SMT Parser: Tutorial and Reference Manual

Sandiway Fong, University of Arizona
September 2024

Background

The hand-built English lexicon is in file `lex.prolog`. Assuming you're in the same directory as the file, to load it, run:

```
?- [lex].
```

After editing the file, you should always reload it and see if there are coding errors.
You do not need to reload the entire parser (`code.prolog`).
Loading `code.prolog` will also load `lex.prolog`.

Worked Example

Let's add the words to LEX to handle the following (Berwick & Chomksy, 2016) sentence:

birds that fly instinctively swim

In Terminal, `cd` to the smtparser directory, start `swipl` and load in the parser.

```
smtparser$ swipl
?- [code].
```

The Reference Manual section explains each of the queries used in the worked example.

If you are unfamiliar with Prolog syntax or the interpreter, please consult the SWI-Prolog quickstart section (https://www.swi-prolog.org/pldoc/man?section=quickstart) first.

Birds

```
?- θR(bird).
false.
```

Add the missing θ-relevant word *bird*, and its plural form *birds*. Edit LEX (`lex.prolog`), add:

```
θR(bird).
```

Reload LEX and test:

```
?- [lex].
true.

?- θR(bird).
true.
```

Test the NUM plural (pl) feature value. Because of its regular morphology, you only need to define the singular form of the common noun.

```
?- num(bird,pl).
false.

?- num(birds,pl).
true.
```

The Initial Workspace (WS) is populated using `word2heads/3` for each word in the input. Test it. Type semicolon (;) if it pauses to obtain other possible answers.

```
?- word2heads(birds,Hd,As).
Hd = [birds],
As = [] ;
Hd = [bird_θ, v_bird:θ:pres, INFL_v:3sg],
As = [] ;
false.
```

Note, it allows *bird* to be a transitive verb as well. For the inflected form *birds*, it returns ϕ-feature `3sg` and TNS feature present (`pres`). This is because the WordNet LEX is active in the background by default. See `vStem/3` below. We can turn off the WordNet LEX using `noMorphy/0`. Then *birds* is only a common noun.

```
?- vStem(birds,V,S).
V = bird,
S = s.

?- noMorphy.
true.

?- word2heads(birds,Hd,As).
Hd = [birds],
As = [] ;
false.
```

From here onwards, we proceed assuming the WordNet LEX has been turned off.

Fly

The verb *fly* is unergative in the worked example. However, LEX has *fly* listed as transitive only as there are two θ-roles listed, one is the complement of *fly*, indicated by verbal root fly_{θ}, the other is associated with v, indicated by $v_{fly:\theta:pres}$. Note there are two entries for the form fly as it can be untensed or have TNS feature `pres`.

```
?- word2heads(fly,Hd,As).
Hd = [fly_θ, v_fly:θ:pres, INFL_v],
As = [] ;
Hd = [fly_θ, v_fly:θ, INFL_v],
```

```
As = [] ;
false.
```

To modify the argument-taking properties of fly, examine `vR/3` in LEX.

```
vR(fly, v:fly:θ, θ).          % they are flying planes
```

We can look for and replicate the `vR/3` entry associated with an unergative verb like *dance*. The empty set symbol (ø) indicates that the verb root does not have a complement.

```
vR(dance, v:dance:θ, ø).      % Mary dances
vR(fly, v:fly:θ, ø).          % birds fly
```

Reload LEX and test. Notice we have 4 forms now, two transitive, two unergative.

```
?- word2heads(fly,Hd,As).
Hd = [fly_θ, v_fly:θ:pres, INFL_v],
As = [] ;
Hd = [fly_ø, v_fly:θ:pres, INFL_v],
As = [] ;
Hd = [fly_θ, v_fly:θ, INFL_v],
As = [] ;
Hd = [fly_ø, v_fly:θ, INFL_v],
As = [] ;
false.
```

The WordNet LEX also lists *fly* as a noun. We will generally have additional entries if `noMorphy/0` is not turned on.

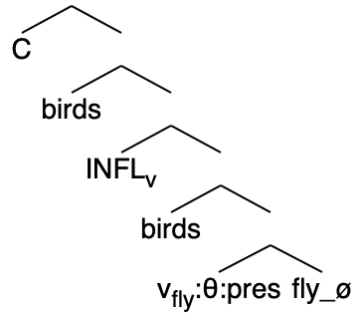
Let's test *birds fly*, using `parse/3`. The variable `SO` will hold the value of the convergent syntactic object.

```
?- parse([birds,fly],SO,L).

Words: birds fly
Initial WS: fly_θ  v_fly:θ:pres  INFL_v  birds
INT/EXT: {C, {birds, {INFL_v, {birds, {v_fly:θ:pres, {fly_θ, birds}}}}}}
Initial Spellout: birds 3pl pres fly birds
Spellout: birds fly birds
Error: spellout inconsistent with original words!
Initial WS: fly_ø  v_fly:θ:pres  INFL_v  birds
INT/EXT: {C, {birds, {INFL_v, {birds, {v_fly:θ:pres, fly_ø}}}}}
Initial Spellout: birds 3pl pres fly
Spellout: birds fly
Parse found!
SO = {C, {birds, {INFL_v, {birds, {v_fly:θ:pres, fly_ø}}}}},
L = [birds, fly] ;

Initial WS: fly_θ  v_fly:θ  INFL_v  birds
INT/EXT: {C, {birds, {INFL_v, {birds, {v_fly:θ, {fly_θ, birds}}}}}}
Initial Spellout: birds 3pl fly birds
Error: Missing TNS
Initial WS: fly_ø  v_fly:θ  INFL_v  birds
```

```
INT/EXT: {C, {birds, {INFL_v, {birds, {v_fly:θ, fly_ø}}}}}
Initial Spellout: birds 3pl fly
Error: Missing TNS
false.
```

