal (get 'Animal (caar args)) t) (apply 'et (cdr args)))
158        ; si la propriété est t, on continue de parcourir les args
159      (t nil) ) )  ; sinon, on retourne faux
```

1.2.10 La fonction apply-conclusion

La fonction `apply-conclusion` met à jour les propriétés de notre individu. Lorsque les conditions d'application d'une règle sont remplies, alors on passe à `t` la propriété relative à la conclusion de cette règle pour notre individu.

```
162 ; NAME : apply-conclusion ; met la conclusion à t
163 ; ARGS :  conclusion
164 ; USAGES : (apply-conclusion (caaddr regles))
```

```
165 ; GLOBALS : Animal, Règles
166 ; CALL : put
167 ; USER : test_realisation_regle
168
169 (defun apply-conclusion (conclusion)
170   (put 'Animal conclusion t) ) ; met la conclusion à t
```

1.2.11 La fonction donne_resultats

La fonction `donne_resultats` permet l'affichage de l'espèce de chat.

Nous avons décidé d'afficher, lorsque toutes les règles ont été parcourues, un `Merci`, qui nous semblait plus de circonstance qu'un `t` ou un `nil`.

La fonction `donne_resultats` vérifie pour chacune des conclusions de l'ensemble de règles, si la propriété associée à la conclusion est à `t` dans notre individu. Si c'est le cas, la fonction imprime le contenu de la conclusion et continue de parcourir les règles et les conclusions suivantes.

```
173 ; NAME : donne_resultats ; affiche les résultats
174 ; ARGS :  Règles
175 ; USAGES : (donne_resultats Règles)
176 ; GLOBALS : Règles, Animal
177 ; CALL : none
178 ; USER : chat
179
180 (defun donne_resultats (regles)
181   (cond
182     ((not regles) 'Merci) ; si on a parcouru toutes les conclusions, on s'arrête
183     ((eq (get 'Animal (car (caddar regles))) t) ; si la conclusion est t
184       (princ (format nil "~a " (car (caddar regles)))) (donne_resultats (cdr regles)) )
185       ; on l'affiche et on continue de parcourir les conclusions
186     (t (donne_resultats (cdr regles))) ) ) ; sinon on continue de parcourir les conclusions
```

1.2.12 L'appel automatique de la fonction chat

Lors du chargement du fichier, la fonction `chat` sera automatiquement appelée.

