Annotating Unscoped Episodic Logical Form: A Tutorial

Version 0.2.2

Gene Kim and Lenhart Schubert

May 15, 2018

1 Introduction

The annotation task is to supply preliminary, unscoped logical forms (ULFs) for sentences, which can later be disambiguated into Episodic Logical forms (ELFs). ULF is very close to a sentence’s surface form, needing only minimal human annotations. The current annotation process involves some automated steps to assist the annotator in speed and accuracy.

Let us begin with an example to understand the annotation process. Consider the sentence “Mary loves to solve puzzles”:

1. Group syntactic constituents (NPs, ADJs, VPs, etc) using round brackets:
   (Mary (loves (to (solve puzzles))));

2. Run a POS tagger over the original sentence:
   (nnp Mary) (vbz loves) (to to) (vb solve) (nns puzzles);

3. Make any necessary corrections to tags, and then use them as dot-extensions in the bracketed sentence (the tag to dot-extension will be automated):
   (Mary.nnp (loves.vbz (to.to (solve.vb puzzles.nns)))); No corrections needed

4. Convert POS extensions to logical-types, and separate tense and plural as operators:
   ([|Mary| (pres love.v) (to (solve.v (plur puzzle.n)))]);
   (|_| ↔ name (proper noun); .v ↔ verbal predicate; .n ↔ nominal predicate; to without an extension is a special reifying operator);

5. Add any necessary implicit operators (typically, type-shifting operators):
   ([|Mary| (pres love.v) (to (solve.v (k (plur puzzle.n)))]));
   (k converts a predicate that is true of ordinary singular or plural entities into a kind – i.e. an abstract individual whose instances are ordinary entities; it is applied whenever we have a common noun phrase lacking a determiner (a so-called “bare noun phrase”).)

2 Basic Annotation Components

In this section, we introduce the core logical type extensions and special operators. As the tutorial proceeds additional operators and extensions will be introduced. A more complete list of extensions and operators appears at the end of this document.
2.1 Logical-type Extensions

Below we list and describe each of the ULF logical type extensions.

- \( .n \) : nominal predicate (mouse, idea, domination, etc.)
- \( .v \) : verbal predicate (run, love, laugh, be, become, etc.)
- \( .a \) : adjectival predicate (happy, green, etc.)
- \( .adv\text{-}a, .adv\text{-}e, .adv\text{-}f, .adv\text{-}s \) : adverbial functions
- \( .cc \) : coordinator (and, or, but, etc.)
- \( .p \) : prepositional predicate (in, with, etc.)
- \( .ps, .pq, .pr \) : preposition-like words used in subordinate clause/question/relative clause formation (while, until, when, if, as though, etc.)
- \( .pro \) : pronoun (him, I, it, etc.)
- \( .d \) : determiner (the, some, few, etc.)
- \( .aux\text{-}s, .aux\text{-}v \) : auxiliary (do, will, may, etc., semantically operating at the sentence level or verb phrase level)
- \( |_| \) : name (i.e. proper noun) (Mary, Star Wars, etc.); not really an extension, but marks the type of an atomic element in the logical form.

2.2 Common Special Operators

ULF has a set of special operators that are written without logical type extensions. They represent operations that are marked morpho-syntactically in English (making it difficult to handle using symbols that correspond to words in the source sentence) or have notable and consistent interpretations in EL.

- \( \text{not} \) : negation
- \( \text{plur} \) : pluralizer
- \( \text{past/pres} \) : tense operators
- \( \text{perf} \) : perfect aspect
- \( \text{prog} \) : progressive aspect
- \( \text{k} \) : kind operator (predicate reifier)
- \( \text{ke} \) : kind of event operator (event-type reifier)
- \( \text{to} \) : action-type reifier (also written \( \text{ka} \))
- \( \text{that} \) : that operator (proposition forming operator)
3 Examples for Getting Started

We begin the tutorial by walking through the annotation of a few example sentences that include most critical and common phenomena. Understanding these examples will act as the foundation upon which we will add less common phenomena in future sections. We start with a sentence that shows the use of a couple reifying operators.

Sentence 1: “Kim knows that Sally likes to run”

1. (Kim (knows (that (Sally (likes (to run)))))
2. (nnp Kim) (vbz knows) (in that) (nnp Sally) (vbz likes) (to to) (vb run)
3. (Kim.nnp (knows.vbz (that.in (Sally.nnp (likes.vbz (to.to run.vb)))))
4. (\|Kim\| ((pres know.v) (that (\|Sally\| ((pres like.v) (to run.v)))))

Like the example in the introduction section, the automatic POS tags (step 2) require no corrections as reflected by the POS tags being unchanged in the following step. Notice that that.in and to.to are mapped to special operators which don’t have logical type extensions. In this example ‘that’ is acting as an operator that maps (the interpretation of) the sentence “Sally likes to run” into a proposition. This is typical of attitude verbs such as ‘know’, and ‘believe’ (or rather, attitude predicates such as ‘know.v’ and ‘believe.v’). Similarly, ‘to’ is forming a kind of action from the verbal predicate ‘run.v’. As we will see, there are instances where the English ‘that’ functions as a determiner (that.d, e.g., that man), as a pronoun (that.pro, e.g., cancel that) or as a relative pronoun (that.rel, e.g., in the dog that barked), and ‘to’ functions as a preposition (to.p, e.g., to Rome).

Also, notice that ‘know’ and ‘like’ become wrapped in tense operators while ‘run’ does not. Since ‘run’ is untensed here and simply forms a kind of action, no tense operator is applied to it. Also, we do not have an operator for future tense. Future tense is handled as a modal auxiliary, will – which is actually a present-tense verb. For a single chain of verbs in a declarative sentence, only the first verb is marked with the tense. As you can see in the above example, operators that create a new sentence context, such as that, allow the introduction of additional tense operators.

Sentence 2: “For John to sleep in is unusual”

1. (((for (John (to (sleep in)))) (is unusual))
2. (in for) (nnp John) (to to) (vb sleep) (in in) (vbz is) (jj unusual)
3. (((for.in (John.nnp (to.to (sleep.vb in.prt)))) (is.vbz unusual.jj))
4. (((for.p (\|John\| (sleep.v in.prt))) ((pres be.v) unusual.a))
5. ((\ke (\|John\| sleep_in.v)) ((pres be.v) unusual.a))

Sentence 2 shows an example where the subject is a kind of event. The subject argument of be.v must be an individual rather than a sentence, and ke turns a sentence (meaning) into the kind of event characterized by that sentence, and kinds are (abstract) individuals. Note also that the particle ‘in’ was initially assigned POS ‘in’ (preposition), which is incorrect – it should be ‘prt’ (a particle, which in a sense completes the verb). Ultimately, we combine the verb and its particle in the ULF, using an underscore.
Sentence 3: “Mary certainly doesn’t like the pizza”

1. (Mary certainly (does n’t (like (the pizza))))
2. (nnp Mary) (rb certainly) (aux does) (rb n’t) (vb like) (dt the) (nn pizza)
3. (Mary.nnp certainly.rb (does.aux n’t.rb (like.vb (the.dt pizza.nn))))
4. (|Mary| certainly.adv-s ((pres do.aux-s) not (like.v (the.d pizza.n))))

Sentence 3 shows the annotation of a sentential adverb, negation, and a lexical determiner. Note that an adverb like certainly preceding the verb phrase has been treated as an immediate sentence constituent, but it could have been bracketed with the verb phrase – postprocessing will produce the same result. A negation or other adverb following a verb is treated as an immediate verb phrase constituent (not bracketed with the verb). This is consistent with standard practice in Penn Treebank annotation, but in postprocessing such sentential operators will be “lifted” to sentence-level (retaining their left-to-right order; e.g., the above sentence is taken to mean “It is certain that it is not the case that Mary likes the pizza”, rather than “It is not the case that it is certain that Mary likes pizza”). Determiners are grouped with their restrictor, forming an NP (in the present case, a definite NP). In postprocessing, determiners are scoped (in general, ambiguously) at the sentence level, binding a variable and accompanied by the restrictor, which restricts the domain of the variable.

4 Short Annotation Summaries by Phenomenon

4.1 Preliminary Comments on Predicates, Modifiers, and Adverbials

Adjectival, prepositional, and verbal phrases can all function as both (1-place) predicates and as predicate modifiers (mapping one predicate to another). Their roles as predicates are most obvious when they are used to ascribe a property to some individual, as in “Alice {is very smart, is in Rome, likes poetry}”. The simplest modifier roles of such predicates are in noun post-modification, as in “food {rich in cholesterol, in the pan, spattering oil}”. As explained fully later on, such postmodifiers add predicates conjunctively to the noun they modify – e.g., we obtain a predicate expressing the property of being food and rich in cholesterol and in the pan and spattering oil.

But addition of postmodifying properties is just a special way to modify nominal predicates. Nominal predicates can also be transformed by premodifying adjective phrases and nouns, as in “harmful phony cancer drugs” (more on this below); adjectival predicates and some prepositional phrase predicates can be transformed by adverbs, as in “very smart” and “madly in love”; and verb phrase predicates can be transformed by adverbials (adverbs, and phrases used like adverbs), as in “He walked away awkwardly, with some difficulty, limping slightly”. Much else will be said about predicates and modifiers later on, but in the following subsection we just mention basic uses of adjective phrases and examples of adverbs modifying such phrases, since even the simplest sentences often involve such constituents.
4.2 Adjectival Predicates and Modifiers

Adjectives and adjective phrases (APs) like “happy”, “very happy”, “numerous”, “surprisingly numerous”, etc. can play multiple semantic roles; most often they function either as predicates (expressing a property of some entity), as in examples (a) and (c) below, or as predicate modifiers (transforming a property), as in examples (b) and (d) below:

(a) “Alice is happy”

(|Alice| ((pres be.v) happy.a))

(b) “Alice has a happy disposition”

(|Alice| ((pres have.v) (a.d (happy.a disposition.n))))

(c) “His off-shore accounts are surprisingly numerous”

((his.d (off-shore.a (plur account.n))) ((pres be.v) (surprisingly.adv-a numerous.a)))

(d) “His quite numerous off-shore accounts are illegal”

((his.d ((quite.adv-a numerous.a) (off-shore.a (plur account.n)))) ((pres be.v) illegal.a))

So in (a), “happy” ascribes a property to Alice, while in (b), it transforms the predicate disposition.n to a more specific one, (happy.a disposition.n). In (c), off-shore.a functions as a predicate modifier, while (surprisingly.adv-a numerous.a) functions as a predicate. In (d), both numerous.a and off-shore.a function as predicate modifiers, while illegal.a functions as a predicate.

Here we should note that we are in effect allowing some “sloppiness” in making type distinctions, because we are using the .a extension for both the predicate role and predicate modifier role of adjectives. This sloppiness is repaired in postprocessing ULFs, by use of an attr type-shifting operator, which converts a predicate to a predicate modifier. So, (happy.a disposition.n) will be converted to ((attr happy.a) disposition.n), (quite.adv-a numerous.a) will be converted to (attr (quite.adv-a numerous.a)), and similarly for off-shore.a in (c) and (d). These repairs are easy to implement, because the attr operator is needed for an AP just in case it modifies a noun. (Well, there’s a slight complication: A few adjectives, like “former”, “consummate”, “utter”, etc., can only function as modifiers, and in these cases the postprocessing changes former.a, consummate.a, etc., to former.attr, consummate.attr, etc., rather than (attr former.a), (attr consummate.a), etc.)

A further point here is that numeral adjectives like “5”, “five”, “many”, “numerous”, “few”, “several”, etc., also appear to function as quantifying determiners in sentences like “Tommy found 5/five/many/numerous/... insects” (see below). In fact, in such cases you can annotate the adjectives as 5.d, five.d, many.d, numerous.d, etc. However, this annotation doesn’t work when the adjectives are themselves modified, as in “around 5”, “very many”, “quite numerous”, etc., because the adverbs used here are predicate modifiers, and as such cannot modify a determiner. Therefore we view determiners like 5.d, five.d, many.d, numerous.d, etc., as abbreviations of (nquan 5.a), (nquan five.a), (nquan many.a), (nquan numerous.a), etc., as further illustrated in a later subsection. Here nquan is a type-shifting operator that converts a predicate to a determiner, and as such it can also convert APs with modifiers, e.g., (nquan (about.adv-a 5.a)), (nquan (about.adv-a five.a)), (nquan (very.adv-a many.a)), etc.
By the way, in ULF we also use bare numerals like 0, 1, 2, 3, ..., without any extensions, but these are regarded as names denoting numbers (which are abstract entities). But obviously, these names are related to the corresponding predicates, more specifically, zero\(\cdot a\) is equivalent to \((= 0)\) (the property of being equal to 0), five\(\cdot a\) is equivalent to \((= 5)\) (the property of being equal to 5), etc.

Finally, noun phrases that start with an adjective (numeral or otherwise) can also be used generically. For example, in “Six adults can carry a coffin”, we are not talking about a specific set of six adults, but about the kind of collective entity whose realizations are sets of six adults. In this case the ULF of the subject is \((k \ (six\cdot a \ (plur \ adult\cdot n)))\), where \(k\) is the kind-forming operator, as illustrated further in the following section.

4.3 Reifiers

As we have already shown in the introductory examples, our annotations have operators that convert predicates, sentences, and attitudes to objects in the language – reifiers. Consider the examples below concerning reification.

(a) “Ants are widespread”
\[\((k \ (plur \ ant\cdot n)) \ \((pres \ be\cdot v) \ \text{widespread}\cdot a)\)\]

(b) “Every child likes pets”
\[\((\text{every}\cdot d \ child\cdot n) \ \((pres \ like\cdot v) \ \ (k \ (plur \ pet\cdot n)))\)\]

(c) “I know that John is a man”
\[\((i\cdot pro \ \((pres \ know\cdot v) \ \(\text{that} \ \(\{|John| \ \((pres \ be\cdot v) \ man\cdot n)\)\)\)\)\]

(d) “Kim believes that every galaxy harbors life”
\[\(|Kim| \ \((pres \ believe\cdot v) \ \(\text{that} \ \((\text{every}\cdot d \ galaxy\cdot n) \ \((pres \ harbor\cdot v) \ \(k \ life\cdot n)\)\)\)\)\]

(e) “John knows he’s right”
\[\(|John| \ \((pres \ know\cdot v) \ \(\text{tht} \ \(\text{he}\cdot pro \ \((pres \ be\cdot v) \ right\cdot a)\)\)\)\]

(a,b,d) show the usage of the kind-operator \((k)\), which converts a predicate to an object. This operator is used whenever a predicate is treated as an object in the language. One simple method for identifying whether something is reified, is whether the word is being used as an argument to something other than a predicate-control verb (e.g. \(\text{be}\cdot v\), \(\text{feel}\cdot v\), \(\text{impress}\cdot v\), etc.). (c-e) show the usage of the that-operator \((\text{that})\), which converts attitudes to an object. It is most widely used with attitudinal verbs (e.g. believe, tell, hope, etc.). Notice in example (e), the word “that” doesn’t actually appear in the sentence although it is clearly an attitude. In such cases we use \(\text{tht}\) operator, which is semantically equivalent to \(\text{that}\) but signals the fact that “that” did not appear in the sentence.

4.3.1 Kinds of actions and events

Special versions of the kind-forming operator exist for actions and events, \(\text{to}\) and \(\text{ke}\), respectively. (We can also use \(\text{ka}\) – “kind of action” – as synonym for \(\text{to}\).) Kinds of actions are often expressed in English as infinitives, such as \(\text{to ski}\). Correspondingly, we form a kind of action in ULF by applying \(\text{to}\) to the VP meaning. Kinds of events may be expressed
in English by a sentence preceded by ‘for’, as was seen in sentence (2). This is coded by applying ke to an untensed sentence meaning; (the instances of the kind thus formed are events of that kind). Note how this contrasts with applying ‘that’ to a sentence meaning, forming an object encapsulating the particular propositional content, or claim, expressed by a sentence. Examples:

(f) “John likes to ski”
   (|John| ((pres like.v) (to ski.v)))

(g) “Mary is trying to ignore an itch”
   (|Mary| ((pres prog) (try.v (to (ignore.v (an.d itch.n)))))

(h) “For John to sleep in is unusual”
   ((ke (|John| sleep_in.v)) ((pres be.v) unusual.a))

It’s worth mentioning here that certain gerunds and NPs can also express kinds of actions and kinds of events, as in “John dislikes indoor smoking”, which may mean that he dislikes performing that kind of action, or that he dislikes that kind of event going on. In such cases we use to (or ka) to form the interpretation as a kind of action and the basic kind-forming operator k to form a kind of episode from the nominal.