Let's examine the parse below briefly. The root is non-complement-taking `fly_ø`, and the head v selects for the root *fly*, second-selects for a θ-relevant WS item, and has TNS pres, as indicated by head `v_fly:θ:pres`. FormCopy ensures only the top copy of *birds* at the surface subject position (spec-INFL) is pronounced, the lower copy in θ-position is necessary for INT.

```
SO = {C, {birds, {INFL_v, {birds, {v_fly:θ:pres, fly_ø}}}}}
```



Let's also check LEX for *swim* as a verb. In this case, using `word2heads/3`, we see *swim* is already present as an unergative verb.

```
?- word2heads(swim,Hd,As).
Hd = [swim_ø, v_swim:θ:pres, INFL_v],
As = [] ;
Hd = [swim_ø, v_swim:θ, INFL_v],
As = [] ;
false.
```

This is confirmed by the `vR/3` lexical entry:

```
vR(swim, v:swim:θ, ø).        % eagles swim
```

that

In this implementation, relative clauses are introduced by a relative pronoun C_{rel}. If you look in LEX for relevant information concerning 'C_rel', you will find a definition for (special word) `spWord/2`:

```
%% for relativization, at parse time
%%   spellout feature word(W)

spWord(who,['C_rel':word(who)]). % ambiguous between noun and relative
pronoun
spWord(whom,['C_rel':word(whom)]).
spWord(that,['C_rel':word(that)]).
```

[Note: in LEX, there is both `spWord/1` and `spWord/2`. These should be consolidated in a future update.]

The listing in `spWord/2` means the parser can convert an input word *that* into the head `'C_rel'`. The feature `word(that)` is not used in Merge. Using `word2heads/3`, we see:

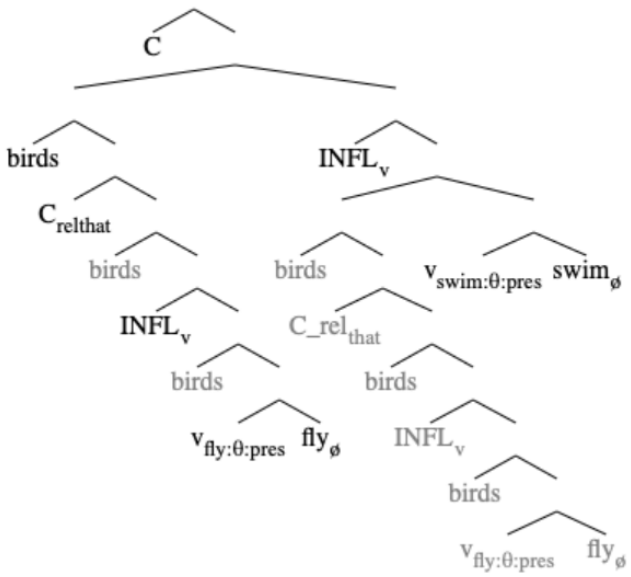
```
?- word2heads(that,Hs,As).
Hs = [C_rel:word(that)],
As = [] ;
false.
```

We can check LEX lookup for the entire sentence using `words2heads/3`. (We omit `As` below for expository brevity.) The correct Initial WS highlighted below is the one that has both *fly* and *swim* as unergatives with TNS pres. (We currently assume both the relative clause and matrix clause must contain tensed verbs. This should be updated to permit untensed relative clauses as long as nonfinite marker *to* is present in English, e.g. *the desire to fly*.)

```
?- words2heads([birds,that,fly,swim],As,WS).
1. WS = [swim_ø, v_swim:θ:pres, INFL_v, fly_θ, v_fly:θ:pres, INFL_v, C_rel_that,
   birds] ;
2. WS = [swim_ø, v_swim:θ, INFL_v, fly_θ, v_fly:θ:pres, INFL_v, C_rel_that, birds] ;
3. WS = [swim_ø, v_swim:θ:pres, INFL_v, fly_ø, v_fly:θ:pres, INFL_v, C_rel_that,
   birds] ;
4. WS = [swim_ø, v_swim:θ, INFL_v, fly_ø, v_fly:θ:pres, INFL_v, C_rel_that, birds] ;
5. WS = [swim_ø, v_swim:θ:pres, INFL_v, fly_θ, v_fly:θ, INFL_v, C_rel_that, birds] ;
6. WS = [swim_ø, v_swim:θ, INFL_v, fly_θ, v_fly:θ, INFL_v, C_rel_that, birds] ;
7. WS = [swim_ø, v_swim:θ:pres, INFL_v, fly_ø, v_fly:θ, INFL_v, C_rel_that, birds] ;
8. WS = [swim_ø, v_swim:θ, INFL_v, fly_ø, v_fly:θ, INFL_v, C_rel_that, birds] ;
false.
```

The parse given by Initial WS #3 above is:

{{birds, {$C_{relthat}$, {birds, {$INFL_v$, {birds, {$v_{swim:θ:pres}$, $swim_ø$}}}}}}, {$INFL_v$, {{birds,{$C\_rel_{word(that)}$,{birds,{$INFL_v$,{birds,{$v_{swim:θ:pres}$,$swim_ø$}}}}}}, {$v_{fly:θ:pres}$, $fly_ø$}}}}}



[Note: there is a slight discrepancy in rendering between $C_{relthat}$ and C_rel_{that}. Not critical, but should be fixed.]