```
190 (chat) ; on appelle la fonction chat
```

1.3 Exemples d'utilisation du programme

Pour lancer notre système expert, il convient de lancer lisp avec la commande `lisp`, puis de charger le fichier `SE_12.lisp` avec la commande `(load "SE_12.lisp")`. Ensuite la fonction `chat` est appelée automatiquement. Si on souhaite relancer plusieurs fois d'affilée la fonction `chat`, il convient alors de la rappeler avec la commande `(chat)` au toplevel.

Nous n'avons pas prévu de message particulier s'il n'y a aucune correspondance, le résultat apparaît simplement comme vide. Néanmoins, cela ne devrait pas se produire car notre système expert permet d'identifier une race de chat, hors les deux premières questions permettent de confirmer que l'animal est bien un chat.
Ainsi, si aucune race n'a pu être identifiée, alors le résultat apparaît comme : `Résultat: chat`.
En revanche, si l'animal testé n'était pas un chat, alors le résultat apparaît comme `Résultat:` (lorsque la fonction est lancée initialement via le chargement du fichier) ou `Résultat: Merci` (lorsque la fonction a été appelée au toplevel par l'utilisateur).

FIGURE 1 – Si on répond "t" aux deux premières questions (qui permettent de s'assurer qu'on veut identifier un chat vivant), alors le résultat obtenu est "chat".

FIGURE 2 – Lorsque le programme est exécuté via le chargement du fichier, si on répond "nil" à toutes les questions, on obtient une sortie vide.

FIGURE 3 – Lorsque le programme est exécuté au toplevel avec la commande (chat), si on répond "nil" à toutes les questions, on obtient un résultat vide puis "Merci" s'affiche.

Ainsi, nous avons pu constater qu'il y a une légère différence de comportement selon que la fonction `chat` est appelée lors du chargement du fichier ou au toplevel. Lorsque la fonction est exécutée lors du chargement du fichier, le `Merci` final ne s'affiche pas. En revanche, il s'affiche lorsque la fonction est appelée au toplevel. Nous ne nous attarderons pas plus sur ce fait.

Par ailleurs, nous avons prévu dans notre programme la gestion des entrées utilisateur non reconnues. Dans une telle situation, le programme demande à nouveau à l'utilisateur d'entrer une valeur valide.

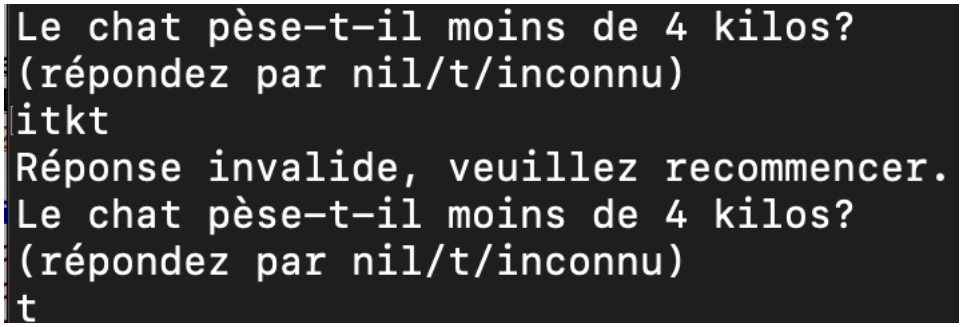

FIGURE 4 – Si l'entrée utilisateur n'est pas valide, l'utilisateur est invité à entrer de nouveau une commande valide.

Nous allons tester chacune de nos 24 espèces de chat en rentrant les caractéristiques de chaque espèce (telles qu'elles sont détaillées dans le fichier `SE_12_data.lisp`).

FIGURE 5 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Manx'.

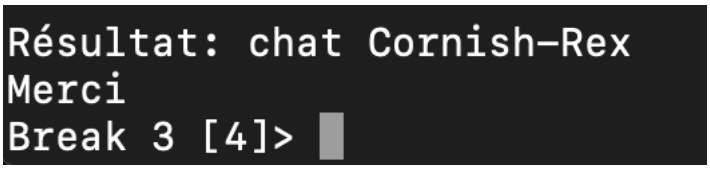

FIGURE 6 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Cornish-Rex'.

FIGURE 7 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Japonais-Bobtail'. Ici l'appel à la fonction (chat) s'est produit automatiquement lors du chargement initial du fichier.

FIGURE 8 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Siamois'. Ici l'appel à la fonction (chat) s'est produit automatiquement lors du chargement initial du fichier.

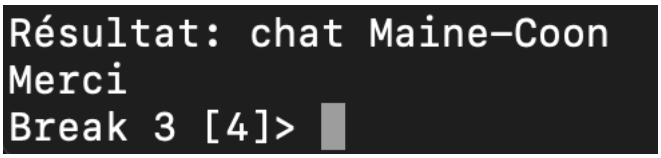

FIGURE 9 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Maine-Coon'.

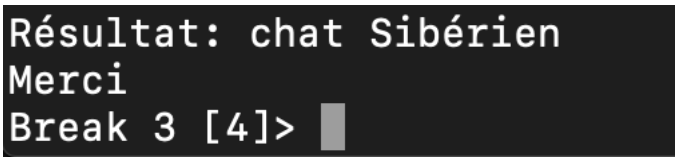

FIGURE 10 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Sibérien'.

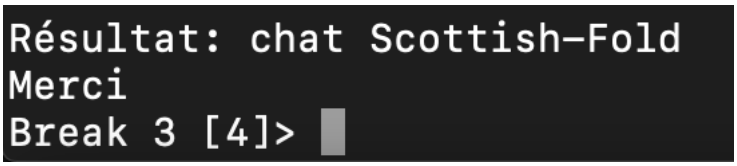

FIGURE 11 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Scottish-Fold'.

FIGURE 12 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Persan'.

FIGURE 13 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Abyssin'.

FIGURE 14 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Sphynx'.

FIGURE 15 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Himalayen'.

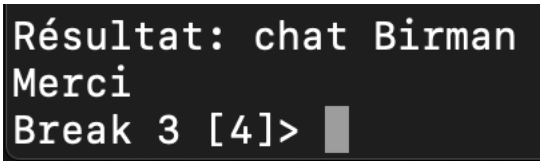

FIGURE 16 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Birman'.

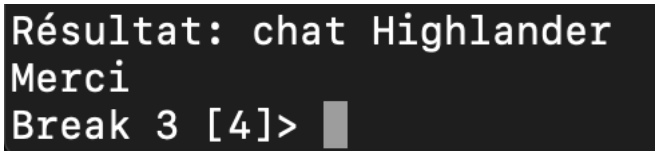

FIGURE 17 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Highlander'.

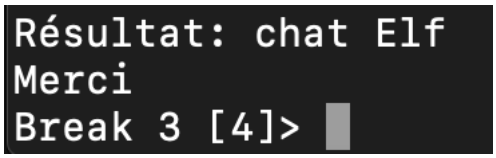

FIGURE 18 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Elf'.

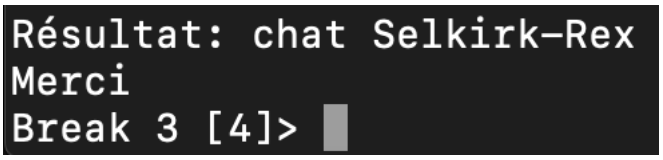

FIGURE 19 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Selkirk-Rex'.

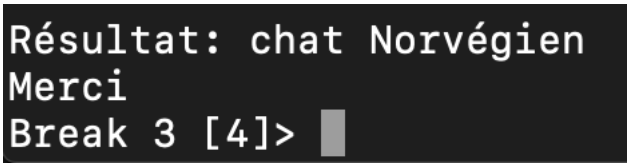

FIGURE 20 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Norvégien'.

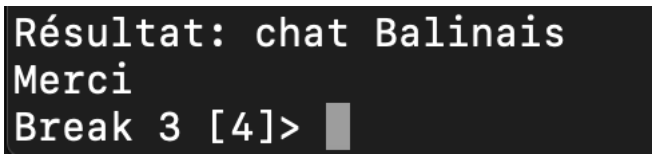

FIGURE 21 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Balinais'.

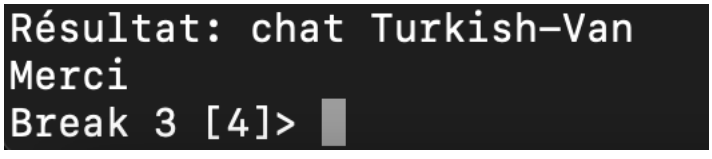

FIGURE 22 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Manx'.

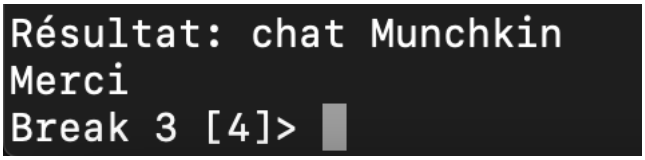

FIGURE 23 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Munchkin'.

FIGURE 24 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat LaPerm'.

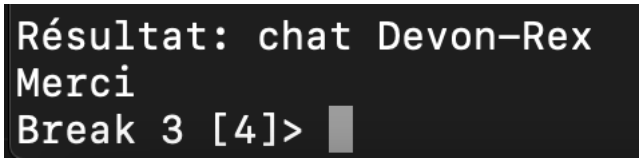

FIGURE 25 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Devon-Rex'.

FIGURE 26 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Chartreux'.

FIGURE 27 – Lorsque nous entrons toutes ses caractéristiques, nous obtenons en résultat 'chat Bambino'.

Annexes

.1 L'intégralité du fichier "SE_12.lisp"

Voici l'intégralité du fichier `SE_12.lisp`.

