Annotating Unscoped Episodic Logical Form: A Tutorial

Version 1.0.0

<Authors>

Contents

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 close to

a sentence's surface form, requiring annotations at a level similar to constituency trees, but with the underlying semantic types in mind. The annotation process is manual with automatic sanity checking of common mistakes and type inconsistencies.

Let us begin with an example to understand the high-level annotation process. Consider the sentence *"Mary loves to solve puzzles"*.

1. Group syntactic constituents (NPs, ADJPs, VPs, etc):
   `(Mary (loves (to (solve puzzles))));`
2. Label the parts of speech (POS):
   `(nnp Mary) (vbz loves) (to to) (vb solve) (nns puzzles);`
3. Merge the previous two steps, using dot-extension for POS:
   `(Mary.nnp (loves.vbz (to.to (solve.vb puzzles.nns))));`
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 logical and macro 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").)

In practice this will all be done at once, or broken down just into the grouping of syntactic constituents and then the final annotation. However, when first learning ULF annotations, breaking it down into all of the above steps can simplify the process—especially for those with a background in syntax.

2 Basic Annotation Components

In this section, we introduce the core logical type extensions and special operators. As the tutorial proceeds, the significance . A more complete list of extensions and operators appears at the end of this document.

2.1 Listing of Logical Type Extensions and Special Operators

This section lists the logical-type extensions and special operators for quick reference.

2.2 Logical-type Extensions

- `.n` : nominal predicate (*mouse*, *idea*, *domination*, etc.)
- `.v` : verbal predicate (*run*, *love*, *laugh*, etc.)
- `.a` : adjectival predicate (*happy*, *green*, etc.)
- `.adv` : adverbial function

  - `.adv-a` : action/attribute modifying function (*quickly*, *angrily*, *confidently*, *very*, *quite*, *entirely, extremely*, etc.)
  - `.adv-e` : event modifying function (*here*, *yesterday*, etc.)
  - `.adv-s` : sentence modifying function (*definitely*, *probably*, etc.)
  - `.adv-f` : sentence-frequency modifying function (*twice*, *regularly*, *usually*, etc.)
- `.cc` : coordinator (*and*, *or*, *but*, etc.); however, the truth-functional connectives *and, or, not* can also be used as-is
- `.p` : prepositional predicate, taking a noun complement (*in*, *with*, etc.)
- `.p-arg` : argument-marking preposition (as in *ask **for** ..., tamper **with** ..., etc.*)
- `.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.
- `.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?")
- `.pro` : pronoun (*him*, *I*, *it*, etc.)
- `.d` : determiner (*the*, *some*, *few*, etc.)
- `.aux-s`, `aux-v` : auxiliary (*do*, *will*, *may*, etc. acting at the sentence or verb phrase level.)
- `.rel` : relative pronoun (*who*, *that*)
- `|_|` : 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.
- `.x` : expletive (*Damn*, *Ow*, etc.) §30.2
- `.gr` : greeting (*hi.gr*) §31
- `{_}.[suffix]` : implicit referent §8.4

2.3 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.

- `not` : negation
- `plur` : pluralizer
- `past`/`pres` : tense operators
- `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
- `voc`, `voc-O` : vocatives (`voc-O` is for vocatives explicitly marked with a "O") §27

2.4 Macros

ULF has *macros* which allow movement and simplifications of logical constructions that are simply communicated in language.

- `n+preds` : post-nomial modification §13.7
- `np+preds` : post-pronominal modification §13.7
- `pu` : phrasal utterance §30.1
- `'s` : post-possession §20
- `sub` : "substitute", topicalized (or left-moved) constituents §12
- `rep` : "replace", rightward-movement of constituents §16.3

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))))))`
5. `(|Kim| ((pres know.v) (that (|Sally| ((pres like.v) (to run.v))))))`

Like the example in the introduction section, we break down the annotation process to five steps. 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 (in accordance with English grammar). 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.prt (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. Ultimately, we combine the verb and its particle in the ULF, using an underscore. The underscore is used when multiple words in English form a single ULF atom. This is often the case with multi-word expressions involving particles. In cases where the additional word has a specified, regular interpretation in ULF, a dash is used to attach a word. For example, `mother-of.n` uses a dash because the 'of' is a marker of a relational noun. Dash-connected predicates can be used even when the attached word is not present in the annotated utterance, whereas underscore-connected predicates require the attached word to appear in the utterance.

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))))`
5. `(|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.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 future resolution steps, determiners are scoped at the sentence level, binding a variable and accompanied by the restrictor, which restricts the domain of the variable.

The remainder of this document will walk through one phenomenon at a time and describe how it is annotated.

4 The Components of a ULF

The ULF consists of atomic symbols and brackets that recursively group them together. At its core the subtrees of the ULF, including atomic symbols correspond to certain possible semantic types that control for how they can interact with other logical objects. The parentheses indicate which subtrees combine together to make new logical object. Because of this, we will never see parentheses surrounding a single element in ULF—we need at least two things to combine into a new object.

This view of ULFs is generalized with *macro* rewriting operations, which are triggered by particular pre-defined symbols. This allows the ULF to retain English word order in the face of linguistic phenomena such as movement and simplify the annotation interface for phenomena that have complex internal semantic structures (e.g. relative clauses). The parentheses still delimit how objects combine. However, the combination will not necessarily be the composition of the semantic types of the constituents. The details differ by *macro* operation.

There are also certain assumed post-processing steps to reduce the annotation overhead in cases such as interleaved arguments and modifiers of verbs and embedded sentence-level adverbs (see §13).

5 Basic Relationships between Predicates, Arguments, and Modifiers

Adjectival, prepositional, and verbal phrases can all function as both (1-place) *predicates* (expressing a property of some entity) and as *predicate modifiers* (transforming a property). 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 phrases are in noun post-modification, as in "food {*rich in cholesterol*, *in the pan*, *spattering oil*}". As explained fully later on (§13.7), such post-modifiers add predicates conjunctively to the noun they modify, i.e. we obtain a predicate expressing the property of being food *and* rich in cholesterol *and* in the pan *and* spattering oil.

The addition of post-modifying properties is simply a special way to modify nominal predicates. Nominal predicates can also be transformed by pre-modifying adjective phrases and nouns, as in "*harmful phony cancer* drugs" (more on this below, §5.2); 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 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.

5.1 Supplying Arguments to Verbs

All arguments except for the subject argument are supplied at the same brackting level of the verb and placed on the right—in concordance with English grammar. We will call this a *flat* construction of arguments. If we picture the ULF as a tree, these arguments would all be at the same level of the tree. For comparison, the alternative would be a nested, or curried, construction where each argument is supplied one at a time. This is further generalized to allow interleaved adverbial modifiers of the verb phrase.

(a) "I put the book on the table"

```
(i.pro ((past put.v) (the.d book.n) (on.p-arg (the.d table.n))))
```

(b) "He raised me a glass"

```
(He.pro ((past raise.v) me.pro (a.d glass.n)))
```

(c) "The dog grabbed the ball quickly"

```
((the.d dog.n) ((past grab.v) (the.d ball.n) quickly.adv-a))
```

(d) "I sold my car like a pro for a fortune"

```
(i.pro ((past sell.v) (my.d car.n)
                      (adv-a (like.p (a.d pro.n)))
                      (for.p-arg (a.d fortune.n))))
```

The adverbial modifiers can even come before the verb, but still be in a flat construction.

(e) "She quickly confused the criminals"

```
(she.pro (quickly.adv-a (past confuse.v) (the.d (plur criminal.n))))
```

5.2 Adjective 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, as in examples (a) and (c) below, or as predicate modifiers, 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.mod-a numerous.a)))
```

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

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

So in item (a), "happy" ascribes a property to Alice, while in item (b), it transforms the predicate `disposition.n` to a more specific one, `(happy.a disposition.n)`. In item (c), `off-shore.a` functions as a predicate modifier, while `(surprisingly.mod-a numerous.a)` functions as a predicate. In item (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 `mod-a` or `mod-n` type-shifting operator, which convert predicates into adjective and noun modifiers, respectively. So, `(happy.a disposition.n)` will be converted to `((mod-n happy.a) disposition.n)`, `(quite.mod-a numerous.a)` will be converted to `(mod-n (quite.adv-a numerous.a))`, and similarly for `off-shore.a` in item (c) and item (d). These repairs are easy to implement, because the `mod-a` and `mod-n` operators are needed for an AP just in case it modifies an adjective or a noun, respectively. (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.mod-n`, `consummate.mod-n`, etc., rather than `(mod-n former.a), (mod-n 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".[1] 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. 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.mod-a 5.a))`, `(nquan (about.mod-a five.a))`, `(nquan (very.mod-a many.a))`, etc.[2]

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.a (plur adult.n)))`, where `k` is the kind-forming operator.[3]

5.3 Copula Handling "be"

For the sake of uniformity, the copula, `be.v`, in our annotations will assume a binary operator that takes an object type argument as the subject and a predicate type subject as its object. Usually, this will be perfectly natural, item (a). This leads to an perhaps unintuitive annotation for sentences where the object of the copula is a noun phrase with an "a" or "an" determiner, item (b).

(a) "I am happy"

```
(i.pro ((pres be.v) happy.a))
```

[1] See §19 for more on numbers.

[2] `nquen` is discussed further in §8.

[3] See §6 for details on the kind-forming operator, `k`.

(b) "I am a lawyer"

```
(i.pro ((pres be.v) (= (a.d lawyer.n))))
```

In order to avoid deletion of surface words, we opt to introduce an equality operator for item (b) to make it be type consistent with `be.v`.

6 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.n)) ((pres be.v) widespread.a))
```

(b) "Every child likes pets"

```
((every.d child.n) ((pres like.v) (k (plur pet.n))))
```

(c) "I know that John is a man"

```
(i.pro ((pres know.v) (that (|John| ((pres be.v) (= (a.d man.n)))))))
```

(d) "Kim believes that every galaxy harbors life"

```
(|Kim| ((pres believe.v)
    (that ((every.d galaxy.n) ((pres harbor.v) (k life.n))))))
```

(e) "John knows he's right"

```
(|John| ((pres know.v) (tht (he.pro ((pres be.v) right.a)))))
```

Items (a), (b) and (d) show the usage of the kind-forming 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. `be.v`, `feel.v`, `impress.v`, etc.[4]). Bare nouns that act as object in the language will often be annotated with `k`.

Items (c) to (e) show the usage of the that-operator (`that`), which converts sentences to a propositional object. It is most widely used with attitudinal verbs (e.g. `believe.v`, `tell.v`, `hope.v`, etc.). Notice in item (e), the word "that" doesn't actually appear in the sentence although it is clearly an attitude. In such cases we use the `tht` operator, which is semantically equivalent to `that` but signals the fact that "that" did not appear in the sentence.

[4]See sections 7 and 13.2 for further discussion on predicate-control verbs

6.1 Kinds of actions and events

Special versions of the kind-forming operator exist for actions and events, `ka` ("kind of action") and `ke` ("kind of event"), respectively. We can also use `to` as a synonym for `ka`. Kinds of actions are often expressed in English as infinitives, such as *to ski*. Correspondingly, we form a kind of action in ULF by applying `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 in §3. This is coded by applying `ke` to an untensed sentence meaning; the instances of the kind thus formed are events of that described by that sentence. 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 `ka` to form the interpretation as a kind of action or the basic kind-forming operator `k` to form a kind of episode from the nominal. The "-ing" form of verbs participate in a variety of semantic phenomena. Please see §34.2 for a discussion of the different ways they can be annotated in ULF.