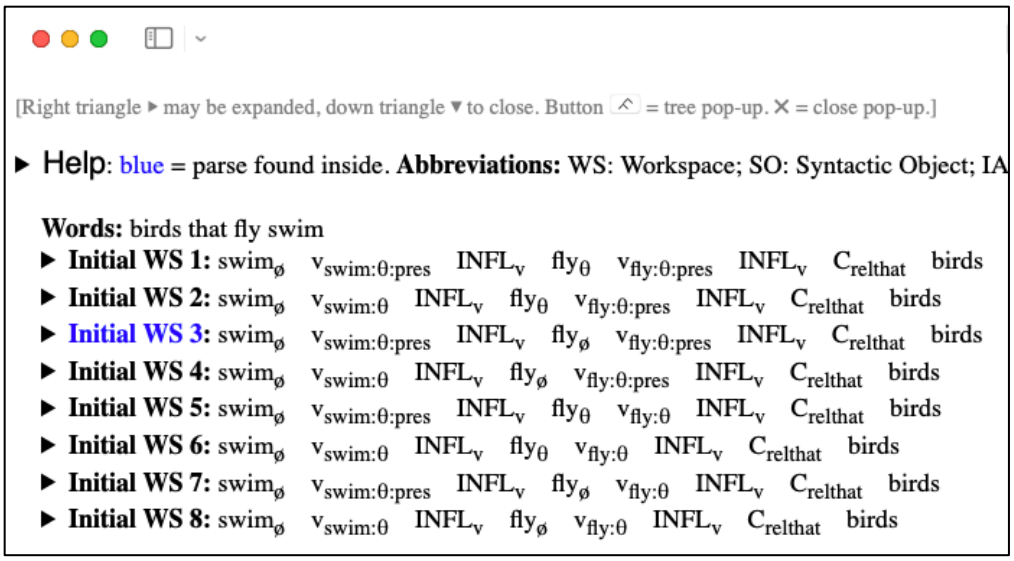
For completeness, here is the Terminal output of the `parse/3` command:

```
?- parse([birds,that,fly,swim],SO,L).

Words: birds that fly swim
Initial WS: swim_ø  v_swim:θ:pres  INFL_v  fly_θ  v_fly:θ:pres  INFL_v  C_rel_that
birds
INT/EXT: {C, {birds, {INFL_v, {birds, {v_fly:θ:pres, {fly_θ, {birds, {C_rel_that,
{birds, {INFL_v, {birds, {v_swim:θ:pres, swim_ø}}}}}}}}}}}}
Initial Spellout: birds 3pl pres fly birds that 3pl pres swim
Spellout: birds fly birds that swim
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}, {INFL_v,
{{birds,{C_rel_that:,{birds,{INFL_v:,{birds,{v_swim:θ:pres:,swim_ø:}}}}}},
{v_fly:θ:pres, {fly_θ, birds}}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly birds
Spellout: birds that swim fly birds
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}, {INFL_v,
{{birds,{C_rel_that:,{birds,{INFL_v:,{birds,{v_swim:θ:pres:,swim_ø:}}}}}},
{v_fly:θ:pres, {fly_θ, {birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}}}}}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly birds that 3pl pres swim
Spellout: birds that swim fly birds that swim
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}, {INFL_v,
{{birds,{C_rel_that:,{birds,{INFL_v:,{birds,{v_swim:θ:pres:,swim_ø:}}}}}},
{v_fly:θ:pres, {fly_θ, {birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}}}}}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly birds that 3pl pres swim
Spellout: birds that swim fly birds that swim
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}, {INFL_v,
{{birds,{C_rel_that:,{birds,{INFL_v:,{birds,{v_swim:θ:pres:,swim_ø:}}}}}},
{v_fly:θ:pres, {fly_θ, birds}}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly birds
Spellout: birds that swim fly birds
Error: spellout inconsistent with original words!
INT/EXT: {C, {birds, {INFL_v, {birds, {v_fly:θ:pres, {fly_θ, {birds, {C_rel_that,
{birds, {INFL_v, {birds, {v_swim:θ:pres, swim_ø}}}}}}}}}}}}
Initial Spellout: birds 3pl pres fly birds that 3pl pres swim
Spellout: birds fly birds that swim
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}, {INFL_v,
{{birds,{C_rel_that:,{birds,{INFL_v:,{birds,{v_swim:θ:pres:,swim_ø:}}}}}},
{v_fly:θ:pres, {fly_θ, {birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}}}}}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly birds that 3pl pres swim
Spellout: birds that swim fly birds that swim
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_that, {birds, {INFL_v, {birds, {v_swim:θ:pres,
swim_ø}}}}}}, {INFL_v,
```

{{birds,{C_rel_*that:*,{birds,{INFL_*v:*,{birds,{v_*swim:θ:pres:*,swim_*ø:*}}}}}},
{v_*fly:θ:pres*, {fly_*θ*, {birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*swim:θ:pres*,
swim_*ø*}}}}}}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly birds that 3pl pres swim
Spellout: birds that swim fly birds that swim
Error: spellout inconsistent with original words!
INT/EXT: {C, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*, {fly_*θ*, {birds, {C_rel_*that*,
{birds, {INFL_*v*, {birds, {v_*swim:θ:pres*, swim_*ø*}}}}}}}}}}}}
Initial Spellout: birds 3pl pres fly birds that 3pl pres swim
Spellout: birds fly birds that swim
Error: spellout inconsistent with original words!
Initial WS: swim_*ø* v_*swim:θ* INFL_*v* fly_*θ* v_*fly:θ:pres* INFL_*v* C_rel_*that* birds
Initial WS: swim_*ø* v_*swim:θ:pres* INFL_*v* fly_*ø* v_*fly:θ:pres* INFL_*v* C_rel_*that*
birds
INT/EXT: {C, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*swim:θ:pres*,
swim_*ø*}}}}}}, {INFL_*v*,
{{birds,{C_rel_*that:*,{birds,{INFL_*v:*,{birds,{v_*swim:θ:pres:*,swim_*ø:*}}}}}},
{v_*fly:θ:pres*, fly_*ø*}}}}}
Initial Spellout: birds that 3pl pres swim 3pl pres fly
Spellout: birds that swim fly
Error: spellout inconsistent with original words!
INT/EXT: {C, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*,
fly_*ø*}}}}}}, {INFL_*v*,
{{birds,{C_rel_*that:*,{birds,{INFL_*v:*,{birds,{v_*fly:θ:pres:*,fly_*ø:*}}}}}},
{v_*swim:θ:pres*, swim_*ø*}}}}}
Initial Spellout: birds that 3pl pres fly 3pl pres swim
Spellout: birds that fly swim
Parse found!
SO = {C, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}},
{INFL_*v*, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}},
{v_*swim:θ:pres*, swim_*ø*}}}}},
L = [birds, that, fly, swim] ;

