Structures in the TRIPS LexiconManager

1 Introduction

This document describes the structures that are present in the TRIPS LexiconManager and their uses.

2 lexicon-db

This lexicon-db is the top-level lexicon structure. The lexicon-db structure has the following fields:

- word-table : hash table of vocabulary-entry structures (see section 3), with the word form (including all morphological variants) as keys, as in (gethash 'w::talking (lexicon-db-word-table *lexicon-data*)).
- synt-table : hash table of syntax-template structures (see section 6) keyed by the template name as in (gethash 'agent-theme-xp-templ (lexicon-db-synt-table *lexicon-data*)).
- lf-ontology : no longer used (used to point to what is now om::*lf-ontology*)
- kr-ontology : no longer used (used to point to what is now om::*kr-ontology*)
- base-forms : list of all base word forms in the lexicon data files (same as lxm::*lexicon-data*).

3 vocabulary-entry

The vocabulary-entry structure holds all the information for a word that is specified in the LexiconManager data files, but it does not integrate any semantic or domain-specific information (i.e., the om is not called to build this structure) (compare the lex-entry structure, section 4). The vocabulary-entry structure has the following fields:

- name : system-internal unique identifier, e.g., w::take1402
- particle : e.g., w::off in entry for *take off*
- remaining-words : list of non-initial words in multi-word expressions, e.g., (w::the w::hell) for *what the hell*.
- word : the orthographic base form, e.g., w::take.
- pos : part of speech, e.g., w::n (noun), w::v (verb), etc. There is a list of TRIPS pos categories at the top of lexicon-interface.lisp.
- pronunciation : currently not used
- wfeats : list of lexically specified morphological and/or syntactic features, e.g., the irregular forms for take: ((W::MORPH (:FORMS (-VB) :PAST W::TOOK :PASTPART W::TAKEN :ING W::TAKING))). This field is used to handle irregular morphology or to define syntactically restricted uses of certain lexemes, for example *got* as an alternate past participle form for gotten: ((W::morph (:forms NIL)) (W::vform W::pastpart)).
- senses : a list of sense-definition structures (see section 5.2).
- boost-word: t for domain-independent words that should be treated identically with the domain-specialized entries for boosting, such as function words like *a*, *the*, *every*, *only*, etc. These are referred to in the system as *core words*.

4 lex-entry

This is the structure that the parser gets when it queries the lxm for a word (the parser gets a list of these structures if the word has multiple senses or there are multi-word entries with the same 1st word). The lex-entry structure is constructed for a particular word from the information specified in its word-sense-definition structure (see section 5.3) via the make-lexicon-entry function in lexicon-DB.lisp.

The lex-entry structure has the following fields:

- id : system-internal unique identifier (same as vocabulary-entry-name).
- words : list of words: for single-word entries, this is just e.g., (w::take) or (w::taking); an

example for a multi-word entry is (w::taking w::care) or (w::the w::united w::states). Note that particles don't show up here, so for *take off* the :words field would be (w::take) and the particle appears embedded in the lex-entry-description (see below).

- pref : numeric value used in parser scoring functions. This can be hand-specified in the lexicon. I thought the default for this was 1, but it seems to be .99 for the words I've inspected.
- description : list containing information for pos, particle, orthography (w::lex), lf-type (e.g., lf::consume), the semantic feature vector, list of roles and their restrictions, the var, ... in short, all the information the parser wants to see for a word.
- boost-word : t if the preference for this word should be multiplied by a boost factor (usually 1.03) to reflect in-domain (or core word) status (same as vocabulary-entry-boost-word, see above).

lex-entry structures can be accessed directly in the lxm with: (get-word-lex-entries-from-DB 'w::word *lexicon-data*) – this returns a list of lex-entry structures that have the same 1st word. The lex-entry structure has a counterpart defined in the Parser.

Note that the lex-entry-structure uses semantic information from the om to construct the complete representation of the semantic vector for the word and its semantic roles and restrictions, which only happens when the parser asks for a word, while the vocabulary-entry-structure does not.

5 sense-definitions

5.1 basic-sense-definition

The basic-sense-definition structure is :included in sense-definition structures and word-sense-definition structures (described below). It has the following fields:

- pos : part of speech, e.g., w::n, w::v; same as vocabulary-entry-pos.
- lf : the lf type, e.g., lf::consume
- sem : list of semantic features, e.g., (W::SEM ($ F::ABSTR-OBJ (F::GRADABILITY +) (F::SCALE F::OTHER-SCALE) (F::MEASURE-FUNCTION -) (F::INTENTIONAL -) (F::INFORMATION -) (F::CONTAINER -))).
- pref : same as lex-entry-pref.
- non-hierarchy-lf : function words and idiomatic expressions are assigned non-hierarchical lfs, i.e. something called an lf but it's not part of the lf ontology, e.g., (W::LF (W::OK)). I'm not sure why these are needed.
- syntax : sense-specific syntactic features. e.g., mass, case and agreement features; vform (base, present, past, passive, ...) for verbs; certain features used to restrict applications of grammar rules e.g., ((ADVBL-NECESSARY +)); Syntactic information can also be specified in vocabulary-entry-wfeats.
- boost-word : same as lex-entry-boost-word and vocabulary-entry-boost-word.
- meta-data : a meta-data structure, see section 5.1.1.

5.1.1 meta-data

The meta-data structure is used to record bookkeeping information about the lexical entries, such as when and why they were added and when and why they were changed. The meta-data structure has the following fields:

- origin : typically the project or domain that was the impetus for the addition, e.g., calo, monroe – sometimes this is the name of a student who adds the entry.
- entry-date : the date the item is added – it would be nice to time-stamp this to make it standard, right now the lexicographers just type it in.
- mod-date : date that the entry is modified – a more accurate name would be last-modified since entries can be modified multiple times.
- comments : a place for the lexicographer to enter commentary about the entry but it isn't often used – usually comments are put in cvs logs or in the data files as lisp comments.

