1 A more realistic lexicon

1.1 Features

We define a set of agreement features:

ghci 2> :i Feat
data Feat = Masc | Fem | Neutr | MascOrFem | Sg | Pl | Fst | Snd | Thrd |
      | Nom | AccOrDat | Pers | Refl | Wh | Tense | Infl | On | With | By | To |

Agreement morphology consists of feature bundles:

```
ghci 3> features
    [Masc, Fem, Neutr, MascOrFem, Sg, Pl, Fst, Snd, Thrd, Nom, AccOrDat, Pers, Refl, Wh, Tense, Infl, On, With, By, To, From]
```

We also define grammatically relevant subsets of these features:

```
ghci 4> :i Agreement
```

Finally, we define a function that will eliminate the underspecified gender feature `MascOrFem` whenever the fully specified gender features `Masc` or `Fem` are added to the feature bundle:
prune :: Agreement → Agreement
prune fs = if (Masc ∈ fs ∨ Fem ∈ fs) then (delete MascOrFem fs) else fs

1.2 Syntactic categories
We can now define syntactic categories as a list consisting of a phonological representation, a category label, an agreement feature bundle and a subcategorization list:
| ghci 22 | Cat "goldilocks" "NP" [Thrd, Fem, Sg] []
|         | "goldilocks" NP [Thrd, Fem, Sg] |
| ghci 23 | Cat "" "NP" [Thrd, Fem, Sg] []
|         | "" NP [Thrd, Fem, Sg] |
| ghci 24 | Cat "littlemook" "NP" [Thrd, Masc, Sg] []
|         | "littlemook" NP [Thrd, Masc, Sg] |
| ghci 25 | Cat "every" "DET" [Sg] []
|         | "every" DET [Sg] |
| ghci 26 | Cat "all" "DET" [Pl] []
|         | "all" DET [Pl] |
| ghci 27 | Cat "some" "DET" [] []
|         | "some" DET [] |
| ghci 28 | Cat "several" "DET" [Pl] []
|         | "several" DET [Pl] |
| ghci 29 | Cat "a" "DET" [Sg] []
|         | "a" DET [Sg] |
| ghci 30 | Cat "did" "AUX" [] []
|         | "did" AUX [] |
| ghci 31 | Cat "helped" "VP" [Tense] [Cat "" "NP" [AccOrDat] []]
|         | "helped" VP [Tense] |
| ghci 32 | Cat "and" "CONJ" [] []
|         | "and" CONJ [] |
We define 4 functions that enable us to extract the individual components of the categories (we could have used record syntax instead and these would have been automatically defined):

\begin{verbatim}
ghci 33> phon $ Cat "helped" "VP" [Tense] [Cat "" "NP" [AccOrDat] []]
"helped"
\end{verbatim}

\begin{verbatim}
ghci 34> catLabel $ Cat "helped" "VP" [Tense] [Cat "" "NP" [AccOrDat] []]
"VP"
\end{verbatim}

\begin{verbatim}
ghci 35> fs $ Cat "helped" "VP" [Tense] [Cat "" "NP" [AccOrDat] []]
[Tense]
\end{verbatim}

\begin{verbatim}
ghci 36> subcatList $ Cat "helped" "VP" [Tense] [Cat "" "NP" [AccOrDat] []]
["" NP [AccOrDat]]
\end{verbatim}

1.3 The lexicon

We are now ready to define our lexicon as a function from strings (the words themselves) to lists of categories (lists b/c some words might be ambiguous). See the Lexicon module for many examples.

\begin{verbatim}
lexicon "us" = [Cat "us" "NP" [Pers, Fst, Pl, AccOrDat] []]
\end{verbatim}

\begin{verbatim}
lexicon "who" = [Cat "who" "NP" [Wh, Thrd, MascOrFem] [], Cat "who" "REL" [MascOrFem] []]
\end{verbatim}

\begin{verbatim}
lexicon "every" = [Cat "every" "DET" [Sg] []]
\end{verbatim}

\begin{verbatim}
lexicon "woman" = [Cat "woman" "CN" [Sg, Fem, Thrd] []]
\end{verbatim}

\begin{verbatim}
lexicon "women" = [Cat "women" "CN" [Pl, Fem, Thrd] []]
\end{verbatim}

\begin{verbatim}
lexicon "cheered" = [Cat "cheered" "VP" [Tense] []]
\end{verbatim}

\begin{verbatim}
lexicon "cheer" = [Cat "cheer" "VP" [Infl] []]
\end{verbatim}

\begin{verbatim}
lexicon "did" = [Cat "did" "AUX" [] []]
\end{verbatim}

\begin{verbatim}
lexicon "didn’t" = [Cat "didn’t" "AUX" [] []]
\end{verbatim}
1.4 Combining syntactic categories