4.4 Predicate complements vs. object complements

Although the predicate complement order in ULF annotation can simply follow surface order, an important distinction that we must recognize is whether a complement is a predicate or an object argument. Consider the following sentences.

(a1) “John made Mary a bookshelf”

(a2) “John made Mary a superintendent”

Although the sentences have the same surface structure, “a bookshelf” should be interpreted as an object, (a.d bookshelf.n), and “a superintendent” should be interpreted as a predicate, superintendent.n. See the following list of sentences for further practice in differentiating the two types of complements:

(b) “I found him an apartment” – (a.d apartment.n)

(c) “I found him a little apartment” – (a.d (little.a apartment.n))

(d) “I found him a nuisance” – nuisance.n

(e) “I found the house on fire” – (on.p (k fire.n))

(f) “The burglar sounded the alarm” – (the.d alarm.n)

(g) “The burglar sounded angry” – angry.a

(h) “The burglar sounded a little angry” – (a_little.adv-a angry.a)

(i) “The burglar sounded alarmed” – alarmed.a
Please try out the practice annotation problems for this section to ensure understanding of this distinction.

The category of “a little” is a bit tricky to identify; e.g., consider two possible meanings of “Mary had a little lamb”, namely, she owned a small lamb, or she consumed a small amount of lamb meat. In the former, “little” modifies “lamb” and “a” provides a determiner to produce a noun phrase (NP), while in the latter “a” combines with “little” to form a determiner a_little.d meaning “a small amount of”. But “a little” can also be an adverb, a_little.adv, as seen in (h) above, or in “It rained a little, and then the sun came out”.

4.5 Determiners

4.5.1 Lexical Determiners

Lexical determiners are annotated by bracketing the determiner with the semantic restrictor and giving the lexical entry a_.d type extension. Determiners combine (only) with nominal predicates, and the combination (which is inherently unscoped in this annotation) introduces an individual or quantifies over individuals for which the nominal predicate holds.

- “We ate some bread”
  (we.pro ((past eat.v) (some.d bread.n)))

- “Every boy owns a toy”
  ((every.d boy.n) ((pres own.v) (a.d toy.n)))

- “Few big dogs are yappy”
  ((few.d (big.a (plur dog.n))) ((pres be.v) yappy.a))

4.5.2 Determiners with a generic reading

The determiners ‘a’ and ‘an’ can have a generic reading, meaning “just about any” rather than the typical meaning “a particular”. We mark these cases specially (with operators a-gen.d and an-gen.d), since they have a very different semantic interpretation. Also, ‘the’ can be used generically, meaning “the kind”. In this case we use ‘the-gen.d’. Adjectival modification as in (c) was discussed earlier, and will be further illustrated in section 4.10.1 in a broader context. Note that examples (d) and (e) include phenomena that have not been described at this point in the document. Section 4.8 describes that details of auxiliaries such as ‘would.aux-s’, and section 4.12 explains the representation of tacitly relational nouns like friend, and an implicit referent (using place-holder *s). In (e), we use |Javan|.a to mark the adjective as name-like, and the adverb ‘now’ is represented as now.adv-e to indicate that it is an event- (or situation-) modifying adverb.

(a) “Every boy loves a dog”
  ((every.d boy.n) ((pres love.v) (a-gen.d dog.n)))

(b) “A dog loves his master”
  ((a-gen dog.n) (pres love.v) (his.d master.n))

(c) “Sam enjoys a good sandwich”
  (|Sam| ((pres enjoy.v) (a-gen.d (good.a sandwich.n))))
(d) “I would help a friend”
   (I.pro ((pres would.aux-s) (help.v (a-gen.d (friend-of.n *s)))))

(e) “The Javan tiger is now extinct”
   ((the-gen.d ([Javan].a tiger.n)) ((pres be.v) now.adv-e extinct.a))

4.6 Generated Determiners

Operators fquan and nquan are used to generate determiners from predicates. fquan is applied to fractional predicates and nquan to counting predicates.

(a) “Two out of ten voters are moderate”
   (((fquan (= .2)) (plur voter.n)) ((pres be.v) moderate.a))

(b) “Tommy found five insects.”
   ([Tommy] ((past find.v) ((nquan (= 5)) (plur insect.n))))
   ([Tommy] ((past find.v) ((nquan five.a) (plur insect.n))))
   ([Tommy] ((past find.v) ((nquan 5.a) (plur insect.n))))
   ([Tommy] ((past find.v) (five.d (plur insect.n))))
   ([Tommy] ((past find.v) (k (five.a (plur insect.n)))))

(c) “Tommy found these five insects.”
   ([Tommy] ((past find.v) (these.d (five.a (plur insect.n)))))
   ([Tommy] ((past find.v) (these.d (5.a (plur insect.n)))))

(d) “Almost all cats hunt.”
   (((fquan (almost.adv-a all.a)) (plur cat.n)) (pres hunt.v))

(e) “Very few people dislike all dogs”
   (((nquan (very.adv-a few.a)) (plur person.n))
    ((pres dislike.v) (all.d (plur dog.n)))))

   Note the various acceptable treatments of “five” in (b), depending on how we represent the predicate meaning of “five”, and whether we make use of the abbreviation of (nquan five.a) as five.d, discussed earlier. The fifth variant is based on the fact that a noun without a determiner can denote a kind, even if there are premodifying adjectives. Indeed, if we replace “found five insects” by “found large insects” or simply “found insects”, the logical form using k is the only possible one: respectively (k (large.a (plur insect.n)) and (k (plur insect.n)). But what does it mean for Tommy to find a kind whose realizations are sets of insects? Well, we take it to mean that he found a realization of that kind! This again involves a postprocessing step.
4.6.1 “Headless” noun phrases (i.e., lacking the noun)

In cases where the determiner occurs all on its own, the entire restrictor is implicit (such as in (a) and (b) below), the determiners can be annotated as pronouns, e.g., many.pro. These will be treated as abbreviations of phrases with an implicit restrictor predicate, e.g., (many.d {ref1}.n) (equivalently, ((nquan many.a) {ref1}.n)). If not specified in the ULF, default restrictors are automatically introduced in postprocessing. If such an annotation is not possible because of a constructed determiner or partially specified restrictors (such as (c-e)), an implicit referent is written in place of the head-noun predicate. Implicit referents are further discussed later in the tutorial, but generally have the syntax {ref#}.[suffix].

(a) “Those are nice”
   (Those.pro ((pres be.v) nice.a))
   ((Those.d {ref1}.n) ((pres be.v) nice.a))

(b) “Many gave their lives”
   (Many.pro ((past give.v) (their.d (plur life.n))))
   ((Many.d {ref1}.n) ((past give.v) (their.d (plur life.n))))

(c) “Nearly a hundred died”
   (((nquan (nearly.adv-a (= 100))) {ref1}.n) (past die.v))

(d) “These three are nice”
   ((these.d (three.a {ref1}.n)) ((pres be.v) nice.a))

(e) “The rich get richer”
   ((the.d (rich.a {ref1}.n)) ((pres get.v) (more.adv-a rich.a)))

Post-nominally modified noun phrases are a common form that appear headless in a determiner. Below are examples of this case (see Section 4.10.5 for more details on post-nominally modified noun phrases). Example (g) additionally includes a relative clause, which has a special regular representation – this has not yet been added to the guidelines.

(f) “Those in the display case are nice”
   ((Those.d (n+preds {ref1}.n (in.p (the.d (display.n case.n))))) ((pres be.v) nice.a))

(g) “Many who served gave their lives”
   ((Many.d (n+preds {ref1}.n (who.rel (past serve.v))))
   ((past give.v) (their.d (plur life.n))))

4.7 Passive Voice

Passive voice is annotated with the pasv operator.

• “He was pushed”
  (he.pro (past (pasv push.v)))

• “She is given an award”
  (she.pro ((pres (pasv give.v)) (an.d award.n)))
There are two important features to keep in mind. `pasv` is a lexical operator (like `plur`), so it takes narrower scope than phrasal operators (such as adverbs or reifying operators). Also, we drop the copula that accompanies the passive construction, since its semantic signal is captured by `pasv` and the tense operator.

### 4.8 Modal Auxiliaries

Annotation of modifiers in ULF requires distinguishing predicate modifiers from sentence modifiers. For adverb(ial)s, this distinction is made with adv-a extensions or operators, versus adv-s, adv-e, adv-f extensions or operators; e.g., *gracefully, completely, without difficulty* are predicate modifiers while *perhaps, surprisingly, in my opinion* are sentence modifiers. This is discussed further in a later section.

For modal auxiliaries, the predicate modifier vs. sentence modifier distinction is marked with -v and -s suffixes, respectively. For example, *must* is represented as `(pres must.aux-v)` in *The cadet must (i.e., is obligated to) obey*, whereas it is represented as `(pres must.aux-s)` in *John must have left*. Note that modal auxiliaries are never untensed, i.e., they are implicitly in present or past tense. This is made explicit in the ULF.

Here are some examples, followed by an exhaustive enumeration.

(a) “This rocket can reach Mars”

```
((this.d rocket.n) ((pres can.aux-v) (reach.v |Mars|)))
```

(b) “This mission can fail”

```
((this.d mission.n) ((pres can.aux-s) fail.v))
```

(c) “You may sit down”

```
(you.pro ((pres may.aux-v) sit_down.v))
```

(d) “The prisoner may escape”

```
((the.d prisoner.n) ((pres may.aux-s) escape.v))
```

(e) “I will send Timmy a toy”

```
(i.pro ((pres will.aux-s) (send.v |Timmy| (a.d toy.n))))
```

(f) “I constantly admonish him, but he just will not listen”

```
((i.pro constantly.adv-f ((pres admonish.v) he.pro)) but.cc
(h.he.pro just.adv-s ((pres will.aux-v) not listen.v)))
```

(g) “I constantly admonished him, but he just wouldn’t listen”

```
((i.pro constantly.adv-f ((past admonish.v) he.pro)) but.cc
(h.he.pro just.adv-s ((past will.aux-v) not listen.v)))
```

(h) “Pterodactyls could fly”

```
((k (plur pterodactyl.n)) ((past can.aux-v) fly.v))
```

(i) “The sea level could rise”

```
((the.d (sea.n level.n)) ((pres could.aux-s) rise.v))
```
To reiterate, modal auxiliary verbs (1) are in present or past tense, and (2) can have sentence-level and VP-level meanings. The sentence-level meanings most often express a possibility or expectation (at present or in the past), and the VP-level meanings typically express an ability, permission, or obligation (but some meanings of will/would and do/did deviate from this general pattern):

- **can** – (pres can. aux-v) if it means something like “presently able to”
  “This rocket can reach Mars”
  – (pres can. aux-s) if it simply refers to a possibility
  “This mission can fail”

- **could** – (pres could. aux-v) if it means something like “presently able to”
  “I could easily climb over that fence”
  – (pres could. aux-s) if it simply refers to a possibility
  “The sea level could rise”
  – (past can. aux-v) if it means roughly “able-to in the past”
  “Pterodactyls could fly”
  – (past can. aux-s) if it refers to a possibility from a past perspective
  “He was well aware that he could fail”

- **may** – (pres may. aux-v) if it means something like “presently permitted to”
  “You may sit down”
  – (pres may. aux-s) if it simply refers to a possibility
  “The prisoner may escape”

- **might** – (pres might. aux-s) if it simply refers to a possibility
  “He might faint”
  – (past may. aux-s) if it refers to a possibility from a past perspective
  “He knew that he might faint”

- **must** – (pres must. aux-s) for present certainty or necessity
  “A piece of the puzzle must be missing”; “What goes up must come down”;
  “He must return at once”; “He must be punished”
- (past must.aux-s) for a past certainty or necessity
  “He knew that the child must be nearby, and that he must keep searching”

will – (pres will.aux-s) or (pres futr) for an expectation, at present
  “The sun will rise”

- (pres will.aux-v) for present willfulness (esp. resistance)
  “No matter how much I cajole him, he just will not cooperate”

would – (pres would.aux-s) if it refers to a conditional possibility
  “I would go to Mars {if I were an astronaut, if I were to be offered the chance to go}”

- (past will.aux-s) or (past futr) if it refers to the future from a past perspective
  “He knew that he would not see her again”

- (past will.aux-v) for past willfulness (esp. resistance)
  “No matter how much I cajoled him, he just would not cooperate”

shall – (pres shall.aux-s) or (pres futr) for a firm expectation or suggestion (at present)
  “We shall overcome”, “Shall we go to the beach?”

should – (pres should.aux-s) for a firm expectation at present
  “He should arrive at any moment”

- (past should.aux-s) for a strong (but perhaps disconfirmed) expectation in the past
  “He knew that John should have arrived already”

- (pres should.aux-v) for a present obligation [* but see comment below]
  “John should study harder”

- (past should.aux-v) for a past obligation (perhaps violated) [* but see comment below]
  “John knew that he should {study, have studied} harder”

- (if).ps ... (pres should.aux-s) for a present conditional possibility
  “I’m ready for that, should it happen” (i.e., “if it should happen”)

- (if).ps ... (past should.aux-s) for a past conditional possibility
  “I was ready for that, should it happen” (i.e., “if it should happen”)

ought – (pres ought.aux-s) for a strong, present expectation
  “That ought to do the trick!”, “John ought to be awake by now”

- (past ought.aux-s) for a strong, past expectation
  “He went to the pharmacy, thinking that his prescription ought to be ready”

- (pres ought.aux-v) for a present obligation [* but see comment below]
  “He ought to study harder”

- (past ought.aux-v) for a past obligation [* but see comment below]
  “He knew that he ought to study harder”

do – (pres do.aux-v) for present emphasis
  “I do appreciate it”
(pres do.aux-s) in subject-auxiliary inversion and negation
“Do you have a car?”, “He did not speak”

did – (past do.aux-v) for past emphasis
“I did lock the door”

(past do.aux-s) in past tense subject-auxiliary inversion and negation
“Did you lock the door?”, “I did not lock the door”

(is – (pres be-destined.aux-v) for a presently scheduled or mandated action
“He is to appear in court tomorrow”

was – (past be-destined.aux-s) future in the past
“Babbage was never to succeed in building the Analytical Engine”

(past be-destined.aux-v) for an event scheduled/mandated event in the past
“He was to appear in court that day”

Progressive & passive uses of “be” are discussed elsewhere. Note that the above modal “is” and “was” senses, like other modal auxiliaries, have no untensed (be/being/been) forms; e.g., *“He has been to appear in court”.

We can also add some unusual items, marginally functioning as modal auxiliaries:

dare – (pres dare.aux-v) for present daring (often with “not”) 
“Dare he leave?”, “He daren’t leave”
(cf., “He doesn’t dare to leave”, where “dare” is a main verb)

dared – (past dare.aux-v) for past daring (often with “not”) 
“He dared not speak up”; **“He dared speak up”
(but, “He dared to speak up”, where “dare” is a main verb)

need – (pres need.aux-v) for presently needing or requiring
“Need you be so negative?”, “School dropouts need not apply”;
(In “You need to study”, “need” is a main verb)

had better – (pres had_better.aux-s) for a present requirement
“Someone had better warn him”
(Note the scope ambiguity – that’s why we want aux-s, not aux-v)

(past had_better.aux-s) for a past requirement
“I realized that someone had better warn John”

Also, “used to”, and “have to” are sometimes regarded as modal auxiliaries. However, we treat “used to” as an aspectual phrase below. [But from the web, “used to” can be broken up: “Rossetti once told me that Hannay, when he first knew him, used to be so hard up that he used never to be at home in the daytime”; thus we have “used never to ...”] Further, we regard “have” in “have to” as a regular main verb, since it can be tenseless and unlike genuine modal auxiliaries can be preceded by another modal auxiliary, as in “I will have to think about it”.

