TP3

!!!) Petit problème avec le code, la traduction est faite deux fois, la première retourne le bon résultat, la deuxième retourne false, je n'ai pas réussit à trouver la raison pour ce "bug". De plus, les fonctions accepte la phrase [p,p,p] ou encore [p,groa,gru,p,p,p,p, gru,groa]. Car c'est de la concaténation de pauses qui est possible, par contre traduire [p,groa,gru,p,p,p,p, gru,groa] retournera le nombre de pauses minimale.

EXO (1

.1)

On a les expressions Expre, et la phrase Phrase.

Expre : = atGa | atDr | atFr | deGa | deDr | deFr | m | s | [ε]

[ε] = la liste contenant le mot vide.

[,] = ce symbole symbolise la concaténation entre deux expressions. on a alors que une Phrase est la concaténation d'expressions.

Phrase = Phrase , {Expre} - la concaténation d'une phrase à une expression donne une nouvelle phrase.

Phrase$_{\{,|\varepsilon\}}$ = [{Expre}]=[{LH}]

Définissons la phrase suivante $[atGa, atDr, atFr, \varepsilon]$:

on a le premier élément $atGa \in Expre \rightarrow [atDr, atFr, \varepsilon]$
le deuxième élément $atDr \in Expre \rightarrow [atFr, \varepsilon]$
le troisième élément $atFr \in Expre \rightarrow [\varepsilon]$
finalement $[\varepsilon] \in Expre$

On sait que Phrase = Phrase , {Expre} - Alors le but est de itérativement prouver que le premier élément de la phrase est une expression, jusqu'à avoir un liste vide.

.2)

Pour vérifier si une phrase AtGa AtFr DeGa est programme du langage humain, il suffit de faire appel à la règle phraseH().

Exemple : **phraseH([atGa, atFr, deGa]).**

.3)

On a que 3 mots différents, et les expressions c'est des différents concaténations de ces 3 mots.

Mots : = groa | gru | grou

On a les expressions Expre, et la phrase Phrase.

Expre : = groa gru | gru groa | gru gru | grou groa | groa grou | grou grou | gru grou | grou gru | [ε] | [p]

[ε] = la liste contenant le mot vide.

[p] = la pause entre chaque expression.

[,] = ce symbole symbolise la concaténation entre deux expressions. on a alors que une Phrase est la concaténation d'expressions.

Phrase = Phrase , p, {Expre} - la concaténation d'une phrase à une expression donne une nouvelle phrase (attention à la pause).

$\textbf{Phrase}_{\{,|\varepsilon\}}$ = [{Expre}]=[{LO}]

Définissons la phrase suivante $[groa\ gru, p, grou\ groa, p, gru\ gru, \varepsilon]$:

on a le premier élément $groa\ gru \in Expre \rightarrow [grou\ groa, p, gru\ gru, \varepsilon]$
le deuxième élément $grou\ groa \in Expre \rightarrow [gru\ gru, \varepsilon]$
le troisième élément $gru\ gru \in Expre \rightarrow [\varepsilon]$
finalement $[\varepsilon] \in Expre$

On sait que Phrase = Phrase , {Expre} - Alors le but est de itérativement prouver que le premier élément de la phrase est une expression, jusqu'à avoir un liste vide, pour trouver les éléments on cherche la pause, quand on le trouve ce qui a avant est l'expression, ce qui se trouve après est le reste de la phrase.

.4)

La pause n'est pas vraiment nécessaire, car on peut créer des règles d'inférence qui prennent les mots deux à deux, et il n'y aura pas de confusion, et si on a une phrase qui a un nombre impair de mots, on l'accepte pas.

Imaginons par contre que nous introduisons une nouvelle expression danser, qui aurait la traduction Groa en Langage Ogre, alors la phrase Groa Gru Groa peut vouloir dire danser et attaquer droite, ou alors attaquer gauche et danser. Avec les pauses, on peut dire ce qu'on veut sans problèmes, Groa P Gru Groa, veut dire danser et ensuite attaquer à droite.

.5)

Pour vérifier si une phrase Groa Gru P Grou Groa P Gru Gru est programme du langage ogre, il suffit de faire appel à la règle phraseOg().

