# Computational Formal Semantics Notes: Part 6

Adrian Brasoveanu*

December 3, 2013

## Contents



# 1 A more realistic lexicon

**ghci 1>** : *l 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* |
*From* -- Defined at Lexicon.hs:6:6 instance Bounded Feat – Defined at Lexicon.hs:6:180 instance Enum Feat – I

*Code based on *Computational Semantics with Functional Programming* by Jan van Eijck & Christina Unger, http://www.computational-semantics.eu.

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

Agreement morphology consists of feature bundles:

```
ghci 4>  :i Agreement
    type Agreement = [Feat]    -- Defined at Lexicon.hs:11:6
```

We also define grammatically relevant subsets of these features:

```
ghci 5>  :t gender
    gender :: Agreement → Agreement
```

```
ghci 6> gender features
    [Masc, Fem, Neutr, MascOrFem]
```

```
ghci 7> number features
    [Sg, Pl]
```

```
ghci 8> person features
    [Fst, Snd, Thrd]
```

```
ghci 9> gcase features
    [Nom, AccOrDat]
```

```
ghci 10> pronType features
    [Pers, Refl, Wh]
```

```
ghci 11> tense features
    [Tense, Infl]
```

```
ghci 12> prepType features
    [On, With, By, To, From]
```

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:

$prune :: Agreement \to Agreement$
$prune\ fs = \mathbf{if}\ (Masc \in fs \lor Fem \in fs)\ \mathbf{then}\ (delete\ MascOrFem\ fs)\ \mathbf{else}\ fs$

**ghci 13>** *gender features*
$[Masc, Fem, Neutr, MascOrFem]$

**ghci 14>** *prune $ gender features*
$[Masc, Fem, Neutr]$

**ghci 15>** *number features*
$[Sg, Pl]$

**ghci 16>** *prune $ number features*
$[Sg, Pl]$

## 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 17>** *:i Cat*
**data** *Cat* = *Cat Phon CatLabel Agreement* [*Cat*] -- Defined at Lexicon.hs:28:6 instance Eq Cat – Defined at Lexic

**ghci 18>** *:i Phon*
**type** *Phon* = *String* -- Defined at Lexicon.hs:26:6

**ghci 19>** *:i CatLabel*
**type** *CatLabel* = *String* -- Defined at Lexicon.hs:25:6

**ghci 20>** *:i Agreement*
**type** *Agreement* = [*Feat*] -- Defined at Lexicon.hs:11:6

Here are a couple of examples of categories:

**ghci 21>** *:t Cat*
$Cat :: Phon \to CatLabel \to Agreement \to [Cat] \to Cat$

```
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):

**ghci 33>** *phon* $ *Cat* `"helped"` `"VP"` [*Tense*] [*Cat* `""` `"NP"` [*AccOrDat*] []]
`"helped"`

**ghci 34>** *catLabel* $ *Cat* `"helped"` `"VP"` [*Tense*] [*Cat* `""` `"NP"` [*AccOrDat*] []]
`"VP"`

**ghci 35>** *fs* $ *Cat* `"helped"` `"VP"` [*Tense*] [*Cat* `""` `"NP"` [*AccOrDat*] []]
[*Tense*]

**ghci 36>** *subcatList* $ *Cat* `"helped"` `"VP"` [*Tense*] [*Cat* `""` `"NP"` [*AccOrDat*] []]
[`""` *NP* [*AccOrDat*]]

## 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.

*lexicon* `"us"` = [*Cat* `"us"` `"NP"` [*Pers*, *Fst*, *Pl*, *AccOrDat*] []]

*lexicon* `"who"` = [*Cat* `"who"` `"NP"` [*Wh*, *Thrd*, *MascOrFem*] [], *Cat* `"who"` `"REL"` [*MascOrFem*] []]

*lexicon* `"every"` = [*Cat* `"every"` `"DET"` [*Sg*] []]

*lexicon* `"woman"` = [*Cat* `"woman"` `"CN"` [*Sg*, *Fem*, *Thrd*] []]

*lexicon* `"women"` = [*Cat* `"women"` `"CN"` [*Pl*, *Fem*, *Thrd*] []]

*lexicon* `"cheered"` = [*Cat* `"cheered"` `"VP"` [*Tense*] []]

*lexicon* `"cheer"` = [*Cat* `"cheer"` `"VP"` [*Infl*] []]

*lexicon* `"did"` = [*Cat* `"did"` `"AUX"` [] []]

*lexicon* `"didn't"` = [*Cat* `"didn't"` `"AUX"` [] []]

## 1.4 Combining syntactic categories

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

$$
\begin{aligned}
&\mathit{combine} :: \mathit{Cat} \rightarrow \mathit{Cat} \rightarrow [\mathit{Agreement}]\\
&\mathit{combine}\ \mathit{cat1}\ \mathit{cat2} =\\
&\quad [\mathit{feats} \mid \mathit{length}\ (\mathit{gender}\ \mathit{feats}) \leqslant 1,\\
&\qquad \mathit{length}\ (\mathit{number}\ \mathit{feats}) \leqslant 1,\\
&\qquad \mathit{length}\ (\mathit{person}\ \mathit{feats}) \leqslant 1,\\
&\qquad \mathit{length}\ (\mathit{gcase}\ \mathit{feats}) \leqslant 1,\\
&\qquad \mathit{length}\ (\mathit{pronType}\ \mathit{feats}) \leqslant 1,\\
&\qquad \mathit{length}\ (\mathit{tense}\ \mathit{feats}) \leqslant 1,\\
&\qquad \mathit{length}\ (\mathit{prepType}\ \mathit{feats}) \leqslant 1]\\
&\quad \textbf{where}\\
&\qquad \mathit{feats} = \mathit{prune} \circ \mathit{nub} \circ \mathit{sort}\ \$\ \mathit{fs}\ \mathit{cat1} +\!\!+\ \mathit{fs}\ \mathit{cat2}
\end{aligned}
$$

For example:

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

**ghci 38>** $\mathit{combine}\ \mathit{cat1}\ \mathit{cat2}$
$[[\mathit{Fem}, \mathit{Sg}, \mathit{Thrd}, \mathit{Tense}]]$

**ghci 39>** $\mathit{combine}\ \mathit{cat1}\ \mathit{cat3}$
$[\,]$

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

$$
\begin{aligned}
&\mathit{agree} :: \mathit{Cat} \rightarrow \mathit{Cat} \rightarrow \mathit{Bool}\\
&\mathit{agree}\ \mathit{cat1}\ \mathit{cat2} = \neg \circ \mathit{null}\ \$\ \mathit{combine}\ \mathit{cat1}\ \mathit{cat2}
\end{aligned}
$$

**ghci 40>** $\mathit{agree}\ \mathit{cat1}\ \mathit{cat2}$
$\mathit{True}$

**ghci 41>** $\mathit{agree}\ \mathit{cat1}\ \mathit{cat3}$
$\mathit{False}$

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

$$
\begin{aligned}
&\mathit{assign} :: \mathit{Feat} \rightarrow \mathit{Cat} \rightarrow [\mathit{Cat}]\\
&\mathit{assign}\ f\ c@(\mathit{Cat}\ \mathit{phon}\ \mathit{label}\ \mathit{fs}\ \mathit{subcatlist}) =\\
&\quad [\mathit{Cat}\ \mathit{phon}\ \mathit{label}\ \mathit{fs}'\ \mathit{subcatlist} \mid\\
&\qquad \mathit{fs}' \leftarrow \mathit{combine}\ c\ (\mathit{Cat}\ \texttt{""}\ \texttt{""}\ [f]\ [\,])]
\end{aligned}
$$

**ghci 42>** *assign Tense* $ *Cat* "run" "VP" [*Pl*] []
    ["run" *VP* [*Pl*, *Tense*]]

## 1.5 String preprocessing

Finally, we do some preprocessing of incoming strings to 'smooth out' various idiosyncracies. The *scan* and *preproc* functions at the end of the *Lexicon* module do this. Their definitions are repeated below for convenience.

