Abdelkader Belloundja - Colin Blake

Homework

1 Grammaire catégorielle

Donner un fragment de grammaire catégorielle combinatoire pour le français couvrant les exemples donnés en annexe :

Question 1.1: Primitives *On vous demande de donner la liste des catégories primitives de la grammaire et de définir un lexique.*

Les primitives sont les suivantes : Phrase, GrNom, Nom, VerbeInf, Ponct, PP où PP est un Participe Passé. Nous utilisons également les types annotés par exemple GrNom[Masc] pour définir un groupe nominal masculin singulier, cela permet plus de robustesse.
Par exemple la phrase "La souris est méchant" ne sera pas acceptée car "méchant" est considéré comme un adjectif masculin.

Nous définissons les catégories suivantes,

```
# Déterminants
DetMasc :: GrNom[Masc]/Nom[Masc]
DetFem :: GrNom[Fem]/Nom[Fem]
DetMascPlur :: GrNom[MascPlur]/Nom[MascPlur]
DetFemPlur :: GrNom[FemPlur]/Nom[FemPlur]
```

* Déterminants masculin singulier (Le, un, ...)
* Déterminants féminin singulier (La, une, ...)
* Déterminants masculin pluriel (Les, des, ...)
* Déterminants féminin pluriel (Les, des, ...) Nous garderons cette distinction par genre et par nombre dans les autres catégories aussi.

```
# Adjectifs
AdjMascL :: Nom[Masc]/Nom[Masc]
AdjFemL :: Nom[Fem]/Nom[Fem]
AdjMascR :: Nom[Masc]\\Nom[Masc]
AdjFemR :: Nom[Fem]\\Nom[Fem]
AdjMascPL :: Nom[MascPlur]/Nom[MascPlur]
AdjFemPL :: Nom[FemPlur]/Nom[FemPlur]
AdjMascPR :: Nom[MascPlur]\\Nom[MascPlur]
AdjFemPR :: Nom[FemPlur]\\Nom[FemPlur]
```

Un adjectif L se place à gauche du nom (par exemple "gentil"), alors qu'un adjectif R se place à droite ("noir"). Certains adjectifs peuvent se placer aussi bien à gauche qu'à droite, c'est notamment le cas de "méchant".

```
# Verbes
VbIntransSM :: Phrase\\GrNom[Masc]
VbIntransSF :: Phrase\\GrNom[Fem]
```

```
VbTransSM :: VbIntransSM/GrNom
VbTransSF :: VbIntransSF/GrNom
VbIntransPM :: Phrase\\GrNom[MascPlur]
VbIntransPF :: Phrase\\GrNom[FemPlur]
VbTransPM :: VbIntransPM/GrNom
VbTransPF :: VbIntransPF/GrNom
```

Un verbe intransitif n'attend qu'un argument à gauche, c'est le cas de "dort". Un verbe transitif a quant à lui besoin d'un argument à sa droite également, "pourchasse" par exemple.

```
# Extras
AnteposMasc :: VbIntransSM/VbTransSM
AnteposFem :: VbIntransSF/VbTransSF
AdvM :: VbIntransSM\\VbIntransSM
AdvF :: VbIntransSF\\VbIntransSF
AdvTM :: VbTransSM\\VbTransSM
AdvTF :: VbTransSF\\VbTransSF
```

Les adverbes qualifient des verbes ("mange paisiblement") aussi bien transitifs qu'intransitifs dans le cas général, les antéposition correspondent à la catégorie des groupes nominaux qui peuvent se placer à gauche d'un verbe transitif en tant qu'argument, par exemple "le" dans "Le chat le mange" est correct, bien que "le" soit l'argument cible du verbe, alors que "Le chat la souris mange" n'est pas correct.