Exemple : **phraseOg([groa, gru, p, grou, groa, p, gru, gru]).**

.6)

Pour traduire une phrase on a besoin de trois choses, un tableau **Tab{LDep}** qui contient la phrase, le pointeur qui indique la position dans le tableau **Pt**, et finalement du programme **Prog**.

Le domaine sémantique est alors : $Tab\{LDep\} \times Pt \times Prog$

Le résultat sera un **Tab{LArr}** et un **Pt**, qui ont les mêmes rôles que avant, ce tableau contient les expressions traduites.

On a alors : $Tab\{LH\} \times Pt \times Prog \rightarrow Tab\{LO\} \times Pt$ comme domaine sémantique pour cette traduction. Le Tableau de départ contient des expressions qu'on veut traduire, et celui d'arrivé des expressions traduites.

.7)

Le Tab de départ contient des expressions en LH, et la traduction souhaité est LO. Soit $\rightarrow$ la représentation de la traduction.

Règle de base : $[\varepsilon] \rightarrow [\varepsilon]$ -la traduction du mot vide, est le mot vide.

Traduire un mot : $[\{LH\}, \varepsilon] \rightarrow [\{LO\}, \varepsilon]$ -a chaque expression du $\{LH\}$ lui correspond un mot du $\{LO\}$.

Concaténation : $[X] \in \{LH\} \wedge [Y] \in \{LH\} \Longrightarrow [X, Y] \in \{LH\}$

Phrase : $[X, Y] \in \{LH\} = [X] \in \{LH\} \wedge [Y] \in \{LH\} \rightarrow [X] \in \{LO\} \wedge [Y] \in \{LO\} = [X, Y] \in \{LO\}$ -Ou $[Y] \in \{LO\}$ correspond à la traduction de $[X] \in \{LH\}$.

Pause : $[X] \in \{LO\} \wedge [Y] \in \{LO\} \Longrightarrow [X], [Y] = [X, p, Y] \in \{LO\}$ -la concaténation en LO doit être faite a travers le connecteur p.

Vu que la règle de concaténation peut être étendue à des phrases de longueur N, on peut appliquer ces règles simples à toute phrase.

.8)

Définissons la phrase suivante $[atGa, deDr, m, s, \varepsilon]$:

On fait exactement de la même façon que quand on voulait prouver que c'était une phrase (1.1), sauf qui ici on traduit aussi l'expression.

On a le premier élément $atGa \rightleftharpoons groa\ gru \rightarrow [deDr, m, s, \varepsilon] \wedge [groa\ gru]$
deuxième $deDr \rightleftharpoons groa\ grou \rightarrow [m, s, \varepsilon] \wedge [groa\ gru, p, groa\ grou]$
troisième $m \rightleftharpoons gru\ grou \rightarrow [s, \varepsilon] \wedge [groa\ gru, p, groa\ grou, p, gru\ grou]$
quatrième $s \rightleftharpoons grou\ gru \rightarrow [\varepsilon] \wedge [groa\ gru, p, groa\ grou, p, gru\ grou, p, grou\ gru]$
finalement $[\varepsilon] \rightleftharpoons [\varepsilon] \rightarrow [groa\ gru, p, groa\ grou, p, gru\ grou, p, grou\ gru, \varepsilon]$

Ici il faut juste faire attention à remettre la pause pour séparer les expressions.

$A \rightleftharpoons B$ - veut dire que A équivaut à B dans les langages souhaités dans la traduction.

.9)

Pour traduire la phrase AtGa AtFr DeGa en langage ogre, il suffit de faire appel à la règle fromHtoO().

Exemple : **fromHtoO([atGa, atFr, deGa], Resultat).**
Resultat = [groa, gru, p, gru, gru, p, grou, groa]

Pour traduire la phrase Groa Gru P Grou Groa P Gru Gru en langage humain, il suffit de faire appel à la règle fromOtoH().

Exemple :
fromOtoH([groa, gru, p, grou, groa, p, gru, gru], Resultat).
Resultat = [atGa, atFr, deGa]

EXO (2

.1)

On a que 3 mots différents, et les expressions c'est des différents concaténations de ces 3 mots.

Mots : = bru | bra | bro

On a les expressions Expre, et la phrase Phrase.