7 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.mod-a angry.a)`

(i) "The burglar sounded alarmed" – `alarmed.a`

7.1 "a little"

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, while in the latter "a" combines with "little" to form a determiner `a_little.d` meaning "a small amount of". A few examples below.

(a) "I saw a little lamb" – `(a.d (little.a lamb.n))`

(b) "I added a little salt" – `(a_little.d salt.n)`

(c) "It rained a little, and then the sun came out" – `((past rain.v) a_little.adv-a)`

Notice that item (b) cannot be interpreted with `(a.d (little.a salt.n))`. That is because `a.d` cannot be used with mass terms (such as *salt*), but `a_little.d` can. "a little" can also be an adverb which modifies adjectives or verbs, (`a_little.mod-a` or `a_little.adv-a`), as seen in item (h) and item (c), respectively.

8 Determiners

8.1 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 with nominal predicates (and occasionally prepositional predicates), and the combination (which is inherently unscoped in this annotation) introduces an individual or quantifies over individuals for which the restrictor predicate holds.

(a) "We ate some bread"

```
(we.pro ((past eat.v) (some.d bread.n)))
```

(b) "Every boy owns a toy"

```
((every.d boy.n) ((pres own.v) (a.d toy.n)))
```

(c) "Few big dogs are yappy"

```
((few.d (big.a (plur dog.n))) ((pres be.v) yappy.a))
```

(d) "Such friends are hard to find"

```
((Such.d (plur friend.n)) ((pres be.v) (hard.a (to find.v))))
```

(e) "Such a storm can ruin a city"

```
((Such.d (= (a.d storm.n))) ((pres can.aux-v) (ruin.v (a-gen.d city.n))))
```

Notice that in item (e) "a storm" is annotated as `(= (a.d storm.n))`. This is because the "a" is acting as a vacuous quantifier, similar to the quantifiers in predicative statements, e.g. "John is a lawyer". The fact that this is vacuous can be observed by the fact that it only appears in singular quantification of "such". A plural version of item (e) would be "Such storms can ruin a city". "Such" can also have an adjective interpretation. See §8.3, on generated determiners, for an example of such a use.

It's also worth noting that the distinction between numeral adjectives such as "five" and apparent quantifiers such as "several", "many", or "few" is quite blurry. After all, we can modify any of these ("around five", "at least several", "very many", "very few"), so that we need to assume adjectival versions of all of them. Conversely, `five.d` can be used as an abbreviation of `(nquan five.a)` (see §19 and §8.3 for details), so it would not be unreasonable to assume that the numeral adjectives also have a direct lexical meaning as determiners, rather than being abbreviations for `(nquan --.a)` expressions.

8.2 Determiners with a generic reading

The determiners "a" and "an" can have *generic* readings, 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. "The" can also be used generically, meaning "the kind". In this case we use `the-gen.d`. Adjectival modification as in item (c) was discussed earlier, and will be further illustrated in §13.1, in a broader context. Note that items (d) and (e) include phenomena that have not been described at this point in the document. §10 describes that details of auxiliaries such as `would.aux-s`, and §20 explains the representation of tacitly relational nouns like *friend*, and an implicit referent (using place-holder `*s`). In item (e), we use `|Javan|.a` to mark the adjective as name-like (see §18), and the adverb "now" is represented as `now.adv-e` to indicate that it is an event- (or situation-) modifying adverb (see §13).

(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.d 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))
```

8.3 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.mod-a all.a)) (plur cat.n)) (pres hunt.v))
```

(e) "Very few people dislike all dogs"

```
(((nquan (very.mod-a few.a)) (plur person.n))
       ((pres dislike.v) (all.d (plur dog.n))))
```

(f) "All four such remarks were unacceptable"

```
((All.d (four.a (such.a (plur remark.n)))) ((past be.v) unacceptable.a))
(((nquan ((mod-a All.a) four.a)) (such.a (plur remark.n)))
 ((past be.v) unacceptable.a))
```

Note the various acceptable treatments of "five" in item (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. In practice we recommend using the simplest annotation (`five.d` here) to reduce the amount of post-processing necessary for use.

Item (f) shows how multiple equivalent annotations exist with chained determiners because of different combinations of adjective interpretations of the determiners.

8.4 "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 items (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 items (c) to (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.mod-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.mod-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 §13.7 for more details on post-nominally modified noun phrases). Item (g) additionally includes a relative clause, which has a special representation using the `.rel` suffix to identify the relativizer.

(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))))
```

8.5 "Headless" partitives

Partitives are phrases that identify parts, members, or amounts of individuals, e.g. *some members of us*, *most portions of the pie*, *most parts of the water*. In these initial examples, we look at the most straightforward type of partitives which use explicit relational nouns (for more info on relational nouns, see §20.1)

(a) "some members of us"

```
(some.d (plur (member-of.n us.pro)))
```

(b) "most portions of the pie"

```
(most.d (plur (portion-of.n (the.d pie.n))))
```

(c) "most parts of the water"

```
(most.d (plur (part-of.n (the.d water.n))))
```

"Headless" partitives require special attention because unlike general "headless" noun phrases partitives without explicit nouns require some specialization in Episodic Logic to properly identify the correct noun. You've probably noticed that the examples above with explicitly stated partitive nouns sound a bit strange. That is, it sounds perfectly natural to say *some of us*, *most of the pie*, and *most of the water* and, in fact, perhaps sounds more natural than the examples given above. We'll handle these cases in the following way.

(d) "ten of us"

```
(ten.d (of.p us.pro))
```

(e) "the ten of us"

```
(the.d (ten.a (of.p us.pro)))
```

(f) "much of the pie"

```
(much.d (of.p (the.d pie.n)))
```

(g) "almost ten of us"

```
((nquan (almost.mod-a ten.a)) (of.p us.pro))
```

(h) "almost all of us"

```
((fquan (almost.mod-a all.a)) (of.p us.pro))
```

(i) "both of us"

```
(both.d (of.p us.pro))
```

Notice that we simply annotate "of" as `of.p` rather than supplying the head noun. Now, why don't we treat these like the other headless noun phrases in supplying the head noun that is being omitted? This is because there are headless partitives that don't have obvious relational nouns. For instance, what is the headed equivalent of *much of the pie*? *much parts of the pie*, *much amount of the pie*, *much pieces of the pie* all break our linguistic intuitions because none of these constructions are acceptable in English. Determining the correct noun requires an analysis of the meaning in relation to each noun's logical interpretation. We leave this to a future disambiguation step due to the specialization it requires.

We've also allowed `of.p` to be used in contexts where only nouns are allowed, e.g. as an argument to a determiner. In a disambiguated interpretation, `of.p` will need to be disambiguated to `part-of.n`, `member-of.n`, and `amount-of.n` to ensure that the semantics fully match. It is this correspondence that allows us to treat these `of.p` to act like nouns. We'll also be able to identify when `of.p` is in a partitive usage and requires such disambiguation when it is acting as a noun.

Sometimes even the "of" can be omitted in English if the determiner makes it clear that this is a partitive construction. In these cases, please supply a `{of}.p` to make the partitive construction clear in the ULF. The rest of the annotation is done in the same way.

(j) "half the pie"

```
(half.d ({of}.p (the.d pie.n)))
```

(k) "almost half the pie"

```
((fquan (almost.mod-a half.a)) ({of}.p (the.d pie.n)))
```

(l) "all the cars"

```
(all.d ({of}.p (the.d (plur car.n))))
```

There are some more complex examples of partitives that require either rewording or modification of the partitive 'of'. Item (m) uses post-nominal modification (`n+preds` §13.7) and item (n) uses emphatic adjectives (`entire-em.a` §28.1), both of which will be introduced in later sections.

(m) "ten of us at the party who had the egg nog"

```
(ten.d (n+preds (of.p us.pro)
                (at.p (the.d party.n))
                (who.rel ((past have.v) (the.d (egg.n nog.n))))))
```

(n) "almost the entire cake"

reworded as "almost all of the entire cake"

```
((fquan (almost.mod-a {all}.a)) ({of}.p (the.d (entire-em.a cake.n))))
```

Now we'll go through a few tricky cases that look like partitives but aren't to help clarify the distinctions and demonstrate the underlying issues that lead to the difference in annotation.

(o) "The ten in the building are trapped"

```
((The.d (ten.a (n+preds {ref}.n
                        (in.p (the.d building.n)))))
 ((pres be.v) trapped.a))
```

(p) "Much in the pie is healthy"

```
((Much.d (n+preds {stuff}.n
                  (in.p (the.d pie.n))))
 ((pres be.v) healthy.a))
```

(q) "Those in the forest"

```
(Those.d (n+preds {ref}.n (in.p (the.d forest.n))))
```

All of these examples use the preposition "in" instead of "of". The clearest and fastest way to realize that this can't be partitive is that by using "in", we cannot form a relational predicate that denotes parts. Really, it wouldn't make sense to make a relational noun with "in" since "in" really supplies an orthogonal meaning from the sortal noun that is the head of the `n+preds`. To understand further, please refer to §20.1 which discusses relational nouns in depth.

9 Passive Voice

Passive voice is annotated with the `pasv` operator.

(a) "He was pushed"
`(he.pro (past (pasv push.v)))`

(b) "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 we consider it to correspond to an identity operator. The syntactic marking of the passive voice can be reduced to the following construction.

be + *<past participle>*

We consider the copula (*be*) to correspond to `be.aux-v`, which is semantically an identity operator (i.e. $(\lambda P.P)$). Therefore,

`(<tense> (pasv <verb>))` $\equiv$ `((<tense> be.aux-v) (pasv <verb>))`

This is important when a passive construction appears alongside subject-auxiliary inversions, such as questions. See §17 for details on these inversions.

(c) "Is Jared liked by dogs?"