We define a way to combine the feature bundles of 2 categories (the empty list [] indicates failure to combine):

\[
\text{combine} :: \text{Cat} \to \text{Cat} \to [\text{Agreement}]
\]

\[
\text{combine } \text{cat1 } \text{cat2} =
\left[\text{feats} \mid \text{length} (\text{gender feats}) \leq 1,
\text{length} (\text{number feats}) \leq 1,
\text{length} (\text{person feats}) \leq 1,
\text{length} (\text{gcase feats}) \leq 1,
\text{length} (\text{pronType feats}) \leq 1,
\text{length} (\text{tense feats}) \leq 1,
\text{length} (\text{prepType feats}) \leq 1\right]
\]

where
\[
\text{feats} = \text{prune} \circ \text{nub} \circ \text{sort} \circ \text{fs cat1} + + \text{fs cat2}
\]

For example:

```
ghci 37> let \{ cat1 = Cat "goldilocks" "NP" [Thrd, Fem, Sg] [];
    \    cat2 = Cat "runs" "VP" [Tense, Sg] [];
    \    cat3 = Cat "run" "VP" [Tense, Pl] [] \}
ghci 38> combine cat1 cat2
[[Fem, Sg, Thrd, Tense]]
ghci 39> combine cat1 cat3
[]
```

We can determine whether 2 categories agree this way: they agree if we combine them and we don’t get an empty list.

\[
\text{agree} :: \text{Cat} \to \text{Cat} \to \text{Bool}
\]

\[
\text{agree } \text{cat1 } \text{cat2} = \neg \circ \text{null} \circ \text{combine cat1 cat2}
\]

```
ghci 40> agree cat1 cat2
True
```

```
ghci 41> agree cat1 cat3
False
```

Finally, we define a function in which a particular agreement feature is assigned to a category:

\[
\text{assign} :: \text{Feat} \to \text{Cat} \to [\text{Cat}]
\]

\[
\text{assign } f @ (\text{Cat } \text{phon label } fs \text{ subcatlist}) =
\left[\text{Cat } \text{phon label } fs' \text{ subcatlist} \mid
fs' \leftarrow \text{combine } f (\text{Cat } " " " [f] []))\right]
\]
1.5 String preprocessing

Finally, we do some preprocessing of incoming strings to ‘smooth out’ various idiosyncrasies. The \textit{scan} and \textit{preproc} functions at the end of the \textit{Lexicon} module do this. Their definitions are repeated below for convenience.

\begin{verbatim}
scan :: String → String
scan [] = []
scan (x:xs) | x ∈ ".,?" = ' ': x: scan xs
   | otherwise = x : scan xs

preproc :: Words → Words
preproc [] = []
preproc "." = []
preproc "?" = []
preproc ("," : xs) = preproc xs
preproc ("did" : "not" : xs) = "didn’t" : preproc xs
preproc ("nothing" : xs) = "no" : "thing" : preproc xs
preproc ("nobody" : xs) = "no" : "person" : preproc xs
preproc ("something" : xs) = "some" : "thing" : preproc xs
preproc ("somebody" : xs) = "some" : "person" : preproc xs
preproc ("everything" : xs) = "every" : "thing" : preproc xs
preproc ("everybody" : xs) = "every" : "person" : preproc xs
preproc ("less" : "than" : xs) = "less_than" : preproc xs
preproc ("more" : "than" : xs) = "more_than" : preproc xs
preproc ("at" : "least" : xs) = "at_least" : preproc xs
preproc ("at" : "most" : xs) = "at_most" : preproc xs
preproc (x : xs) = x : preproc xs
\end{verbatim}

1.6 The lexer

We are now ready to take an incoming string, identify the lexical items it contains and extract their categories from the lexicon.