Initial WS: swim_*ø* v_*swim:θ* INFL_*v* fly_*ø* v_*fly:θ:pres* INFL_*v* C_rel_*that* birds
Initial WS: swim_*ø* v_*swim:θ:pres* INFL_*v* fly_*θ* v_*fly:θ* INFL_*v* C_rel_*that* birds
Initial WS: swim_*ø* v_*swim:θ* INFL_*v* fly_*θ* v_*fly:θ* INFL_*v* C_rel_*that* birds
Initial WS: swim_*ø* v_*swim:θ:pres* INFL_*v* fly_*ø* v_*fly:θ* INFL_*v* C_rel_*that* birds
Initial WS: swim_*ø* v_*swim:θ* INFL_*v* fly_*ø* v_*fly:θ* INFL_*v* C_rel_*that* birds
false.

The command `report.` will produce a compact and browser-friendly output, which can be expanded.



Instinctively

Finally, we need to add the adverb *instinctively* to LEX. Adverbs are listed by `advWord/1`.

```
advWord(carefully,v+_).        % + permits unkeyed match
advWord(furiously,v+_).
```

We assume adverbs must seek a verb to modify, in the theory these are vP selecting adverbs. The second parameter is v+_, this means it selects for a phrase headed by v+_. Verbs are inserted into the Initial WS as a cluster of heads, usually INFL, v and the verb root. These heads are clustered together using the + notation. Underscore (_) here means we permit the adverb to match any vP. (If valued using the key to a cluster (instead of underscore), the selection will keyed too. Keying is clearly incorrect for adverbs.)

We add:

```
advWord(instinctively,v+_).
```

and reload LEX, checking the entry using `word2heads/2`.

```
?- [lex].
true.

?- word2heads(instinctively,L,As).
L = [instinctively:v+_],
As = [] ;
false.
```

Then we parse the complete sentence as follows:

```
?- parse([birds,that,fly,instinctively,swim],SO,L).
Words: birds that fly instinctively swim
Initial WS: swim  instinctively_v  fly  C_rel_that  birds
Initial WS: swim_ø  v_swim:θ:pres  INFL_v  instinctively_v  fly  C_rel_that  birds
```

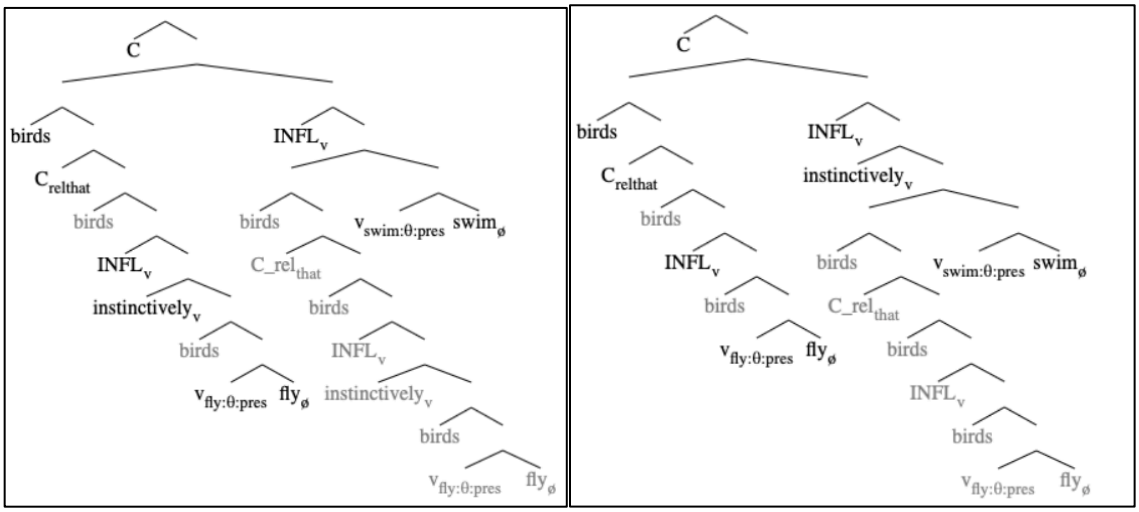
Initial WS: swim_*ø* v_*swim:θ* INFL_*v* instinctively_*v* fly C_rel_*that* birds
Initial WS: swim instinctively_*v* fly_*θ* v_*fly:θ:pres* INFL_*v* C_rel_*that* birds
Initial WS: swim_*ø* v_*swim:θ:pres* INFL_*v* instinctively_*v* fly_*θ* v_*fly:θ:pres* INFL_*v* C_rel_*that* birds
…
Initial Spellout: birds that 3pl pres fly instinctively 3pl pres swim
Spellout: birds that fly instinctively swim
Parse found!
SO = {C, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {instinctively_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}}}, {INFL_*v*, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {instinctively_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}}}, {v_*swim:θ:pres*, swim_*ø*}}}}},
L = [birds, that, fly, instinctively, swim] ;

INT/EXT: {C, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}}}, {INFL_*v*, {instinctively_*v*, {{birds,{C_rel_*that:*,{birds,{INFL_*v:*,{birds,{v_*fly:θ:pres:*,fly_*ø:*}}}}}}}, {v_*swim:θ:pres*, swim_*ø*}}}}}
Initial Spellout: birds that 3pl pres fly instinctively 3pl pres swim
Spellout: birds that fly instinctively swim
Parse found!
SO = {C, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}}}, {INFL_*v*, {instinctively_*v*, {{birds, {C_rel_*that*, {birds, {INFL_*v*, {birds, {v_*fly:θ:pres*, fly_*ø*}}}}}}}, {v_*swim:θ:pres*, swim_*ø*}}}}},
L = [birds, that, fly, instinctively, swim] ;