Question 1.2: Règles d'inférence *Vous utiliserez les cinq règles d'inférence données en cours (application gauche, application droite, composition gauche, composition droite, type raising)*

Les applications gauche et droite sont les règles avec lesquelles nous avons davantage construit notre grammaire, les compositions et type raising sont autorisés mais apportent beaucoup d'ambigüité lorsqu'activées. Cependant, ces deux dernières ont l'avantage de pouvoir créer des arbres de dérivations qui se lisent de gauche à droite, ce qui correspond davantage au flot de la langue naturelle.

Question 1.3: Inspiration *Vous pouvez vous inspirer des exemples en anglais donnés en cours*
Nous avons commencé par créer une grammaire très spécifique au fragment (qui n'acceptait strictement que les phrases des exemples), puis avons essayer de la généraliser progressivement pour accepter d'autres phrases syntactiquement correctes sans pour autour accepter trop de phrases agrammaticales. Pour cela, nous avons étendu la base de donnée de phrases en rajoutant d'autres phrases qui sont correctes et nous servir de jeu de données test.
Nous avons estimé que la phrase "Le chat que mon voisin lui donne mange" ne semblait pas naturelle pour nous (on dirait plutôt "Le chat que mon lui donne est en train de manger" ou "Mon voisin lui donne un chat qui mange") et donc trop compliquée à analyser.

2 Test de la grammaire

Testez votre grammaire en python en utilisant le module CCG de nltk

Question 2.1: Robustesse *Dire si votre grammaire est robuste, c'est-à dire si elle accepte des phrases manifestement agrammaticales ou si elle est stricte, c'est-à-dire qu'elle accepte des phrases grammaticales et ne donne pas d'analyse pour les phrases agrammaticales.*

Nous avons essayé de faire une grammaire stricte, elle ne gère pas le fait qu'une phrase doive commencer par une majuscule mais vu que nous n'utilisons pas de points non plus cela semblait superflu. Lorsque

nous n'activons pas le Type Raising, voici quelques phrases courtes qui sont générées aléatoirement par la grammaire, ces dernières semblent majoritairement correctes :

```
Elle dort
un chat le donne
Il mange
Elle le pourchasse
Elle la donne
Il donne quoi
un voisin mange
Il dort
Elle mange
le rat dort
Elle la pourchasse
Elle qui mange mange
Un fromage noir mange
La sœur mange
Elle mange quoi
Qui donne ses dents ?
la souris dort
Le chat mange
Le rat dort
Il est mangé
Elle le mange
mon chat dort
le fromage mange
Qui donne quoi ?
Un voisin dort
La souris dort
Le voisin mange
Il est méchant
```

Si nous activons le Type Raising, de nombreuses phrases agrammaticales deviennent possibles, notamment parce que la egstion du genre et du nombre ne suit pas correctement (On peut type raising un nom masculin pour qu'il rentre dans un pattern où il ne devrait pas). Ce phénomène est dû à la liberté qu'apporte le type raising, il faut donc trouver un bon compromis.

```
Le chat dort
quoi dorment
Elle donne quoi
ses voisin dort
Il mange
un voisin dorment
quel rat dort
Il dort
quel dents méchant dort
quoi la donne
Il attrape paisiblement Un rat
ses sœur mange quoi
Elle mange
```

```
Le rat mange
quoi le donne
Elle le donne
quelle fromage est méchant
Un voisin mange
quoi dort
Elle dort
quel chat mange
la voisin dort
la sœur attrape Le fromage
quoi mange
Le voisin mange
le souris dort
Un sœur dort
La dents dorment
Un souris la pourchasse
Le souris dort
ses voisin mange
quel dents la donne
quelle voisin est mangée
```

Question 2.2: Ambigüité *A partir des résultats obtenus, quelles sont les causes d'ambiguité que vous identifiez. Si vous passiez à l'échelle en augmentant la taille du vocabulaire quelles seraient selon vous les conséquences en termes d'ambiguité ?*