```
(((pres be.aux-v) |Jared| ((pasv like.v) (by.p-arg (k (plur dog.n))))) ?)
```

The argument that would normally be the subject is sometimes applied in the passive voice using the "by" preposition. This is annotated with `by.p-arg` as seen in item (c).

9.1 Verbal vs. Adjectival Passives

Distinguishing between adjectival derivations of verbs and passive voice can be very tricky because in many cases the semantic differences between these two interpretations are subtle or not apparent at all. Compare the example from §29 with:

"John was frightened by his coworkers"
`(|John| ((past (pasv frighten.v)) (by.p-arg (his.d (plur (coworker-of.n *s))))))`

Though not an exhaustive test, we will primarily rely on the following test to distinguish between adjectival and passive readings.

> *"A verb is passive if we can reasonably use 'by' to supply the agent/subject while retaining the same meaning of the word."*

Harwood, in his 2017 Studia Linguistica paper, lists ten tests for distinguishing adjectival passives from verbal passives. I will summarize his discussion here. The tests below can be used when the *by-subject* test is not conclusive. All of the examples in this discussion are directly from Harwood's discussion.

Two contexts in which the passive participle is unambiguously adjectival are: (1) as a pre-nominal modifier (e.g. "the broken window") and (2) as a complement to adjective-taking verbs (e.g. "The window looks broken"). For reference, here are a few adjective-taking verbs: *appear*, *sound*, *become*, *remain*, *look*.

Using these syntactic constructions, we can identify diagnostics for distinguishing verbal and adjectival passives in ambiguous contexts.

1. *Duration adverbials*
   Indicates verbal passives.
   (a) The womble was defeated in a few minutes
   (b) *the defeated in a few minutes womble
   (c) *The womble appears defeated in a few minutes

2. *Rationale clauses* (in order to)
   Indicates verbal passives.
   (a) The money was stolen to pay the bills.
   (b) *the stolen to pay the bills money
   (c) *The money looks stolen to pay the bills

3. *Manner adverbials*
   Indicates verbal passives.
   (a) The womble was defeated quickly.
   (b) *the defeated quickly womble
   (c) *The womble looks defeated quickly.

4. *Instrument phrases*
   Indicates verbal passives.
   (a) The womble was defeated with a sturdy baseball bat.
   (b) *the defeated with a baseball bat womble
   (c) *The womble looks defeated with a sturdy baseball bat.

5. *by phrases*
   Indicates verbal passives.
   (a) The womble was defeated by a gang of chavs.
   (b) *the defeated by a gang of chavs womble
   (c) *The womble looks defeated by a gang of chavs.

6. *Eventive verbs*
   Some verbs are only acceptable in eventive(non-adjectival) contexts.
   (a) The womble was followed wherever he went.
   (b) *the followed womble
   (c) *The womble looks followed.

7. *Idiom chunks*
   Some idioms are only acceptable in eventive(non-adjectival) contexts.
   (a) Not much headway was made today.
   (b) *Not much headway appears made today.

8. *Raising verbs*
   Passivized raising verbs are only acceptable in eventive(non-adjectival) contexts.
   (a) Uncle Bulgaria was believed to have fled the country.
   (b) *the believed to have fled the country womble
   (c) *Uncle Bulgaria seems believed to have fled the country.

9. *Progressive passive*
   Indicates verbal passives.

	Duration adverbials	Rationale clauses	Manner adverbials	Instrument phrases	*by* phrases	Eventive verbs	Idiom chunks	Raising verbs	Progressive passives	*un*-prefixing
Verbal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Adjectival	✗	✗	✗	✗	✗	xmark	✗	✗	✗	✓

Table 1: Table of diagnostics for verbal and adjectival passives.

(a) The womble was being badly defeated.
(b) *the being badly defeated womble
(c) *The womble looks being badly defeated.

10. *un-prefixing*
Indicates adjectival passives.

(a) *Uncle Bulgaria was unfollowed.
(b) the undefeated womble
(c) Uncle Bulgaria looks undefeated.

See Table 1 for a quick summary of the behavior of the diagnostic tests.

In cases where the adjectival and passive readings have no differences in meaning, it does not really matter which annotation is given since we should be able to draw the same inferences from both.

10 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.

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
 (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
 (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))
```

(j) "He might faint"

```
(he.pro ((pres might.aux-s) faint.v))
```

(k) "He knew that he might faint"

```
(he.pro ((past know.v) (that (he.pro ((past might.aux-s) faint.v)))))
```

(l) "I do appreciate it"

```
(i.pro ((pres do.aux-v) (appreciate.v it.pro)))
```

(m) "He didn't sleep"

```
(he.pro ((past do.aux-s) not sleep.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). Here is an exhaustive enumeration of modal auxiliaries and how they are annotated.

can – `(pres can.aux-v)` if it means something like "presently able to (or permitted 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"

– `(pres must.aux-v)` for present obligation
"A cadet must obey orders"

will – `(pres will.aux-s)` 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 – `(cf will.aux-s)` if it refers to a conditional possibility in counterfactual construction
"I would go to Mars {if I were an astronaut, if I were to be offered the chance to go}"

– `(past will.aux-s)` 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"

– `(pres would.aux-s)` for present expectation (especially in questions)
"Would you take him with you?"

shall – `(pres shall.aux-s)` 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 ... (cf should.aux-s)` for a present conditional possibility
"I'm ready for that, should it happen" (i.e., "if it should happen")

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

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

– `(pres ought_to.aux-v)` for a present obligation [* but see comment below]
"He ought to study harder"

– `(past ought_to.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-to.aux-v)` for a presently scheduled or mandated action
"He is to appear in court tomorrow"

– `(pres be.aux-v)` transparent operator for passives as necessary
"Is Bob liked by his coworkers?"

was – `(past be-to.aux-s)` future in the past or event scheduled/mandated in the past
"Babbage was never to succeed in building the Analytical Engine"

– `(cf be-to.aux-s)` same as above but in a counterfactual context
"If he were to leave, the department would collapse"

– `(past be.aux-v)` transparent operator for passives as necessary
"Was Bob liked by his coworkers?"

Progressive and 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"

better – `(pres better.aux-s)` for a present requirement
"Someone better warn him"
(Note the scope ambiguity – that's why we want aux-s, not aux-v)

had best – `(pres had_best.aux-s)` for a present requirement
"Someone had best warn him"
(Note the scope ambiguity – that's why we want aux-s, not aux-v)

– `(past had_best.aux-s)` for a past requirement
"I realized that someone had best warn John"

Also, "used to", and "have to" are sometimes regarded as modal auxiliaries. However, we treat these as simple verbs that take kinds of action arguments (in the form of infinitives). This is because both can occur under other auxiliaries or aspectual operators, "I had used to always have a coffee after lunch" and "I will have to think about it". As a word of caution, the words "used to" can be interpreted as many different ways depending on subtle syntactic contexts. Please take a look at §29.1 when encountering a sentence with "used to" in it to be sure that the proper annotation is used.

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.

11 Aspect Annotation (Extension Over Time)

Aspect is generally captured in our annotations by operators reflecting perfect "have" and progressive "be -ing", as well as by the lexical entries in our annotations (e.g., *daily*) and less frequently as multi-word modifier phrases (e.g., *every day*, *for an hour*). Perfect and progressive tense are marked morpho-syntactically in English and require special treatment, which we describe here. The other forms are all treated in the same manner as other modifiers and we refer to that section, §13, for details.

11.1 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 `prog` and the tense operator. The last example is interesting because there are two copulas in the surface form – one to capture the tense and another to capture the progressive aspect. For clarity, here are the syntactic markings for the perfect and progressive aspects.

Perfect: *have* + *<past participle>*

Progressive: *be* + *<-ing verb>*

Please look at the examples above to verify this. 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.

(h) "John is under arrest and is being questioned"

```
(|John| (((pres be.v) (under.p (k arrest.n))) and.cc
         ((pres prog) (pasv question.v))))
```

12 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.a (of.p-arg *h)))))
```

(b) "About that, I know nothing"

```
(sub (about.p that.pro)
     (I.pro ((pres know.v) (no.d (n+preds thing.n *h)))))
```

(c) "Fight our fears, we must"

```
(sub (fight.v (our.d (plur (fear-of.n *s)))) (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. 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
             (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 Love.n)
 ((pres be.v) not
  (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 used an operator `rep` which is very similar to `sub` but in the other direction. See §16.3 for details.

13 Modifiers

Modifiers are operators that map predicates to predicates or sentences to sentences. They correspond closely with the syntactic class of adverbs. First we'll look at single word modifiers of verb phrases. An important restriction is that *modifiers cannot modify other modifiers directly*.

(a) "Jim is running quickly"

```
(|Jim| ((pres prog) (run.v quickly.adv-a)))
```

(b) "John saw Mary yesterday"

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

(c) "Mary undoubtedly spoke up"

```
(|Mary| undoubtedly.adv-s (past speak_up.v))
```

(d) "John sees Mary regularly"

```
(|John| ((pres see.v) |Mary| regularly.adv-f))
```

We distinguish between four different types of verb phrase modifiers: *action* modifiers, *event* modifiers, *sentence* modifiers, and *frequency* modifiers. These correspond to the suffix tags `adv-a`, `adv-e`, `adv-s`, and `adv-f`, respectively. *Action* modifiers map an action described by a verb phrase to a new action often describing a manner or a purpose (e.g., *ran quickly, run to catch the bus*). *Event* modifiers add some information about the event described by the sentence such as the time or location (e.g. *in the forest, along the river, at noon*). *Sentence* modifiers comment on the sentence, but do not modify its meaning, such as writer commentary (e.g., *surprisingly*). *Frequency* modifiers specify repetitive occurrence of the type of event described by the sentence they modify (e.g. *daily*, *regularly*, *every week*). Below are some additional examples of the different types.

(e) "John politely greeted Mary"

```
(|John| (politely.adv-a (past greet.v) |Mary|))
```

(f) "Mary spoke up confidently"

```
(|Mary| ((past speak_up.v) confidently.adv-a))
```

(g) "The next meeting will probably be canceled"

```
((the.d (next.a meeting.n)) ((pres will.aux-s) 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))
```

Notice that modifiers are supplied at the same bracketing level as the verb and its arguments. This allows us to handle various interleavings of arguments and modifiers that can occur in English. Strictly speaking, `adv-e`, `adv-s`, and `adv-f` operate over sentences and `adv-a` operates over complete verb phrases. This can be handled in post by lifting the sentence-level operators above the sentence and action modifiers above the verb phrase within which they occur.

Additionally, adjective and noun modifiers have their own types, `mod-a` and `mod-n`, respectively.

(m) "I am very happy"

```
(i.pro ((pres be.v) (very.mod-a happy.a)))
```

(n) "This is a fake diamond"

```
(this.pro ((pres be.v) (= (a.d (fake.mod-n diamond.n)))))
```

13.1 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 §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.

1. "I spilled a burning hot melting pot"

```
(I.pro ((past spill.v) (a.d ((burning.a hot.a) (melting.n pot.n)))))
```

13.2 Predicate Complements vs Predicate Modifiers

Similar to the subtle distinction we need to make between predicate and object complements (see §7) we also need to distinguish predicate complements from predicate modifiers which can look deceivingly similar.

For example, "lost in thought" in "I sat lost in thought" seems semantically similar to "I remained lost in thought" in that "lost in thought" gets applied to the subject alongside the main verb. However, in the former case "lost in thought" is optional for this interpretation of "sat". That is, "sat" has the same meaning in "I sat" vs "I sat lost in thought". This is in contrast to "I remain" vs "I remain lost in thought". "remain" has distinct interpretations

in the two cases indicating that rather than "lost in thought" modifying the main verb, it is triggering a new sense with a predicate argument. This is the same for other predicate complement/argument verbs: "He looks" vs "He looks tired"; *"That seems" vs "That seems unlikely"

We annotate predicate modifiers by wrapping the predicate in the appropriate `adv-*` type-shifter. It turns out that when adverbs are further resolved into Episodic Logic, their meaning is, in part, a concurrent predicate application of the appropriate element (the subject for `adv-a`, the event for `adv-e`, etc.) so the semantic intuition that these predicates are also being applied to the subject is captured in this way.

(a) "I sat lost in thought"

```
(I.pro ((past sit.v) (adv-a (lost.a (adv-e (in.p (k thought.n)))))))
```

(b) "I fell asleep contented"

```
(I.pro ((past fall.v) asleep.a (adv-a contented.a)))
```

(c) "I woke up refreshed"