Initial Spellout: birds that 3pl pres fly 3pl pres swim instinctively
…

We obtain the following two parses:

{C, {{birds, {$C_{relthat}$, {birds, {$INFL_v$, {$instinctively_v$, {birds, {$v_{fly:θ:pres}$, $fly_ø$}}}}}}}, {$INFL_v$, {{birds,{$C\_rel_{that:}$,{birds,{$INFL_{v:}$,{$instinctively_{v:}$,{birds,{$v_{fly:θ:pres:}$,$fly_{ø:}$}}}}}}}, {$v_{swim:θ:pres}$, $swim_ø$}}}}}
{C, {{birds, {$C_{relthat}$, {birds, {$INFL_v$, {birds, {$v_{fly:θ:pres}$, $fly_ø$}}}}}}}, {$INFL_v$, {instinctively$_v$, {{birds,{$C\_rel_{that:}$,{birds,{$INFL_{v:}$,{birds,{$v_{fly:θ:pres:}$,$fly_{ø:}$}}}}}}}, {$v_{swim:θ:pres}$, $swim_ø$}}}}}



Reference Manual: LEX

Basic Notation

Generally, we write H_F for head H has feature F.
Features are separated by a colon (:), e.g. $H_{F1:F2}$ means head H has features F1 and F2, and so on.

In LEX, we write heads as H:F1..:Fn (plain text), portrayed as `H_F1..:Fn` in Terminal output (underscore used to separate the name of the head from its features, note: no underscore in LEX), and $H_{F1..:Fn}$ in Browser output (features are subscripts).

{…} represents a set formed by Merge. Sets may be nested (hierarchical structure).
{…} represents a set formed by FormSet.

IA: internal θ-argument.
EA: external θ-argument.

For θ-Merge, the first and 2nd features are important.
H_θ means head H takes a θ-relevant complement.
Examples: adjectival phrase {$long_\theta$, corridor}, verb with internal argument {$like_\theta$, sea_{the}}.

$H_{F:\theta}$ means head H takes a θ-relevant specifier (2nd Merge).
Example: v here selects for complement *like* and θ-relevant specifier {EA, {$v_{like:\theta}$, {$like_\theta$, sea_{the}}}.

Nouns

Checklist

For each new noun:

1. Define `θR/1`.
2. Check to see `morphyStem/3` works for stemming the plural form.
   Define `num/2` for plural form.
3. Check its `ϕ`-features using `ϕ/2`.
4. Decide whether you need to define `case/2`, `whR/1` and `animate/1`.

θ-relevant

Nouns are θ-relevant, and selected by heads with complement or specifier (2nd Merge) feature θ.
Define `θR/1`. Sample entries:

```
θR(ball).
θR(boat).
θR(bombing).
θR(car).
θR(corridor).
θR(city).
```

NUM feature

θ-relevant nouns are assumed to be grammatically singular unless otherwise defined. Plurals are defined using `num/2`. To determine whether a noun is plural we enlist WordNet morphy for stemming.

```
morphyStem(Noun,n,Root)
?- morphyStem(ball,n,Root).          Root = ball.
?- morphyStem(balls,n,Root).         Root = ball.
```

Then if *ball* + *s* = *balls*, we infer *balls* is plural.
We also have a rule implemented that states if a noun ends in -*y*, plural form ends in -*ies*.

```
?- morphyStem(lady,n,Root).          Root = lady.
?- morphyStem(ladies,n,Root).        Root = lady.
```

That is, *lady* - *y* +*ies* = *ladies*.

But, morphy doesn't always work:

```
?- morphyStem(men,n,Root).     Root = men.
```

We cannot infer from identical form and root (reported by morphy) that *men* is plural.
For these cases, we list them directly using num/2.

```
num(men,PL) :- !, PL = pl.     % red
num(women,PL) :- !, PL = pl.   % red
```

So:

```
?- num(men,NUM).          NUM = pl.
?- num(men,sg).           false.
```

Currently, we test `num/2` for plurality only, i.e. we will only call `num/2` with 2nd argument set to `pl`. The following queries are never made:

```
?- num(man,NUM).          false.
?- num(man,sg).           false.
```

ϕ-features

`ϕ(N,Phi)` returns `ϕ`-feature values `Phi` for a noun `N`. θ-relevant nouns are assumed to be 3sg unless otherwise defined. A noun `N` that satisfies `num(N,pl)` is 3pl.

Pronouns `ϕ`-feature values are specified using `per/2`. These definitions will be accessed by `ϕ/2`. Sample entries:

```
per('I','1sg').
per(we,'1pl').
per(us,'1pl').
per(ourselves,'1pl').
```

```
per(you,'2pl').
per(yourselves,'2pl').
per(they,'3pl').
per(them,'3pl').
per(themselves,'3pl').
```

Case

Case is only checked at EXT, not part of Merge syntax. Define restrictions on Case using caseR/2. If not defined here, assume unrestricted. Sample entries.

```
caseR('I',nom).
caseR(me,acc).
caseR(we,nom).
caseR(us,acc).
caseR(they,nom).
caseR(them,acc).
caseR(whom,acc).        % tricky: whom did John see?
```

wh-feature

wh-relevant nouns are defined using `whR/1`. Sample entries:

```
whR(what).
whR(who).
whR(whom).
```

Other features on nouns

For relative clause formation, the animacy property seems to select between relative pronouns *who* and *that*.

The man/mechanic who I saw
**The man/mechanic that I saw*
The shop that I saw
**The shop who I saw*

Define `animate/1`. Sample entries:

```
animate(old).
animate(man).
animate(mechanic).
animate(men).
animate(woman).
animate(women).
```

Determiners

Do not project in this theory, i.e. no DPs. Therefore nothing selects for a determiner.

Determiners are essentially a bundle of semantic features associated with a head noun, e.g. *the* is definite, *a* indefinite. In this implementation, we simply say *the* (and *a*) are a feature on a noun.