Expre : = bru bra bro | bro bra bru | bro bro bro | bra bru bro | bro bru bra | bra bra bra | bru bru bra | bra bru bru | [ε] | [p]

[ε] = la liste contenant le mot vide.
[p] = la pause entre chaque expression.
[,] = ce symbole symbolise la concaténation entre deux expressions. on a alors que une Phrase est la concaténation d'expressions.
Phrase = Phrase , p, {Expre} - la concaténation d'une phrase à une expression donne une nouvelle phrase (attention à la pause).
Phrase$_{\{,|\varepsilon\}}$ = [{Expre}]=[{LO}]

Définissons la phrase suivante [$bru\ bra\ bro, p, bro\ bra\ bru, p, bra\ bru\ bru, \varepsilon$] :

on a le premier élément $bru\ bra\ bro \in Expre \rightarrow [bro\ bra\ bru, p, bra\ bru\ bru, \varepsilon]$
le deuxième élément $bro\ bra\ bru \in Expre \rightarrow [bra\ bru\ bru, \varepsilon]$
le troisième élément $bra\ bru\ bru \in Expre \rightarrow [\varepsilon]$
finalement $[\varepsilon] \in Expre$

On sait que Phrase = Phrase , {Expre} - Alors le but est de itérativement prouver que le premier élément de la phrase est une expression, jusqu'à avoir un liste vide, pour trouver les éléments on cherche la pause, quand on le trouve ce qui a avant est l'expression, ce qui se trouve après est le reste de la phrase.

.2)

Le langage des ogres a 3 mots a disposition, et accepte des expressions qui contiennent 2 mots, ces expressions acceptent aussi les répétitions de mot, ce qui fait donc un maximum de $3^2 = 9$ expressions différentes, les ogres utilisent alors 8 des 9 expressions possibles.

Le langage des barbares a aussi 3 mots à disposition, donc les barbares pourraient avoir exactement le même fonctionnement que les ogres, ils décident néanmoins de utiliser des expressions de 3 mots chaque avec répétition possible, ce qui fait un total de $3^3 = 27$, alors qu'ils utilisent que 8 expressions différentes sur les 27 possibles.

.3)

Pour vérifier si une phrase Bru Bra Bro P Bro Bra Bru P Bra Bru Bru est programme du langage barbare, il suffit de faire appel à la règle phraseBa().

Exemple :

phraseBa([bru, bra, bro, p, bro, bra, bru, p, bra, bru, bru]).

.4)

Pour traduire une phrase on a besoin de trois choses, un tableau **Tab{LDep}** qui contient la phrase, le pointeur qui indique la position dans le tableau **Pt**, et finalement du programme **Prog**.

Le résultat sera un **Tab{LArr}** et un **Pt**, qui ont les mêmes rôles que avant, ce tableau contient les expressions traduites.

$$Tab\{LO\} \times Pt \times Prog \rightarrow Tab\{LB\} \times Pt$$

.5)

Traduire la pause : $[p] \rightarrow [p]$ -la pause se traduit tout simplement par la pause.

Les règles d'inférence définies en 1.7) sont toutes applicables ici, du coup on garde les mêmes, juste que le langage de départ et d'arrivé ont changé, ici on veut traduire de LO à LB, mais vu que tout mot a une traduction directe.

.6)

Définissons la phrase suivante $[groa\ gru, p, groa\ grou, p, grou\ gru, \varepsilon]$:

On a le premier élément $groa\ gru \rightleftharpoons bru\ bra\ bro \rightarrow [groa\ grou, p, grou\ gru, \varepsilon] \wedge [bru\ bra\ bro]$
deuxième $groa\ grou \rightleftharpoons bro\ bru\ bra \rightarrow [grou\ gru, \varepsilon] \wedge [bru\ bra\ bro, p, bro\ bru\ bra]$
troisième $grou\ gru \rightleftharpoons bra\ bru\ bru \rightarrow [\varepsilon] \wedge [bru\ bra\ bro, p, bro\ bru\ bra, p, bra\ bru\ bru]$
finalement $[\varepsilon] \rightleftharpoons [\varepsilon] \rightarrow [bru\ bra\ bro, p, bro\ bru\ bra, p, bra\ bru\ bru, \varepsilon]$

$A \rightleftharpoons B$ - veut dire que A équivaut à B dans les langages souhaités dans la traduction.