Si nous n'activons pas le type raising, les phrases sont assez peu ambigüe. Il y a un peu d'ambigüité qui est intrinsèque à la construction de la langue française cependant, notamment en présence de plusieurs foncteurs comme "de", "par", "à", ..., plus on a de connecteurs dans la phrase plus cette dernière devient ambigüe.
Le nombre de parses par phrase de la base de donnée se trouve en annexe.
Lorsque nous activons le Type Raising, cet effet combinatoire explose notamment quand il y a de nombreux étapes de dérivations qui contiennent des types primitifs. Si une phrase est ambigüe sans type raising, en activant cette règle l'effet est encore plus amplifié.
Si l'on augmentait la taille du vocabulaire, il n'y aurait pas plus d'ambigüité sauf si l'on rajoute des mots dont le type crée de l'ambigüité (par exemples des foncteurs "ou", "car"), c'est la longueur de la phrase qui rajoute principalement l'ambigüité

3 Sémantique

Ajouter une composante de calcul sémantique à votre grammaire en vous inspirant de la page nltk
Nous avons implémenté une sémantique par continuation. Notre lambda calcul implémente un attribut exists et un opérateur booléen "et", l'opérateur 'ou' n'est pas implémenté pusiqu'il ne nous a pas servi dans cette sémantique. Les variables existentielles peuvent être instanciées par la suite de la dérivation.
Nous sommes globalement satisfait de notre sémantique bien que quelques points posent encore problème, le fragment de texte est assez chargé et nous avons essayer de faire des compromis pour avoir cette version finale qui nous semble être la meilleure (du moins la plus correcte) que nous ayons fait :

— La sémantique du mot "très" n'est implémentée, cela impliquerait d'avoir une notion de degré sur les adjectifs, certains sont quantifiables et d'autres non.

— Les adverbes tels que "paisiblement" sont considérés comme des adjectifs, en réalité ça ne devrait pas être le cas puisqu'ils qualifient le verbe et non le sujet de la phrase
— Pour des raisons de typage, nous ne voulons pas avoir des lambda dans les 'et', par exemple (x. P) & (z y. Q) n'est pas correct, il faut sortir les lambdas. Une façon de le faire a été de remplacer ces lambdas abstraction par des quantifieurs existentiels, potentiellement instanciables par le futur. Cela compromet légèrement la sémantique (on ne sait plus vraiment faire la distinction entre "le" et "un") mais permet d'avoir des formules logiques correctes.
— L'opérateur "et" duplique la sémantique, il ne devrait pas, idéalement nous devrions avoir dans certains cas que quelques morceaux d'information complémentaires.

Les sémantiques que notre algorithme renvoie se trouvent en annexe.

4 CKY

Implanter un algorithme d'analyse CKY « 1-best » pour CCG destiné à tester votre grammaire. Cela se décompose en :

Question 4.1: Format *Définir un format de lexique qui permet de pondérer les entrées lexicales*

Nous avons reconstruit notre parser CKY de zéro (et le parser de grammaire), cela nous permet plus de flexibilité et de possibilité sur la syntaxe de notre grammaire. Il est notamment possible dans la grammaire de définir des Weight qui prennent en paramètre une règle d'inférence et deux types puis un poids, par exemple Weight("¡", GrNom, Phrase
GrNom) = 1.5, cela permet de définir les valeurs de la fonction ϕ de l'algorithme Viterbi-CKY. Il est également possible de donner un poids à des règles terminales avec la syntaxe fromage =¿(3.0) Nom[Masc], bien que ce ne soit pas utilisé dans l'algorithme Viterbi, nous voulions nous laisser cette possibilité qui pourrait servir par exemple pour les foncteurs comme "et" qui peuvent créer de l'ambigüité.

Question 4.2: Pondérations *L'algorithme CKY « 1-best » suppose un système de pondérations. Vous définirez une méthode de pondération ad-hoc dans le cadre de cet exercice :*
Cela suppose de pondérer les dérivations CCG, c'est-à-dire les règles lexicales et les règles d'inférence.