```
1  ; Nom ..... : SE_12.lisp
2  ; Rôle ..... : Système expert pour id un chat
3  ; Auteur ..... : Avrile Floro
4  ; Version ..... : V0.1 du 22/12/23
5  ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6  ; Usage : lisp puis (load "SE_12.lisp") puis (chat)
7
8  (load "SE_12_data.lisp")
9
10 ; NAME : chat, lance le programme d'id du chat
11 ; ARGS : none
12 ; USAGES : (chat)
13 ; GLOBALS : prop-basique, Règles
14 ; CALL : initialisation_prop, init_conclu, pose_questions, parcourt_regles, donne_resultats
15 ; USER : top level
16
17 (defun chat ()
18   (initialisation_prop prop-basique) ; initialise les propriétés à inconnu
19   (init_conclu Règles) ; initialise les conclusions à nil
20   (princ "Bonjour, je suis un système expert permettant d'identifier un chat.")
21   (terpri)
22   (pose_questions prop-basique)  ; pose les ? pour les prop basiques
23   (terpri)
24   (parcourt_regles Règles); on parcourt les règles pour modifier les conclusions
25   (terpri)
26   (princ "Résultat: ")
27   (donne_resultats Règles) )
28
29
30 ; NAME : initialisation_prop ; initialise les propriétés à inconnu
31 ; ARGS : prop-basique
32 ; USAGES : (initialisation_prop prop-basique)
33 ; GLOBALS : prop-basique, Animal
34 ; CALL : put
35 ; USER : chat
36
37 (defun initialisation_prop (prop-basique)
38   (cond
39       ((not prop-basique)) ; si on a parcouru toutes les prop, on s'arrête
40       (t (put 'Animal (caar prop-basique) 'inconnu)
41         ; on met la propriété à inconnu et on continue
42         (initialisation_prop (cdr prop-basique)) ) ) )
43
44
45 ; NAME : init_conclu ; initialise les conclusions à nil
46 ; ARGS : Règles
47 ; USAGES : (init_conclu Règles)
48 ; GLOBALS : Règles, Animal
49 ; CALL : put
50 ; USER : chat
51
52 (defun init_conclu (regles)
53   (cond
```