```
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
```

## 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:

```
type Words = [String]
lexer :: String → Words
lexer = preproc ∘ words ∘ (map toLower) ∘ scan
```

**ghci 43>** *lexer* "I loved her."
    ["i","loved","her"]

**ghci 44>** *lexer* "She despised me."
    ["she","despised","me"]

Then we extract their categories from the lexicon and collect them:

$$
\begin{aligned}
&\mathit{lookupWord} :: (\mathit{String} \to [\mathit{Cat}]) \to \mathit{String} \to [\mathit{Cat}]\\
&\mathit{lookupWord}\ \mathit{db}\ w = [c \mid c \leftarrow \mathit{db}\ w]\\
&\mathit{collectCats} :: (\mathit{String} \to [\mathit{Cat}]) \to \mathit{Words} \to [[\mathit{Cat}]]\\
&\mathit{collectCats}\ \mathit{db}\ \mathit{words} =\\
&\quad \mathbf{let}\ \mathit{listing} = \mathit{map}\ (\lambda x \to (x, \mathit{lookupWord}\ \mathit{db}\ x))\ \mathit{words}\\
&\qquad \mathit{unknown} = \mathit{map}\ \mathit{fst}\ (\mathit{filter}\ (\mathit{null} \circ \mathit{snd})\ \mathit{listing})\\
&\quad \mathbf{in}\ \mathbf{if}\ \mathit{unknown} \not\equiv []\ \mathbf{then}\ \mathit{error}\ (\texttt{"unknown words: "} +\!\!+\ \mathit{show}\ \mathit{unknown})\\
&\qquad \mathbf{else}\ \mathit{initCats}\ (\mathit{map}\ \mathit{snd}\ \mathit{listing})\\
&\mathit{initCats} :: [[\mathit{Cat}]] \to [[\mathit{Cat}]]\\
&\mathit{initCats}\ [] = [[]]\\
&\mathit{initCats}\ (\mathit{cs} : \mathit{rests}) = [c : \mathit{rest} \mid c \leftarrow \mathit{cs}, \mathit{rest} \leftarrow \mathit{initCats}\ \mathit{rests}]
\end{aligned}
$$

**ghci 45>** *collectCats lexicon* $ *lexer* `"I loved her."`
[["i" *NP* [*Pers*, *Fst*, *Sg*, *Nom*], "loved" *VP* [*Tense*], "her" *NP* [*Pers*, *Thrd*, *Sg*, *AccOrDat*, *Fem*]]]

**ghci 46>** *collectCats lexicon* $ *lexer* `"She despised me."`
∗ ∗ ∗*Exception* : *unknown words* : ["despised"]

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

**ghci 47>** : *l ParserNoMvt*

We first define 3 useful functions:

(*i*) a function from trees to categories

**ghci 48>** : *t t2c*
*t2c* :: *ParseTree Cat Cat* → *Cat*

**ghci 49>** : *i ParseTree*
**data** *ParseTree nonterminal terminal* = *EmptyTree* | *Leaf terminal* | *Branch nonterminal* [*ParseTree nonterminal terminal*] -- Defined at BasicDef.hs:6:6 instance (Eq nonterminal, Eq

**ghci 50>** : *t Cat* `"goldilocks"` `"NP"` [*Thrd*, *Fem*, *Sg*] []
*Cat* `"goldilocks"` `"NP"` [*Thrd*, *Fem*, *Sg*] [] :: *Cat*

```
ghci 51> Cat "goldilocks" "NP" [Thrd, Fem, Sg] []
    "goldilocks" NP [Thrd, Fem, Sg]
```

```
ghci 52> :t Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] [])
    Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] []) :: ParseTree nonterminal Cat
```

```
ghci 53> Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] [])
    "goldilocks" NP [Thrd, Fem, Sg]
```

```
ghci 54> t2c $ Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] [])
    "goldilocks" NP [Thrd, Fem, Sg]
```

```
ghci 55> Cat "runs" "VP" [Tense, Sg] []
    "runs" VP [Tense, Sg]
```

```
ghci 56> Leaf (Cat "runs" "VP" [Tense, Sg] [])
    "runs" VP [Tense, Sg]
```

```
ghci 57> t2c $ Leaf (Cat "runs" "VP" [Tense, Sg] [])
    "runs" VP [Tense, Sg]
```