[* Note: the suggested aux-v forms of “should” and “ought” are questionable, because with a VP-level modification like
((some.d person.n) (should.aux-v (help.v |John|))), or
((some.d person.n) (ought.aux-v (help.v |John|))),
i.e., “Someone {should, ought to} help John”, we can’t explain the ambiguity of such a sentence – we capture the meaning “There is someone who should help John”, but not the meaning “It ought to happen that someone helps John” (sort of a “socially distributed” obligation). But if we use aux-s instead of aux-v for these obligation-implying senses of “should” and “ought”, how do we distinguish them from those that just suggest an expectation? Maybe we need an additional ((pres,past) ought-to-happen.aux-s) operator? Or maybe all uses of should/ought are sentence-level, but when they express an obligation and the subject is definite, they strongly suggest that the obligation falls on that subject. This remains an open question.

4.9 Aspect Annotation (Extension Over Time)
Aspect is generally captured by by the perfect “have” and progressive “be” auxiliaries in our annotations, as well as by the lexical entries in our annotations (e.g., daily, used to) and less frequently as multi-word modifier phrases (e.g., every day, for an hour). These are all treated in the same manner as other modifiers and we refer to that section for annotation details. Perfect and progressive tense are marked morpho-syntactically in English and require special treatment, which we describe next.

4.9.1 Special Cases – Perfect and Progressive
Perfect and progressive aspects are annotated with operators perf and prog, respectively. They are sentence-level operators, but (as with modifiers in general) the annotation will keep them in surface order and the operators will be lifted in post (similar to auxiliaries and negation).

(a) “He has left Rome”
   (he.pro ((pres perf) (leave.v |Rome|)))
(b) “He is sleeping”
   (he.pro ((pres prog) sleep.v))
(c) “He has been sleeping”
   (he.pro ((pres perf) (prog sleep.v)))
(d) “He will be sleeping”
   (he.pro ((pres will.aux-s) (prog sleep.v)))
(e) “He may have been sleeping”
   (he.pro ((pres may.aux-s) (perf (prog sleep.v))))
(f) “She had been given an award”
   (she.pro ((past perf) ((pasv give.v) (an.d award.n))))
(g) “She was being honored”
   (she.pro ((past prog) (pasv honor.v)))
Notice that we drop the copula that accompanies the progressive aspect, since its semantic signal is captured by \textit{prog} and the tense operator. The last example is interesting because there are two copulas introduced – one to capture the tense and another to capture the progressive aspect.

Now that we’ve seen all three forms of ‘be’ (main verb, in a progressive, in a passive construction) see the sentence below that includes all three.

\textit{(h) “John is under arrest and is being questioned”}

\[
|John| (\text{(pres be.v) (under.p (k arrest.n))) and.cc ((pres prog) (pasv question.v)))
\]

### 4.10 Modifiers (again)

Modifiers are operators that map predicates to predicates or sentences to sentences. They correspond closely with the syntactic class of adverbs. See the examples below to get a feel for what constitutes a modifier:

(a) “Jim is very happy”

\[
|Jim| (\text{(pres be.v) (very.adv-a happy.a))})
\]

(b) “John saw Mary yesterday”

\[
|John| (\text{((past see.v) |Mary|) yesterday.adv-e)}
\]

(c) “Mary undoubtedly spoke up”

\[
|Marry| (\text{undoubtedly.adv-s (past speak_up.v)}), or equivalently,

|Marry| undoubtedly.adv-s (past speak_up.v))
\]

(d) “John sees Mary regularly”

\[
|John| (\text{(pres see.v) |Mary|) regularly.adv-f})
\]

We distinguish between four different types of modifiers: \textit{action/attribute} modifiers, \textit{event} modifiers, \textit{sentence} modifiers, and \textit{frequency} modifiers. These correspond to the suffix tags -a, -e, -s, and -f, respectively. \textit{Action/attribute} modifiers map the predicate to a new predicate (e.g., \textit{ran quickly}, \textit{very smart}). \textit{Event} modifiers add some information about the event described by the sentence (e.g. \textit{in the forest, along the river, at noon}). \textit{Sentence} modifiers comment on the sentence, but do not modify its meaning, such as writer commentary (e.g., \textit{surprisingly}). \textit{Frequency} modifiers specify repetitive occurrence of the type of event described by the sentence they modify (e.g. \textit{daily, regularly, every week}). Below are some additional examples of the different types.

(e) “John politely greeted Mary”

\[
|John| (\text{politely.adv-a ((past greet.v) |Mary|)})
\]

(f) “Mary spoke up confidently”

\[
|Marry| (\text{(past speak_up.v) confidently.adv-a})
\]

(g) “The next meeting will probably be canceled”

\[
((\text{the.d (next.a meeting.n)}) (\text{pres will.aux) probably.adv-s (pasv cancel.v)})
\]
(h) “John saw Mary yesterday”

```
(John (past see.v) (Mary) yesterday.adv-e))
```

(i) “The meeting took place here”

```
(The.d meeting.n) (past take_place.v) here.adv-e))
```

(j) “The meeting went on interminably”

```
(The.d meeting.n) (past go_on.v) interminably.adv-e))
```

(k) “John saw Mary twice”

```
(John (past see.v) (Mary) twice.adv-f))
```

(l) “John usually wakes up early”

```
(John (usually.adv-f (pres wake_up.v) early.adv-e)))
```

### 4.10.1 Predicates as Modifiers

When modifying non-verbal predicates with other predicates (e.g. (burning.a hot.a)) we omit the type-shifting operators from predicates to predicate modifiers for annotator simplicity. Adding these operators is a completely deterministic process and will be handled in post. If you are curious, please refer to Appendix A.1 for a complete description of the type-shifting operators and the method of inserting them from the present form. Below is an example with multiple predicate modifications.

“I spilled a burning hot melting pot”

```
1. (I (spilled (a ((burning hot) (melting pot))))))
2. (prp I) (vbd spilled) (dt a) (vbg burning) (jj hot) (nn melting) (nn pot)
3. (I.prp (spilled.vbd (a.dt ((burning.jj hot.jj) (melting.nn pot.nn))))))
4. (I.pro ((past spill.v) (a.d ((burning.a hot.a) (melting.n pot.n))))))
```

Also notice that the automatic POS tag for ‘burning’ was incorrect and needed correction. In this example ‘burning’ is acting as an adjective (similar to ‘red hot’) rather than a verb.

### 4.10.2 Gaps, and topicalization

Sometimes constituents are “moved” from their normal syntactic position, leaving “gaps” (also referred to as “holes” or “traces”) in those positions. For example, “Those dogs, I’m afraid of” can be viewed as a rearrangement of “I’m afraid of those dogs”, but with the final noun phrase placed at the front of the sentence to make it salient, leaving a gap after of. Constituent fronting of this type is called **topicalization**.

In forming ULFs for such sentences, we might instead put the topicalized constituent in its “normal” place, but it turns out to be better to use a different approach, for preserving pragmatic information and for uniform treatment of related phenomena (especially certain types of relative clauses and wh-questions, as will be seen). This uniform approach employs a macro, `sub`, and a special “hole variable”, `*h`. Here are some examples:
(a) “Those dogs, I’m afraid of”
   (sub (those.d (plur dog.n)) (I.pro ((pres be.v) (afraid_of.a *h))))

(b) “About that, I know nothing”
   (sub (about.p-arg that.pro) (I.pro (((pres know.v) nothing.pro) *h))))

(c) “Fight our fears, we must”
   (sub (fight.v (our.d (k (plur fear.n)))) (we.pro ((pres must.aux-v) *h))))

(d) “Swiftly, the fox ran away”
   (sub swiftly.adv-a ((the.d fox.n) (((past run.v) away.adv-a) *h))))

Note that various types of phrases can be topicalized, including noun phrases, prepositional phrases, verb phrases, and adverbial phrases. The `sub` macro expects two arguments, namely the ULF of the dislocated (“filler”) phrase, and the ULF of an arbitrarily complex clause with a hole variable *h* somewhere in it, usually at or near the end of the clause. In postprocessing, lambda-abstraction of the hole-variable may be used to make the semantic connection between the filler and the hole explicit.

Occasionally (mostly in old or poetic English) topicalization occurs at the verb phrase level rather than the sentence level, as in

(e) “Let me not to the marriage of true minds admit impediments”
   (you).pro (((pres let.v) me.pro) not.adv-s
   (sub (to.p-arg (the.d (marriage-of.n (k (true.a (plur mind.n))))))
   (admit.v (k (plur impediment.n)) *h))))

(from a Shakespearean sonnet). In fact, the next line of the sonnet contains another such instance:

(f) “Love is not love which alters when it alteration finds”
   (k (pres be.v) not.adv-s
   (k (n*preds love.n
   (which.rel ((pres alter.v)
   (when.ps (it.pro (sub (k alteration.n) ((pres find.v) *h))))))))))

Here the direct object, `alteration`, is moved to the front of the verb phrase. An interesting ambiguity in this example is that the relative clause “which alters when ...” might not modify the second occurrence of `love` as assumed in the above ULF, but rather the first, i.e., “Love which alters when ... is not love”; in other words, the relative clause may have been right-shifted over “is not love”, much as if the wording had been, “Love is not love if it alters ...”. In such a case, we would have left the relative clause “free-floating” in the main verb phrase, much as if it were an adverbial headed by `if`.

4.10.3 Modifying phrases (adverbials)

EL provides the operators, adv-a, adv-e, adv-s, and adv-f to construct complex adverbials from predicates (typically, derived from prepositional phrases). The operator names correspond to the suffixes of lexical adverbs: .adv-a, .adv-e, .adv-s, and .adv-f.
(a) “I read with my glasses”
   (i.pro ((pres read.v) (adv-a (with.p (my.d glasses.n))))))
(b) “Wycliffe played the piece with great passion”
   ([Wycliffe] (((past play.v) (the.d piece.n))
   (adv-a (with.p (great.a passion.n))))))
(c) “I like to read in the park”
   (i.pro ((pres like.v) (to (read.v (adv-e (in.p (the.d park.n)))))))
(d) “Without a doubt, John was at school today”
   ((adv-s (without.p (a.d doubt.n)))
   ([John] (((past be.v) (at.p (k school.n))) today.adv-e))))
(e) “Most likely, John went to the store”
   ((adv-s (most.adv-a likely.a))
   ([John] ((past go.v) (to.p-arg (the.d store.n))))))
(f) “Eve eavesdrops on her friends, usually intentionally or knowingly”
   ([Eve] (((pres eavesdrop.v) (on.p-arg (her.d (plur friend.n))))
   (usually.adv-f (intentionally.adv-a or.cc knowingly.adv-a))))
(g) “Suddenly {,} she left”
   (Suddenly.adv-e (she.pro (past leave)))
(h) “Sullenly {,} she left”
   (sub sullenly.adv-a (she.pro ((past leave.v) *h))))

Notice the contrast between the last two examples. In the last sentence, the initial manner adverb is treated as topicalized, to facilitate its “lowering” to the VP level in postprocessing.

4.10.4 Clausal and verb-phrase adverbials

The examples of adverbials above do not involve verbs, but many adverbials are formed from clauses or verb phrases. Semantically, clausal adverbials are modifiers at the sentence level, though like some of the adverbials in the previous subsection, they can appear in sentence-premodifying, postmodifying, or internal positions:

(a) “Although the sun shone, the air was chilly”
   ({{Although.ps ((the.d sun.n) (past shine.v)))
   ((the.d air.n) ((past be.v) chilly.a))})
(b) “The air was chilly, even though the sun shone”
   (((the.d air.n) ((past be.v) chilly.a))
   (even_though.ps ((the.d sun.n) (past shine.v))))
(c) “The icy wind, though the sun shone, chilled him to the bone”

(((the.d (icy.a wind.n)) (though.ps ((the.d sun.n) (past shine.v))))

(((past chill.v) he.pro) (adv-a (to.p (the.d bone.n)))))

(d) “Mangoes are delicious when they are ripe”

(((k (plur mango)) ((pres be.v) delicious.a

(when.ps (they.pro ((pres be.v) (ripe.a)))))

(e) “Mangoes are delicious when they are from India”

(((k (plur mango)) ((pres be.v) delicious.a

(when-if.ps (they.pro ((pres be.v) (from.p |India|))))))

We distinguish temporal and atemporal when in the last two examples, coding the atemporal version as when-if.ps in view of its semantic similarity to if.

Verb phrase adverbials use the -ing (participial) form or to-infinitive form of the verb, and may be VP-modifiers (adv-a adverbials), episode-modifiers (like adv-e adverbials), or proposition-modifiers (like adv-s adverbials):

(f) “The quarterback walked away, limping noticeably”

(((the.d quarterback.n) (((past walk.v) away.adv-a) (adv-a (prog (limp.v noticeably.adv-a)))))

(g) “Limping noticeably, the quarterback walked away”

(sub (adv-a (prog (limp.v noticeably.adv-a)))

(((the.d quarterback.n) (((past walk.v) away.adv-a) *h))))

(h) “The quarterback stumbled, while backing up”

(((the.d quarterback.n) (((past stumble.v) (while.ps ({he}.pro ((past prog) back_up.v))))

(i) “Considering her inexperience, she did very well”

((adv-s (consider.v (her.d inexperience.n)))

(she.pro ((past do.v) (adv-a (very.adv-a good.a))))

(j) “The outfielder ran to catch the ball”

(((the.d outfielder.n) (((past run.v) (adv-a ((for).p (to (catch.v (the.d ball.n))))))))

(k) “To put it bluntly, he failed”

(((adv-s ((for).p (to (put.v it.pro bluntly.adv-a))))

(he.pro (past fail.v))))

In (i), we have treated “considering her inexperience” much as we would have treated “despite her inexperience” or “in light of her inexperience”, though some might want to expand the sentence meaning to something like “if I consider her inexperience, I conclude that she did very well”. But any such expansions should be deferred to postprocessing. Note also that we have rendered very well as the adv-a transform of very good, since very.adv-a needs to operate on a predicate, not on an adverb.
In (j), we have represented the purpose adverbial “to catch the ball” by introducing a covert preposition \( for \) that can take the action type \( (to \ (catch.v \ (the.d \ ball.n))) \) as its complement. (We can read the covert \( for \) as \textit{for-purpose}.) In (k), the adverbial comments on the sentence as a whole, hence the \texttt{adv-s} operator; but again the \textit{to}-infinitive is used to express a purpose.

### 4.10.5 Phrasal post-nominal modifiers

For post-nominal modifiers, we introduce macros \texttt{n+preds} and \texttt{np+preds} to simplify the annotation. Since post-nominal modifiers can only add to the meaning of the noun, these macros map to a lambda expression with a conjunction of the listed properties. The syntactic forms of these macros are:

\[
\begin{align*}
\texttt{n+preds} & \ [\text{noun (incl. any arguments)}] \ [\text{predicate} \ 1] \ [\text{predicate} \ 2] \ \ldots \\
\texttt{np+preds} & \ [\text{noun phrase}] \ [\text{predicate} \ 1] \ [\text{predicate} \ 2] \ \ldots
\end{align*}
\]

See examples below.

(a) “A table with three legs”
\[
\texttt{(a.d \ n+preds \ table.n \ (with.p \ ((nquan \ three.a) \ (plur \ leg.n))))}
\]

(b) “The explosion in the city”
\[
\texttt{(the.d \ n+preds \ explosion.n \ (in.p \ (the.d \ city.n)))}
\]

(c) “The hawk circling overhead”
\[
\texttt{(the.d \ n+pred hawk.n \ (circle.v \ (adv-a \ overhead.a)))}
\]

(d) “John, totally exhausted, ...”
\[
\texttt{(n+preds \ |John| \ (totally.adv-a \ exhausted.a))}
\]

(e) “John, feeling tired, ...”
\[
\texttt{(np+preds \ |John| \ (feel.v \ tired.a))}
\]

Note that we use \texttt{n+preds} for restrictive postmodifiers – ones that further limit what entities the noun phrase as a whole can refer to; while we use \texttt{np+preds} for nonrestrictive postmodifiers, i.e., ones that just add supplementary information about the entity, which is already identified by the NP without the postmodifier(s). In English, nonrestrictive postmodifiers are usually separated from the NP they supplement by a comma.

