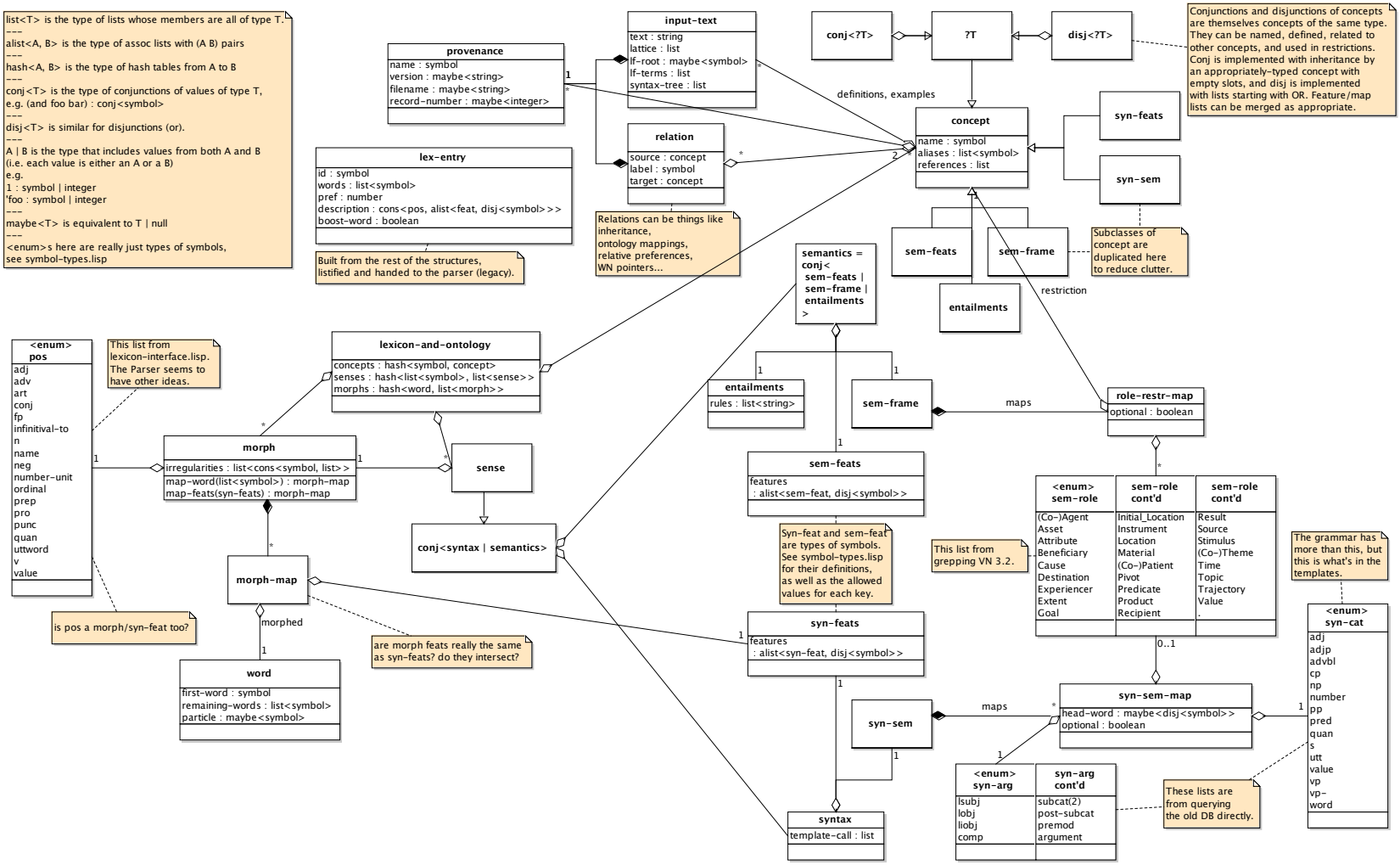
list<T> is the type of lists whose members are all of type T.
---
alist<A, B> is the type of assoc lists with (A B) pairs
---
hash<A, B> is the type of hash tables from A to B
---
conj<T> is the type of conjunctions of values of type T,
e.g. (and foo bar) : conj<symbol>
---
disj<T> is similar for disjunctions (or).

A | B is the type that includes values from both A and B
(i.e. each value is either an A or a B)
e.g.
1 : symbol | integer
'foo : symbol | integer
---
maybe<T> is equivalent to T | null

<enum>s here are really just types of symbols,
see symbol-types.lisp
provenance
name : symbol
version : maybe<string>
filename : maybe<string>
record-number : maybe<integer>
input-text
text : string
lattice : list
lf-root : maybe<symbol>
lf-terms : list
syntax-tree : list
conj<?T>
?T
disj<?T>
Conjunctions and disjunctions of concepts are themselves concepts of the same type. They can be named, defined, related to other concepts, and used in restrictions. Conj is implemented with inheritance by an appropriately-typed concept with empty slots, and disj is implemented with lists starting with OR. Feature/map lists can be merged as appropriate.
definitions, examples
concept
name : symbol
aliases : list<symbol>
references : list
syn-feats
syn-sem
relation
source : concept
label : symbol
target : concept
lex-entry
id : symbol
words : list<symbol>
pref : number
description : cons<pos, alist<feat, disj<symbol>>>
boost-word : boolean
Built from the rest of the structures, listified and handed to the parser (legacy).
Relations can be things like inheritance, ontology mappings, relative preferences, WN pointers...
sem-feats
sem-frame
entailments
Subclasses of concept are duplicated here to reduce clutter.
restriction
semantics = conj< sem-feats | sem-frame | entailments >
<enum> pos
adj
adv
art
conj
fp
infinitival-to
n
name
neg
number-unit
ordinal
prep
pro
punc
quan
uttword
v
value
This list from lexicon-interface.lisp. The Parser seems to have other ideas.
is pos a morph/syn-feat too?
lexicon-and-ontology
concepts : hash<symbol, concept>
senses : hash<list<symbol>, list<sense>>
morphs : hash<word, list<morph>>
morph
irregularities : list<cons<symbol, list>>
map-word(list<symbol>) : morph-map
map-feats(syn-feats) : morph-map
sense
morph-map
morphed
word
first-word : symbol
remaining-words : list<symbol>
particle : maybe<symbol>
are morph feats really the same as syn-feats? do they intersect?
conj<syntax | semantics>
entailments
rules : list<string>
sem-frame
maps
role-restr-map
optional : boolean
sem-feats
features
: alist<sem-feat, disj<symbol>>
Syn-feat and sem-feat are types of symbols. See symbol-types.lisp for their definitions, as well as the allowed values for each key.
<enum> sem-role
(Co-)Agent
Asset
Attribute
Beneficiary
Cause
Destination
Experiencer
Extent
Goal
sem-role cont'd
Initial_Location
Instrument
Location
Material
(Co-)Patient
Pivot
Predicate
Product
Recipient
sem-role cont'd
Result
Source
Stimulus
(Co-)Theme
Time
Topic
Trajectory
Value
.
This list from grepping VN 3.2.
The grammar has more than this, but this is what's in the templates.
syn-feats
features
: alist<syn-feat, disj<symbol>>
0..1
syn-sem-map
head-word : maybe<disj<symbol>>
optional : boolean
<enum> syn-cat
adj
adjp
advbl
cp
np
number
pp
pred
quan
s
utt
value
vp
vp-
word
syn-sem
maps
<enum> syn-arg
lsubj
lobj
liobj
comp
syn-arg cont'd
subcat(2)
post-subcat
premod
argument
These lists are from querying the old DB directly.
syntax
template-call : list