```
(I.pro ((past wake_up.v) (adv-a refreshed.a)))
```

(d) "I woke up in a state of confusion"

```
(I.pro ((past wake_up.v)
        (adv-a (in.p (a.d (n+preds state.n (of.p (k confusion.n))))))))
```

(e) "I walked away feeling good"

```
(I.pro ((past walk.v) away.adv-a (adv-a (feel.v good.a))))
```

(f) "I returned home a changed man"

```
(I.pro ((past return.v) (adv-a ({to}.p (k home.n)))
                        (adv-a (= (a.d (changed.a man.n))))))
```

(g) "I arrived at the meeting well prepared"

```
(I.pro ((past arrive.v) (adv-e (at.p (the.d meeting.n)))
                        (adv-a (well.mod-a prepared.a))))
```

13.3 Modifying phrases (adverbials)

ULF 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 (plur glass.n))))))
```

(b) "Wycliffe played the piece with great passion"

```
(|Wycliffe| ((past play.v) (the.d piece.n)
                           (adv-a (with.p (k (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) "I slept poorly yesterday"

```
(I.pro ((past sleep.v) poorly.adv-a yesterday.adv-e))
```

(e) "She left at noon last Friday"

```
(She.pro ((past leave.v)
          (adv-e (at.p noon.pro))
          (adv-e ({during}.p ({the}.d (last.a |Friday|.n))))))
```

(f) "She will leave on Friday"

```
(She.pro ((pres will.aux-s) (leave.v (adv-e (on.p |Friday|)))))
```

(g) "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)))
```

(h) "Most likely, John went to the store"

```
((adv-s (most.mod-a likely.a))
 (|John| ((past go.v) (to.p-arg (the.d store.n)))))
```

(i) "Eve eavesdrops on her friends, usually intentionally or knowingly"

```
(|Eve|
 ((pres eavesdrop.v) (on.p-arg (her.d (plur (friend-of.n *s))))
                     (usually.adv-f (intentionally.adv-a or.cc
                                     knowingly.adv-a))))
```

(j) "Suddenly {,} she left"

```
(Suddenly.adv-e (she.pro (past leave.v)))
```

(k) "Sullenly {,} she left"

```
(sub sullenly.adv-a (she.pro ((past leave.v) *h)))
```

One might argue that *yesterday* and *today* in items (d) and (g) should really be treated as pronouns, in view of examples like "Yesterday was a good day". This might lead us to expand `yesterday.adv-e` to `(adv-e ({during}.p yesterday.pro))`, where `{during}.p` is a covert constituent. But to keep annotations as simple as possible, we allow both `yesterday.adv-e` and `yesterday.pro`.

Names of weekdays, months, etc., present a similar predicament. We do seem to need covert constituents in representing *last Friday* in item (e), namely `{during}.p`, `{the.d}`, and *Friday* needs to be treated as a name-like nominal predicate, `|Friday|.n`. (Such name-related predicates are further discussed in section 18.) But in item (f), *Friday* seems to function simply as a name. We could express this occurrence by expanding `|Friday|` into ({the}.d ({next}.a |Friday|.n)), in order to keep the meaning of *Friday* unambiguous. But again, we opt instead for simplicity of the ULF, by allowing both a nominal-predicate version and a version as a proper name. We would also employ the proper name, `|Friday|`, in a generic use such as "Friday is my favorite day of the week", rather than forming a kind, `(k |Friday|.n)`.

Note as well the contrast between items (j) and (k). In item (k), the initial manner adverb is treated as topicalized, to facilitate its "lowering" to the VP level in postprocessing.

Adverbial modifiers also seem to be able to form from nominal predicates, particularly distance related ones, e.g. "walked a great distance". These require special note because unlike the other predicates discussed so far, these predicate don't seem to modify the meaning by applying to the subject or the event as a whole. That is, "I walked a great distance" means neither that I am a great distance or that the event described is a great distance. Rather, there seems to be an implicit prepositional relation. Below are examples of how we handle this.

(l) "I walked a great distance"

```
(I.pro ((past walk.v) (adv-a ({for}.p (a.d (great.a distance.n))))))
```

(m) "I walked ten miles"

```
(I.pro ((past walk.v) (adv-a ({for}.p (ten.d (plur mile.n))))))
```

(n) "I threw the discus 100 meters"

```
(I.pro ((past throw.v) (the.d discus.n)
                       (adv-a ({for}.p (100.d (plur meter.n))))))
```

(o) "The temperature rose 20 degrees"

```
((The.d temperature.n) ((past rise.v)
                        (adv-a ({by}.p (ds temperature "20 degrees")))))
```

(p) "The fundraiser fell $1000 short"

```
((The.d fundraiser.n) ((past fall.v)
                       (adv-a ((adv-a ({by}.p (ds currency "$1000")))
                               (short-of.a {ref}.pro)))))
```

The `ds` operator is for capturing domain specific grammars (e.g. time, standard of measurement, currencies, addresses, etc.) which don't necessarily follow basic English grammar but have their specific rules for interpreting a complex phrase. Please refer to §24 for details.

13.4 Generalized Sentential Adverbial Modification

So far we've discussed how predicates can modify other predicates (i.e. `.v`,`.n`,`.a` modifying each other), using adverbs to modify actions/predicates (e.g. `adv-a`), and using adverbs to modify the event/proposition (e.g. `adv-e`,`adv-s`). The first two types compose readily, with some implicit type-shifters and the third case composes readily once the modifier is lifted to the event or proposition level. However, there are some cases where an adverb that semantically acts at a event or proposition level seems to modify a predicate:

frequently happy, *briefly happy*, *surprisingly happy*

We need to capture this in a way that distinguishes it from an actual sentence-level modification. Items (1) and (2) demonstrates this ambiguity.

(1) *The surprisingly happy man went home*

(2) *The happy man surprisingly went home*

Notice that the meanings of these two sentences are undeniably different. In item (1) the fact that the man is happy is surprising whereas in item (2) the fact that he went home is surprising. In that way, in the item (1) *surprising* is modifying *happy*.

In order to capture a local adverb application we simply scope the adverb *only* around the object that it modifies. This means that sentence-level adverbs that act on the sentence in which they're embedded must be supplied flat (i.e. at the same bracketing level as the predicate's other arguments and modifiers). Semantically the adverbs are still sentence-level so they must be expanded locally. Please see §A.9 for details on this expansion. Let's look at some examples.

(a) "I was surprisingly happy"

```
(I.pro ((past be.v) (surprisingly.adv-s happy.a)))
```

(b) "I was, surprisingly, happy"

```
(I.pro ((past be.v) surprisingly.adv-s happy.a))
```

(c) "The surprisingly happy man went home"

```
((The.d ((surprisingly.adv-s happy.a) man.n)) ((past go.v) (k home.n)))
```

(d) "The happy man surprisingly went home"

```
((The.d (happy.a man.n)) surprisingly.adv-s ((past go.v) (k home.n)))
```

(e) "I fell asleep, briefly forgetting my anxiety"

```
(I.pro (((past fall.v) asleep.a)
        (adv-a (briefly.adv-e (forget.v (my.d anxiety.n))))))
```

(f) "I have become a frequently returning member"

```
(I.pro ((pres perf)
        (become.v (a.d ((adv-a (frequently.adv-f return.v))
                        (member-of.n *ref))))))
```

In items (b) and (d) the sentence adverbial is supplied at a bracketing with at least two other elements so they will be interpreted as lifting to the sentence-level. The related items (a) and (c) are modifying the adjective meaning so they are scoped singly around the modified predicates. Items (e) and (f) show that this can be done with `adv-e` and `adv-f` as well and that this can be used to modify verbs as well as adjectives.

13.5 Verb-phrase adverbials

The examples of adverbials above do not involve verbs, but many adverbials are formed from verb phrases. Verb phrase adverbials in English 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). The bracketing practices are exactly the same. Here are a few examples.

(a) "The quarterback walked away, limping noticeably"

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

(b) "Limping noticeably, the quarterback walked away"

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

(c) "Considering his inexperience, he did very well"

```
((adv-s (consider.v (his.d inexperience.n)))
 (he.pro ((past do.v) (adv-a (very.mod-a good.a)))))
```

(d) "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)))))))
```

(e) "To put it bluntly, he failed"

```
((adv-s ({for}.p (to (put.v it.pro bluntly.adv-a))))
 (he.pro (past fail.v)))
```

In item (c), we have treated "considering his inexperience" much as we would have treated "despite his inexperience" or "in light of his inexperience", though some might want to expand the sentence meaning to something like "if I consider his inexperience, I conclude that he 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.mod-a` needs to operate on a predicate. ULF does not have modifiers that modify other modifiers directly.

In item (d), we have represented the purpose adverbial "to catch the ball" by introducing a covert preposition `{for}.p` that can take the action type `(to (catch.v (the.d ball.n)))` as its complement. (We can read the covert `{for}.p` as *for-purpose*.) In item (e), the adverbial comments on the sentence as a whole, hence the `adv-s` operator; but again the *to*-infinitive is used to express a purpose. This is a common pattern for purpose clauses that are supplied with a infinitive.

13.6 Clausal adverbials

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. Clausal adverbials are always mediated by a subordinating coordinator (e.g. *although*, *while*, *so*, some instances of *because*) which are annotated with a `.ps` extension. This stands for *sentential preposition* since in ULF we think of them as prepositions in that they relate two things, but which operate over sentences instead of objects.

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

The sentential prepositions are always annotated as curried functions—there is a separate bracketing for each sentence argument. Similar to sentential modifiers, the clausal adverb (the sentential preposition with one of the arguments supplied) can appear anywhere in the second sentence argument position. In post these will be lifted to the sentence level. Item (a) shows the canonical (and post-processed) form of clausal adverbial modifiers. Items (b) and (c) show how the clausal adverb can appear in other locations of the second argument sentence.

We distinguish temporal and atemporal *when* in items (d) and (e), coding the atemporal version as `when-if.ps` in view of its semantic similarity to *if*.

13.6.1 Shortened clausal adverbials

When the subject of the main verb corefers to the subject of a clausal adverbial, the subject of the clause may be omitted, along with the copula, e.g. "Mangoes are delicious when they are ripe" → "Mangoes are delicious when ripe". In these shortened clausal adverbials, the appropriate referential pronoun, tense, and copula are added as appropriate and the clausal adverb should be scoped within the verb phrase to ensure compositional access to the subject.

(a) "Mangoes are delicious when ripe"

```
((k (plur Mango.n)) ((pres be.v) delicious.a
                     (when.ps ({they}.pro ((pres {be}.v) ripe.a)))))
```

(b) "I fell asleep while contented"

```
(i.pro ((past fall.v) asleep.a
        (while.ps ({i}.pro ((past {be}.v) contented.a)))))
```

(c) "The party sat while lost in thought"

```
((The.d party.n) ((past sit.v)
          (while.ps ({they}.pro ((past {be}.v)
                                 (lost.a (adv-e (in.p (k thought.n)))))))))
```

(d) "John woke up while in a state of confusion"

```
(|John| ((past wake_up.v)
         (while.ps ({he}.pro ((past {be}.v)
                              (in.p (a.d (n+preds state.n
                                                  (of.p (k confusion.n))))))))))
```

(e) "I sat while thinking"

```
(I.pro ((past sit.v)
        (while.ps ({I}.pro ((past prog) think.v)))))
```

(f) "The quarterback stumbled, while backing up"

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

You may have noticed that many of these examples are similar to the predicate modifier examples shown in §13.2. Semantically, these *are* similar. Once the coreference between the two subjects are resolved the semantic analysis of these clausal adverbs closely parallel those of predicate modifiers. These examples are analyzed in terms of shortened clausal adverbs rather than lengthened predicate modifiers because the prepositions in these examples parallel those of clausal adverbs.

13.7 Post-nominal Modifiers

For post-nominal modifiers, we introduce macros `n+preds` and `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:

```
(n+preds [noun (incl. any arguments)] [predicate 1] [predicate 2] ...)
(np+preds [noun phrase] [predicate 1] [predicate 2] ...)
```

See examples below.

(a) "A table with three legs"

```
(a.d (n+preds table.n (with.p ((nquan three.a) (plur leg.n)))))
```

(b) "The explosion in the city"

```
(the.d (n+preds explosion.n (in.p (the.d city.n))))
```

(c) "The hawk circling overhead"

```
(the.d (n+pred hawk.n (circle.v (adv-a overhead.a))))
```

(d) "John, totally exhausted, ..."