Le calcul des poids se fait au moment de la construction des dérivations individuelles, notre algorithme n'élague pas au fur et à mesure les arbres qui n'auront vraisemblablement pas le meilleur poids, nous calculer les poids de chaque arbre. Nous gardons alors toutes les dérivations qui ont le meilleur poids, même si elles sont plusieurs.
La définition des poids a été intégrée directement à la grammaire, nous pouvons définir un poids pour une combinaison d'un combinateur, d'un fils gauche et d'un fils droit, cela correspond à la formule ψ du cours. Le poids d'une dérivation est alors le poids de ses différents fils multipliés avec le poids de la combinaison.
Exemple :

```
Weight("<", GrNom, Phrase\\GrNom) = 1.5
```

Correspond à définir la transition $Phrase \rightarrow (GrNom, Phrase$
$GrNom)$ par le combinateur ApplLeft comme ayant un poids de 1.5

Les poids ont été définis arbitrairement et ne sont pas complets, ils peuvent dépendre de ce que l'on recherche (avoir une structure correspondant à une lecture de gauche à droite par exemple).

Question 4.3: NLTK Tree *Votre implémentation devra suivre l'interface existante dans nltk et renvoyer une seule analyse, celle de poids maximal sous forme d'objet nltk.Tree.*

Puisque nous n'utilisons pas la bibliothèque nltk, nous avons ajouté une méthode à la classe de nos dérivations s'appellant "to nltk tree" et permettant de convertir ces derniers en arbre nltk et travailler avec ces derniers.

5 Travail supplémentaire

— Nous avons implémenté un parser pour les dérivations qui nous permette plus de liberté, notamment pour ajouter des poids, de nouveaux opérateurs sémantiques
— Nous avons implémenté un lambda calcul complet permettant de gérer des opérateurs binaires, des quantifieurs et des prédicats
— Nous avons implémenté les différentes types de la grammaire catégorielle, ces derniers peuvent être annotés, avec un pattern matching pour l'unification des types
— Nous avons implémenté le parser CKY qui construit les dérivations en utilisant nos autres librairies, nous avons veillé à ce que l'implémentation de nouvelles règles d'inférence soit efficace
— Nous avons réalisé un affichage des dérivations plus compact que celui fournit par nltk, qui permet une meilleure visibilité
— Nous avons réalisé des tests unitaires qui couvrent la majorité de notre code