```
54      ((not regles)) ; si on a parcouru toutes les conclusions, on s'arrête
55      (t (put 'Animal (caaddr (car regles)) 'nil)
56        ; on met la conclusion à nil et on continue
57        (init_conclu (cdr regles)) ) ) )
58
59
60
61  ; NAME : pose_questions ; parcourt les prop basiques appelle get-basic-value poiur les ?
62  ; ARGS :  prop-basique
63  ; USAGES : (pose_questions prop-basique)
64  ; GLOBALS : prop-basique
65  ; CALL : get-basic-value
66  ; USER : test_realisation_regle, pose_questions
67
68  (defun pose_questions (prop-basique)
69    (cond
70      ((not prop-basique)) ; si on a parcouru toutes les prop, on s'arrête
71      ((get-basic-value (caar prop-basique)) ; on pose la question
72        (pose_questions (cdr prop-basique)) ) ) ) ; on continue de parcourir les prop
73
74
75  ; NAME : put ; met des valeurs dans les propriétés
76  ; ARGS :  symbole, attribut, valeur
77  ; USAGES : (put 'Animal conclusion t)
78  ; GLOBALS : Animal, Règles
79  ; CALL : none
80  ; USER : initialisation_prop, init_conclu, apply-conclusion, get-basic-value
81
82  (defun put (symbole attribut valeur) (setf (get symbole attribut) valeur) valeur)
83
84
85  ; NAME : get-basic-value ; màj prop basiques d'après les réponse de l'utilisateur
86  ; ARGS :  prop &aux base read
87  ; USAGES : (get-basic-value (caar prop-basique))
88  ; GLOBALS : prop-basique, Animal
89  ; CALL : put
90  ; USER : test_realisation_regle, pose_questions
91
92  (defun get-basic-value (prop &aux base read)
93    (cond
94      ((not (eq (get 'Animal prop 'inconnu) 'inconnu)) (get 'Animal prop))
95        ; si la propriété n'est pas inconnue, on retourne sa valeur
96      ((setq base (assoc prop prop-basique))
97        ; sinon on affiche la question et on lit la réponse
98        (princ (cdr base))
99        (terpri)
100       (princ "(répondez par nil/t/inconnu)")
101       (terpri)
102       (setq read (read))
103       (cond
104           ; on met à jour la propriété en fonction de la réponse
105         ((eq read 't) (put 'Animal prop t))
106         ((eq read 'nil) (put 'Animal prop nil) t)
107         ((eq read 'inconnu) (put 'Animal prop 'inconnu))
108         (t (princ "Réponse invalide, veuillez recommencer.")
109           ; si la réponse est invalide, on recommence
110           (terpri) (get-basic-value prop)) ) ) )
111
112
113
114 ; NAME : parcourt_regles ; parcourt les règles et appelle test_realisation_regle
```