```
ghci 58> :t Branch (Cat "" "S" [] []) [Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] []), Leaf (Cat "runs" "VP" [Tens
    Branch (Cat "" "S" [] []) [Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] []),
    Leaf (Cat "runs" "VP" [Tense, Sg] [])] :: ParseTree Cat Cat
```

```
ghci 59> Branch (Cat "" "S" [] []) [Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] []), Leaf (Cat "runs" "VP" [Tense, S
    ["" S [] : "goldilocks" NP [Thrd, Fem, Sg] "runs" VP [Tense, Sg]]
```

```
ghci 60> t2c $ Branch (Cat "" "S" [] []) [Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] []), Leaf (Cat "runs" "VP" [Te
    "" S []
```

(*ii*) a function that checks whether 2 trees agree

```
ghci 61> :t agreeC
  agreeC :: ParseTree Cat Cat → ParseTree Cat Cat → Bool
```

```
ghci 62> agreeC (Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] [])) (Leaf (Cat "runs" "VP" [Tense, Sg] []))
  True
```

```
ghci 63> agreeC (Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] [])) (Leaf (Cat "runs" "VP" [Tense, Pl] []))
  False
```

(*iii*) a function that assigns an agreement feature to a category

```
ghci 64> :t assignT
  assignT :: Feat → ParseTree Cat Cat → [ParseTree Cat Cat]
```

```
ghci 65> assignT Nom $ Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] [])
  ["goldilocks" NP [Fem, Sg, Thrd, Nom]]
```

```
ghci 66> assignT Sg $ Branch (Cat "" "S" [] []) [Leaf (Cat "goldilocks" "NP" [Thrd, Fem, Sg] []), Leaf (Cat "runs"
  [["" S [Sg] : "goldilocks" NP [Thrd, Fem, Sg] "runs" VP [Tense, Sg]]]
```

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], [])]
```

```
ghci 71> leafP "NP" [Cat "runs" "VP" [Tense, Sg] []]
  []
```

```
ghci 72> :t leafP "VP"
  leafP "VP" :: PARSER Cat Cat
```

```
ghci 73> leafP "VP" [Cat "runs" "VP" [Tense, Sg] []]
  [("runs" VP [Tense, Sg], [])]
```

```
ghci 74> leafP "VP" [Cat "goldilocks" "NP" [Thrd, Fem, Sg] []]
  []
```

### 2.2 Parser for sentences

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

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

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

```
ghci 78> parseSent $ [Cat "goldilocks" "NP" [Thrd, Fem, Sg] [], Cat "runs" "VP" [Tense, Sg] [], Cat "quickly" "Adv
  [(["" S [] : "goldilocks" NP [Fem, Sg, Thrd, Nom] ["" VP [Sg, Tense] : "runs" VP [Tense,
  Sg]]], ["quickly" AdvP []])]
```

### 2.3 NP, DET, CN and PP parsers

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

```
ghci 80> parseNP [Cat "every" "DET" [Sg] [], Cat "princess" "CN" [Sg, Fem, Thrd] []]
  [(["" NP [Fem, Sg, Thrd] : "every" DET [Sg] "princess" CN [Sg, Fem, Thrd]], [])]
```

```
ghci 81> parseDET [Cat "every" "DET" [Sg] []]
    [("every" DET [Sg],[])]
```

```
ghci 82> parseCN [Cat "princess" "CN" [Sg,Fem,Thrd] []]
    [("princess" CN [Sg,Fem,Thrd],[])]
```

```
ghci 83> parsePrep [Cat "with" "PREP" [With] []]
    [("with" PREP [With],[])]
```

```
ghci 84> parsePP [Cat "with" "PREP" [With] [],Cat "every" "DET" [Sg] [],Cat "princess" "CN" [Sg,Fem,Thrd] []]
    [(["" PP [Fem,Sg,Thrd,AccOrDat,With] : "with" PREP [With] ["" NP [Fem,Sg,Thrd,
    AccOrDat] : "every" DET [Sg] "princess" CN [Sg,Fem,Thrd]]],[])]
```