6 Annexes

6.1 Nombre de parses

```
Un chat dort  :  1 parses valides
Il dort  :  1 parses valides
Le chat dort paisiblement  :  1 parses valides
Le chat noir dort  :  1 parses valides
Le méchant chat dort  :  2 parses valides
Le très méchant chat dort  :  2 parses valides
Le chat de la sœur dort  :  2 parses valides
Le chat de la sœur de mon voisin dort  :  9 parses valides
Le chat qui dort est noir  :  2 parses valides
Le chat mange la souris  :  2 parses valides
Le chat mange un chat  :  2 parses valides
Un chat mange un chat  :  2 parses valides
Le chat la mange  :  1 parses valides
Il la mange  :  2 parses valides
Quel chat mange la souris ?  :  5 parses valides
Qui mange la souris ?  :  10 parses valides
Quelle souris mange le chat ?  :  5 parses valides
La souris est mangée par le chat  :  2 parses valides
Elle est mangée par le chat  :  4 parses valides
Quelle souris est mangée par le chat ?  :  4 parses valides
Le chat mange la souris avec ses dents  :  4 parses valides
Le chat la mange avec ses dents  :  2 parses valides
Avec quoi le chat mange la souris ?  :  2 parses valides
Le rat donne un fromage  :  2 parses valides
Le rat donne un fromage à la souris  :  2 parses valides
Un fromage est donné  :  1 parses valides
```

```
Un fromage est donné à la souris  :  2 parses valides
Un fromage est donné par le rat  :  2 parses valides
Le chat lui donne la souris  :  2 parses valides
Un fromage est donné par le rat à la souris  :  8 parses valides
A quelle souris un fromage est donné par le rat ?  :  8 parses valides
Le chat le donne à la souris  :  2 parses valides
Il le lui donne  :  5 parses valides
mon voisin lui donne le chat  :  2 parses valides
Il souhaite que mon voisin lui donne le chat  :  2 parses valides
Il souhaite donner le chat  :  5 parses valides
Il souhaite donner le chat à mon voisin  :  5 parses valides
Le chat de mon voisin pourchasse la souris  :  4 parses valides
Le chat pourchasse et attrape la souris  :  2 parses valides
Le chat de mon voisin pourchasse et attrape la souris  :  4 parses valides
La souris dort et le chat de mon voisin attrape la souris  :  16 parses valides
Le chat dort et la souris dort  :  4 parses valides
Le chat et la souris dorment  :  2 parses valides
le chat mange la souris et le fromage  :  5 parses valides
la souris de mon chat et le fromage dorment  :  4 parses valides
un fromage très méchant mange le fromage  :  2 parses valides
le voisin de mon chat mange mon voisin  :  4 parses valides
Le méchant chat attrape et mange la souris paisiblement  :  4 parses valides
Le chat est méchant  :  2 parses valides
La souris est mangée par mon voisin  :  2 parses valides
La souris est mangée par mon voisin avec ses dents  :  4 parses valides
Le fromage mange avec ses souris  :  2 parses valides
Le méchant chat noir est noir et méchant  :  6 parses valides
Le rat mange ses dents avec ses dents  :  4 parses valides
La souris est mangée  :  1 parses valides
La souris est mangée par ses dents  :  2 parses valides
La souris mange un rat paisiblement  :  2 parses valides
```