```
; ARGS :  Règles
; USAGES : (parcourt_regles Règles)
; GLOBALS : Règles
; CALL : test_realisation_regle
; USER : chat

(defun parcourt_regles (regles)
  (cond
     ((not regles)) ; si on a parcouru toutes les règles, on s'arrête
     ((test_realisation_regle (car regles)) (parcourt_regles (cdr regles)))
       ; si la règle est vraie, on continue de parcourir les règles
     (t (parcourt_regles (cdr regles)) ) ) )
       ; sinon on continue de parcourir les règles


; NAME : test_realisation_regle ; vérifie la réalisation d'une règle et appelle
; apply-conclusion si elle est vraie
; ARGS :  Règles
; USAGES : (test_realisation_regle (car regles)
; GLOBALS : Règles
; CALL : apply-conclusion
; USER : parcourt_regles

(defun test_realisation_regle (regles)
  (cond
     ((not regles) nil) ; si on a parcouru toutes les règles, on s'arrête
     ((apply (caadr regles) (cdadr regles))
       ; on applique la fonction de la règle aux arguments de la règle
       (apply-conclusion (caaddr regles)) ) ) )
         ; on applique la conclusion de la règle


; NAME : et ; vérifie si toutes les propriétés sont vraies
; ARGS :  &rest args
; USAGES : (et (est-vivant X) (est-chat X))
; GLOBALS : Animal, Règles
; CALL : none
; USER : test_realisation_regle (via apply)

(defun et (&rest args)
  (cond
     ((not args))  ; si on a parcouru tous les args, retourne vrai
     ((equal (get 'Animal (caar args)) t) (apply 'et (cdr args)))
       ; si la propriété est t, on continue de parcourir les args
     (t nil) ) )  ; sinon, on retourne faux


; NAME : apply-conclusion ; met la conclusion à t
; ARGS :  conclusion
; USAGES : (apply-conclusion (caaddr regles))
; GLOBALS : Animal, Règles
; CALL : put
; USER : test_realisation_regle

(defun apply-conclusion (conclusion)
  (put 'Animal conclusion t) ) ; met la conclusion à t


; NAME : donne_resultats ; affiche les résultats
; ARGS :  Règles
; USAGES : (donne_resultats Règles)
```