```
(np+preds |John| (totally.mod-a exhausted.a))
```

(e) "John, feeling tired, ..."

```
(np+preds |John| (feel.v tired.a))
```

(f) "The lunch today was good"

```
((The.d (n+preds lunch.n today.a)) ((past be.v) good.a))
```

Note that we use `n+preds` for restrictive postmodifiers – ones that further limit what entities the noun phrase as a whole can refer to; while we use `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.

The interpretation of "today" in example (f) may seem strange since "today" is generally thought of as an adverb or a pronoun. In the final meaning, this will be reflected in our representation as well. For deictic temporal terms such as "today" we allow adjectival, adverbial, and pronoun meanings which all rely on the pronoun reference at its core. `today.a` is defined as `({during}.p today.pro)`. §13.3 discusses the adverbial-pronoun relation. §15.1 describes a generalized form of this post-modification for cases that interleave arguments and modifiers.

14 Relative clauses

Relative clauses also postmodify nouns or noun phrases, but they often involve gaps, and thus make use of the `sub` macro introduced ealier. In the following examples, only item (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, ..."

```
(np+preds |John| (who.rel ((pres be.v) lawyer.n)))
```

(b) "The manager whom you met"

```
(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, but `tht.rel` marks those that are not realized)

(e) "A man not of this world"

```
(A.d (n+preds man.n (sub tht.rel (not (*h (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 ((who.rel 's) 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 ((which.rel 's) (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|.n))
                                     (adv-e *h))))))
```

(m) "The couch whereon 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, the first argument of the `sub` macro must include a relative pronoun – `who.rel`, `that.rel`, `tht.rel`, or `which.rel`. Note further that in item (i) 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 coreferential 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 postprocessing, 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 postprocessing as well, writing them in ULF as `whose.dr`, `when.pr`, `where.pr`, `whereon.pr`, etc.

15 Derived Nominals

Unlike most nouns, nouns that are derived from verbs and adjectives can have post-nominally supplied arguments (with or without a preposition) as well as adverbs that work on the nominalized verb or adjective phrase rather than the outer sentence. These post-nominally supplied arguments can simply be supplied after the noun in a flat format.

(a) "The sale of the car to me by Sally for $400"

```
(the.d (sale.n (of.p-arg (the.d car.n))
               (to.p-arg me.pro)
               (by.p-arg |Sally|)
               (for.p-arg (ds currency "$400"))))
```

(b) "His belief that the Earth is flat"

```
(his.d (belief.n (that ((the.d |Earth|.n) ((pres be.v) flat.a)))))
```

For adverbs that act on the nominalized event `n+post` should be used to restrict the lifting of the adverb.

(c) "The sale yesterday"

```
(The.d (n+post sale.n yesterday.adv-e))
```

As described in the following subsection, §15.1, on generalized nominal post-modification, if post-nominally supplied arguments are mixed with predicates then `n+post` should be used.

15.1 Generalized Noun Post-modification/complementation (`n+post`)

It turns out that there are instances of post-nominal modifiers and arguments that are interleaved with each other, where the scoping of the modifiers need to be specified in relation to the arguments. This is similar to how in verb phrases arguments and adverbs can be interleaved with each other even if the adverb acts on the full verb phrase or even the sentence. Here is an example with noun post-modification.

"the similarity of Kepler 438b in many respects to our planet"

Notice that since "Kepler 438b" and "our planet" are arguments it would be inappropriate to supply them as arguments via `n+preds`. First of all, they're not predicates. Moreover, even if they were to be predicate arguments, the `n+preds` expansion would apply that predicate rather than supply it as a predicate argument.

We introduce the macro `n+post`, a generalization of `n+preds` to handle this. `n+post` takes a noun followed by one or more post-nominal predicates, terms, prepositionally marked arguments, or adverbs. Predicates are handled in the same way as `n+preds`, terms and prepositionally marked arguments are supplied as arguments to the noun, and adverbs are made to modify the nominalized verb.

(a) "the similarity of Kepler 438b in many respects to our planet"

```
(the.d (n+post similarity.n
               (of.p-arg |Kelper 438b|)
               (in.p (many.d (plur respect.n)))
               (to.p-arg (our.d planet.n))))
```

(b) "the promise by John yesterday to tidy up his room"

```
(the.d (n+post promise.n
               (by.p-arg |John|)
               yesterday.adv-e
               (to (tidy.v up.adv-a (his.d room.n)))))
```

(c) "the idea going around that vaccinations cause autism"

```
(the.d (n+post idea.n
               (go.v around.adv-a)
               (= (that ((k (plur vaccination.n))
                         ((pres cause.v) (k autism.n)))))))
```

Notice that not all term arguments are necessarily marked as an argument with a preposition, such as the last argument of `n+post` in example (b). It's important to note that only nouns derived from verbs or adjectives can take arguments and adverbs. Thus plain nouns that are not derived from verb or adjectives will not require `n+post`. Rather `n+preds` can be used instead. See the previous section on derived nouns, Section 15 for more discussion on this topic.

Another tricky issue is that even sentence/event-level adverbs are handled specially by `n+post`. That is, they are inserted into the modified event. This is to allow sentences such as "I just heard today about the promise by John yesterday to tidy up his room". So the scoping is important in sentences such as "I heard the promise by John yesterday" whether "yesterday" is scoping over the "promise" event or the "heard" event.

Though rare, there are also cases of post-nominally supplied predicate arguments, such as the examples shown below. To our knowledge, these will always be accompanied by an argument marking preposition so they can be disambiguated from modifiers of noun. For example, "as a chicken" is supplying the predicate `chicken.n` as an argument to `disguise.n` and `brilliant.n` is a predicate argument to `view.n`.

(d) "his disguise as a chicken for the party"

```
(his.d (n+post disguise.n
               (as.p-arg chicken.n)
               (for.p (the.d party.n))))
```

(e) "the view of him at this moment as brilliant"

```
(the.d (n+post view.n
               (of.p-arg him.pro)
               (at.p (this.d moment.n))
               (as.p-arg brilliant.n)))
```

16 It-clefts, extraposition, and *there*-sentences

It-clefts, it-extraposition, and *there*-sentences are are all constructions where a portion of the sentence is right-shifted in the sentence for reduced processing load. Each of these will be annotated with specific mechanisms that reflect the correspondence between their syntactic realizations and their semantic meaning.

The difference between clefts and extrapositions are not consistent in the linguistic literature so we will define what we mean in this document. We restrict 'clefts' to constructions starting with *it* + *be* and ends in a relative clause indicating the semantic equivalent of the predicate or NP following *it* + *be* filling the relative clause gap[5]. We restrict 'extraposition'

[5]There are some more subtle issues related to the semantic of *it-clefts* which are discussed in Appendix A.5.

to constructions what start with "it" and end with a nominalized action, event, or proposition which corefers to "it". For other right-shifted meanings, we will simply say they are right-shifted or rightwardly displaced. These can be rewritten by moving the rightshifted clause to the appropriate location in the sentence. Here are examples.

- **It-cleft**

  It was Mary who arrived first

  It was Rome that I went to

- **It-extraposition**

  It's surprising that Mary arrived first

  It's surprising for Mary to arrive first

- **Rightward Displacement**

  Someone left a message whom we don't know

  How frustrated are they with their kids?

16.1 It-clefts

It-clefts can be interpreted as a paraphrase of a sentence which adds emphasis to a particular part of the sentence by rightshifting the rest of the sentence meaning in a relative clause[6]. To keep the annotation process as simple as possible, we simply introduce a special sense of *'it'* for clefts, `it-cleft.pro`, which allows proper analysis of the following `be.v` statement with two curried arguments. That is, the right-shifted relative clause will be supplied as an additional argument to `be.v` in this special construction. Here are examples followed by a discussion of some tricky cases or aspects and the translation process.

(a) "It was Mary who arrived first"

```
(It-cleft.pro (((past be.v) |Mary|)
               (who.rel ((past arrive.v) first.adv-a))))
```

(b) "It was Mary he gave the book to"

```
(It-cleft.pro (((past be.v) |Mary|)
               (sub tht.rel
                    (he.pro ((past give.v) (the.d book.n) (to.p-arg *h))))))
```

(c) "It was Rome that he traveled to"

```
(It-cleft.pro (((past be.v) |Rome|)
               (sub that.rel (he.pro ((past travel.v) (to.p-arg *h))))))
```

(d) "It was to Rome that he traveled"

```
(It-cleft.pro (((past be.v) (to.p-arg |Rome|))
               (sub that.rel (he.pro ((past travel.v) *h)))))
```

[6]The it-cleft construction also adds a presupposition that the relative clause is satisfied by something. For example, *"It might have been Mary who went home"* presupposes that *someone* went home.