### 4.10.6 Relative clauses

Relative clauses also postmodify nouns or noun phrases, but they often involve gaps, and thus make use of the \texttt{sub} macro and hole variable \( *h \) introduced earlier. In the following examples, only (a) does not require these devices, because the relative pronoun in that example is in subject position, and as such “already in the right place”.

(a) “John, who is a lawyer, ...”
\[
\texttt{(np+preds \ |John| \ (who.rel \ ((pres \ be.v) \ lawyer.n)))}
\]

(b) “The manager whom you met”
\[
\texttt{(The.d \ n+preds \ manager.n \ (sub \ who.rel \ (you.pro \ ((past \ meet.v) \ *h))))}
\]
(c) “The car that you bought”
   (The.d (n+preds car.n (sub that.rel (you.pro ((past buy.v) *h))))))

(d) “The car you bought”
   (The.d (n+preds car.n (sub tht.rel (you.pro ((past buy.v) *h))))))
   (tht.rel, that.rel are synonyms)

(e) “A man not of this world”
   (A.d (n+preds man.n (sub tht.rel (not (*h (of.p (this.d world.n)))))))
   or, using non- to negate a predicate,
   (A.d (n+preds man.n (non- (of.p (this.d world.n))))))

(f) “The manager, whom you met”
   (np+preds (The.d manager.n) (sub who.rel (you.pro ((past meet.v) *h))))

(g) “The woman at the door whose brother you met”
   (The.d (n+preds woman.n (at.p (the.d door.n)))
   (sub (the.d ((poss-by who.rel) (brother-of.n *s)))
   (you.pro (past meet.v) *h))))

(h) “The manager, whose house we passed”
   (np+preds (The.d manager.n)
   (sub (the.d ((poss-by who.rel) house.n)) (we.pro ((past pass.v) *h))))

(i) “The dog on the beach, whose owner you know”
   (np+preds (The.d (n+preds dog.n (on.p (the.d beach.n))))
   (sub (the.d ((poss-by which.rel) (owner-of.n *s)))
   (you.pro ((pres know.v) *h))))

(j) “The White House, which was designed by James Hoban”
   (np+preds (The.d White House|n
   (which.rel ((past (pasv design.v)) (by.p-arg [James Hoban|))))

(k) “The street where you live”
   (The.d (n+preds street.n
   (sub (at-loc.p which.rel) (you.pro ((pres live.v) (adv-e *h)))))))

(l) “The time when dinosaurs roamed [on] the Earth”
   (The.d (n+preds time.n
   (sub (at-time.p which.rel) ((k (plur dinosaur.n))
   ((past roam.v) (on.p (the.d [Earth|})) (adv-e *h))))))

(m) “The couch wheron he reclined”
   (The.d (n+preds couch.n
   (sub (on-loc.p which.rel) (he.pro ((past recline.v) (adv-e *h)))))))
Note that for relative clauses, argument (1) of the sub macro must include a relative pronoun – who.rel, that.rel, that.rel, or which.rel. Note further that in (j) we have rendered whose owner as “the owner of which.rel” rather than “the owner of who.rel”, because the relative determiner whose, unlike the relative pronoun who, does not imply reference to a person – it may be a person or any other type of thing, and here it is a set of dogs. (We could have used that.rel instead of which.rel to maintain the person/non-person ambiguity of whose, but in this case the referent is clearly non-human.)

Like whose, the relative prepositions when, where, whereon, etc., need to be decomposed in the ULF to expose a relative pronoun (here, which.rel), since in postprocessing we need to have a pronoun that is co-referential with the entity whose type is specified by the main noun of the noun phrase. For example, in the street where you live, where needs to be expanded so that it contains a relative pronoun that refers to the street being described. Hence we render where as (at-loc.p which.rel), where which.rel refers to the particular street. In post-processing, the argument of street.n would become a variable, and that same variable would replace which.rel. We may in future leave the decomposition of relative determiner whose and of the “relative sentential prepositions” when, where, whereon, etc., to post-processing as well, writing them in ULF as whose.dr, when.pr, where.pr, whereon.pr, etc.

4.10.7 It-clefts, extraposition, and there-sentences

TODO
- “It was Mary who arrived first”
- “It was Mary he gave the book to”
- “It’s surprising that Mary arrived first”
- “He saw to it that Mary would get the book”
- “There is a tavern in the town” (existential)
- “(Over) there is the tavern I like” (adverbial; same as “Away ran the wolf”, later??!)
- “There but for the grace of God is where I would be”

4.10.8 Yes-no questions

Syntactically, the simplest questions are declarative questions, which only require addition of a question mark:

(a) “Bob has left?”

((Bob ((pres perf) leave.v)) ?)

(b) “Bob has left, hasn’t he?”

((Bob ((pres perf) leave.v)) .?)

(c) “Bob hasn’t left yet, has he?”

((Bob ((pres perf) not.adv-s (leave.v yet.adv-e))) .?)

It’s worth mentioning that despite the superficial similarity of yes-no questions (especially declarative ones) to declarative statements, they are of different semantic types. Roughly speaking, the meaning of a question is taken to be its true answer(s) in each possible world. For example, the question in (a) denotes the fact that Bob has left (if in actuality he has),
or that he has not left (if in actuality he has not); similarly in non-actual possible worlds. You can take the question mark as signalling this distinct semantics.

In the “tag questions”, (b) and (c), we don’t code the tag explicitly, but use ‘?’ as shown, indicating that the speaker/writer presumes truth, but asks anyway. The period only has pragmatic significance – the semantics of ‘?’ is the same as for ‘?’.

More commonly, yes-no questions involve subject-auxiliary inversion:

(e) “Has Bob left?”

(((pres perf) |Bob| leave.v) ?)

(f) “Is Bob well-liked by his co-workers?”

(((pres be.v) |Bob| (well-liked.a (by.p-arg (his.d (plur co-worker.n)))))) ?

(g) “Isn’t Bob going to leave?”

(((pres be.v) not.adv-s |Bob| (going_to.v leave.v)) ?)

(h) “Didn’t she eventually hire Bob?”

(((past do.aux-s) not.adv-s she.pro eventually.adv-e (hire.v |Bob|)) ?)

(i) “Will he have finished the assignment by that time?”

(((pres will.aux-s) he.pro (perf (finish.v (the.d assignment.n) (by.p (that.d time.n)))))) ?

Note that “subject-auxiliary inversion” is a somewhat inaccurate term; for example, in (e) and (f) the main copular verb is fronted. In British English have is sometimes fronted: “Have you a pencil?”; and in old or poetic English other main verbs may be fronted: “Hear ye not?” For pragmatic reasons we retain the subject-auxiliary inversion in the ULF, even though post-processing will probably rearrange constituents into declarative-question-like form. For example, the ULF for (f) may be rearranged into

(((pres be.v) not.adv-s |Bob| (going_to.v leave.v)) ?).

We should note that some other subject-verb inversion are seen occasionally, in particular in sentences beginning with an adverbial, and some imperatives:

“Under the tree sat Bob”

“Under the ice have been found new deep-sea creatures”

“Away ran the wolf”

“Merrily did we drop, below the kirk, below the hill, ...” (Coleridge)

“Get thee to a nunnery” (Shakespeare’s Hamlet)

“Take you a course, get you a place, ...” (John Donne)

It appears that the locative inversions in the first three examples involve interchange of the entire intransitive verb phrase, not just the tensed verb, with the subject. Subject-object ambiguities may result if we retain such an inversion in the ULF, so the inversion should probably be undone; whereas the initial adverbial should probably be treated as topicalized. Under these assumptions, the ULF for the second example would be
The fourth example also involves topicalization and subject-auxiliary inversion, very much as in a question like “How did we drop?”. In the imperative examples we would likewise retain the surface ordering.

4.10.9 Wh-questions (constituent questions)

The simplest kinds of wh-questions have the same form as declarative sentences, either because the constituent being questioned is the subject, or because the embedded wh-constituent is left in place, rather than being fronted:

(a) “Who arrived?”
   (Who.pro (past arrive.v))
(b) “You did what?”
   ((You.pro ((past do.v) what.pro)) ?)

But again, the most common forms of wh-questions involve subject-auxiliary inversion, and additionally the wh-constituent is fronted, leaving a gap. So in essence, such sentences consist of a wh-constituent preceding an “inverted sentence” of the same form as a yes-no question (but containing a gap):

(c) “Whom did you invite?”
   ((sub who.pro ((past do.aux-s) you.pro (invite.v *h))) ?)
(d) “Why did you fix it?”
   ((sub Why.adv-s ((past do.aux-s) you.pro (fix.v it.pro *h))) ?)
(e) “With what did you fix it?”
   ((sub (adv-a (with.p what.pro))
      ((past do.aux-s) you.pro (fix.v it.pro *h))) ?)
(f) “On which topic have you decided to focus?”
   ((sub (on.p-arg (which.d topic.n))
      ((pres perf) you.pro (decide.v (to (focus.v *h)))))) ?)
(g) “What topic have you decided to focus on?”
   ((sub (what.d topic.n)
      ((pres perf) you.pro (decide.v (to (focus.v (on.p-arg *h))))))) ?)
(h) “How smart is he?”
   ((sub (How.adv-a smart.a) ((pres be.v) he.pro *h)) ?)
(i) “How quickly can you say ‘desserts’ backward?”
   ((sub (adv-a (how.adv-a quick.a))
      ((pres can.aux-v) you.pro (say.v "desserts" (adv-a backward.a) *h))) ?)
(j) “Which sandwiches were ordered by which guests?”
   
   \[
   \text{(sub (\text{which}.d (\text{plur sandwich}.n)))}
   \]
   
   \[
   \text{((past (\text{pasv order}.v))) \ast h (by.p-arg (\text{which}.d (\text{plur guest}.n))))} ?
   \]

As in the case of yes-no questions, there are occasional examples in old and poetic English of main-verb inversions (for main verbs other than \text{be}):

“Dear heart, how like you this?” (Sir Thomas Wyatt)

“Why wayle we then?” (Edmund Spenser)

“Why brook’st thou, ignorant horse, subjection?” (John Donne)

“Why bows the side-box from its inmost rows?” (Alexander Pope)

We would still form ULFs as in the case of auxiliaries, i.e., the embedded inverted sentential ULF starts with a tensed verb, followed immediately by the subject noun phrase, followed by any verb complements or adjuncts.

4.11 Names

Names must distinguish between \textit{true} names and \textit{predicate} names. True names are those that can be used without a preceding determiner, while predicate names requires a preceding determiner. Notice that we need \textit{the} Delaware River for (b) to be grammatical.

(a) Mary is beautiful [\textit{good}!]

(b) Delaware River is beautiful [\textit{bad}!]

True names are annotated with surrounding pipes \texttt{| |}. Spaces and capitalization are preserved in the pipes. For readers familiar with Lisp, this corresponds to Lisp’s escape symbols. Below are some examples of true name annotation:

- Mary \rightarrow \texttt{|Mary|}
- John \rightarrow \texttt{|John|}
- Three Mile Island \rightarrow \texttt{|Three Mile Island|}
- The Hague \rightarrow \texttt{|The Hague|}
- New York \rightarrow \texttt{|New York|}

Predicate names are annotated with surrounding pipes and followed by the noun suffix \texttt{|n}. Below are some examples of this annotation:

- Delaware River \rightarrow \texttt{|Delaware River|n}
- Eiffel Tower \rightarrow \texttt{|Eiffel Tower|n}

The semantic information from the name would be extracted with a separate module, since it requires extensive interaction with the surface form. For example, “Three Mile Island” being an island. For cases such as “his name is John” or “love is a four-letter word” where the string is referred to as the word itself rather than what it means, the quotes are elided so we annotate them as we would object quotes (see Section \textsection 4.14.4).

\[
\text{(his}.d\text{name}.n) ((\text{pres be}.v) (= "John"))
\]

\[
("love" ((\text{pres be}.v) ((\text{four-}.a letter}.n) word}.n))
\]
4.12 Possessives

Possessives are semantically handled with the binary predicate poss-by. See basic examples below and further discussion following:

(a) “The kindergarten’s boisterous children”
\[((\text{the.d kindergarten.n}) \ 's) (\text{boisterous.a (plur child.n))})\]
\[((\text{the.d (poss-by (the.d kindergarten.n)) (boisterous.a (plur child.n))}))\]

(b) “My dogs are happy”
\[((\text{My.d (plur dog.n)) ((pres be.v) happy.a))\]
\[((\text{the.d (poss-by Me.pro) (plur dog.n)) ((pres be.v) happy.a))\]

(c) “The dogs are mine”
\[((\text{The.d (plur dog.n)) ((pres be.v) mine.a))\]
\[((\text{The.d (plur dog.n)) ((pres be.v) (poss-by me.pro))}\]

(d) “That is John’s dog”
\>((\text{That.pro ((pres be.v) (= ((|John| \ 's) dog.n))))\]
\>((\text{That.pro ((pres be.v) (= (\text{the.d ((poss-by |John|) dog.n)})}))\]

(a) is the bare possessive phrase and (b) uses possessive phrase as an argument. (c) shows a usage where poss-by is separated from the rest of the phrase. (d) shows a predicative use of the possessive where we wrap the possessive NP with (= ..) to turn it into a predicate.

4.12.1 Relational Predicates in Possession

Relational predicates (e.g. sister of, child of, etc.) are handled by creating relational predicates P-of. This naming convention was chosen because the postnominal genitive (e.g. “the father of John”) strongly prefers a relational interpretation. *s and *ref are anaphoric variables used to mark the participant of the relation: *s for internal relations where the possessive represents a relation involving the possessor, and *ref for external relations where the noun is relational to some external entity.1 Below are examples of annotations for these cases.

Possessives with Internal Relations

(e) “John’s boisterous children”
\>(((|John| \ 's) (\text{boisterous.a (plur (child-of.n *s))}))\]
\>((\text{the.d ((poss-by |John|) (boisterous.a (plur (child-of.n *s))}))}\]

(f) “My children are happy”
\>((\text{My.d (plur (child-of.n *s)) ((pres be.v) happy.a))\]
\>((\text{the.d ((poss-by Me.pro) (plur (child-of.n *s)) ((pres be.v) happy.a))}\]

1The internal relations seem to parallel the linguistic notion of inalienable possession, i.e., there is necessarily a possessor, though English does not grammatically mark this variant of possession as some languages do.
(g) “The children are mine”

(((The.d (plur (child-of.n *s))) ((pres be.v) mine.a))
  (((The.d (plur (child-of.n *s))) ((pres be.v) (poss-by me.pro)))))

(h) “That is John’s child”

((That.pro ((pres be.v) (= ((|John| ’s) (child-of.n *s)))))
  ((That.pro ((pres be.v) (= (the.d ((poss-by |John|) (child-of.n *s)))))))

(i) “The boisterous children of John”

((The.d (boisterous.a (plur (child-of.n |John|))))

(e-h) show examples of internal relations in a possessive phrase and correspond respectively to (a-d) basic cases. Notice that the child-of-n relation uses the *s anaphoric variable in this case, but not in (a). This allows unambiguous location of the argument from the rest of the LF for (e). In (a) “the kindergarten” does not partake in the ‘child-of’ relation. (i) shows a postnominal genitive, aka of-possession, which strongly correlates with a relational predicate interpretation (e.g. “the dog of John” and “the children of the kindergarten” are not acceptable to most speakers or do not preserve the ‘’s interpretation).