We first identify the lexical items:

\begin{verbatim}
type Words = [String]
lexer :: String → Words
lexer = preproc ∘ words ∘ (map toLower) ∘ scan
\end{verbatim}

\begin{verbatim}
ghci 42> assign Tense $ Cat "run" "VP" [Pl] []
["run" VP [Pl, Tense]]
\end{verbatim}

\begin{verbatim}
ghci 43> lexer "I loved her."
["i","loved","her"]
\end{verbatim}

\begin{verbatim}
ghci 44> lexer "She despised me."
["she","despised","me"]
\end{verbatim}
Then we extract their categories from the lexicon and collect them:

\[
\text{lookupWord} :: (\text{String} \to [\text{Cat}]) \to \text{String} \to [\text{Cat}]
\]

\[
\text{collectCats} :: (\text{String} \to [\text{Cat}]) \to \text{Words} \to [[\text{Cat}]]
\]

\[
\text{collectCats} \ db \ \text{words} =
\]

\[
\begin{align*}
\text{let } & \text{listing} = \text{map} \ (\lambda x \rightarrow (x, \text{lookupWord} \ db \ x)) \ \text{words} \\
& \text{unknown} = \text{map} \ \text{fst} \ (\text{filter} \ (\text{null} \circ \text{snd}) \ \text{listing}) \\
\text{in } & \text{if} \ \text{unknown} \neq [] \text{ then error } \text{("unknown words: "} + \text{show unknown)} \\
& \text{else initCats} \ (\text{map} \ \text{snd} \ \text{listing})
\end{align*}
\]

\[
\begin{align*}
\text{initCats} :: [[\text{Cat}]] & \to [[\text{Cat}]] \\
\text{initCats} \ [] & = [] \\
\text{initCats} \ (\text{cs} : \text{rests}) & = [\text{cs} : \text{rests} | c \leftarrow \text{cs}, \text{rest} \leftarrow \text{initCats} \ \text{rests}]
\end{align*}
\]

2 Parsing a more realistic English fragment (w/o mvt)

\[
\text{ghci 45}> \ \text{collectCats} \ \text{lexicon} \ \text{lexer} \ "I \ loved \ her."
[["i" \ NP \ [\text{Pers}, Fst, Sg, Nom], "loved" \ VP \ [\text{Tense}], "her" \ NP \ [\text{Pers}, Thrd, Sg, AccOrDat, Fem]]]
\]

\[
\text{ghci 46}> \ \text{collectCats} \ \text{lexicon} \ \text{lexer} \ "She \ despised \ me."
\*
\*
\*
\text{Exception : unknown words: ["despised"]}
\]

\[
\begin{align*}
\text{ghci 47}> \ :l \ \text{ParserNoMvt}
\end{align*}
\]

We first define 3 useful functions:

(i) a function from trees to categories

\[
\text{ghci 48}> \ :t \ \text{t2c}
\]

\[
\begin{align*}
t2c :: \text{ParseTree} \ \text{Cat} \ \text{Cat} & \to \text{Cat}
\end{align*}
\]

\[
\text{ghci 49}> \ :i \ \text{ParseTree}
\]

\[
\begin{align*}
\text{data} \ \text{ParseTree} \ \nonterminal \ \text{terminal} & = \text{EmptyTree} \mid \text{Leaf} \ \text{terminal} \\
& \mid \text{Branch} \ \nonterminal \ [\text{ParseTree} \ \nonterminal \ \text{terminal}] \\
\end{align*}
\]

\[
\text{ghci 50}> \ :t \ \text{"goldilocks"} \ \text{"NP"} \ [\text{Thrd}, \text{Fem}, \text{Sg}] \ []
\]

\[
\begin{align*}
\text{Cat} \ "\text{goldilocks"} \ \text{"NP"} \ [\text{Thrd}, \text{Fem}, \text{Sg}] \ [] : \text{Cat}
\end{align*}
\]
(ii) a function that checks whether 2 trees agree
(iii) a function that assigns an agreement feature to a category

We can now start defining our parser combinators (including the basic parsers).