(e) "It is Jaime for whom we are looking"

```
(It-cleft.pro (((pres be.v) |Jaime|)
               (sub (for.p-arg whom.rel)
                    (we.pro ((pres prog) (look.v *h))))))
```

(f) "It was because he was ill (that) we decided to return."

```
(It-cleft.pro (((past be.v) (because.ps (he.pro ((past be.v) ill.a))))
               (that.rel (we.pro ((past decide.v) (to return.v))))))
```

(g) "It was in September that he first found out about it."

```
(It-cleft.pro
 (((past be.v) (adv-e (in.p |September|)))
  (sub that.rel
       (he.pro (first.adv-a (past find_out.v) (about.p-arg it.pro) *h)))))
```

(h) "It was on foot that he went there."

```
(It-cleft.pro (((past be.v) (adv-a (on.p (k foot.n))))
               (sub that.rel
                    (he.pro ((past go.v) there.adv-e *h)))))
```

(i) "It was greedily and speedily that Homer Simpson drank his beer."

```
(It-cleft.pro (((past be.v) (greedily.adv-a and.cc speedily.adv-a))
               (sub that.rel
                    (|Homer Simpson| ((past drink.v) (his.d beer.n) *h)))))
```

(j) "It is to address a far-reaching problem that Oxfam is launching this campaign"

```
(It-cleft.pro
 (((pres be.v) (to (address.v (a.d (far-reaching.a problem.n)))))
  (sub that.rel
       (|Oxfam| ((pres prog) (launch.v (this.d campaign.n)
                 (adv-a ({for}.p *h))))))))
```

(k) "It was because she was so lonely all the time that she decided to move out."

```
(It-cleft.pro
 (((past be.v)
   (because.ps (she.pro ((past be.v) (so.mod-a lonely.a)
                         (adv-e ({during}.p (all.d ({of}.p (the.d time.n)))))))))
  (that.rel (she.pro ((past decide.v) (to (move.v out.adv-a)))))))
```

(l) "It could be Mary he gave the book to"

```
(It-cleft.pro
 ((past can.aux-s) ((be.v |Mary|)
                    (sub tht.rel
                         (he.pro ((past give.v) (the.d book.n) (to.p-arg *h)))))))
```

(m) "Is it the knave that stole the tarts?"

```
(((pres be.v) it-cleft.pro
  (the.d knave.n) (that.rel ((past steal.v) (the.d (plur tart.n))))) ?)
```

(n) "It was conceivably but not very likely me who fell asleep"

```
(It-cleft.pro
 (((past be.v)
   (conceivably.adv-s but.cc (adv-s (not (very.mod-a likely.a))))
   me.pro)
  (who.rel ((past fall.v) asleep.a))))
```

This construction has a close correspondence to topicalization in general. Every one of these examples can be rewritten by removing the "*it* + *be*" and relativizer and undoing any pied-piping without modification of the truth-value meaning. It does, of course, reduce the emphasis of the topicalized phrase and result in awkward sentences. For example, *"It was Rome that he traveled to"* can be rewritten as *"Rome, he traveled to"* and *"It is Jaime for whom we are looking"* as *"Jaime, we are looking for"*.

Item (i) shows that this it-cleft construction needs to be resolved before other syntactic expansion. In this case, the coordination of two adverbs needs to be factored, but only after resolving the slot of the coordinated adverb. Furthermore, it-clefts in general do not correctly resolve semantic types in the surface form.

Items (g) and (h) display the correct annotation for a very subtle distinction made in it-clefts. Notice that in both annotations the topicalized phrase is the full adverbial (i.e. including the `adv-*` operator). This is an important distinction as argument marking prepositions and adverbial prepositions seem much preferred to direct predicate arguments in it-clefts. See the following examples as evidence.

?"It was in love that he was"

"It was on the bookshelf {?that is was, that it lay}"

?"It was out of order that the question was declared"

Mapping from this cleft to non-cleft construction details in the Appendix A.5.

16.2 It-extraposition

This is a phenomenon that at first glance looks very similar to *it-clefts*, but have distinct semantic effects. Whereas in *it-clefts* the "*it* + *be*" simply indicates a reordering of the remainder of the sentence, in *it-extraposition* the *"it"* corefers to the reified sentence or action that is supplied in a right-shifted location. The structural similarities are reflected in similar annotation methods. Here we annotate *"it"* as `it-extra.pro` and the right-shifted argument is supplied as an additional *curried* argument to the verb in which `it-extra.pro` itself participates as an argument.

(a) "It's surprising that Mary arrived first"

```
(It-extra.pro (((pres be.v) surprising.a)
               (that (|Mary| ((past arrive.v) first.adv-a)))))
```

(b) "He saw to it that Mary would get the book"

```
(He.pro (((past see.v) (to.p-arg it-extra.pro))
         (that (|Mary| ((past will.aux-s) (get.v (the.d book.n)))))))
```

(c) "It was frustrating that I burned the potatoes"

```
(It-extra.pro (((past be.v) frustrating.a)
               (that (I.pro ((past burn.v) (the.d (plur potato.n)))))))
```

(d) "Did it surprise you that that happened?"

```
(((past do.aux-s) it-extra.pro
  (surprise.v you.pro (that (that.pro (past happen.v))))) ?)
```

(e) "We suggested it to them that we leave today"

```
(We.pro (((past suggest.v) it-extra.pro (adv-a (to.p them.pro)))
         (that (we.pro ((pres leave.v) today.adv-e)))))
```

(f) "Nobody believes it for a second that Newt will get the nomination"

```
(Nobody.pro (((pres believe.v) it-extra.pro (adv-e (for.p (a.d second.n))))
             (that (|Newt| ((pres will.aux-s) (get.v (the.d nomination.n)))))))
```

(g) "It is unusual for John to sleep in"

```
(It-extra.pro (((pres be.v) unusual.a) (ke (|John| sleep_in.v))))
```

(h) "I find it hard to exercise everyday"

```
(I.pro (((pres find.v) it-extra.pro hard.a)
        (to (exercise.v everyday.adv-f))))
```

Notice that in all cases we get a grammatical (if awkward) sentence with the same meaning if we replace the *"it"* with the right-ward displaced argument. For example, *"It's surprising that Mary arrived first"* has the same meaning as *"That Mary arrived first is surprising"*. The *"it"* can appear as a non-subject argument as well, as shown in items (b), (e), (f) and (h). For these cases, we still supply the coreferring expression as an extra argument to the verb that *"it"* participates as an argument.

This phenomenon can occur in the context of a question, including auxiliary inversion as shown in item (d). And finally, the coreferring expression is often a proposition, as shown in the first handful of examples, but can also be a kind of event or kind of action as shown by items (g) and (h), respectively.

16.3 General Rightward Displacement

In addition to the more constrained *it-cleft* and *it-extraposition* phenomena, English allows rightward displacement of almost any clause in appropriate context. Since this truly is a case of movement, rather than restructured sentence meaning or coreference, we introduce the `rep`, or *replace* operator, which is the reverse of the `sub` operator, in that the marked location in the first argument is *replaced by* the second argument. For `rep` we will annotate the marked location with `*p`, for *placeholder*.

(a) "Someone left a message whom we don't know"

```
(rep ((Some.d (n+preds person.n *p)) ((past leave.v) (a.d message.n)))
     (sub whom.rel (we.pro ((pres do.aux-s) not (know.v *h)))))
```

(b) "Susan said something again that nobody expected"

```
(rep (|Susan| ((past say.v) (Some.d (n+preds thing.n *p)) again.adv-s))
     (sub that.rel (nobody.pro ((past expect.v) *h))))
```

(c) "Some guy was there with red hair"

```
(rep ((Some.d (n+preds guy.n *p)) ((past be.v) there.adv-e))
     (with.p (red.a hair.n)))
```

(d) "How frustrated are they with their kids?"

```
((rep (sub (How.adv-a (frustrated.a (adv-a *p))) ((pres be.v) they.pro *h))
      (with.p (their.d (plur kid.n)))) ?)
```

(e) "What actually happened that was so entertaining?"

```
((rep ((What.d *p) (actually.adv-s (past happen.v)))
      (that.rel ((past be.v) (so.mod-a entertaining.a)))) ?)
```

(f) "What do you think they did that upset everyone?"

```
((rep (sub (What.d *p) ((pres do.aux-s) you.pro
                        (think.v (tht (they.pro ((past do.v) *h))))))
      (that.rel ((pres upset.v) everyone.pro))) ?)
```

Please study items (d) to (f) to get comfortable with interactions between `sub`, `rep`, and question inversions. Please see the §A.6 for an explicit definition of `rep` and an example walkthrough of the macro expansion.

16.4 Existential *there*-sentences

This phenomenon is quite distinct from the others we have just discussed, so it is pretty easy to distinguish from the other cases. The treatment of existential *"there"* is superficially similar to "it-clefts" in that we specially interpret *"there"* and *"be"*, if present, and restructure the sentence to get the appropriate semantics.

We will always treat `there.pro` as 'existential-there', whereas all other forms of *"there"* are annotated as an adverb, `there.adv-e`, or an adjective, `there.a`. This is based on the observation that *"there"* generally cannot replace or be replaced by NPs. *"I went there"* cannot be changed to **"I went school"* or **"I went the store"* (*"I went home"* is an exception). Similarly, *"I go to the park"* cannot be changed to **"I go to there"*.

(a) "There is a tavern in the town"

```
(There.pro (((pres be.v) (a.d tavern.n)) (adv-e (in.p (the.d town.n))))
```

(b) "There exist two major variants"

```
(There.pro ((pres exist.v) (two.d (major.a (plur variant.n)))))
```

(c) "There occurred a strange incident"

```
(There.pro ((past occur.v) (a.d (strange.a incident.n))))
```

(d) "Is there a test today?"

```
(((pres be.v) there.pro (a.d test.n) today.adv-e) ?)
```

Notice that existential *"there"* can occur with verbs other than *"be"* (items (b) and (c)). Also, it can occur with an inverted question (item (d).

17 Questions

17.1 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 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 item (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" of items (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 `?`. More commonly, yes-no questions involve *subject-auxiliary inversion*.

(d) "Has Bob left?"

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

(e) "Is Bob well-liked by his co-workers?"

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

(f) "Isn't Bob going to leave?"

```
(((pres be.v) not |Bob| (going_to.v leave.v)) ?)
```

(g) "Didn't she eventually hire Bob?"

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

(h) "Will he have finished the assignment by that time?"

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

(i) "Was Mary politely handed the book by John?"

```
(((past be.aux-v) |Mary|
  (politely.adv-a (pasv hand.v) (the.d book.n) (by.p-arg |John|))) ?)
```

Note that "subject-auxiliary inversion" is a somewhat inaccurate term; for example, in items (d) and (e) 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 postprocessing will probably rearrange constituents into declarative-question-like form. For example, the ULF for (f) may be rearranged into

```
((|Bob| ((pres be.aux-v) not (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

```
(sub (adv-e (under.p (the.d ice)))
     ((k (new.a (deep.a sea.n) (plur creature.n)))
      ((pres pref) (pasv find.v) *h)).
```

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.

In the rare instance where a modifier or an argument must scope outside of the inverted verb or auxiliary, supply those modifiers and arguments flat following the remaining components of the verb phrase. For example, *"Was he happy holding the balloon?"* is annotated as

```
(((past be.v) he.pro happy.a (adv-a (hold.v (the.d balloon.n)))) ?)
```

In general the mapping rule is roughly as follows.

```
   ((V/AUX SUBJ REST EXTRA1 ... EXTRAn) ?)
⇒ ((SUBJ (V/AUX REST EXTRA1 ... EXTRAn)) ?)
```

17.2 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)) ?)
```

Note that we can also have multiple wh-constituents in a question.

(c) "Which sandwiches were ordered by which guests?"

```
(((which.d (plur sandwich.n))
  ((past (pasv order.v)) (by.p-arg (which.d (plur guest.n))))) ?)
```

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).

(d) "Whom did you invite?"

```
((sub who.pro ((past do.aux-s) you.pro (invite.v *h))) ?)
```

(e) "Why did you fix it?"

```
((sub Why.adv-s ((past do.aux-s) you.pro (fix.v it.pro *h))) ?)
```

(f) "With what did you fix it?"

```
((sub (adv-a (with.p what.pro))
      ((past do.aux-s) you.pro (fix.v it.pro *h))) ?)
```

(g) "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))))) ?)
```

(h) "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)))))) ?)
```

(i) "How smart is he?"

```
((sub (How.mod-a smart.a) ((pres be.v) he.pro *h)) ?)
```

(j) "How quickly can you say 'desserts' backward?

```
((sub (adv-a (how.mod-a quick.a))
      ((pres can.aux-v) you.pro (say.v (\" (plur dessert.n) \") backward.adv-a *h)))
?)
```

(k) "Which sandwiches did you give to which guests?"

```
((sub (which.d (plur sandwich.n))
      ((past do.aux-s) you.pro
       (give.v *h (to.p-arg (which.d (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 *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.

17.3 Lexical and Prepositional Wh-questions (.pq)

For preposition wh-questions that are lexical, we use the `.pq` extension. This includes `when.pq`, `where.pq`, and `how.pq`, which roughly map to *at time which*, *at location which*, and *by means of which*, respectively.

(a) "Where did you go?"