Possessives with External Relations

(j) “My side is winning”

(((My.d (side-of.n *ref)) ((pres prog) win.v))
  (((the.det ((poss-by Me.pro) (side-of.n *ref))) ((pres prog) win.v)))

(k) “John’s mothers keep wandering away”

(((|John| ’s) (plur (mother-of.n *ref)))
  ((pres keep.v) (ka (wander.v away.adv-a))))
  (((the.d ((poss-by |John|) (plur (mother-of.n *ref)))))
  ((pres keep.v) (ka (wander.v away.adv-a))))

(l) “This is her side”

((This.pro ((pres be.v) (= (her.d (side-of.n *ref))))))
  ((This.pro ((pres be.v) (= (the.d ((poss-by she.pro) (side-of.n *ref)))))))

(j) shows an example where the sentence context pushes the predicate meaning away from the internal relation reading. (k) and (l) are ambiguous between an internal and external reading. ‘mothers’ in (k) could refer to John’s mother and step-mother (internal), but just as likely a group of mothers he’s in charge of guiding (external). Similarly, ‘side’ could refer to ‘side of her’, which would be an internal relation reading, but just as likely her side of some partitioned area, or opposing players in a game, etc.

The lines between relational and non-relational nouns, and internally relational and externally relational possession are fuzzy – dependent on both grammatical signals and semantic concepts. These criteria are designed to be relatively simple to follow and to be conservative in our designation of relational possession.

Criteria for Relational Predicates

1. The noun must have two participants to be satisfied, or even possible to interpret. For example, a father without a child is not a father, nor can a side exist without being
the side of something. Relational nouns often have sortal alternatives denoting the same entity without the relation (father – man, birthday – day), though this is not always true, e.g. side, weight, pinnacle, etc. These exceptions tend to be functional nouns, which describe an entity’s intrinsic property as opposed to a relation between two distinct entities. Under this definition body parts are not relational since they can exist independently of a person without losing the noun meaning (e.g. a hand grown in a test tube would still be a hand).

2. The noun can be used in post-nominal genitive construction (e.g. father of John) and preserve the original relation meaning. **Beware**, the post-nominal genitive is not the same as the double genitive (father of John ≠ father of John’s). Only the post-nominal genitive strongly prefers relational nouns (e.g. “This is a book of Bob’s” is okay, but not *“This is a book of Bob”*).

**Criterion for Internally Relational Possession**

1. Satisfies the criteria for relational predicates in possessives.

2. The interpretation where the possessor participates in the relation is heavily favored, in the context of the given sentence. For example, in the sentence “This is her child” (without further context), the interpretation “child-of her(s)” in the offspring (or legal parent) sense, is heavily favored. So in this case you would use (child-of.n *s), even though other interpretations are possible in certain contexts (e.g., as “the child the nanny is caring for”). By contrast, “This is her side” (without further context) does not heavily favor a reading as “side of her(s)”, i.e., a side of her body; it could be her side of some partitioned area, or opposing players in a game, or of a debatable issue, etc., so you would use (side-of.n *ref). On the other hand, if the sentence was “She was reclining on her side”, the interpretation “side of her(s)” in the bodily sense is heavily favored, so you would use (side-of.n *s) in this case.

If the criterion for internally related possession is not satisfied, but the criteria for relational predicates in possessives is, then it is assumed to be an externally related possession and annotated with *ref.

For reference, here are two short lists of words whose most common senses are relational and not relational (by criterion 1), respectively.

**Commonly Relational:** mother, father, daughter, son, uncle, (other kinship terms), birthday, pet, enemy, sake, side, top, bottom, edge, pinnacle, (other views or areas of objects), temperature, weight, (other functional properties)

**Commonly Not Relational:** hand, hair, leg, (other body parts), dog, table, wheel, door, etc.

### 4.12.2 Relational Nouns Outside of Possessives

Relational nouns can be used outside of possessive contexts, and we still want to annotate them as relational. The Criteria for Relational Predicates described for possessives holds outside of possessives as well. Below are a few clarifying examples:

(m) “We reached the pinnacle”

(We.pro ((past reach.v) (the.d (pinnacle-of.n *ref)))))
(n) “I started at the foot of the mountain”
(((I.pro (past start.v)) (adv-e (at.p (the.d (foot-of.n (the.d mountain.n))))))

(o) “Legs ache when they are strained”
(((k (plur leg.n)) (pres ache.v)) (adv-e (when.ps (they.pro (pres (pasv strain.v))))))

(p) “I was surprised by the weight”
((I.pro (((past be.v) surprised.a) (by.p-arg (the.d (weight-of.n *ref)))))

(q) “I met some mothers”
((I.pro ((past meet.v) (some.d (plur (mother-of.n *ref))))))

(r) “I met some sisters”
((I.pro ((past meet.v) (some.d (plur sister.n))))

The difference between examples (q) and (r) is likely most surprising. This arises from
the fact that “mother” does not have an alternate, non-relational sense that can be used in
the context whereas “sister” may mean a nun, which is not relational.

4.12.3 Role Nouns and Other Context-Dependent Relational Nouns

There are a number of nouns that have both a relational sense and a non-relational sense,
so the criteria must be checked every time to verify that they are satisfied. An example
below:

(s) “This is my residence” – (my.d (residence-of *s)), relational
“This is a nice residence” – (nice.a residence.n), not (necessarily) relational

A common and tricky class of these nouns is role nouns, e.g. tutor, pilot, bouncer,
mascot, pet, etc. These job-like terms are non-relational in general use, rather denoting an
agent that habitually and/or professionally holds a particular relation with various entities.
These are annotated with simple predicates, except where it is explicitly relational – “She’s
my tutor”. Some examples:

(t) “I saw a mascot today” – (a.d mascot.n)
“I saw my school’s mascot today” – ((my.d school.n) (mascot-of *s))

(u) “Johnny wants to become a captain” – (a.d captain.n)
“I met the captain of the USS Alabama” – (captain-of.n (the.d |USS Alabama|.n))

4.12.4 Verbal Possession

Verbal possession (e.g. “John has a dog”) is annotated with have.v. The criteria for anno-
tating relative predicates and internal/external variants are the same here. Examples:

(v) “John has a dog”
(|John| ((pres have.v) (a.d dog.n)))

(w) “Mary has an assistant”
(|Mary| ((pres have.v) (an.d (assistant-of.n *s))))
<table>
<thead>
<tr>
<th>Personal Pronoun</th>
<th>Possessive Determiner</th>
<th>Possessive Pronoun</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>my</td>
<td>mine</td>
</tr>
<tr>
<td>you</td>
<td>your</td>
<td>yours</td>
</tr>
<tr>
<td>he</td>
<td>his</td>
<td>his</td>
</tr>
<tr>
<td>she</td>
<td>her</td>
<td>hers</td>
</tr>
<tr>
<td>it</td>
<td>its</td>
<td>its</td>
</tr>
<tr>
<td>we</td>
<td>our</td>
<td>ours</td>
</tr>
<tr>
<td>they</td>
<td>their</td>
<td>theirs</td>
</tr>
<tr>
<td>one</td>
<td>one’s</td>
<td>one’s</td>
</tr>
<tr>
<td>NP (John)</td>
<td>NP’s (John’s)</td>
<td>NP’s (John’s)</td>
</tr>
</tbody>
</table>

Table 1: Listing of personal pronouns with corresponding possessive determiners and possessive pronouns.

(x) “Bambi’s father had antlers”

(((the.d ((poss-by Bambi) (father-of.n *s))) ((past have.v) (plur antler.n))))

(y) “I have the poster ready for the conference”

(I.pro (((pres have.v) (the.d poster.n))
  (ready.a (adv-a (for.p (the.d conference.n))))))

(z) “She had her audience in stitches”

(She.pro (((past have.v)
  (the.d ((poss-by she.pro) (audience-of.n *s))))
  in_stitches.a))

Notice that in examples (y) and (z) have.v takes two complements, one NP and one monadic predicate. Please keep in mind that “have” can indicate the perfect aspect (see Section 4.9).

4.12.5 Possessive Determiners and Pronouns

Some examples have already shown uses of possessive determiners. These have special interpretations that rely on the corresponding pronoun. For example, my.d is rewritten as (i.pro ‘s) (which itself is a macro – see Appendix A.6 for details). Possessive pronouns have a similar mapping. For reference, here is a table of possessive determiners and pronouns in relation to the basic pronouns.

4.13 Comparatives, superlatives, etc.

Comparatives involve explicit or implicit comparison of measures on some scale, as in “π is greater than 3”, or “Al is older than Bill”. To treat all such examples systematically, we first need to digress into measures.

[The examples in the comparatives (sub)sections still need indenting repaired, and some uncertainties resolved.]
4.13.1 Measures

We often measure entities quantitatively in terms of units of some sort, such as miles, grams, minutes, light years, miles per hour, city blocks, etc. English and other languages specify quantities using the same kinds of phrases as are used to describe entities of other sorts. For example, *twenty nautical miles* has the same structure as *twenty naval vessels*, and so we would expect the ULF forms to be similar:

\[
(\text{twenty.a (nautical.a (plur mile.n)))},
\]
\[
(\text{twenty.a (naval.a (plur vessel.n)))},
\]

(before formation of kind-level or quantified terms from these nominal predicates).

What is unusual about such quantities, however, is their identity conditions. For example, while it’s hard to find an alternative specification equivalent to *three naval vessels*, we could specify a distance of twenty nautical miles equivalently as *twenty-three miles*, *thirty-seven kilometers*, etc. It appears that all these expressions map to some common abstract length or distance entity. Since such distance entities are produced by combining a number with a unit of length, theorists often regard units of measurement as functions on numbers, e.g., *(mi 23) or (km 37)*, where these have abstract lengths/distances as values (in this case, approximately the same). However, we do not switch to such representations here, in part because staying closer to the surface structure should make machine learning easier, and in part because the functional view requires invention of length “functions” used in informal and technical language such as paces, horse-lengths, hair-widths, football-fields, Planck-lengths, Earth-radii, astronomical-units, etc. By retaining the compositional structure of such units, we may also be able to infer the approximate length-amounts they refer to, if we know for instance what the standard length of a football field is, or what the Earth’s radius is. Similar comments apply to measures for other quantities, such as duration, weight, volume, speed, acceleration, temperature, etc.

So when we form terms from predicates like *(twenty.a (nautical.a (plur mile.n)))*, we will do so by applying the kind-forming operator *k*; so

\[
(k (\text{twenty.a (nautical.a (plur mile.n))}))
\]

is regarded as a kind whose realizations are abstract distances along specific paths, as in “In the last twenty nautical miles of its journey, the ship rounded the Cape of Good Hope.” Note that this would not be easy to express using functions for units.

4.13.2 Comparing quantities

We annotate comparatives using the special relations *more-than* and *less-than*, which relate two quantities. (You can think of them as special adjectival predicates, of the same general semantic type as *near.a, fond-of.a, divisible-by.a*, etc.) The simplest cases are ones where the quantities are directly specified in English; e.g.,

(a) “One mile is more than one kilometer”

\[
((k (\text{one.a mile.n})) ((\text{pres be.v}) (\text{more-than} (k (\text{one.a kilometer.n}))))))
\]

(b) “The width of the Lower Mississippi is more than a mile”

\[
((\text{the (width-of.f (the |Lower Mississippi|.n)))})
\]

\[
((\text{pres be.v}) (\text{more-than} (k (\text{one.a mile.n}))))])
\]

Note: Though *width* is a noun, we can write *width-of.f* instead of *width-of.n*, to indicate that this relation is necessarily single-valued, i.e., a function (when measured
at a particular cross-section of an object); contrast this with parent-of.n (there are typically two), and even mother-of.n, where under special circumstances, such as birth mother and adoptive mother, there may be more than one. But measure nouns such as size, length, weight, square (of), as well as “topological” part-selectors such as surface (of), interior (of), etc., can typically be treated as functions.) Of course mathematical functions such as sine, ratio, derivative, etc., can also be coded with .f (but beware: e.g., the square root of 1 can be 1 or -1 when we’re considering the real numbers; when in doubt, use .n.)

(c) “The Lower Mississippi is more than one mile wide”
(cf., “very wide”, i.e., more than one mile, like very, functions as an adverbial)

((the |Lower Mississippi|.n) ((pres be.v) ((adv-a (more-than (k (one.a mile.n)))) wide.a)))

We can modify this relation (once it’s applied to the second element in the comparison), as in

(d) “One mile is quite a bit more than one kilometer”

((k (one.a mile.n)) ((pres be.v) ((adv-a (quite.adv-a a_bit.a)) (more-than (k (one.a kilometer.n))))))

(e) “One mile is 600 meters more than a kilometer”

((k (one.a mile.n)) (pres be.v) ((adv-a ((num 600) (plur meter.n))) (more-than (k (one.a kilometer.n))))))

(f) “One mile is more than the length of 5 city blocks.

((k (one.a mile.n)) (pres be.v) (more-than (the.d (length-of.f (k (five.a ((nn city.n) (plur block.n))))))))

(g) “The size of the cosmos was initially much less than one micron”

((the.d (size-of.f (the |cosmos|.n))) ((past be.v) initially.adv-e (much.adv-a (less-than (k (one.a micron.n))))))

Note that the representations for more than are similar to what we would use for equivalent phrasings such as exceeds or is in excess of. Also more modifying a noun (i.e., as adjective or quantifying determiner) often means a greater {amount / number} of, as in more money, while applied to an adjective it means to a greater degree. However, more in the former sense of a greater {amount / number} of can be recast technically in terms of degrees as to-a-greater-degree much (with much regarded as an adjective – it can also function as an adverb, as in much further, much to my surprise, and as a quantifying determiner, as in Much rain fell). So in, principle all comparisons can be framed in terms of degrees. [There’s more to be said about much, e.g., (adv-a (very.adv-a much.a)), but perhaps not here...]
4.13.3 More complex comparatives

The quantities being compared were explicit in all the above examples. But when the comparative phrase (headed by an -er word, like “more” or “faster”) modifies another phrase and/or contains a sentential complement (as in “He walks faster than I do / I jog”, then either or both quantities being compared become implicit – and we leave them that way in ULF, while “marking” the locations where the quantities implicitly occur with special adverbs ‘degree1’ and ‘degree2’ (for comparands 1 an 2 respectively). So an apparent sentence of form [...degree1...] will actually serve as a term of type the degree d such that [...(to-degree d)...]. Furthermore, macros more-x-than and less-x-than are introduced to handle these constructs while preserving word order. (see more formal expansions below):

(a) “John jogged further than five miles”

```
(|John| (past jog.v)
  (more-x-than (adv-e (degree1 far.a))
    (k (five.a (plur mile.n)))))
```

(expanding more-x-than macro)

```
(((|John| (past jog.v) (adv-e (degree1 far.a))))
  more-than
  (k (five.a (plur mile.n))))
```

(for “jogged more than five miles” we would use (adv-e (degree1 much.a)) instead of (adv-e (degree1 far.a))). So this says that the degree d1 such that John jogged far to-degree d1, is more than five miles. Postprocessing will make the quantities compared explicit.

(...degree1...) expands to

```
(the.d (:l d1 (...(to-degree d1)...)), and similarly
(...degree2...) expands to
(the.d (:l d2 (...(to-degree d2)...)).
```

If we amplify further to much further in (g), then in the ULF we need to bracket more-x-than with its second argument before applying the adverb:

```
(((|John| (past jog.v)
  (much.adv-a more-x-than) (adv-e (degree1 far.a))
    (k (five.a (plur mile.n))))))
```

(more-x-than expansion with modifier.)

```
(((|John| (past jog.v) (adv-e (degree1 far.a)))
  (much.adv-a (more-than (k (five.a (plur mile)))))))
```

(b) “This pizza is more (food) than I can eat”

```
((This.d pizza.n) (pres be.v)
  (more-x-than ((degree1 {much}.a) food.n)
    (I.pro ((pres can.aux) (eat.v {(k ((degree2 much.a) food.n))])))
  )
```

(or, if food is missing, ((degree1 much.a) stuff.n))
more-than
(I.pro (pres can.aux) (eat.v ((k ((degree2 much.a) food.n)))))

(c) “More food than I could eat was on the plate”
((more-x-than (k ((degree1 {much}.a) food.n))
   (I.pro (past can.aux-v) (eat.v ((k ((degree2 much.a) food.n))))))
   (past be.v) (on.p (the.d plate.n)))

(d) “The size of the cosmos is more than I can fathom”
((the (size-of.f (the |cosmos|.n))) more-than
(i.pro (pres can.aux-v) (fathom.v (a-gen.d ((degree2 big.a) (size-of.f (the |cosmos|.n)))))))

Here we need to recognize that there is a gap at the end of the sentence, for the thing
being fathomed—which is a certain size of cosmos, i.e., a size that is big to a certain
degree. We can assume axiomatically that the degree to which a size is big equals that
size; so when ’degree2’ is expanded to make ’d2’ explicit, d2 becomes equated to the
(possible) size of the cosmos.