2.1 Parser for leaf nodes

We begin with a parser for leaf nodes:

ghci 67> :t leafP
leafP :: CatLabel → PARSER Cat Cat

ghci 68> :i CatLabel
    type CatLabel = String -- Defined at Lexicon.hs:25:6

ghci 69> :t leafP "NP"
leafP "NP" :: PARSER Cat Cat

ghci 70> leafP "NP" [Cat "goldilocks" "NP" [Thrd, Fem, Sg] []]
   [[["goldilocks" NP [Thrd, Fem, Sg],[]]]]
2.2 Parser for sentences

```haskell
ghci 75> :t parseSent
parseSent :: PARSER Cat Cat
```

```haskell
ghci 76> :i PARSER
    type PARSER input category = Parser input (ParseTree category input) -- Defined at ParserCombinators.hs:49:6
```

```haskell
ghci 77> parseSent $ [Cat "goldilocks" "NP" [Thrd, Fem, Sg] [], Cat "runs" "VP" [Tense, Sg] [], ["runs" VP [Tense, Sg], []], ["" S [] : "goldilocks" NP [Fem, Sg, Thrd, Nom] ["" VP [Sg, Tense] : "runs" VP [Tense, Sg]], []]
```

```haskell
ghci 78> parseSent $ [Cat "goldilocks" "NP" [Thrd, Fem, Sg] [], Cat "runs" "VP" [Tense, Sg] [], Cat "quickly" "AdvP"
```

2.3 NP, DET, CN and PP parsers

```haskell
ghci 79> parseNP [Cat "goldilocks" "NP" [Thrd, Fem, Sg] []]
    ["goldilocks" NP [Thrd, Fem, Sg], []]
```

```haskell
ghci 80> parseNP [Cat "goldilocks" "NP" [Thrd, Fem, Sg] [], Cat "princess" "CN" [Sg, Fem, Thrd] []]
    ["princess" CN [Sg, Fem, Thrd], []]
```

11
2.4 VP parser

VPs are assembled by means of a rule that parses a VP first and then check that the following items in the list of remaining inputs match the sub-categorization list of the VP. If they do, those items are subsumed under the VP branch.

We also have a rule for finite VPs resulting from combining an auxiliary and a VP:
2.5 Bringing it all together

Finally, we can assemble all of these functions into a single function that takes us from strings directly to parse trees.

```
ghci 92> :l ParserNoMvt
```

```
ghci 93> prs "I loved her."
[["" S [] : "i" NP [Sg,Fst,Nom,Pers] ["" VP [Tense] : "loved" VP [Tense] "her" NP [Pers, Thrd,Sg,AccOrDat,Fem]]]]
```

```
ghci 94> prs "I loved her.!!0"
["" S [] : "i" NP [Sg,Fst,Nom,Pers] ["" VP [Tense] : "loved" VP [Tense] "her" NP [Pers, Thrd,Sg,AccOrDat,Fem]]]
```

```
ghci 95> prs "She didn’t love me."
[["" S [] : "she" NP [Fem,Sg,Thrd,Nom,Pers] ["" VP [] : "didn’t" AUX [] ["" VP [Infl] : "love" VP [Infl] "me" NP [Pers,Fst,Sg,AccOrDat]]]]
```

```
ghci 96> prs "She didn’t love me.!!0"
["" S [] : "she" NP [Fem,Sg,Thrd,Nom,Pers] ["" VP [] : "didn’t" AUX [] ["" VP [Infl] : "love" VP [Infl] "me" NP [Pers,Fst,Sg,AccOrDat]]]]
```

```
ghci 97> prs "She despised me."
***Exception: unknown words: ["despised"]
```

We also have a convenience function that generates \TeX-\text{compilable} trees (based on code by Christina Unger):
ghci 98> writeTree2Tex ((prs "I loved her.") !! 0)
   " S "
   /\    /
  l::NP VP
  /\    /
 LOVED::VP HER::NP

ghci 99> writeTree2Tex ((prs "She didn’t love me.") !! 0)
   " S "
   /\    /
  SHE::NP VP
   /\    /
  DIDN’T::AUX VP
    /\    /
   LOVE::VP ME::NP

ghci 100> writeTree2Tex ((prs "The dwarf didn’t defeat the giant.") !! 0)
   " S "
   /\    /
  NP VP
  /\    /
 THE::DET DWARF::CN
   /\    /
 DIDN’T::AUX VP
    /\    /
   DEFEAT::VP NP
      /\    /
     THE::DET GIANT::CN