6.2 Sémantiques

```
"Un chat dort"
 (\R_0, x_1. (exists x_2. (chat(x_2) & (R_0(x_2) & dort(x_2)))))

"Il dort"
 (\R_2, x_3. (masculin(x_3) & (R_2(x_3) & dort(x_3))))

"Le chat dort paisiblement"
 (\R_0, x_1. (chat(x_1) & ((paisible(x_1) & R_0(x_1)) & dort(x_1))))

"Le chat noir dort"
 (\R_0, x_1. ((chat(x_1) & noir(x_1)) & (R_0(x_1) & dort(x_1))))

"Le méchant chat dort"
 (\R_0, x_1. ((chat(x_1) & méchant(x_1)) & (R_0(x_1) & dort(x_1))))
```

```
"Le très méchant chat dort"
 (\R_0, x_1. ((chat(x_1) & méchant(x_1)) & (R_0(x_1) & dort(x_1))))

"Le chat de la sœur dort"
 (\R_0, x_1. (exists x_8. (soeur(x_1) & ((chat(x_8) & (R_0(x_8) & dort(x_8))) & appartient(x_
x_1)))))

"Le chat de la sœur de mon voisin dort"
 (\R_0, x_1. (exists x_106. (voisin(x_1) & (possede(moi, x_1) & ((exists x_107. (soeur(x_106)
& ((chat(x_107) & (R_0(x_107) & dort(x_107))) & appartient(x_107, x_106)))) & appartient(x_10
x_1))))))

"Le chat qui dort est noir"
 (\R_0, x_1. (chat(x_1) & ((R_0(x_1) & noir(x_1)) & dort(x_1))))

"Le chat mange la souris"
 (\R_0, x_1. (exists y_6. (chat(x_1) & (souris(y_6) & (mange(x_1, y_6) & R_0(x_1))))))

"Le chat mange un chat"
 (\R_0, x_1. (exists y_27. (chat(x_1) & (exists x_30. (chat(x_30) & (mange(x_1, x_30) & R_0(x

"Un chat mange un chat"
 (\R_0, x_1. (exists y_57. (exists x_61. (chat(x_61) & (exists x_62. (chat(x_62) & (mange(x_6
x_62) & R_0(x_61)))))))
"Le chat la mange"
 (\R_0, x_1. (exists y_71. (chat(x_1) & (féminin(y_71) & (mange(x_1, y_71) & R_0(x_1))))))

"Il la mange"
 (\R_2, x_3. (exists y_74. (masculin(x_3) & (féminin(y_74) & (mange(x_3, y_74) & R_2(x_3))))))

"Quel chat mange la souris ?"
 (\x_0. (exists y_3. ((souris(y_3) & (mange(x_0, y_3) & inconnu(x_0))) & chat(x_0))))

"Qui mange la souris ?"
 (\x_0. (exists y_4. ((souris(y_4) & (mange(x_0, y_4) & inconnu(x_0))) & humain(x_0))))

"Quelle souris mange le chat ?"
 (\y_0, x_1. (exists y_4. ((chat(y_4) & (mange(x_1, y_4) & inconnu(x_1))) & souris(y_4))))

"La souris est mangée par le chat"
 (\R_0, x_1. (exists y_0. (souris(y_0) & (mange(x_1, y_0) & (chat(x_1) & R_0(y_0))))))

"Elle est mangée par le chat"
 (\R_2, x_3. (exists y_0. (féminin(y_0) & (mange(x_3, y_0) & (chat(x_3) & R_2(y_0))))))

"Quelle souris est mangée par le chat ?"
 (\y_0, x_1. (exists y_2. ((mange(x_1, y_2) & (chat(x_1) & inconnu(y_2))) & souris(y_2))))

"Le chat mange la souris avec ses dents"
```

```
 (\R_0, x_1. (exists z_38. (exists y_39. (chat(z_38) & (souris(y_39) & (mange(z_38, y_39)
& (dents(x_1) & (possede(z_38, x_1) & (utilise(z_38, x_1) & R_0(z_38)))))))))

"Le chat la mange avec ses dents"
 (\R_0, x_1. (exists z_23. (exists y_24. (chat(z_23) & (féminin(y_24) & (mange(z_23, y_24)
& (dents(x_1) & (possede(z_23, x_1) & (utilise(z_23, x_1) & R_0(z_23)))))))))

"Avec quoi le chat mange la souris ?"
 (\x_0, z_1, y_2. ((objet(x_0) & inconnu(x_0)) & (chat(y_2) & (souris(z_1) & (mange(y_2,
z_1) & utilise(y_2, x_0))))))

"Le rat donne un fromage"
 (\R_0, x_1. (exists y_25. (exists s_26. (exists x_27. (fromage(x_27) & (rat(x_1) & (donne(x_
x_27, y_25) & R_0(x_1)))))))

"Le rat donne un fromage à la souris"
 (\R_0, x_1, y_2. (exists s_19. (exists x_20. (fromage(x_20) & (souris(y_2) & (donne(x_1,
x_20, y_2) & (rat(x_1) & R_0(x_1)))))))

"Un fromage est donné"
 (\R_0, x_1. (exists y_0. (exists x_2. (fromage(x_2) & (donne(x_2, x_2, y_0) & R_0(x_2))))))

"Un fromage est donné à la souris"
 (\R_0, x_1. (exists y_5. (exists x_7. (fromage(x_7) & (donne(x_1, x_7, y_5) & (souris(y_5)
& R_0(x_7)))))))

"Un fromage est donné par le rat"
 (\R_0, x_1. (exists y_0. (exists x_2. (fromage(x_2) & (donne(x_2, x_2, y_0) & (rat(x_2)
& R_0(x_2)))))))

"Le chat lui donne la souris"
 (\R_0, z_1, y_2, x_3. (souris(x_3) & ((chat(z_1) & (donne(z_1, x_3, y_2) & R_0(z_1))) &
cible(y_2))))

"Un fromage est donné par le rat à la souris"
 (\R_0, x_1. (exists y_29. (exists x_31. (fromage(x_31) & (donne(x_1, x_31, y_29) & (rat(x_1)
& (souris(y_29) & R_0(x_31)))))))

"A quelle souris un fromage est donné par le rat ?"
 (\x_0. (exists y_10. (exists x_11. (fromage(x_11) & (donne(x_11, x_11, y_10) & (rat(x_11)
& (souris(y_10) & inconnu(y_10)))))))

"Le chat le donne à la souris"
 (\R_0, x_1, y_2, s_3. (exists s_5. (masculin(s_5) & (souris(s_5) & (donne(x_1, s_5, s_5)
& (chat(x_1) & R_0(x_1)))))))

"Il le lui donne"
 (\R_15, z_16, y_17, x_18. (masculin(x_18) & ((masculin(z_16) & (donne(z_16, x_18, y_17)
& R_15(z_16))) & cible(y_17))))
```