. 7 & 8)

Pour traduire la phrase AtGa DeDr M S en langage barbare, il suffit de faire appel à la règle fromHtoB().

Exemple : **fromHtoB([atGa, deDr, m, s], Resultat).**
Resultat = [bru, bra, bro, p, bro, bru, bra, p, bru, bru, bra, p, bra, bru, bru]

Pour traduire la phrase Bru Bra Bro P Bro Bra Bru P Bra Bru Bru en langage humain, il suffit de faire appel à la règle fromBtoH().

Exemple :
fromBtoH([bru, bra, bro, p, bro, bra, bru, p, bra, bru, bru], Resultat).
Resultat = [atGa, atDr, s]

!!!

Même dans le code je fais bien le passage de humain à ogre, puis à barbare, et vise versa, les fonctions **fromOtoB()**. (ogre to barbare), ainsi que **fromBtoO()**. (barbare to ogre) sont utilisables.

!!!

Pour traduire la phrase Groa Gru P Groa Grou en langage barbare, il suffit de faire appel à la règle fromOtoB().

Exemple : **fromOtoB([groa, gru, p, groa, grou], Resultat).**
Resultat = [bru, bra, bro, p, bro, bru, bra]

Pour traduire la phrase Bru Bra Bro P Bro Bra Bru en langage ogre, il suffit de faire appel à la règle fromBtoO().

Exemple :
fromBtoO([bru, bra, bro, p, bro, bra, bru], Resultat).
Resultat = [groa, gru, p, gru, groa]

.9)

Définissons la phrase suivante $[atGa, deDr, m, s, \varepsilon]$ D'abord on le traduit en langage Ogre : (exo 1.8)

On a le premier élément $atGa \rightleftharpoons groa\ gru \rightarrow [deDr, m, s, \varepsilon] \wedge [groa\ gru]$
deuxième $deDr \rightleftharpoons groa\ grou \rightarrow [m, s, \varepsilon] \wedge [groa\ gru, p, groa\ grou]$
troisième $m \rightleftharpoons gru\ grou \rightarrow [s, \varepsilon] \wedge [groa\ gru, p, groa\ grou, p, gru\ grou]$
quatrième $s \rightleftharpoons grou\ gru \rightarrow [\varepsilon] \wedge [groa\ gru, p, groa\ grou, p, gru\ grou, p, grou\ gru]$
finalement $[\varepsilon] \rightleftharpoons [\varepsilon] \rightarrow [groa\ gru, p, groa\ grou, p, gru\ grou, p, grou\ gru, \varepsilon]$

Maintenant on traduit $[groa\ gru, p, groa\ grou, p, gru\ grou, p, grou\ gru, \varepsilon]$
de Ogre à Barbare :

On a le premier élément $groa\ gru \rightleftharpoons bru\ bra\ bro \rightarrow [groa\ grou, p, gru\ grou, p, grou\ gru, \varepsilon] \wedge$
$[bru\ bra\ bro]$
deuxième $groa\ grou \rightleftharpoons bro\ bru\ bra \rightarrow [gru\ grou, p, grou\ gru, \varepsilon] \wedge [bru\ bra\ bro, p, bro\ bru\ bra]$
troisième $gru\ grou \rightleftharpoons bru\ bru\ bra \rightarrow [grou\ gru, \varepsilon] \wedge [bru\ bra\ bro, p, bro\ bru\ bra, p, bru\ bru\ bra]$
quatrième $grou\ gru \rightleftharpoons bra\ bru\ bru \rightarrow [\varepsilon] \wedge [bru\ bra\ bro, p, bro\ bru\ bra, p, bru\ bru\ bra, p, bra\ bru\ bru]$
finalement $[\varepsilon] \rightleftharpoons [\varepsilon] \rightarrow [bru\ bra\ bro, p, bro\ bru\ bra, p, bru\ bru\ bra, p, bra\ bru\ bru, \varepsilon]$

$$[atGa, deDr, m, s, \varepsilon]$$
$$\rightleftharpoons$$
$$[bru\ bra\ bro, p, bro\ bru\ bra, p, bru\ bru\ bra, p, bra\ bru\ bru, \varepsilon]$$