(e) “John is taller than Bill”
((|John| (pres be.v) (degree1 tall.a))
 more-than ((|Bill| (pres be.v) (degree2 tall.a))))

(f) “Mary is smarter than anyone (else)”
((|Mary| (pres be.v) (degree1 smart.a))
 more-than
((any.d (n+pred person.n (distinct-from.a |Mary|)))
 (pres be.v) (degree2 smart.a))))

In an upward-entailing context, any.d is equivalent to every.d, except that it prefers
to take narrow scope.

(g) “The bed is wider than the door is high”
(((the bed.n) (pres be.v) (degree1 wide.a))
 more-than ((the door.n) (pres be.v) (degree2 high.a))))

(h) “The movie was more silly than scary”
(((the movie.n) (past be.v) (degree1 silly.a)) more-than
((the movie.n) (past be.v) (degree2 scary.a))))

(i) “A man taller than Bill showed up”
(((a.d (n+pred man.n
(that.rel ((*h (degree1 tall.a)) more-than (Bill (degree2 tall.a)))))
(past show-up.v))

35
i.e., “A man such that the degree to which he is tall is more than the degree to which Bill is tall showed up”. Seems complicated; but this would also extend to “A bed wider than the entrance is high was delivered”.

(j) “John sneezed more than Mary coughed”

\[
\text{((} |\text{John}\text{| (past sneeze.v) (adv-a (degree1 much.a)))}}
\text{more-than}
\text{((} |\text{Mary}\text{| (past cough.v) (adv-a (degree2 much.a)))}}
\]

(k) “The pizza was bigger than I expected (it to be)”

\[
\text{((the.d pizza.n) (past be.v) (degree1 big.a))}
\text{more-than}
\text{((I.pro (past expect.v) ((the.d pizza.n) be.v (degree2 big.a)))}}
\]

The assumption about expect here is that it takes a single complement of form (NP VP[to-inf]), rather than an NP object and a separate infinitive complement, because otherwise we can’t account for the scope ambiguity of, e.g., someone in “John expects someone to give him a ride”. If “someone to give him a ride” sentential, we can restrict the scope of someone to that sentence, so that it stays within the scope of expects.

(l) “John baked fewer cookies than Mary wanted to eat”

\[
\text{((} |\text{John}\text{| (past bake.v) ((nquan (degree1 many.a)) (plur cookie.n)))}}
\text{less-than (} |\text{Mary}\text{| (past want.v) (to eat.v ((nquan (degree2 many.a)) (plur cookie.n))))}}
\]

(m) “For some reason I feel more alive at night”

\[
\text{((adv-s (for.p (some.d reason.n))}
\text{((l.pro ((pres feel.v) (degree1 alive.a) (adv-e (at.p (k night.n))))))}
\text{more-than}
\text{((i.pro ((pres feel.v) (degree2 alive.a) (adv-e (at.p (k {ref1}.n))))))}}
\]

i.e., we leave open the problem of filling in a reference predicate. This is a reasonable strategy, because in general the reference predicate, if omitted, is context-dependent. For example, if the preceding sentence is “I’m not a morning person”, then the reference predicate seems to be “morning time”.

So filling in {ref1}.n becomes a postprocessing problem, just like anaphora (pronoun) reference determination. Perhaps by default we contrast night-time with day-time (an antonymic relation), and this could be coded as

\[
\text{((adv-e (at.p (k (n+pred time.n (\{t (not (t night-time.n))\))))))}}
\]

This unfortunately still requires lambda abstraction (despite the use of ’n+pred’, which often avoids lambdas), so we might allow ourselves a predicate modifier ’non-‘, so that we can express “non-night-time”:

\[
\text{((adv-e (at.p (k (n+pred time.n (non- night-time.n))))))}
\]

We use a hyphen at the end of the ’non’ because this normally occurs as a prefix, e.g., in non-scientist, non-expert, etc.
[TODO: Add subsections on superlatives, equatives (as ... as ...), and excessives (too ... to/for ...). Superlatives make explicit, partially implicit, or implicit reference to a class: “2 is the smallest prime”, “7 is the largest prime less than 10”; “The swift is the fastest flyer” (among flying birds, in horizontal flight); “Cheetahs run fastest” (among animal species); “That toddler has the cutest smile” (possible for any toddler? any child? any person?). In the ULFs, it’s probably sufficient to specify the comparison class – possibly consisting of or including a {ref} predicate. It shouldn’t be necessary to spell things out like “For all primes x not equal to 7, 7 is larger than x”. Equatives are much like comparatives, but they substitute “equal to or greater than” for “more than”, and “equal to or less than” for “less that”, with an implicature of approximate equality; “Alice is as smart as Bob” (we can add without contradiction, “in fact, smarter”, thus canceling the implicature). Also saying “at least as smart as” suppresses the implicature, as does “almost as smart as”. Excessives seem to relate some gradable property to a type of purposeful action or attribute ... more thought required; “Too heavy to carry”, “That car is too small for me”...]

4.13.4 Superlatives

Superlatives require the identification of a quality that is maximized by a given statement over a set of statements. In the simplest examples, the quality is an adjective (e.g. tall.a), the set of statements is characterized by a noun (e.g. mammal.n), and the unique statement is characterized by an individual (e.g. giraffe.n). These cases can be delineated with a straightforward macro, most-n, that maps the adjective and the noun to a lambda expression describing the individual that maximizes over the set. The exact mapping is given at the end of the section.

However, there are cases where the set of statements is characterized by an adverb (“Cheetahs run fastest in a straight line”) or by the full sentence (“Today I ran the fastest I have ever”), which require their own macros for proper handling. All the macros work similarly at a superficial level – they specify the adjective maximized and the characterization of the comparison set of sentences. The macro identifies the corresponding maximizing statement from the context to map to the explicit semantic representation. The macros are most-* where * is replaced by the characterizing type (e.g. n, adv, s, etc.). See below for examples:

(a) 2 is the smallest prime
   (2 ((pres be.v) (= (the.d (most-n small.a prime.n)))))

(b) 7 is the largest prime number less than 10
   (7 ((pres be.v) (= (the.d (most-n large.a (n+preds (prime.a number.n) (less-than 10)))))))

(c) The swift is the fastest flyer
   (((The-gen.d swift.n) ((pres be.v) (= (the.d (most-n fast.a flyer.n))))))

(d) Cheetahs run the fastest (among animal species)
   (((k (plur Cheetah.n)) ((pres run.v) (adv-a (most-n fast.a {ref}.n))))
       (most-n fast.a (animal.a (plur species.n))))
(e) The toddler has the cutest smile

((The.d toddler.n) (pres have.v) (the.d (most-n cute.a (smile-of.n *s)))))

(f) The blue whale is the largest mammal

((The-gen.d blue_whale.n) (pres be.v) (the.d (most-n large.a mammal.n))))

(g) The largest mammal weighs about 200 tons

((The. d (most-n large.a mammal.n)) (pres weigh.v) (k ((about.adv-a 200.a) (plur ton.n)))))

(h) The human foot is narrowest at the heel

((The-gen.d (human.n foot.n)) (pres be.v) (the.d (most-pc narrow.a (adv-e (at.p (the.d {ref}.n))))))

(i) First Class mail usually arrives the fastest

((most-max (k (First_Class.a mail.n))) usually.adv-s ((pres arrive.v) (adv-a (most-pc fast.a (I.pro ((past perf) {run}.v (adv-a {fast}.a)) (adv-a (most-class (of.p {run}.v) (adv-a (most-class (of.p us.pro)))))

(j) Today, I ran the fastest I had ever

((most-max (Today.adv-e (I.pro ((past run.v) (most-pc fast.a (I.pro ((past perf) {run}.v (adv-a {fast}.a)) (adv-a (most-class (of.p them.pro)))))

(k) The Burj Khalifa in Dubai is the tallest building by several hundred feet

((most-max (np+preds (The.d |Burj Khalifa|.n) (in.p |Dubai|))) (pres be.v) (= (the.d ((most-pc tall.a building.n)))))) by.p-diff (k ((several.a hundred.a) feet.n)))))))

(l) I hit the table hardest of them all

(i.pro (((past hit.v) (most-max (the.d table.n))) (adv-a (most-pred hard.a)) (adv-a (most-class (of.p them.pro))))))

(m) I hit the table hardest of us all

((most-max i.pro) (((past hit.v) (the.d table.n)) (adv-a (most-pred hard.a)) (adv-a (most-class (of.p us.pro)))))

(n) Cheetahs run fastest in a straight line

[TODO(GK): need to update so that most-class is different for noun-ind vs. pred-pred]

Notice in (i) that usually.adv-s will need to be lifted above the full comparison whereas in (j) Today.adv-e will stay at sentence level for the correct interpretation. That is for (i) it is usually the case the First Class mail arrives faster than any other mail, while for (j), the run from today was faster than all other runs. Now if we replace usually.adv-s with a *.adv-e, say Today.adv-e, we get the interpretation that the first class mail arrives today faster than any other mail arrives today.

Definitions of Macros

(most-n A.a N.n) =>
( :1 x (x N.n) and
  (forall y
    ((y N.n) and (not (y = x))))
38
\[ \rightarrow ((x \text{ (degree}1 \text{ A.a}) \text{ more-than } (y \text{ (degree}2 \text{ A.a})))\]

\[ S[(\text{most-pc A.a Pc})/(\text{most-max Pm})] \Rightarrow \]  
\[ S[(\text{most-pc A.a Pc}) \leftarrow (\text{degree}1 \text{ A.a}); (\text{most-max Pm}) \leftarrow \text{ Pm}] \Rightarrow \text{ e1} \]

### 4.13.5 Excessives

(a) The temperature is too hot for comfort by 20 degrees

\[(\text{The.d temperature.n}) ((\text{pres be.v}) \text{ (}\text{}}\]

### 4.14 Punctuation

..........................................................

Notes for new section:

Drop all basic punctuation that is not semantically relevant (e.g. only pragmatically relevant). For the most part, we'll capture the meaningfulness of punctuation in the bracketing (e.g. commas). Only annotate the following:

!: for imperatives (show example that makes it ambiguous without)

?: for questions

..........................................................

In general, we try to keep punctuation close to surface text as possible. As we have seen already in the tutorial, apostrophes for contractions are dropped and the contraction expanded to the constituent words in analysis and apostrophes for possessives are expanded to the possessive semantic representation. For the remaining punctuation, we keep a surface-like representation.

#### 4.14.1 Sentence-level Punctuation

Punctuation such as periods, exclamation marks, and question marks indicate the end of the sentence as well as the nature of the sentence itself. These are annotated at the end of the sentence to match the surface text. So the bracketing structure is:

\[(S \, \text{.}), (S \, !), (S \, ?), (S \, ;)\]

The backslashes on period and semi-colon is to retain compatibility with Lisp. In formal writing, the annotations would follow the surface text exactly since questions and imperatives. In less formal writing, questions and imperatives may not be marked in the surface text.

(a) “She is happy?”

\[ ((\text{She.pro ((}\text{pres be.v) happy.a})) \text{ ?}) \]

(b) “She is happy.”

\[ ((\text{She.pro ((}\text{pres be.v) happy.a})) \text{ \text{.}}) \]

(c) “Go home!”

\[ ((\text{you}.\text{pro ((}\text{pres go.v) home.n})) \text{ !}) \]

(d) “John, go home!”

\[ ((\text{John}\, \text{\text{.}}, ((\text{pres go.v) home.n})) \text{ !}) \]
(e) “You’re falling asleep, go to bed!”
    (((You.pro (pres (prog fall_asleep.v))) \,
      (((you).pro ((pres go.v) (adv-a (to.p (k bed.n)))))) !))

(f) “You’re falling asleep, so go to bed!”
    (((You.pro (pres (prog fall_asleep.v))) \,
      (so.dm (((you).pro ((pres go.v) (adv-a (to.p (k bed.n)))))) !)))

Notice that in example (c), we introduce an implicit argument of the listener with {you}.pro. However, in example (d), we use the explicit argument that exists in the sentence. These can also be followed by complete sentences such as example (e). Example (f) is the same as example (e) but with the discourse marker ‘so.dm’.

Semantically these are prefix operators, but we can simply mark them at the end. Since they are unary operators that mark a complete sentence, no ambiguity is introduced by post-marking these prefix operators.

4.14.2 Commas

Common uses of commas are annotated in place of occurrence, as shown in the previous section. However, for non-restrictive relative clauses or appositives (see Section 4.10.5) the commas are dropped since the information carried by the comma is captured by the rest of the annotation.

4.14.3 *New* Quotes

Quotes can be categorized into two types, mentioned quotes and integrated quotes, according to the interaction between the content of the quote and the type system. [TODO: change to be example-driven (just added this so far so as not to forget)]

1. mentioned quotes Mentioned quotes are characterized by being able to replace it with an arbitrary piece of text and retain the superficial grammaticality of the sentence. For example, consider the sentence
   “Love” is a four-letter word.
   We can replace “Love” with “Until tomorrow” a retain grammaticality, though the sentence would be patently false (“Until tomorrow” as more than four letters and isn’t even a word).
   We annotate mentioned quotes by interpreting the content and wrapping it in ( "  "). So the example above would be annotated:
   ((( "  
     (k Love.n) ")) ((pres be.v) (= (a.d ((four.a letter.n) word.n)))))
   The interpretation within the quote is dependent on the context which becomes apparent with the annotation for “Love” is a transitive verb.
   ((( "  
     love.v ")) ((pres be.v) (a.d (transitive.a verb.n)))))

40
There will be cases where the context will not fully disambiguate the word, such as the first example sentence (notice that it is a possibility that “Love” in that sentence refers to the transitive verb). Please annotate the most readily available interpretation. For single words this will likely be kind terms.

If the quoted material is not interpretable due to being a different language or gibberish, simply retain the quoted string. “bonsoir, monsieur”, was the sharp reply.

(“bonsoir, monsieur” ((past be.v) (= (the.d (sharp.a reply.n)))))

2. integrated quotes

We call quotes that do not retain grammaticality when replaced by an arbitrary quote integrated quotes. This is because the type structure of the quoted material is integrated into the surrounding sentence. Consider the sentence: John’s new ”theory” is confusing. If we replace ”theory” with ”Until tomorrow” it is no longer grammatical.

In these cases we simply ignore the quotes since the quote is not relevant for the semantic type structure. Although these quotes hold information that is important for further resolution of meaning, they turn out to be too difficult to properly handle. Since these integrated quotes don’t need to be constituents in the formula, in general these require lexical marking of the quote. Even so, we lose parts of the surface form that are not captured by ULFs (spaces, punctuation, capitalization, etc.) that could be relevant for exact processing of the quoted material.

(in fact this has a correspondence even to non-quoted material. A sentence could be self-referential of it’s surface form, which the ULF alone would not be able to handle: “The interpretation of this sentence has more atoms than the original number of words.”)

[TODO: discuss quote dilemma and the fact that our annotation decision loses needed information that regards exact surface form.]

[TODO: add examples from email correspondence]

4.14.4 Quotes

Quotes are categorized into two different types: object quotes and integrated quotes. These two categorizations are handled differently.