"mon voisin lui donne le chat"
 (\R_0, z_1, y_2, x_3. (chat(x_3) & ((voisin(z_1) & (possede(moi, z_1) & (donne(z_1, x_3, y_2) & R_0(z_1)))) & cible(y_2))))

"Il souhaite que mon voisin lui donne le chat"
 (\P_2, Q_3, R_4, x_5. (exists y_20. (exists s_21. souhaite((chat(y_20) & ((voisin(s_21) & (possede(moi, s_21) & (donne(s_21, y_20, x_5) & R_4(s_21)))) & cible(x_5))), x_5))))

"Il souhaite donner le chat"
 (\R_2, x_3. (exists y_40. (exists s_41. souhaite((chat(s_41) & (masculin(x_3) & (donne(x_3, s_41, y_40) & R_2(x_3)))), x_3))))

"Il souhaite donner le chat à mon voisin"
 (\R_2, x_3, y_4. (exists s_8. souhaite((chat(s_8) & (voisin(y_4) & (possede(moi, y_4) & (donne(x_3, s_8, y_4) & (masculin(x_3) & R_2(x_3)))))), x_3)))

"Le chat de mon voisin pourchasse la souris"
 (\R_0, x_1. (exists y_41. (exists x_43. (voisin(x_1) & (possede(moi, x_1) & ((chat(x_43) & (souris(y_41) & (pourchasse(x_43, y_41) & R_0(x_43)))) & appartient(x_43, x_1)))))))

"Le chat pourchasse et attrape la souris"
 (\R_0, x_1. (exists y_12. ((chat(x_1) & (souris(y_12) & (pourchasse(x_1, y_12) & R_0(x_1)))) & (chat(x_1) & (souris(y_12) & (attrape(x_1, y_12) & R_0(x_1)))))))

"Le chat de mon voisin pourchasse et attrape la souris"
 (\R_0, x_1. (exists y_27. ((exists x_29. (voisin(x_1) & (possede(moi, x_1) & ((chat(x_29) & (souris(y_27) & (pourchasse(x_29, y_27) & R_0(x_29)))) & appartient(x_29, x_1))))) & (exist x_31. (voisin(x_1) & (possede(moi, x_1) & ((chat(x_31) & (souris(y_27) & (attrape(x_31, y_27) & R_0(x_31)))) & appartient(x_31, x_1))))))))

"La souris dort et le chat de mon voisin attrape la souris"
 (\R_0, x_1. (exists y_102. ((exists y_105. (exists x_106. (voisin(x_1) & (possede(moi, x_1) & ((chat(x_106) & (souris(y_105) & (attrape(x_106, y_105) & R_0(x_106)))) & appartient(x_106, x_1))))))) & (souris(y_102) & (R_0(y_102) & dort(y_102))))))

"Le chat dort et la souris dort"
 (\R_0, x_1. (exists y_156. ((souris(x_1) & (R_0(x_1) & dort(x_1))) & (chat(y_156) & (R_0(y_1 & dort(y_156))))))

"Le chat et la souris dorment"
 (\R_0, x_1. (exists y_165. ((chat(x_1) & (R_0(x_1) & dort(x_1))) & (souris(y_165) & (R_0(y_1 & dort(y_165))))))