```
((sub Where.pq ((past do.aux-s) you.pro (go.v *h))) ?)
```

(b) "When will you arrive?"

```
((sub When.pq ((pres will.aux-s) you.pro (arrive.v *h))) ?)
```

(c) "How did you see me?"

```
((sub How.pq ((past do.aux-s) you.pro (see.v me.pro *h))) ?)
```

17.4 Reified questions

Recall that declarative sentences can be type-shifted to become individuals (and thus arguments of predicates) using reification operator operator `that` (and this very sentence is an example, containing a reified sentence as object argument of *recall*). Similarly, yes-no questions and wh-questions can be reified, using operators `whether` and `ans-to` respectively.

(a) "I don't know whether/if it will rain"

```
(I.pro ((pres do.aux-s) not
        (know.v (whether (it.pro ((pres will.aux-s) rain.v))))))
(I.pro ((pres do.aux-s) not
        (know.v (if (it.pro ((pres will.aux-s) rain.v))))))
```

(b) "I know what you did last summer"

```
(I.pro ((pres know.v)
        (ans-to (sub what.pro
                     (you.pro
                      ((past do.v) *h
                       (adv-e ({during}.p ({the}.d (last.a summer.n)))))))))))
```

Note that *if* would be interpreted as `whether` in item (a)—its use for question reification is quite different from its use as a conditional (subordinating) conjunction `if.ps`, as in *"I'll be surprised if it rains"*. `whether` and `if` are aliases of each other, similar to how `to` and `ka` are aliases. The annotation should use the symbol that corresponds to the word used in the sentence, defaulting to `whether` if it is implicit.

18 Names

Names must distinguish between *true* names and *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 *the* Delaware River for item (b) to be grammatical.

(a) Mary is beautiful [good!]

(b) Delaware River is beautiful [bad!]

True names are annotated with surrounding pipes `|_|`. Whitespace 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 → `|Mary|`
- John → `|John|`
- Three Mile Island → `|Three Mile Island|`
- The Hague → `|The Hague|`
- New York → `|New York|`

Predicate names are annotated with surrounding pipes and followed by the noun suffix `|_|.n`. Below are some examples of this annotation.

- Delaware River → `|Delaware River|.n`
- Eiffel Tower → `|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 §22).

```
((his.d name.n) ((pres be.v) (= (\" |John| \")))
((\" (k love.n) \") ((pres be.v) ((four.a letter.n) word.n)))
```

19 Numbers

Bare numerals like 0, 1, 2, 3, ..., without any extensions are regarded as *names* denoting numbers (which are abstract entities). These names are related to the corresponding predicates, more specifically, `zero.a` is equivalent to `(= 0)` (the property of being equal to 0), `five.a` is equivalent to `(= 5)` (the property of being equal to 5), etc. These can also be written as `0.a` or `5.a` to reflect the surface text. Numbers can also act as determiners, e.g. `5.d`, which is equivalent to `(nquan 5.a)`, which is itself equivalent to `(nquan (= 5))`. For examples and a discussion of numbers acting as determiners see §5.2.

These correspondences are important when annotating more complex examples so that we know how to appropriately modify the numerical logical entities or so that we know what forms are available. For example, *"almost"* is an adjective modifier, so when we have

a phrase like *"I ate almost five pizzas"* we know that we can modify the adjective form of *five* and construct the rest of the sentence appropriately, like the following.

```
(i.pro ((past eat.v) ((nquan (almost.mod-a five.a)) (plur pizza.n))))
```

Similarly, we know for phrases like *"The five dogs" five* can act as an adjective—like most determiners—to form `(the.d (five.a (plur dog.n)))`.

There are also cases where numbers are used as labels rather than to denote the number itself, e.g. *the 1990s* means the years labeled by the number 1990. So we use pipes to mark numbers as names separate from numbers themselves.

(a) The late 1990s

```
(the.d (late.a (|1990| (plur {year}.n))))
```

(b) His late 20s

```
(his.d (late.a (|20| (plur {year}.n))))
```

(c) 1990 was a great year

```
(|1990| ((past be.v) (= (a.d (great.a year.n)))))
```

(d) 20 years

```
(20.d (plur year.n))
```

Items (a) to (c) show examples of the number used as a label. See the difference against item (d) where *"20 years"* really refers to a particular multiplicity of years, rather than say, a label for certain years in relation to a particular person's age.

20 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"

```
(((the.d kindergarten.n) 's) (boisterous.a (plur child.n)))
```

(b) "My dogs are happy"

```
((My.d (plur dog.n)) ((pres be.v) happy.a))
```

(c) "The dogs are mine"

```
((The.d (plur dog.n)) ((pres be.v) mine.a))
```

(d) "That is John's dog"

```
(That.pro ((pres be.v) (= ((|John| 's) dog.n))))
```

Item (a) is the bare possessive phrase and item (b) uses possessive phrase as an argument, and where the determiner is a shorthand for the possessive *dogs posessed by me*. Item (c) shows a usage where a predicate contains the possessive meaning (*mine* means *possessed by me*). Item (d) shows a predicative use of the possessive where we wrap the possessive NP with `(= ..)` to turn it into a predicate. `'s` is a macro that expands into a specific usage of a more basic `poss-by` operator. §A.7 describes this in detail.

20.1 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.[7] Below are examples of annotations for these cases.

Possessives with Internal Relations

(e) "John's boisterous children"

```
((|John| 's) (boisterous.a (plur (child-of.n *s))))
```

(f) "My children are happy"

```
((My.d (plur (child-of.n *s))) ((pres be.v) happy.a))
```

(g) "The children are mine"

```
((The.d (plur (child-of.n *s))) ((pres be.v) mine.a))
```

(h) "That is John's child"

```
(That.pro ((pres be.v) (= ((|John| 's) (child-of.n *s)))))
```

(i) "The boisterous children of John"

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

Items (e) to (h) show examples of internal relations in a possessive phrase and correspond respectively to the basic cases of items (a) to (d). Notice that the `child-of.n` relation uses the `*s` anaphoric variable in this case, but not in item (a). This allows unambiguous location of the argument from the rest of the LF for item (e). In item (a) *"the kindergarten"* does not partake in the 'child-of' relation. Item (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 and 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))
```

(k) "John's mothers keep wandering away"

```
((((|John| 's) (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)))))
```

[7] The 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.

Item (j) shows an example where the sentence context pushes the predicate meaning away from the internal relation reading. Items (k) and (l) are ambiguous between an internal and external reading. *"mothers"* in item (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. The following 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"*).

Criteria 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.

20.2 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)) (when.ps (they.pro (pres (pasv strain.v)))))
```

(p) "I was surprised by the weight"

```
(I.pro ((past (pasv surprise.v)) (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 items (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.

20.3 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.

(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"*.

(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))`

Personal Pronoun	Possessive Determiner	Possessive Pronoun
I	my	mine
you	your	yours
he	his	his
she	her	hers
it	its	its
we	our	ours
they	their	theirs
one	one's	one's
NP (John)	NP's (John's)	NP's (John's)

Table 2: Listing of personal pronouns with corresponding possessive determiners and possessive pronouns.

20.4 Verbal Possession

Verbal possession (e.g. *"John has a dog"*) is annotated with `have.v`. The criteria for annotating relative predicates and internal/external variants are the same here.

(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))))
```

(x) "Bambi's father had antlers"

```
((((|Bambi| 's) (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)
          (her.d (audience-of.n *s))
          in_stitches.a))
```

Notice that in items (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 §11).

20.5 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 §A.7 for details). Possessive pronouns have a similar mapping. For reference, table 2 lists possessive determiners and pronouns in relation to the basic pronouns.

21 Punctuation

We only mark punctuation as needed to capture semantic content that is not represented elsewhere in the ULF. For example, periods marking the end of the sentence and commas marking subordinate clauses are ignored since the bracketing captures their semantic content. We have two sentence-level operators, `!` for imperatives and `?` for questions, which correspond, at least in part, to their surface-form meanings. Here are some examples.

(a) "She is happy."

```
(She.pro ((pres be.v) happy.a))
```

(b) "She is happy?"

```
((She.pro ((pres be.v) happy.a)) ?)
```

(c) "Go home!"

```
(({you}.pro ((pres go.v) (to.p-arg (k home.n)))) !)
```

(d) "John, go home!" (This imperative includes a vocative term – see §27)

```
((voc |John|) ({you}.pro ((pres go.v) (to.p-arg (k home.n)))) !)
```

(e) "I'm going home!"

```
(I.pro ((pres prog) (go.v (to.p-arg (k home.n)))))
```

(f) "You're falling asleep, go to bed(!)"

```
((You.pro ((pres prog) fall_asleep.v)) {so}.cc
 (({you}.pro ((pres go.v) (to.p-arg (k bed.n)))) !))
```

Notice all the commas and periods were dropped in these examples. Furthermore, the exclamation mark in item (e) is dropped since the exclamation mark is not acting as an imperative. Similarly, the exclamation mark in item (f) is optional, hence the parentheses, and both versions have the same annotation with the `!` imperative marking.

In item (c), we introduce an implicit argument of the listener with `{you}.pro`. In item (d), it might seem like *"John"* is the argument to the sentence, but this is actually a vocative as *"John"* in that sentence primarily funtions to address John. In item (f), notice that the imperative marker only operates on the inner sentence. Also there is an convert conjunction `{so}.cc`. It's easy to see that the sentence has the same meaning with or without *"so"*.

22 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.

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" has 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 in the following manner.

```
((\" (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))))
```

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, we have ways of annotating that, see §24, e.g. *"bonsoir, monsieur", was the sharp reply.*

```
((\" (ds fws "bonsoir, monsieur") \") ((past be.v) (= (the.d (sharp.a reply.n)))))
```

Here are several examples of mentioned quote annotations.

(a) ""Love" is a four-letter word"

```
((\" (k Love.n) \") ((pres be.v) (= (a.d ((four.a letter.n) word.n)))))
```

(b) "Love is a four-letter word"

```
((\" (k Love.n) \") ((pres be.v) (= (a.d ((four.a letter.n) word.n)))))
```

(c) "'O' is a vowel"

```
((\" |O| \") ((pres be.v) (= (a.d vowel.n))))
```

(d) ""Bonsoir, Monsieur," he said archly"

```
(sub (ds fws "Bonsoir, Monsieur") (he.pro ((past say.v) *h archly.adv-a)))
```

(e) ""Goodnight", he said archly. "Quite so", I replied."