5.2 sense-definition

The sense-definition structure appears to be used to hold the word sense data specified in the lexicon data files until they are needed for "full" processing, i.e. when the parser requests the word. The sense-definition structure includes (:include) the basic-sense-definition structure and has the following fields:

- lf-parent : lf-type, e.g., lf::consume
- lf-form : this is the base word (multi-word expressions and particle verbs will be combined

as e.g., take-care, take-off) unless otherwise specified in the lexicon. Multi-word expressions and particle verbs will be combined as e.g., take-care, take-off.

- templ : the syntactic template name, e.g., agent-theme-xp-templ
- params : used in syntactic template definition to allow variables with a default value e.g., ((XP (% W::PP (W::PTYPE W::ON))))).

All the information in the sense-definition structure is simply read directly from the lexicon data files and no information from the om is incorporated.

5.3 word-sense-definition

The word-sense-definition structure includes (:include) the basic-sense-definition structure.The word-sense-definition structure has the following fields:

- word : orthographic form
- roles : list of semantic roles (defined in the lf ontology) for this sense and the feature-list structure associated with each role ((role1 feature-list1) (role2 feature-list2)...)
- mappings : a list of arguments, one for each argument that appears in the template for a given sense, and its associated word-synsem-map structure (see section 7.3 ((lsubj word-synsem-map) (lobj word-synsem-map)...). For verbs, the arguments can be lsubj, lobj, liobj, lcomp. For nouns, the argument is typically subcat; I'm not sure about other parts of speech.
- name : the var identified with this constituent
- remaining-words
- kr-type : corresponding kr-type or nil; used only with myrosia's transform handling.
- specialized : t if a kr-type has been added; used only with myrosia's transform handling.
- transform : domain-specific transform to be used with this sense; used only with myrosia's transform handling.
- coercions : domain-specific coercions to be used with this sense.
- operator : t if the entry is obtained by performing a coercion, otherwise nil.

Word-sense-definition structures can be retrieved like this: (lxm::getworddef 'w::word).

6 syntax-template

These syntax-template structures are stored in the synt-table (see section 2).

The syntax-template structure has the following fields:

- name : name of the template, e.g., agent-theme-xp-templ (same as sense-definition-templ)
- syntax : syntactic features used during parsing; e.g., case, agreement, etc.
- mappings : an A-list of arguments (lsubj, lobj, subcat) and their associated synsem-map structure (see 7.2), e.g., ((arg1 . synsem-map1) (arg2 . synsem-map2)...).

7 synsem mappings

These structures are used when matching a grammatical argument category (subject, object, complement, indirect object) with an lf role (agent, theme, lf::of, lf::val, etc.). All lf roles have associated semantic features (if only the defaults) that must unify with the semantic features associated with the grammatical (syntactic) arguments during parsing. The semantic features for the lf roles are not specified directly in the templates – rather they are specified in the lf ontology data files and they are combined with the templates on demand, that is, when the parser asks for an entry.

7.1 basic-synsem-map

The basic-synsem-map structure is included in the synsem-map and word-synsem-map structures (below). The basic-sysnem-map structure has the following fields:

- name : grammatical argument category used by parser, e.g., LSUBJ, LOBJ, LCOMP, SUBCAT
- slot : corresponding slot in LF, e.g., lf::agent, lf::theme, lf::of, lf::val
- optional : t if this mapping can be optional (omitted). For example, consider the template in Figure 1, used with the verb *disconnect*:

  By declaring the lf::co-theme role as optional, this template allows both *He disconnected the boxcar* and *He disconnected the boxcar from the train*.
- maponly : according to a comment in the code: "t if the restriction is only for mapping but not generation, e.g., iobj" – but I still don't understand what this actually does, and there are no examples using this in the current lexicon.

```
(AGENT-THEME-CO-THEME-OPTIONAL-TEMPL
(ARGUMENTS
(LSUBJ (% W::NP) LF::AGENT)
(LOBJ (% W::NP) LF::THEME)
(LCOMP (:parameter xp (:default (% W::PP (w::ptype w::with)))) LF::CO-THEME optional)))
```

Figure 1: agent-theme-co-theme-optional template declaration

7.2 sysnsem-map

The synt-table (see section **??** is made up of sysnem-map structures, one for every template declared in the lexicon data files. The synsem-map structure includes (:include) the basic-synsem-map structure. It has the following fields:

- paramname : name of the parameter (e.g., np, vp, xp) if used – this is a mechanism to allow a parameterized specification as a form of generalization within the syntax-semantics mapping template (e.g., :parameter in the LCOMP in Figure 1.
- default : default value of the parameter if none given in the lexicon definition (e.g., w::pp is the default for the template LCOMP in Figure 1; defaults to the restriction for non-parametric mappings
- required : common restriction present for all parameters – this is a list of the following: lf, sort, lex, var, agr, case); nil for non-parametric mappings

7.3 word-sysnsem-map

The word-sysnem-map structure is used in word-sense-definition-mappings (see section 5.3). This version of the syntactic-semantic mappings is the one sent to the parser when it requests a word. The word-synsem-map structure includes (:include) the basic-synsem-map structure. It has the following fields:

- varname : This is usually the same as basic-synsem-map-name, e.g., LSUBJ, LOBJ,SUBCAT. I haven't seen an example where it is different.
- syntcat : the syntactic category of the argument, e.g., w::pp, w::np
- syntfeat: a list of syntactic features associated with this role mapping, including agreement, case, sort, .. lf, lex and var are in this list too.