```
176 ; GLOBALS : Règles, Animal
177 ; CALL : none
178 ; USER : chat
179
180 (defun donne_resultats (regles)
181   (cond
182       ((not regles) 'Merci) ; si on a parcouru toutes les conclusions, on s'arrête
183       ((eq (get 'Animal (car (caddar regles))) t) ; si la conclusion est t
184         (princ (format nil "~a " (car (caddar regles)))) (donne_resultats (cdr regles)) )
185         ; on l'affiche et on continue de parcourir les conclusions
186       (t (donne_resultats (cdr regles))) ) ) ; sinon on continue de parcourir les conclusions
187
188
189
190 (chat) ; on appelle la fonction chat
```

.2 L'intégralité du fichier "SE_12_data.lisp"

Voici l'intégralité du fichier `SE_12_data.lisp`.

```
; Nom ..... : SE_12_data.lisp
; Role ..... : données pour le système expert
; Auteur ..... : Avrile Floro
; Version ..... : V0.1 du 22/12/23
; Licence ..... : réalisé dans le cadre du cours d'Algo 1
; Usage : lisp puis (load "SE_12.lisp") puis (chat)

; NAME : Individus
; USAGES : liste des individus
; USER : initialisation_prop, init_conclu, put, get-basic-value, et,
;   apply-conclusion, donne_resultats

(setq Individus '(Animal))

; NAME : prop-basique
; USAGES : liste des propriétés de base
; USER : chat, initialisaiton_prop, pose_questions, get-basic-value

(setq prop-basique
   '((est-vivant . "L'animal est-il vivant?")
     (est-chat . "L'animal est-il un chat?")
     (pese<4 . "Le chat pèse-t-il moins de 4 kilos?")
     (pese-4-7 . "Le chat pèse-t-il entre 4 et 7 kilos?")
     (pese>7 . "Le chat pèse-t-il plus de 7 kilos?")
     (est-couleur-solide . "Le chat est-il de couleur solide")
     (est-bicolore . "Le chat est-il bicolore?")
     (est-ecaille-tortue . "Le chat est-il de couleur écaille de tortue?")
     (est-calico . "Le chat est-il de couleur calico?")
     (est-colorpoint . "Le chat est-il de couleur colorpoint?")
     (est-tabby . "Le chat est-il de couleur tabby?")
     (a-poil-court . "Le chat a-t-il le poil court?")
     (a-poil-mi-long . "Le chat a-t-il le poil mi-long?")
     (a-poil-long . "Le chat a-t-il le poil long?")
     (a-poil-frisé . "Le chat a-t-il le poil frisé?")
     (a-pas-de-poil . "Le chat est-il sans poil?")
     (a-courtes-pattes . "Le chat a-t-il de courtes pattes?")
     (a-tête-plate . "Le chat a-t-il la tête plate?")
     (a-tête-fine . "Le chat a-t-il la tête fine?")
     (a-queue-courte . "Le chat a-t-il une queue courte?")
     (a-grandes-oreilles . "Le chat a-t-il de grandes oreilles?")
     (a-oreille-courbée . "Le chat a-t-il les oreilles courbées?")
     (a-oreille-pliée . "Le chat a-t-il les oreilles pliées?")
     (a-oreille-de-lynx . "Le chat a-t-il des oreilles de lynx?") ) )


; NAME : Règles
; USAGES : liste des règles
; USER : chat, init_conclu, parcourt_regles, donne_resultats, put,
;   test_realisation_regle, et, apply-conclusion, donne_resultats

(setq Règles
  '((R_chat
      (et (est-vivant X) (est-chat X))
      (chat X) )
    (R_Manx
       (et (chat X) (pese-4-7 X) (a-poil-court X) (a-queue-courte X))
```