1. object quote

Object quotes are quotes that are semantically treated as a thing in the language. Thus the quote as a whole is being talked about in the sentence, rather than the words in the quote grammatically and semantically interacting with the surrounding sentence. Below we list a few examples:

(1a) “Mary” is a common name.
(2a) He said, “My bagels are the best”.
(3a) ‘o’ is a vowel.
(4a) “bonsoir, monsieur,” was the sharp reply.

The representation of these quotes depend on the complexity and feasibility of obtaining a ULF form of the quote. For quotes that cannot obtain a ULF form because it is too small of a semantic unit (perhaps one word), or in a different language, simply write the
quote surrounded by the quote type (i.e. single or double) as occurring in the surface text preceded by a single quote (’). For ULF interpretable quotes, wrap the interpretation in backslashed quotes as appearing in the text, and add a single quote in front of the opening backslashed quote. The default interpretation of a quote wrapped segment is an object. See below for examples.

(1b) ("Mary" ((pres be.v) (a.d (common.a name.n))))
(2b) (He.pro ((past say.v)
  (’" ((my.d (plur bagel.n)) ((pres be.v) (= (the.d best.a))) "))))
(3b) (’‘o’’ ((pres be.v) (a.d vowel.n)))
(4b) ("'bonsoir, monsieur" ((past be.v) (the.d (sharp.a reply.n)))))

Then we can have axioms for saying such that ∀x,y. [x say.v y] ⇒ [x say.v (that (semantic-interpretation-of y))]

2. integrated quote
Integrated quotes are ones where the words within the quotes are a semantic and grammatical part of the sentence. The normal meanings of the words may be altered due to the quotes, but nevertheless the quoted words are treated as words in the sentence rather than an object. See the examples below:

(1a) Harvey said an inquiry would not be ruled out, “should serious and systemic issues” emerge as a result of the MoD’s own investigations.
(2a) John’s new “theory” is hard to understand.
(3a) We sell “fresh” pastries.
(4a) She read Maitland’s Dark Ages, “which I enjoyed very much”; La Croix on the Customs of the Middle Ages, 16 chapters of Bryce “and liked it more and more”; more chapters of Guizot; Lecky and Stanley’s Eastern Church.

Properly used integrated quotes hold a semantic unit, so that we can notate the quote as a whole and insert into the logical form as one unit. To see this more clearly, consider the following examples:

(a) He welcomed the release “following successful negotiations”
(b) He welcomed the release following “successful” negotiations
(c) He welcomed the release “following successful” negotiations
(d) He welcomed “the release following” successful negotiations
(e) Their claim is that “most if not all” animals are wild.
(f) He welcomed “the release” following successful negotiations

(a), (b), and (e) are grammatically allowed since the quoted expression is a semantic unit. (c) and (d) however are incorrect uses of integrated quotes. They also do not contain a semantic unit. In (c), “following successful” begins a prepositional predicate but does not complete it. (d) similarly starts with a noun phrase (“the release”) and includes the next preposition, but not the remainder of the preposition phrase, so it is incomplete.

For integrated quotes we introduce two alternatives again. For uninterpretable quotes, the quote is surrounded by double quotes and preceded by a backtick (i.e. ’). In this
font, the backtick is simply the open single quote, but still differentiable from the single quote (see ‘ vs ‘). For interpretable quotes, precede the opening backslashed quote with a backtick, which is a semantically transparent operator. The quote is used for pragmatic analysis.

(1b) ... " should.cc ((serious.a and.cc systemic.a) (plur issue.n)) " (pres emerge) ... 
(2b) (((|John| 's) (new.a (" theory.n ")))) 
(pres be.v) (hard.a (to understand.v))) 
(3b) (We.pro ((pres sell.v) (" fresh.a ") (plur pastry.n)))) 
(e’) (((their.det claim.n) ((pres be.v) 
(that (" (fquan (most.a (if.cc (non- all.a))) ") (plur animal.n)) 
(pres be.v) wild.a))))

Multisentence quotes
Quotes can often stretch across sentences, so the sentence-level annotation needs to be able to allow for identification of such cases. To capture this, we simply place the same backslashed quote operators where it appears in the surface text. Generally multisentence quotes will be object quotes, but both are allowed. This way we can mark segments of text that have a starting quote, but no ending quote, and vice-versa. Then in future steps where multiple sentences are analyzed together, the starting quotes (starting with ‘ or ‘) and ending quotes (bare) can be matched to each other to capture to full quote.

1. “A partially quoted sentence.
   (" (A.d ((partially.adv-a quoted.a) sentence.n)))
2. Another partially quoted sentence.”
   (Another.d ((partially adv quoted.a)) sentence.n) "
3. He said, “This long statement.
   (He.pro ((past say.v) \, " (This.d (long.a statement.n))))

So if the quote occurs in the middle of an annotation, simply mark it where it occurs without adding brackets (similar to commas). If the quote starts or ends the sentence, wrap it around the sentence (similar to periods).

Special Cases
Other uses of quote symbols
Quote symbols are sometimes used as special characters, such as to denote the length units feet and inches. In these cases, we expand out the quotation use to the standard unit representation. For example, 5’ 11” should be written as 5ft 11in.

[HOW WOULD THIS ACTUALLY BE USED IN THE LF? IT SEEMS WE NEED SOME OPERATOR TO MERGE QUANTITIES... MAYBE AN AMBIGUOUS AND? IF WE HAVE THE FOLLOWING WITH COMPLEMENTARY UNITS (ft/in, gallon/oz, cup/tbsp/tsp), THEN (quant1 and.cc quant2) → (quant-add quant1 quant2)]
Post-processing Quotes

All of the quote annotation can be read directly into Lisp because of the backslashes. However, characters preceded by a single quote (‘) or a backtick (‘) in Lisp are both interpreted as symbols. So to be able to differentiate object quotes and integrated quotes the annotation need to be post-processed in some way, such as the following:

Notice that single quote wrapped strings keep two single quotes after post-processing. This is because the first quote will act as a symbolizer

\[ "_" \rightarrow (quote-o "_") \]
\[ \'_\' \rightarrow (quote-o |\'_\'|) \]
\[ \"_\" \rightarrow (quote-i "_") \]
\[ \'\'_\' \rightarrow (quote-i |\'\'_\'|) \]
\[ \"\" \rightarrow (quotestart-o \") \]
\[ \'\' \rightarrow (quotestart-i \') \]
\[ \"\" \rightarrow (quotestart-i \") \]
\[ \'\' \rightarrow (quotestart-i \') \]

The notable case here are the single quoted strings. Lisp doesn’t read single quoted characters as strings, so we wrap it in bars so that the characters preserve case. Also, without the bars the ending single quote would be interpreted as an symbol marking operator for the following stream of symbols.

4.14.5 Parentheses

Parentheses annotation is done very similarly to quotes. Please annotated them where they occur in relation to the other words, using a backtick (‘) to mark quotes that integrate into the sentence and a single quote (‘) for those that do not. Again parentheses need to be backslashed for Lisp compatibility. More details will be described after the presented examples.

(a) “For appositives (see Section 4.10.3) the commas are dropped”

(b) “[He] hate[s] to do laundry”

\[ (\"\" (\'\'He.pro \'\') ((pres hate.v) (ka (do.v (k laundry.n)))))) \"

TODO: [Change annotation to distinguish between different types of parentheses (e.g. embedded, interleaved, appositive, domain specific, etc.)]

4.15 Numbers as Names

[TODO: move somewhere more appropriate]

There are cases where numbers are used as labels rather than to denote the number itself: the 1990s means the years modified by the 1990 label. So we use pipes to mark numbers as names separate from numbers themselves.

The late 1990s – (the.d (late.a ((nnp |1990|) (plur year.n))))
His late 20s – (his.d (late.a ((nnp |20|) (plur year.n))))
1990 was a great year – ([1990] ((past be.v) (= (a.d (great.a year.n))))))
20 years – ((nnp 20) (plur year.n))
4.16 Stringify

[TODO: add to it]

Some phenomena in language have their own domain specific grammars (e.g. time, dates, phone numbers, etc.) that do not fit directly into general English grammar. To reduce the learning difficulty, these will simply be wrapped in quotes. These could be further resolved into record syntax using a domain-specific parser. For example: “My number is 555 123 5555” would be annotated

$((\text{my.d number.n}) \text{ (pres be.v) (ds "555 123 5555")))$

[Maybe we should add what domain in which it lies? (ds phone-number "555 123 5555")]

[TODO: convert below subsection to this. Copy below subsection to appendix on possible post-processing]

4.17 Dates, Times, Currency, Addresses, etc.

There are certain categories where we really use domain-specific representations so rather than trying to tie these subgrammars into our general English grammar handling, we use a “record type” to write these down

<record> ::= ($ <record type> <term>1 <term>2 ... <term>n), n \geq 1

For dates and times we have the record type date-time where by convention we use “-” for “unspecified”, and always go in the order year, month, day, hour, minute, second, but stop as soon as there are no more specified items. So “5:30pm” would be ($ date-time -- -- 17 30) and “June 18th, 2017” would be ($ date-time 2017 6 18).

Below are a listing of common record types – please contact us if you come across any other phenomena that should be a record type.

1. Dates and Times

($ date-time <year> <month> <day> <hour> <minute> <second>)

“5:30pm” – ($ date-time -- -- 17 30)

“June 18(th) 2017” – ($ date-time 2017 7 18)

2. Currency

($ currency <currency name> <real number>)

“Five dollars and thirty cents” – ($ currency |dollar| 5.30)

“€30” – ($ currency |euro| 30)

“Three pound seventeen pence” – ($ currency |pound| 3.17)

3. Address

($ us_addr <street #> <street name> <street type> <city/town> <state> <zip>)

“880 Linden Ave” – ($ us_addr 880 |Linden| (k avenue.n))

Notice that for address, we specify us_addr. Since other countries may have significantly different address structures, each will need its own record type. See below for a reference of most US street types and their abbreviations:

- Road (Rd.)
- Way
• Street (St.)
• Avenue (Ave.)
• Boulevard (Blvd.)
• Lane (Ln.)
• Drive (Dr.)
• Terrace (Ter.)
• Place (PL)
• Court (Ct.)

A special case of record type is numbers, which we write down simply by the number (without $ or record type). So “one thousand nine hundred and seventy-four”, “nineteen hundred and seventy-four”, and “nineteen seventy-four” are all annotated as 1974 or the adjective or determiner variants of numbers mentioned in Section 4.6 on generated determiners.

4.18 Ellipsis
TODO [We won’t try to cover all types; start with simple ones like “Alice left, and Bob did too”; “Kim gave Alice a book, and Bob a sweater”; “Bob didn’t turn in his essay but Bill did” (ambiguous possessive ellipsis); etc.]

4.19 Coordination
(a) “I ate pizza and ice cream”

(i.pro ((past eat.v) (set-of (k pizza.n) (k ice_cream.n))))

(b) “Most eyes are brown, green, or blue”

((most.d (plur eye.n))

((pres be.v)

(brown.a or.cc green.a blue.a)))

(c) “Al and Clyde love Mary or Sue”

(((|Al| and.cc |Clyde|)

((pres love.v)

(|Mary| or.cc |Sue|)))

(d) “John went to the store and bought some peanuts”

(|John| (((past go.v) (to.p (the.d store.n))) and.cc

((past buy.v) (some.d (plur peanut.n))))))

(e) “I found a bag of food and drinks”

(i.pro ((past find.v)

(a.d (n+preds bag.n

(of.p (set-of (k food.n) (k (plur drink.n))))))))

(f) “John and Mary hugged each other”

(((set-of |John| |Mary|) ((past hug.v) each_other.pro))
“Bob and I chatted over tea and crackers”

(h) “I bought apples and oranges”

There is ambiguity between the collective and distributive readings of “and”. The collective reading creates a new individual, which is the collection made up of the individuals that are enumerated within in scope of “and”. This is annotated with the operator set-of. The distributive reading is the sentence-level logical conjunction. There are instances where a coordinating conjunction is unambiguously the collective reading of a specific scope. (e), (f), and (g) are examples of this. Also, there are cases where all sensible interpretations of the sentence satisfy both the collective and distributive readings. (a) and (h) are examples of this case. (h) specifically means that the speaker bought a collection of apples and oranges, but in doing so also bought apples and bought oranges. For such cases we default to the collective reading and leave the distributive (and concurrently true) reading to the semantics of the words.

Coordination can also occur at both predicate and individual levels. See that (h), and (d) show coordinated predicates, while (e) show coordinated individuals. However, the arguments must have consistent types, semantically and syntactically, since when the coordinators are scoped, the arguments must be able to distribute coherently. Of course, the collective reading can only occur with coordinated individuals.

or.cc and and.cc have the same scoping ambiguity as quantifiers. This isn’t surprising considering that some.d and all.d quantifiers can be rewritten as or.cc and and.cc statements respectively over the restrictor predicate members.

4.20 Discourse Markers

TODO

4.21 Conditionals

TODO [if ... then ...]

4.22 Fragments


5 Listing of Logical Type Extensions and Special Operators for Reference

This section lists the logical-type extensions and special operators for reference.
5.1 Logical-type Extensions

- \( \cdot n \): nominal predicate (mouse, idea, domination, etc.)
- \( \cdot v \): verbal predicate (run, love, laugh, etc.)
- \( \cdot a \): adjectival predicate (happy, green, etc.)
- \( \cdot adv \): adverbial function
  - \( \cdot adv-a \): action/attribute modifying function (quickly, angrily, confidently, very, quite, entirely, extremely, etc.)
  - \( \cdot adv-e \): event modifying function (here, yesterday, etc.)
  - \( \cdot adv-s \): sentence modifying function (definitely, probably, etc.)
  - \( \cdot adv-f \): sentence-frequency modifying function (twice, regularly, usually, etc.)
- \( \cdot cc \): coordinator (and, or, but, etc.)
- \( \cdot p \): prepositional predicate, taking a noun complement in, with, etc.)
- \( \cdot ps \): sentential preposition, taking a sentential complement (when, before, until, while, if, as though, etc.); such words are also termed “subordinating conjunctions”, but we use “preposition” because Treebank parsers label them as such.
- \( \cdot pq \): single-word prepositional question phrase, taking an inverted sentential complement (when, where, meaning “at what time”, “at what place”, as in “When did he leave”, “Where does he live?”)
- \( \cdot pr \): single-word prepositional relative phrase, heading a relative clause (when, where, etc., meaning “at which” in a temporal or locative sense, as in “the era when...”, or “the place where...”)
- \( \cdot pro \): pronoun (him, I, it, etc.)
- \( \cdot d \): determiner (the, some, few, etc.)
- \( \cdot aux \): auxiliary (do, will, may, etc.)
- \( \cdot rel \): relative pronoun (who, that)
- \( \cdot \) : name (i.e. proper noun) (Mary, Star Wars, etc.); not really an extension, but marks the type of an atomic element in the logical form.

5.2 Special Operators

ULF has a set of special operators that are written without logical type extensions. They represent operations that are marked morpho-syntactically in English (making it difficult to handle using symbols that correspond to words in the source sentence) or have notable and consistent interpretations in EL.

- \( \cdot not \): negation
- \( \cdot plur \): pluralizer
- \( \cdot past/pres \): tense operators
- \( \cdot perf \): perfect aspect
- prog: progressive aspect
- pasv: passive tense
- k: kind operator (predicate reifier)
- ke: kind of event operator (event reifier)
- to: action reifier (also written ka)
- that: that operator (proposition forming operator)
- fquan/nquan: quantifier forming operators
- set-of: set forming operator

### 6 Conclusion

Now that you have gone through this tutorial, you should be equipped to annotate most sentences with the corresponding ULF. For a more thorough guide to annotating ULFs see the complete annotation guide. If while annotating, you see a situation that has not been covered in this tutorial or the complete annotation guide, please contact the project coordinators through the link in the ULF editor so that we may add guidelines for this new annotation situation.

### Appendices

#### A Macros and ULF Relaxations

In the main document we superficially describe the macros and ULF relaxations to give an intuitive explanation of how they fit into the annotation process. Here we will look at the macros and ULF relaxations in depth, describing the process of getting exact ULFs from the version described in these guidelines in depth to show that type coherence and sentence meaning are preserved.

