**Tips**

* 1. Boot up gf shell
     1. gf
     2. gf concreteEng.gf
  2. Run script to import all and generate all
     1. Put into hello.gfs the following

import HelloEng.gf

import HelloFin.gf

linearize Hello World or simply l Hello Word

* 1. gf --run <hello.gfs outside of gf but in the folder.

* 1. Translate phrase/sentences
     1. parse -lang=HelloEng "hello mum" | linearize -lang=HelloFin

* 1. Translate all
     1. parse -lang=HelloEng "hello mum" | linearize
     2. p "this exquisit wine is delicious" | l -all

* 1. Abstract and Concrete alignment esp when adding new abstract fun

|  |  |
| --- | --- |
| Items in Abstract cat | Ensure same as Concrete lincat (esp if new fun with new output type were created) |
| Abstract startcat | Review if need to update for new fun output type ie new Rome |
| Abstract cat | Ensure new fun output type are added |
| Abstract fun | All fun types must be defined/described in Concrete lin esp newly created fun |
| Concrete lin | * + 1. Has all the abstract fun     2. Each item is accompanied with its arguments types as defined in Abstract fun     3. Each item definition includes its argumentsType.s (if applicatble) ++     4. End each line with ; |
| Concrete lincat | Ensure same as Abstract cat (esp if new fun with new output type were created) |

* + Machine generated alternative text:
    Food.gf • 
    Food.gf 
    FoodEng.gf 
    abstract Food = { 
    flags startcat 
    Statement 
    cat 
    Phrase • 
    fun 
    Item 
    NoPrefix 
    Excuseme 
    Fresh, Warm, 
    ; Kind 
    Italian, 
    Fish 
    • Quality 
    Expensive, 
    Statement; 
    Del icious, 
    Boring 
    Very 
    QKi nd 
    Wi ne, 
    This, 
    qnls 
    Cheese, 
    That 
    Phrase 
    Statement 
    Phrase 
    Statement 
    : Quality • 
    : Quality 
    - > Quality 
    : Quality 
    Kind Kind 
    : Kind • 
    : Kind 
    - > Item 
    - > Quality Phrase 
    Item 
    - > Quality Phrase 
    Item 
    - (takes an/ (x) argument(s) of type) 
    (returns product type); 

Food.gf 
FoodEng.gf 
FoodEng.gf 
concrete FoodEng 
lincat 
Phrase, 
Iin 
NoPrefix 
Excuseme 
of 
Food 
Ki nd, 
Item, 
hrase 
phrase 
Quality , 
Statement 
{s 
{s 
Highlights are the 
arguments types 
taken by Abstract 
fun 
. str} , 
Is ite 
ualit 
qnls item qualit 
This 
That kind 
QKind quality kind 
Wi ne 
Cheese 
Fish 
Very quality 
Fresh 
Warm 
Italian 
Expensive 
Delicious 
Boring 
phrase. s} 
"Excuse me" ++ phrase. s} 
item.s ++ "is" ++ qualit .s 
"is • ++ item.s ++ qua lty.s} 
"this" ++ 
"that" ++ 
quality.s ++ kind .s} 
"wine"} 
"cheese"} 
"fish"} 
"fresh"} . 
"warm" } 
"Italian"} 
" expensive"} 
"delicious"} 
"boring"} 

* 1. Shortcuts

|  |  |
| --- | --- |
| e | to exit out of what had been imported ie clear gf memory |
| q | to get out of gf shell |
| rm \*.gfo | remove all the gf object files |
| gt | to generate trees under the startcat (not all cat) |
| gt -cat = abstract\_fun\_name1 | to get all trees/combi under fun\_name1 (not linearized) |
| gt -cat = abstract\_fun\_name1 | l  *Space* | *space* l | to get all trees/combi under fun\_name1 in linearized forms ie proper english |
| r | to reload after saving changes to .gf files |
| gr fun\_name1 ? | to generate any func argument pair (not linearized) of abstract.  NB: If fun\_name1 is not in concrete -> no trees found  But if  1. e to exit  2. import (or i) abstractfile.gf   * 1. Reperform gr fun\_name1?   => fun\_name1 Recipient/argument will be output even though fun\_name1 is missing in concrete |
|  |  |

* 1. Linearize with concrete.gf
     1. Import concrete.gf
     2. l lin\_word1 lin\_word2 (without quotes)
     3. gt -cat = fun\_name1
     4. parse -lang=HelloMornEng "morning dad" | linearize
     5. gr funName1
     6. ? ? | l (to generate a random linearize output of the funName1)

* 1. Know the parser
     1. Food> parse -cat=Statement "Excuse Me is this cheese Italian"

ExcuseMe (qnIs (This Cheese) Italian)

* 1. parse "this delicious cheese is very Italian" | align\_words

* 1. Generate Trees and strings in gf, count lines

|  |  |
| --- | --- |
| generate\_random | build a random tree in accordance with an abstract syntax |
| generate\_random | l | linearize a random tree |
| gr -number=4 | l | generate x random trees |
| generate\_trees | l    Or  gt | l | generate all trees |
| generate\_trees -depth=2 | l | generate tree default depth is 3. Change depth to # |
| generate\_trees -depth=4 | ? wc -l | Count number of lines generated |
| gr -number=2 | l -treebank | Generate a set of trees with their linearizations in different languages |

* 1. Write and read files in CCLAW

|  |  |
| --- | --- |
| gr -number=10 | linearize | write\_file -file=exx.tmp | Write |
| gt | l | write\_file -file=gt2.txt | Write |
| gt -depth=4 | linearize | write\_file -file=gt4.txt | Write |
| read\_file -file=exx.tmp -lines | parse | Read in gf |
| wc -l gt2.txt | Count lines in text file outside of gf |

* 1. Generate Tree Diagram
     1. Inside gf:
        1. parse "this delicious cheese is very Italian" | visualize\_tree -view="eog" OR
        2. parse "this fish is very expensive" | visualize\_tree | wf -file="myfish.dot"
     2. Open new shell in same directory
        1. dot -Tpng myfish.dot > fish.png
     3. code fish.png