Define `det/1`. Sample entries:

```
det(a).
det(every).
det(one).
det(some).
det(the).
```

TODO: some theory for possessive pronouns.

```
%% temporary, fix these later with some structure
det(my).
det(your).
det(his).
det(her).
det(its).
det(our).
det(their).
```

Adjectives

Triggers θ-Merge. Optionally, nominalization can be triggered after θ-Merge.

Example: narrow$_{\theta}$ selects for an object as in *narrow corridor*.
Nominalization is optional as the follow examples demonstrate:

The corridor is narrow
I liked the narrow corridor

Defined in `aR/2`. Sample entries:

```
aR(colorless, θ).
aR(criminal, θ).
aR(dark, θ).
aR(green, θ).       % all color names can be adjectives or nouns, right?
aR(happy, θ).
aR(long, θ).
aR(narrow, θ).
aR(old, θ).
aR(sad, θ).
aR(flying, θ).              % gerunds are adjectival
```

Adverbs

Triggers selection (sMerge).

Example: carefully$_{v}$ seeks a vP to modify in *the mechanic carefully fixed the car*.

Define `advWord/2`. Sample entries:

```
advWord(carefully,v+_).       % + permits keyed match
```

```
advWord(furiously,v+_).
```

(+) keying of selected v is necessary as adverbs are part of the extended verbal projection.

Prepositions

Triggers θ-Merge and secondary selection (s2Merge).

Example: $by_{\theta:v}$ in *the food was eaten by the man.*

$by_{\theta:v}$ selects for a θ-relevant object, *the man*, and θ-Merge constructs $\{by_{\theta:v}, man_{the}\}$.
Preposition $by_{\theta:v}$ selects also for a phrase headed by v.
This is secondary selection, applied after θ-Merge above.
The phrase is $\{v_{v:pass:pres}, \{v_{eat}, \{eat_{\theta:pastp}, food_{the}\}\}\}$, headed by $v_{v:pass:pres}$.
Head $v_{v:pass:pres}$ suppresses θ-selection (a requirement peculiar to $by_{\theta:v}$).
This is so we can identify the θ-role suppressed by $v_{v:pass:pres}$ with the one selected by $by_{\theta:v}$.
Other prepositions, e.g. *with* or *on* as in *with a telescope* or *on the boat*, do not have this secondary selection requirement on vP. In other words, θ-selection by other prepositions will not identified with the main arguments of the vP.

Example derivation with $by_{\theta:v}$.

Initial WS 3: man_{the} $by_{\theta:v}$ $eat_{\theta:pastp}$ v_{eat} $INFL_v$ $v_{v:pass:pres}$ $food_{the}$
WS 1: $\{eat_{\theta:pastp}, food_{the}\}$ man_{the} $by_{\theta:v}$ v_{eat} $INFL_v$ $v_{v:pass:pres}$
WS 2: $\{v_{eat}, \{eat_{\theta:pastp}, food_{the}\}\}$ man_{the} $by_{\theta:v}$ $INFL_v$ $v_{v:pass:pres}$
WS 3: $\{v_{v:pass:pres}, \{v_{eat}, \{eat_{\theta:pastp}, food_{the}\}\}\}$ man_{the} $by_{\theta:v}$ $INFL_v$
WS 4: $\{by_{\theta:v}, man_{the}\}$ $\{v_{v:pass:pres}, \{v_{eat}, \{eat_{\theta:pastp}, food_{the}\}\}\}$ $INFL_v$
WS 5: $\{\{v_{v:pass:pres}, \{v_{eat}, \{eat_{\theta:pastp}, food_{the}\}\}\}, \{by_{\theta:v}, man_{the}\}\}$ $INFL_v$
WS 6: $\{food_{the}, \{INFL_v, \{\{v_{v:pass:pres}, \{v_{eat}, \{eat_{\theta:pastp}, food_{the}\}\}\}, \{by_{\theta:v}, man_{the}\}\}\}\}$
$\{C, \{food_{the}, \{INFL_v, \{\{v_{v:pass:pres}, \{v_{eat}, \{eat_{\theta:pastp}, \sout{food_{the}}\}\}\}, \{by_{\theta:v}, man_{the}\}\}\}\}\}$
the food 3sg pres be en eat by the man
Spellout: the food is eaten by the man

Passivization as main verb θ-role suppression

In the example above, passive $v_{v:pass:pres}$ selects for $v_{eat:\theta}$, which selects for an external argument (θ). Passive $v_{v:pass:pres}$ transforms $v_{eat:\theta}$ into v_{eat}, which then does not project an external argument (θ-role suppression).

Verbs: introducing the verbal cluster

A cluster of heads is inserted into the Initial Workspace (WS) when the parser recognizes a verb.

For example, the copula verb form *is* analyzed as the pair $INFL_{v:3sg}$-$v_{pred:pres}$. The head INFL here has ϕ-features 3sg and selects for a phrase headed by the categorizer v, which, in turn, selects for a phrase headed by a predicate (pred), e.g. *criminal*, as in Chomsky's example *the bombing of the cities is criminal*. v also has TNS feature pres (present). Note the copula has no θ-selection properties. Compare this with a regular verb such as *liked*, which is associated with

the cluster $INFL_v$-$v_{like:\theta:pst}$-$like_\theta$. The categorizer v, here with past TNS (pst) and selecting for root *like*, and *like* take a θ-relevant WS item at 2nd and 1st Merge, respectively.