```
57      ; toutes les couleurs sont aceptées
58      (Manx X) )
59    (R_Cornish-Rex
60      (et (chat X) (pese<4 X) (a-poil-frisé X) (a-tête-fine X) (a-grandes-oreilles X))
61      (Cornish-Rex X) )
62    (R_Bobtail-Japonais
63      (et (chat X) (pese<4 X) (a-queue-courte X))
64      ; presque toutes les couleurs, poils longs ou courts
65      (Bobtail-Japonais X) )
66    (R_Siamois
67      (et (chat X) (pese-4-7 X) (est-colorpoint X) (a-poil-court X) (a-tête-fine X))
68      (Siamois X) )
69    (R_Maine-Coon
70      (et (chat X) (pese>7 X) (est-couleur-solide X) (a-poil-long X) (a-oreille-de-lynx X))
71      (Maine-Coon X) )
72    (R_Siberien
73      (et (chat X) (pese-4-7 X) (a-poil-mi-long X) (a-oreille-de-lynx X))
74      ; toutes les couleurs sont admises
75      (Sibérien X) )
76    (R_Scottish-Fold
77      (et (chat X) (pese-4-7 X) (est-couleur-solide X) (a-tête-plate X) (a-oreille-pliée X))
78      (Scottish-Fold X) )
79    (R_Persan
80      (et (chat X) (pese-4-7 X) (a-poil-long X) (a-tête-plate X))
81      ; toutes les couleurs sont admises
82      (Persan X) )
83    (R_Abyssin
84      (et (chat X) (pese-4-7 X) (est-tabby X) (a-poil-court X) (a-tête-fine X))
85      (Abyssin X) )
86    (R_Sphynx
87      (et (chat X) (pese-4-7 X) (a-pas-de-poil X) (a-tête-fine X))
88      (Sphynx X) )
89    (R_Himalayen
90      (et (chat X) (pese>7 X) (est-colorpoint X) (a-poil-long X))
91      (Himalayen X) )
92    (R_Birman
93      (et (chat X) (pese-4-7 X) (est-colorpoint X) (a-poil-long X))
94      (Birman X) )
95    (R_Highlander
96      (et (chat X) (a-queue-courte X) (a-oreille-courbée X))
97      ; toutes les couleurs sont autorisées et poils courts ou longs
98      (Highlander X) )
99    (R_Elf
100     (et (chat X) (pese-4-7 X) (a-pas-de-poil X) (a-oreille-courbée X))
101     (Elf X) )
102   (R_Selkirk-Rex
103     (et (chat X) (pese-4-7 X) (a-poil-frisé X) (a-tête-plate X))
104     (Selkirk-Rex X) )
105   (R_Norvégien
106     (et (chat X) (pese>7 X) (a-poil-mi-long X) (a-oreille-de-lynx X))
107     (Norvégien X) )
108   (R_Balinais
109     (et (chat X) (pese<4 X) (est-colorpoint X) (a-poil-mi-long X))
110     (Balinais X) )
111   (R_Turkish-Van
112     (et (chat X) (est-bicolore X) (a-poil-mi-long X) (a-oreille-de-lynx X))
113     (Turkish-Van X) )
114   (R_Munchkin
115     (et (chat X) (pese<4 X) (a-courtes-pattes X) (a-tête-plate X))
116     (Munchkin X) )
117   (R_LaPerm
```

```
118       (et (chat X) (pese<4 X) (a-poil-frisé X) (a-oreille-de-lynx X))
119       (LaPerm X) )
120    (R_Devon-Rex
121       (et (chat X) (pese-4-7 X) (a-poil-court X) (a-poil-frisé X) (a-grandes-oreilles X))
122       ; toutes les couleurs sont autorisées
123       (Devon-Rex X) )
124    (R_Chartreux
125       (et (chat X) (pese-4-7 X) (est-couleur-solide X) (a-poil-court X))
126       (Chartreux X) )
127    (R_Bambino
128       (et (chat X) (pese<4 X) (a-pas-de-poil X) (a-courtes-pattes X) (a-grandes-oreilles X))
129       (Bambino X) ) ) )
```