"le chat mange la souris et le fromage"
 (\R_0, x_1. (exists y_281. (chat(x_1) & (exists y_284. ((souris(y_281) & (mange(x_1, y_281) & R_0(x_1))) & (fromage(y_284) & (mange(x_1, y_284) & R_0(x_1)))))))))

"la souris de mon chat et le fromage dorment"

```
 (\R_0, x_1. (exists y_375. ((exists x_376. (chat(x_1) & (possede(moi, x_1) & ((souris(x_376)
& (R_0(x_376) & dort(x_376))) & appartient(x_376, x_1))))) & (fromage(y_375) & (R_0(y_375)
& dort(y_375))))))

"un fromage très méchant mange le fromage"
 (\R_0, x_1. (exists y_391. (exists x_393. ((fromage(x_393) & méchant(x_393)) & (fromage(y_39
& (mange(x_393, y_391) & R_0(x_393)))))))

"le voisin de mon chat mange mon voisin"
 (\R_0, x_1. (exists y_422. (exists x_424. (chat(x_1) & (possede(moi, x_1) & ((voisin(x_424)
& (voisin(y_422) & (possede(moi, y_422) & (mange(x_424, y_422) & R_0(x_424))))) & appartient(
x_1)))))))

"Le méchant chat attrape et mange la souris paisiblement"
 (\R_0, x_1. (exists y_32. (((chat(x_1) & méchant(x_1)) & (souris(y_32) & (attrape(x_1, y_32)
& (paisible(x_1) & R_0(x_1))))) & ((chat(x_1) & méchant(x_1)) & (souris(y_32) & (mange(x_1,
y_32) & (paisible(x_1) & R_0(x_1)))))))))

"Le chat est méchant"
 (\R_0, x_1. (chat(x_1) & (R_0(x_1) & méchant(x_1))))

"La souris est mangée par mon voisin"
 (\R_0, x_1. (exists y_0. (souris(y_0) & (mange(x_1, y_0) & (voisin(x_1) & (possede(moi,
x_1) & R_0(y_0)))))))

"La souris est mangée par mon voisin avec ses dents"
 (\R_0, x_1. (exists z_18. (exists y_19. (souris(y_19) & (mange(z_18, y_19) & (voisin(z_18)
& (possede(moi, z_18) & (dents(x_1) & (possede(z_18, x_1) & (utilise(z_18, x_1) & R_0(z_18)))

"Le fromage mange avec ses souris"
 (\R_0, x_1. (exists z_4. (fromage(z_4) & ((souris(x_1) & (possede(z_4, x_1) & (utilise(z_4,
x_1) & R_0(z_4)))) & mange(z_4)))))

"Le méchant chat noir est noir et méchant"
 (\R_0, x_1. (((chat(x_1) & méchant(x_1)) & noir(x_1)) & ((R_0(x_1) & noir(x_1)) & (R_0(x_1)
& méchant(x_1)))))

"Le rat mange ses dents avec ses dents"
 (\R_0, x_1. (exists z_79. (exists y_80. (rat(z_79) & (exists y_83. (dents(y_80) & (possede(y
y_80) & (mange(z_79, y_80) & (dents(x_1) & (possede(z_79, x_1) & (utilise(z_79, x_1) & R_0(z_

"La souris est mangée"
 (\R_0, x_1. (exists y_0. (souris(y_0) & (mange(x_1, y_0) & R_0(y_0)))))

"La souris est mangée par ses dents"
 (\R_0, x_1. (exists y_5. (souris(y_5) & (mange(x_1, y_5) & (exists y_8. (dents(x_1) & (posse
x_1) & R_0(y_8)))))))

"La souris mange un rat paisiblement"
```

(\R_0, x_1. (exists y_52. (souris(x_1) & (exists x_55. (rat(x_55) & (mange(x_1, x_55) & (paisible(x_1) & R_0(x_1))))))))