The first step the parser takes is to stem the inflected verb form (using morphy), as in (a-i) and (b-i) below. For *is* and *liked*, the suffix is *-s* and *-d*, which are associated with TNS feature pres and pst (to be attached to v), respectively. In (a-iii) and (b-iii), L is the list of heads that enter the Initial WS on behalf of *is* and *liked*.

a.	b.
i. `?- vStem(is,V,S). V=be, S=s`	`?- vStem(liked,V,S).  V=like, S=d`
ii. `?- vEnding(s,TNS). TNS=pres`	`?- vEnding(d,TNS).  TNS=pst`
iii. `?- θ(is,L). L=[`$v_{pred:pres}$`, `$INFL_{v:3sg}$`]`	`?- θ(liked,L). L=[like`$_\theta$`, `$v_{like:\theta:pst}$`, `$INFL_v$`]`

Generally, there may be multiple cluster possibilities for a verb, leading to multiple (distinct) initial WS's. For example, *be* is not only the copula in English, but also a main verb, e.g. *they are aeroplanes*, and the progressive auxiliary *be*, as in *they are eating lunch*. Similary, a verb like *melt* may participate in various alternations, *the ice melted / the sun melted the ice*. We assume LEX lookup will return the appropriate cluster possibilities, e.g. as in (a) and (b).

a.	b.
`?- θ(is,L).`	`?- θ(melted,L).`
`L = [`$v_{pred:pres}$`, `$INFL_{v:3sg}$`] ;`	`L = [melt`$_\emptyset$`, `$v_{melt:\theta:pst}$`, `$INFL_v$`]`
`L = [be`$_\theta$`, `$v_{be:\theta:pres}$`, `$INFL_{v:3sg}$`]`	`L = [melt`$_\theta$`, `$v_{melt:\theta:pst}$`, `$INFL_v$`]`
`L = [`$v_{v:prog:pres}$`, `$INFL_{v:3sg}$`]`	

(Note: $melt_\emptyset$ and $melt_\theta$ signals that *melt* takes no complement (∅ = empty set) and a θ-relevant WS item as complement, respectively.)

Verbs and LEX

Two distinct systems, EXT and Merge, are involved when adding a new verb.

- EXT: requires stemming (parsing) and inflection (for spellout).
- Merge: requires verbal cluster INFL-v-Root and selectional properties to be defined.

EXT: Stemming

Morphy does general verb stemming duties when parsing, `morphyStem(Word,C,Root)`, with `C` = v (for verbs). Examples below:

```
?- morphyStem(likes,v,Root).  Root = like.
?- morphyStem(seen,v,Root).   Root = see.
?- morphyStem(sold,v,Root).   Root = sell.
?- morphyStem(saw,v,Root).    Root = saw.
```

The general interface for all input verb forms is via `vStem(Word,Root,Suffix)`. Examples below:

```
?- vStem(eat,Root,Suffix).    Root = eat, Suffix = .
```

```
?- vStem(eats,Root,Suffix).   Root = eat, Suffix = s.
?- vStem(ate,Root,Suffix).    Root = eat, Suffix = d.
?- vStem(eaten,Root,Suffix).  Root = eat, Suffix = en.
?- vStem(eating,Root,Suffix). Root = eat, Suffix = ing.
```

Suffixes generally recognized are defined in `vEnding(Suffix,TNS)`, summarized below:

1. `''` (empty) (untensed or present tense),
2. `d/ed` (past tense),
3. `s` (3sg present tense),
4. `en` (past participle), and
5. `ing` (present participle).

Endings `d`/`ed` and `''`/`s` are recognized and map to TNS features `pst` (past) and `pres` (present), respectively. `''` alone has the option of no TNS mapping for untensed, e.g. *win* in *to win*. (`s` will separately map to INFL φ-features `3sg` for Agree at EXT.)

Suffixes are computed automatically from sub-string differences between the Word and Root, not possible in the case of *sold* = *sell* + *d*, but done automatically for *sees* = *see* + *s*, *liked* = *like* + *d* and *kicked* = *kick* + *ed*. At present, a rule is also provided for Root ending in *-y* to Word ending in *-ies* and suffix *s*, e.g. as with Word *flies* and verb root *fly*.

Irregular suffixation cases must be explicitly listed as `irregularV(Word,Root,Suffix)` entries, defined in LEX. Sample entries:

```
irregularV(broken,break,en).
irregularV(broke,break,d).
irregularV(saw,see,d).
irregularV(seen,see,en).
```

Note: a default rule is run as a last resort. When no irregular entry exists and it cannot detect the suffix by sub-string, but Morphy says it is a valid verb form, it guesses the suffix is *d* (past tense) or *en* (past participle). For example, Morphy says *ate* comes from verb root *eat*, but does not supply a suffix. Next, suppose *ate* is not listed in `irregularV/3`, then `vStem/3` will return both choices:

```
?- vStem(ate,Root,Suffix).
Root = eat, Suffix = d ;
Root = eat, Suffix = en.
```

Once we add the following entry, the incorrect guess is eliminated.

```
irregularV(ate,eat,d).

?- vStem(ate,Root,Suffix).
Root = eat, Suffix = d
```

EXT: Inflection

For spellout after Merge has converged, each head `H` and its features are externalized via `ext(H,N,Copies,List,OutputFlag), where List` is a sequence of words and bound morphemes.[1] Content heads such as nouns, adjectives, adverbs and verbs spell out as named heads. Functional heads, e.g. C, v and INFL usually will not spell out as words. Instead, they may have features that may spell out as individual words or (bound) morphemes that must be attached to words. For verbs, generally there will be a root head immediately preceded by φ and TNS from functional heads INFL and v, respectively. (φ on INFL comes from Agree with the surface subject.) The mapping φ+TNS+Root to inflected form is defined via `inflectedV(Phi,TNS,Root,Word)`.