##### A.1 Type-shifter Dropping (Predicates as Modifiers)

Section 4.10.1 introduces using predicates as modifiers in the formula. This superficially leads to type-incoherency, but for the predicate combinations that we allow in using predicates as modifiers, we can automatically insert the appropriate type-shifter.

The predicate combinations that are allowed are “non-verbal predicates with other predicates”. The fully explicit type shifting of predicates to predicate-modifiers can be done with the following operators:

- nn - shifts noun predicate to noun predicate modifier
- *nnp - shifts noun phrase to noun predicate modifier
- attr - shifts adjective predicate to noun predicate modifier
- adv-a - shifts any predicate to monadic verb/adjective predicate modifier
*nnp doesn’t fit exactly into the way we have been talking since it’s formal type is an individual, not a predicate. But the principles are the same.

Below are the mapping functions that introduce the operators. They are applied bottom up and after relaxations that change bracketing such as sentence-level operator lifting. The constituent labels are the expected syntactic type (e.g. N for noun), indexed if two of the same type occur in the rule, and the ending quote denotes that it is the interpretation of the syntactic category.

\[
\begin{align*}
(N'N2') & \rightarrow ((nnN1')N2') \\
(N'N') & \rightarrow ((nnpN')N') \\
(A'N') & \rightarrow ((attrA')N') \\
(A1'A2') & \rightarrow ((adv-aA1')A2') \\
(N'A') & \rightarrow ((adv-aN')A')
\end{align*}
\]

Note that verb modifiers are not constructed implicitly. This is because verb modifiers look close to and occur in the same places as sentence modifiers. Thus they are annotated explicitly so we can distinguish verb modifiers from sentence modifiers.

A.2 Post-nominal Modifiers (n+preds and np+preds)

Definitions

\[
\begin{align*}
(n+predsN'Pred1Pred2...PredN) & \equiv \\
& (:l x ((xN')and(xPred1)(xPred2)...(xPredN)))
\end{align*}
\]

\[
\begin{align*}
(np+predsNP'Pred1Pred2...PredN) & \equiv \\
& (the.d(:l x ((x=NP)and(xPred1)(xPred2)...(xPredN))))
\end{align*}
\]

:1 is our ascii writing of lambda (\(\lambda\)). This macro can be applied at anytime.

These macros are introduced in Section 4.10.5 which give many examples of its uses. An important distinction here is that n+preds results in a predicate type and np+preds results in an individual type. In light of their uses in handling restrictive and non-restrictive relative clauses, respectively, this is not so surprising. Also, the type correspond to the first argument of the macro, which makes it easy to remember. Since these macros don’t do any reordering, they can be applied at basically any time.

A.3 Handling Gaps (sub)

Definition

\[
\begin{align*}
(subC[S[\star h]]) & \equiv S[\star h\leftarrowC]
\end{align*}
\]

Example

\[
\begin{align*}
(sub|Juliet|(|Romeo|((preslove.v)*h))) & \rightarrow (|Romeo|((preslove.v)|Juliet|))
\end{align*}
\]

sub is introduced in Section 4.10.2 and the following sections further discuss its uses. sub takes two arguments and substitutes its first argument into all free occurrences of *h in the second sentences. Identifying which occurrences of *h are free can be done with methods similar to avoiding variable capture in lambda-calculus, treating the sub, its first and second arguments similar to the \(\lambda\) symbol, the lambda bound variable, and the formula, respectively.
A.4 Relativizers (that.rel and who.rel)

Definitions

$S_{\text{emb}}[\text{that.rel}] \equiv (\lambda r S_{\text{emb}}[\text{that.rel} \leftrightarrow r])$

$S_{\text{emb}}[\text{who.rel}] \equiv (\lambda r S_{\text{emb}}[\text{who.rel} \leftrightarrow \text{(the.d (:l y ((y person.n) and (y = r))))}])$

Relativizers that.rel and who.rel have a fairly radical affect on the type structure where they lie. They are regarded as a variables that are lambda-abstracted at the level determined by identification of the appropriate type incoherence. What this reduces to is identifying when a sentence is supplied where a predicate is required. This apparent type incoherence is resolved when the sentence is lambda-abstracted and thus converted to a predicate. Notice below that unlike scope raising, relativizer wrapping is not trapped by sentence embedding boundaries.

"... the table that John knew I liked." gets annotated as

$$\text{(the.d (n+preds table.n)}$$

$$\text{(sub that.rel)}$$

$$\text{(|John| ((past know.v) (tht (I.pro ((past like.v) *h))))))}$$)

Which when n+preds and sub are expanded results in:

$$\text{(the.d (:l x ((x table.n) and.cc)}$$

$$\text{(x (|John| ((past know.v) (tht (I.pro ((past like.v) that.rel))))))})$$)

The resolution of that.rel that we want is the following

$$\text{(the.d (:l x ((x table.n) and.cc)}$$

$$\text{(x (:l r (|John| ((past know.v) (tht (I.pro ((past like.v) r))))))})$$)

Notice that if we naively wrap the lowest embedding sentence we get the following unwanted, and type incoherent result. The incoherent operations are underlined at the operator.

$$\text{* (the.d (:l x ((x table.n) and.cc)}$$

$$\text{(x (|John| ((past know.v) (tht (:l r (I.pro ((past like.v) r))))))})$$)

Of course correct resolution of that.rel relies on correct annotations, but that is true of other ULF relaxations as well.

In English relativizers appear directly to the left of the relative clause (what will become the lambda abstracted formula) so we can write simpler mappings from the annotated ULFs to the desired expansions by working on the ULF with relativizer position preserved. This amounts to pre-empting the expansion of sub macros that move the relativizer.

Simplified Relativizer Mappings

$$(\text{sub C[that.rel] F}) \equiv (\lambda r (\text{sub C[that.rel} \leftrightarrow r] F))$$

$$(\text{that.rel VP}) \equiv (\lambda r (\text{r VP}))$$

With these rules we still substitute $r$ for that.rel and abstract the appropriate constituent with a lambda expression. Though this increases the number of rules, these rules don’t require identifying the embedding sentence that is acting as a predicate. The main restriction for using these rules is that the first rule must be applied first if possible since there is some overlap in contexts where these rules apply. There are equivalent rules for who.rel which is replaced by $(\lambda y ((y person.n) and (y = r)))$ rather than $r$.

---

2 If you’re not sure how these expansions work step-by-step, take a look at Appendix Section A.4.1 for a

---

51
A.4.1 Walkthroughs: Handling a Relative Clauses

Now that we have discussed the macros n+preds, sub, and relativizers that.rel and who.rel, we’re at a point where we can use them to handle relative clauses in full. Here we walk through the macro expansions to show that we preserve the overall type structure and sentence meaning of the relative clause.

Walkthrough 1 (basic example)
Consider the formula for “car that you bought”
\[
(n+preds \text{car.n} (\text{sub that.rel} (\text{you.pro ((past buy.v) *h))})),
\]
now we can expand n+preds into the lambda expression
\[
(:l x ((x \text{car.n}) (x (\text{sub that.rel} (\text{you.pro ((past buy.v) *h))))))),
\]
then sub moves the relativizer into the relative clause
\[
(:l x ((x \text{car.n}) (x (\text{you.pro ((past buy.v) that.rel))}))),
\]
then we interpret the relativizer that.rel to *r in a lambda expression
\[
(:l x ((x \text{car.n}) \text{and.cc} (x (:l *r (\text{you.pro (past buy.v) *r})))),
\]
via lambda-conversion becomes
\[
(:l x ((x \text{car.n}) \text{and.cc} (\text{you.pro (past buy.v) x})));
\]
the predicate that is true of an entity if it is a car and you bought it.

Walkthrough 2 (simplified relativizer mapping)
This uses the same sentence as Walkthrough 1, but uses the simplified relativizer mapping. Consider the formula for “car that you bought”
\[
(n+preds \text{car.n} (\text{sub that.rel} (\text{you.pro ((past buy.v) *h))})),
\]
now we can expand n+preds into the lambda expression
\[
(:l x ((x \text{car.n}) (x (\text{sub that.rel} (\text{you.pro ((past buy.v) *h))))))),
\]
then we used the simplified rule \((\text{sub C[that.rel] F}) \equiv (:l *r (\text{sub C[that.rel←*r] F}))\)
\[
(:l x ((x \text{car.n}) \text{and.cc} (x (:l *r (\text{sub *r (you.pro (past buy.v) *h))))))),
\]
then sub moves the *r into the relative clause
\[
(:l x ((x \text{car.n}) (x (:l *r (\text{you.pro (past buy.v) *r}))])),
\]
via lambda-conversion becomes
\[
(:l x ((x \text{car.n}) \text{and.cc} (\text{you.pro (past buy.v) x})));
\]
and the result is the same as Walkthrough 1.

Walkthrough 3 (who.rel)
More subtly, who.rel will be rewritten as \((\text{the.d (:l y ((x \text{person.n) and.cc (x = *r))))})\), i.e. the entity that is a person and is identical with *r, so that, for example, “the manager who you met” is annotated as
\[
\text{np+preds (the.d manager.n)}
\]
\[
\text{(sub who.rel (you.pro ((past meet.v) *h)))},
\]
via np+preds and sub becomes
\[
(\text{the.d (:l y ((y = (the.d manager.n)) and.cc (y (you.pro ((past meet.v) who.rel)))))}),
\]
via who.rel becomes
\[
(\text{the.d (:l y ((y = (the.d manager.n)) and.cc (y (:l *r (you.pro ((past meet.v) (the.d (:l x ((x person.n) and.cc (x = *r))/Authenticating))))))))},
\]
which after *r lambda-conversion becomes
\[
(\text{the.d (:l y
\]
((y = (the.d manager.n)) and.cc
 (you.pro ((past meet.v)
 (the.d (:l x ((x person.n) and.cc (x = y))))))),
since the restrictor for the.d asserts an equality between the quantified individual (x) with
the variable y, we can simplify the quantified individual to y and raise the restrictor to a
predication over y.
((the.d (:l y ((y = (the.d manager.n)) and.cc
 ((you.pro ((past meet.v) y)) and.cc
 (y (:l x ((x person.n) and.cc (x = y))))))))),
simplify with lambda-conversion
((the.d (:l y ((y = (the.d manager.n)) and.cc
 ((you.pro ((past meet.v) y)) and.cc
 (y person.n) and.cc (y = y))))),
reduce tautology to T
((the.d (:l y ((y = (the.d manager.n)) and.cc
 ((you.pro ((past meet.v) y)) and.cc
 (y person.n) and.cc T))))),
simplify trivial and.cc, i.e. (p and.cc T) → p
((the.d (:l y ((y = (the.d manager.n)) and.cc
 ((you.pro ((past meet.v) y)) and.cc
 (y person.n))))),
flatten nested and.cc
((the.d (:l y ((y = (the.d manager.n)) and.cc
 (you.pro ((past meet.v) y))
 (y person.n)))),
i.e., the individual y such that y is the manager, you met y, and y is a person.

Definition of rule used for the.d with equality
S_{le}[(the.d (:l x F[(x = y)]))] ≡ (S_{le}[(the.d (:l x F[(x = y)]))] → y) and.cc (y (:l x F)))
S_{le} is the lowest embedding sentence that contains the bracketed formula. This rule says
that if the restrictor for the.d is a complex lambda predicate containing a positive assertion
of the equality of the quantified variable (here x) with some other individual (here y), we’re
replacing the quantification with the other individual (y). As for the rest of the restrictor,
we can now assert that the substituting individual (y) satisfies the restrictor and lift that
predication to the lowest embedding sentence level, S_{le}.

A.5 Mapping Names to Lisp

It turns out that when we load |_| into Lisp, we won’t be able to distinguish it from _ on
its own if all alphabetic characters between the pipes are upper case, and don’t include any
of the reserved characters of Lisp. For example, |C++| and |WABC-TV| would look the same
as c++ (or C++) and Wabc-TV (or WABC-TV) respectively in Lisp; (whereas |C#| would retain
the pipes, and in fact for c# or C#, pipes would be added by the Lisp reader). So, to map into
Lisp, the annotation is post-processed at the character level so that names in pipes where
Lisp would drop the pipes are prefixed with a blank space. For example, |C++| becomes |
C++| and |WABC-TV| becomes | WABC-TV|. Then when read into Lisp, the pipe-enclosed
symbol will remain pipe-enclosed and thus is identified as a name. The same happens for
name predicates, 

|John| → |John| (no change)
|U.S.A.|.n → | U.S.A..N|
|NY| → | NY|
|Missouri|.n → |Missouri.N|

Note that there is a distinction between using a name, and mentioning it. Names enclosed in pipes are being used (to refer to some entity in the world), whereas names (or other strings) in quotes are mentions of those strings, standing for the literal strings themselves. It turns out that the Lisp string function applied to a name in pipes replaces the pipes by quotes; for example, (string '|John|) evaluates to "John", so a use is converted to a mention. (However, any initial blank characters need to be removed.)

A.6 Possessives

Prenominal Possessive Rewriting Definitions

((NP 's) N[(_-of.n *s)]) ≡ (the.d N[Ns＜-NP])
((NP 's) N[!(_-of.n *s)]) ≡ (the.d (n+preds N (poss-by NP)))

The expansions reflect the post-nominal possession constructions described in Section 4.12, which are different for relational and non-relational possessives. poss-by is a binary predicate indicating general, unspecified possession which is only distinguished from relational possession, which is lexicalized in the relational predicate. Section 4.12 also lists a bunch of formulas before and after this rewriting for reference. An additional layer is built for possessive determiners to further lexicalize and simplify the annotations.

Possessive Determiner Rewriting Definition

my.d ≡ (me.pro 's)

my.d and me.pro can be replaced by any corresponding pair of possessive determiner and personal pronoun (fully listed at Table 1).

Possessive Pronoun Rewriting Definitions

mine.a ≡ (poss-by i.pro)
mine.pro ≡ ((i.pro 's) ref.n)
mine.a, mine.pro, and i.pro can be replaced by corresponding possessive pronoun and personal pronoun. Here are some examples:

That is mine – (that.pro ((pres be.v) mine.a)) → (that.pro ((pres be.v) (poss-by i.pro)))

Mine is red – (mine.pro ((pres be.v) red.a)) → (((i.pro 's) (ref).n) ((pres be.v) red.a)) → ((the.d (n+preds (ref).n (poss-by i.pro))) ((pres be.v) red.a))

B POS Tagset

We use the Penn Treebank POS tagset supplemented with an AUX tag for non-modal verbs (which are annotated as VB in the Penn Treebank). Below is a table of the tags for reference.
Please look at the pos annotation guidelines for in depth discussion of the annotations [http://repository.upenn.edu/cgi/viewcontent.cgi?article=1603&amp;context=cis_reports](http://repository.upenn.edu/cgi/viewcontent.cgi?article=1603%26context=cis_reports).

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential there</td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition or subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NNPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PRP$</td>
<td>Possessive pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
</tr>
<tr>
<td>RBR</td>
<td>Adverb, comparative</td>
</tr>
<tr>
<td>RBS</td>
<td>Adverb, superlative</td>
</tr>
<tr>
<td>RP</td>
<td>Particle</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol</td>
</tr>
<tr>
<td>TO</td>
<td>to</td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
</tr>
<tr>
<td>VBG</td>
<td>Verb, gerund or present participle</td>
</tr>
<tr>
<td>VBN</td>
<td>Verb, past participle</td>
</tr>
<tr>
<td>VBP</td>
<td>Verb, non-3rd person singular present</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb, 3rd person singular present</td>
</tr>
<tr>
<td>WDT</td>
<td>Wh-determiner</td>
</tr>
<tr>
<td>WP</td>
<td>Wh-pronoun</td>
</tr>
<tr>
<td>WP$</td>
<td>Possessive wh-pronoun</td>
</tr>
<tr>
<td>WRB</td>
<td>Wh-adverb</td>
</tr>
</tbody>
</table>