## 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.

```
ghci 85> vpRule [Cat "took" "VP" [Tense] [Cat "" "NP" [AccOrDat] []],Cat "a" "DET" [Sg] [],Cat "sword" "CN" [Sg
    [(["" VP [Tense] : "took" VP [Tense] ["" NP [Neutr,Sg,Thrd] :
    "a" DET [Sg] "sword" CN [Sg,Neutr,Thrd]]]],[])]
```

```
ghci 86> vpRule [Cat "took" "VP" [Tense] [Cat "" "NP" [AccOrDat] []],Cat "a" "DET" [Sg] [],Cat "sword" "CN" [Sg
    [(["" VP [Tense] : "took" VP [Tense] ["" NP [Neutr,Sg,Thrd] :
    "a" DET [Sg] "sword" CN [Sg,Neutr,Thrd]]]],["to" PREP [To],"alice" NP [Thrd,
    Fem,Sg]])]
```

```
ghci 87> vpRule [Cat "gave" "VP" [Tense] [Cat "" "NP" [AccOrDat] [],Cat "" "PP" [To] []],Cat "a" "DET" [Sg] [],C
    []
```

```
ghci 88> vpRule [Cat "gave" "VP" [Tense] [Cat "" "NP" [AccOrDat] [],Cat "" "PP" [To] []],Cat "a" "DET" [Sg] [],C
    [(["" VP [Tense] : "gave" VP [Tense] ["" NP [Neutr,Sg,Thrd] :
    "a" DET [Sg] "sword" CN [Sg,Neutr,Thrd]] ["" PP [Fem,Sg,Thrd,AccOrDat,
    To] : "to" PREP [To] "alice" NP [Fem,Sg,Thrd,AccOrDat]]]],[])]
```

We also have a rule for finite VPs resulting from combining an auxiliary and a VP:

```
ghci 89> parseAux [Cat "didn't" "AUX" [] []]
    [("didn't" AUX [], [])]
```

```
ghci 90> auxVpRule [Cat "didn't" "AUX" [] [], Cat "smile" "VP" [Infl] []]
    [(["" VP [] : "didn't" AUX [] ["" VP [Infl] : "smile" VP [Infl]]], [])]
```

```
ghci 91> auxVpRule [Cat "didn't" "AUX" [] [], Cat "smiled" "VP" [Tense] []]
    []
```

## 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]]]]]
```

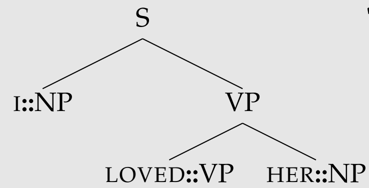
```
ghci 96> prs "She didn't love me." !! 0
    ["" S [] : "she" NP [Fem, Sg, Thrd, Nom, Pers] ["" VP [] : "didn't" AUX [] ["" VP [Infl] :
    "love" VP [Infl] "me" NP [Pers, Fst, Sg, AccOrDat]]]]
```

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

We also have a convenience function that generates `TeX`-compilable trees (based on code by Christina Unger):

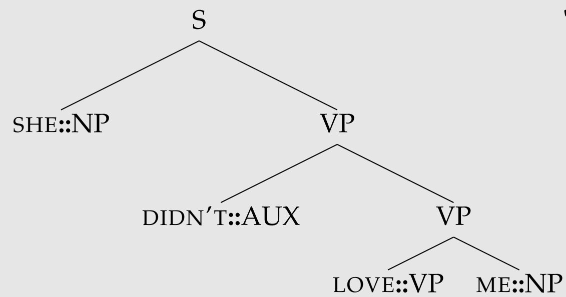
**ghci 98>** *writeTree2Tex* ((*prs* `"I loved her."`) !! 0)

" "




**ghci 99>** *writeTree2Tex* ((*prs* `"She didn't love me."`) !! 0)

" "




**ghci 100>** *writeTree2Tex* ((*prs* `"The dwarf didn't defeat the giant."`) !! 0)

" "