Similarly, `inflectedV(Suffix,Root,Word)` defines rules for regular (concatenative) cases such as *en*+*eat* = *eaten* and *ing* + *eat* = *eating*, as well as *ing* + *like* = *liking*.

```
?- inflectedV(ing,like,Word). Word = liking.
```

Irregular spellout must be listed explicitly in `irregularVspell(Suffix,Root,Word)` as shown below:

```
irregularVspell(en,break,broken).
irregularVspell(ing,see,seeing).

irregularVspell(en,be,been).
irregularVspell(ing,be,being).
irregularVspell(ing,have,having).
irregularVspell(en,have,had).
```

Merge

θ-configurations are built by θ-aware Merge, which relies on selectional properties of roots and other heads in the extended verbal projection. These are listed for individual verbs in `vR(Root,LittleV,R_selects)`. Examples below:

```
vR(arrive, v:arrive, θ).           % a train arrives
vR(be, v:pred, none).              % copula, just a v
vR(be, v:v+_:prog, zero).          % PROG -> be ing (keyed)
vR(dance, v:dance:θ, ø).           % Mary dances
vR(eat, v:eat:θ, θ).               % John ate a sandwich
vR(melt, v:melt:θ, ø).             % ice melted
vR(melt, v:melt:θ, θ).             % sun melted the ice
vR(must, v:v+_:modal:word(must), zero). % must have (no TNS)
vR(saw,  v:saw:θ, θ).              % John sawed the rope
vR(see,  v:see:θ, θ).              % Mary saw John
vR(want, v:want:θ, 'INFL').        % Mary wants to ...
vR(will, v:v+_:modal:word(will), zero).   % will/would have (no TNS)
```

[1] Inputs `N` and `Copies`, not described here track FormCopy heads that will not be pronounced. `N` gives the local position of the head in the structure and `Copies` contains a list of unpronounced positions. Together, they control whether the head generates output or not. If a copy is recognized, i.e. when `N` is listed in `Copies`, EXT is instructed to not generate anything for the head except with respect to Agree, e.g. in the case of Subject-Verb agreement, even if the argument is not spelled out at the edge of INFL, it can output φ-features used for verbal inflection. (Finally, `OutputFlag` controls EXT error signaling and debugging, and can be ignored.)

```
vR(win, v:win:θ, ø).                % Mary won
```

For example, the entry for unaccusative root *arrive* indicates that it is keyed with the head `v:arrive`. This means the categorizer *v* selects for a phrase headed by root *arrive*. The root *arrive* itself selects θ (denoting a θ-relevant WS item). An INFL head keyed to the catgorized v by selection is also generated. θ-aware Merge can then construct the θ-configuration in (c) below, beginning with heads (a) inserted after stemming the input *arrived*. IA represents a θ-relevant item selected from the rest of the WS. In (d), $INFL_v$ selects for the *v* phrase built in (c). $INFL_v$ triggers Search for a θ-relevant item inside the θ-configuration. There is only one possiblity, IA. Internal Merge produces (e). (After the $INFL_v$ phrase has been built, a further addition of declarative complementizer C completes the clause and Merge's task is complete.

a. $INFL_v$ $v_{arrive:pst}$ $arrive_\theta$ IA_φ C
b. $INFL_v$ $v_{arrive:pst}$ $\{arrive_{\theta:pst}, IA_\varphi\}$ C
c. $INFL_v$ $\{v_{arrive:pst}, \{arrive_\theta, IA_\varphi\}\}$ C
d. $\{INFL_v, \{v_{arrive:pst}, \{arrive_\theta, IA_\varphi\}\}\}$ C
e. $\{IA_\varphi, \{INFL_v, \{v_{arrive:pst}, \{arrive_\theta, IA_\varphi\}\}\}\}$ C
f. $\{C, \{IA_\varphi, \{INFL_v, \{v_{arrive:pst}, \{arrive_\theta, IA_\varphi\}\}\}\}$
g. $\{C, \{IA_\varphi, \{INFL_v, \{v_{arrive:pst}, \{arrive_\theta, \cancel{IA_\varphi}\}\}\}\}$ $(IA_\varphi, \cancel{IA_\varphi})$
h. C IA_φ $INFL_{v:\Phi}$ $v_{arrive:pst}$ $arrive_\theta$
i. IA φ pst arrive
j. IA arrived

In (g), FormCopy applies, constructing the c-command relation $(IA_\varphi, \cancel{IA_\varphi})$, the 2nd of which will not be externalized. (h) shows the sequence of heads (in English-particular order) transmitted to Spellout at EXT. The declarative head C does not spell out. Agree applies between surface subject IA and $INFL_v$, valuing φ on $INFL_v$, spelling out as morpheme φ at (i). At step (j), the mapping from the linear sequence φ+pst+*arrive* to *arrived* happens.

In the case of unergative verb *dance*, the root selects ø (the empty set), i.e. it does not take a complement. The categorizer is listed as `v:dance:θ`, this means v selects the root *dance* first, and a second Merge selects θ (denoting a θ-relevant WS item). In the case of verbal form *dances*, θ-aware Merge builds the θ-configuration in (c). The rest of the derivation steps follow those above, except for the φ+pres+*dance* mapping. If EA is a plural noun, e.g *they*, 3pl+pres+*dance* = *dance*, and 3sg+pres+*dance* = *dances* in the case of a singular noun such as *Mary*.

a. $INFL_v$ $v_{dance:pres}$ $dance_ø$ EA_φ C
b. $INFL_v$ $\{v_{dance:pres}, dance_ø\}$ EA_φ C
c. $INFL_v$ $\{EA_\varphi, \{v_{dance:pres}, dance_ø\}\}$ C
d. $\{INFL_v, \{EA_\varphi, \{v_{dance:pres}, dance_ø\}\}\}$ C
e. $\{EA_\varphi, \{INFL_v, \{EA_\varphi, \{v_{dance:pres}, dance_ø\}\}\}\}$ C
f. $\{C, \{EA_\varphi, \{INFL_v, \{EA_\varphi, \{v_{dance:pres}, dance_ø\}\}\}\}\}$
g. $\{C, \{EA_\varphi, \{INFL_v, \{\cancel{EA_\varphi}, \{v_{dance:pres}, dance_ø\}\}\}\}\}$ $(EA_\varphi, \cancel{EA_\varphi})$
h. C EA_φ $INFL_{v:\varphi}$ $v_{dance:pres}$ $dance_ø$

i. EA_{φ} φ pres dance
j. *They dance / Mary dances*