```
(sub (\" Goodnight.gr \") (he.pro ((past say.v) *h archly.adv-a)))
(sub (\" (quite.mod-a so.a) \") (I.pro ((past reply.v) *h)))
```

(f) ""Good morning, Sir," he said cheerily"

```
(sub (\" ((Good.a morning.n) (voc |Sir|)) \")
     (he.pro ((past say.v) *h cheerily.adv-a)))
```

(g) "He said, "My bagels are the best""

```
(He.pro
 ((past say.v)
  (\" ((My.d (plur bagel.n))
       ((pres be.v) (= (the.d (best.a {ref1}.n))))) \")))
```

When mention quotes are missing, as in item (b), we add them to the ULF. We always annotate quotes with a double quote even if the surface word was quoted with single quotes, see item (c). Notice that all the quotes except item (d) are fully interpreted. The quotes in items (e) and (f) are annotated with greetings §31. Finally, we don't have any examples of nested quotes, but it can be handled straight forwardly with `(\" ... (\" ... \") ... \")`. The bracketing around the quotes make the nesting unambiguous.

2. *integrated quotes*

   We call quotes that do not retain grammaticality when replaced by an arbitrary piece of text *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.

   (h) "John's new "theory" of everything is confusing"

   ```
   ((((|John| 's) (new.a (n+preds theory.n (of.p everything.pro))))
    ((pres be.v) confusing.a))
   ```

   (i) "This is his Achilles' heel"

   ```
   (This.pro ((pres be.v) (= (his.d |Achilles' heel|.n))))
   ```

   (j) "Maldacena's "AdS/CFT correspondence" hypothesis is astounding"

   ```
   ((((|Maldacena| 's) ((|AdS/CFT| correspondence.n) hypothesis.n))
    ((pres be.v) astounding.a))
   ```

   (k) "According to him, AI "cannot ever" be achieved"

   ```
   ((adv-s (According_to.p he.pro))
      ((k |AI|.n) ((pres can.aux-s) not ever.adv-e (pasv achieve.v))))
   ```

   (l) "Harvey said an inquiry would not be ruled out, "should serious and systematic issues" emerge."

   ```
   (|Harvey|
    ((past say.v)
     (tht (((an.d inquiry.n)
            ((past will.aux-s) not (pasv rule_out.v)))
           ({if}.ps ((k ((serious.a and.cc systematic.a) (plur issue.n)))
                     ((pres should.aux-s) emerge.v)))))))
   ```

   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 these issues have 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."*

22.1 Incomplete Quotes

For dangling sentence-initial or sentence-final quotation marks (perhaps part of a multi-sentence quotation), such as item (m), we ignore those quote marks. Our focus is on single-sentence ULFs; if in future we tackle multi-sentence ULFs, and these are quoted, we could notate this as `(\" sentential-ULF1 ... sentential-ULFn \")`. Obvious accidental omission of a matching quotation mark, item (n), should be corrected in the ULF. Obviously incomplete utterances such as item (o) should not be annotated. A dangling quote mark

for an integrated quote with unclear boundaries, item (p), should be ignored. However, if the entire sentence is quoted, wrap the sentence in quotes, item (q).

(m) ""I will tell you a story. ... And that's the end of my story.""

```
(I.pro ((pres will.aux-s) (tell.v you.pro (a.d story.n))))
...
(And.cc (that.pro ((pres be.v) (= (the.d (end-of.n (my.d story.n)))))))
```

(n) "He said "I will and promptly regretted it"

```
(He.pro (((past say.v) (\" (I.pro ((pres will.aux-s) {ref1}.v)) \")) and.cc
         (promptly.adv-e (past regret.v) it.pro)))
```

(o) "He said "I will do it and"

Ignore

(p) "He said that he was "truly sorry for having insulted her."

```
(He.pro ((past say.v) (that (he.pro ((past be.v) ((truly.mod-a sorry.a)
    (for.p-arg (ka (perf (insult.v her.pro)))))))))
```

(q) "Did you see?"

```
(\" (((past do.aux-s) you.pro (see.v {ref}.pro)) ?) \")
```

22.2 Interleaved Mention Quote Attribution

Sometimes, particularly in dialogue heavy texts, mention quotes will be attributed in the middle of the quote for stylistic or temporal/causative marking purposes. To handle these we use a special operator `qt-attr` and a specially interpreted symbol `*qt`. `qt-attr` takes one argument, which is a ULF interpretation with `*qt` marking where the quote should lie. Then the contained ULF will be lifted out of the mention quote and replace `*qt` with the mention quote.

(r) "Well, perhaps not," said Alice in a soothing tone, "don't be angry about it""

```
(\" (Well.x (perhaps.adv-s not {ref}.sent))
     (qt-attr (|Alice| ((past say.v) *qt
                        (adv-a (in.p (a.d (soothing.a tone.n)))))))
     (({you}.pro ((pres do.aux-s) not
                  (be.v angry.a (adv-a (about.p it.pro))))) !) \")
```

(s) ""Won't you," he said, interrupting John's monologue, "please get on with it.""

```
(\" ((pres will.aux-s) not you.pro
     (qt-attr (he.pro ((past say.v) *qt
                       (adv-a (interrupt.v ((|John| 's) monologue.n))))))
     please.adv-s (get.v on.adv-a (with.p-arg it.pro))) \")
```

22.3 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 use the domain specific operator `ds`, see §24.

23 Parentheses

Parentheses annotation is done very similarly to quotes. For parentheses where the contents are interleaved with the sentence, simply drop the parentheses. If the contents of the parentheses can be interpreted without the grammatical context of the sentence, then wrap the interpreted contents in `(\( .. \))`. The brackets can be marked with any type by the annotator, `(\( .. \))`, `(\[ .. \])`, or `(\{ .. \})`, but they will all be postprocessed to the same representation. Thus after postprocessing, the brackets will all have uniform representation similar to how all quotes are marked with `\"`.

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

```
(sub (adv-a (for.p (k (plur appositive.n))))
     (\( (({you}.pro ((pres see.v) |Section 4.10.3|)) !) \))
     ((the-gen.d (plur comma.n)) ((pres (pasv drop.v)) *h)))
```

(b) ""[He] hate[s] to do laundry""

```
(He.pro ((pres hate.v) (to (do.v (k laundry.n)))))
```

The notable difference from quotes is that these parenthesized contents will not play into the types whatsoever. That is, the parenthesized elements will be ignored in the composition and simply retained in place for possible pragmatic analysis. That is why in the first example it isn't problematic that the parenthesized element causes an incorrect number of arguments for `sub`.

24 Domain Specific Grammars

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"* is annotated
`((my.d number.n) ((pres be.v) (= (ds phone-number "555 123 5555"))))`.
Here are a list of domains and associated examples. Please let us know if you come across something that seems domain-specific, but isn't listed here.

1. `phone-number`
   ```
   (ds phone-number "555 555-5555")
   (ds phone-number "(555)555-5555")
   (ds phone-number "5555555")
   ```
2. `date-time`
   ```
   (ds date-time "5:30pm")
   (ds date-time "June 18th 2017")
   (ds date-time "quarter after 3")
   ```
3. `currency`
   ```
   (ds currency "$50.12")
   (ds currency "Fifty dollars and 12 cents")
   (ds currency "€30")
   ```

4. address
   (ds address "880 Linden Ave")
   (ds address "Rochester NY 14620")

5. fws (foreign words)
   (ds fws "bonjour monsieur")
   (ds fws "君の名は")
   (ds fws "dm-drogerie markt")

6. temperature
   (ds temp "5 degrees Celsius")
   (ds temp "-12.3°F")

7. length (includes height and distance)
   (ds length "5′11″")
   (ds length "seven meters")
   (ds length "80km")
   ~~(ds length "about 17 miles")~~

8. speed
   (ds speed "17kph")
   (ds speed "50mile per hour")
   (ds speed "8.2 m/s")
   (ds speed "0.8c")
   ~~(ds speed "faster than 2mph")~~

9. percent
   (ds percent "2.5%")
   (ds percent "fifteen percent")
   (ds percent "72.3 percent")

10. unk (unknown domain/uninterpretable)
    (ds unk "whhhatre yooooouuuuse doeeeein")
    (ds unk "001asc21")

Notice that modifying or approximating phrases (about, faster than) are not allowed in the domain specific area. Those should compose with the domain-specific expressions since they are structurally simple English. There are additional categories of weight, acceleration, etc. which are not shown here. See §B.1 for how we intend to further resolve these annotations.

25 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-arg (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))
```

(g) "Bob and I chatted over tea and crackers"

```
((set-of |Bob| I.pro)
 ((past chat.v) (adv-a (over.p (set-of (k tea.n) (k (plur cracker.n)))))))
```

(h) "I bought apples and oranges"

```
(i.pro ((past buy.v) (set-of (k (plur apple.n)) (k (plur orange.n)))))
```

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. Items (e) to (g) are examples of this. There are also cases where all sensible interpretations of the sentence satisfy both the collective and distributive readings. Items (a) and (h) are examples of this case. Item (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 items (b) and (d) show coordinated predicates, while item (c) 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.

25.1 Discourse-Level Connectives

Coordinating conjunctions can be used at a discourse-level by starting a sentence rather than in between two sentence clauses (e.g. "But, that was ..."). We want to distinguish

these uses because a sentence-level coordination with one argument is simply vacuous, while these are not. These are really making statements about the sentence in relation to the discourse context, so we will mark these with an `.adv-s` extension. This naturally makes a correspondence between these uses of the coordinators with unambiguously sentence-level adverbials (e.g. actually, in fact). Notice the similarity between switching between these words "But/Actually/In fact, that was not the case". Now we list some examples.

(a) "But applying this rationale to society yields bizarre results"

```
(But.adv-s ((ka ((apply.v (this.d rationale.n)) (to.p-arg (k society.n))))
            ((pres yield.v) (k (bizarre.a (plur result.n))))))
```

(b) "And of course, this is what happened"

```
(And.adv-s ((adv-s (of.p (k course.n)))
 (this.pro ((pres be.v) (np+preds what.pro (past happen.v))))))
```

One must be careful to distinguish discourse connectives which are about propositions with adverbs about events.

(c) "And then, he slipped on a banana peel"

```
(And.adv-s (then.adv-e (he.pro ((past slip.v) (adv-a (on.p (a.d (banana.n peel.n)))))))))
```

26 Ellipsis

Elided text that is not supported mechanisms described elsewhere in this document, we simply use the curly-bracket notation for manually filling in the elided elements. The curly-brackets are wrapped around segments that normally correspond to surface text and indicate that the corresponding words were not present in the actual text.

(a) "Alice left, and Bob did too"

```
((|Alice| (past leave.v)) and.cc
 (|Bob| ((past do.aux-s) (leave.v too.adv-s))))
```

(b) "Bob didn't turn in his essay but Bill did"

```
((|Bob| ((past do.aux-s) not (turn_in.v (his.d essay.n)))) but.cc
 (|Bill| ((past do.aux-s) (turn_in.v (his.d essay.n)))))
```

27 Vocatives

Vocatives, used for addressing the individual at whom the speech is directed, play primarily a discursive role and are largely detached, syntactically, from the sentences in which they occur. Thus, in ULFs we will mark vocatives with a `voc` operator which can occur free-floating in the formula and is lifted out of the sentence in postprocessing. Since vocatives can be complex expressions themselves, the vocative expression should be interpreted into ULFs (similar to how we handle mentioned quotes §22).

(a) "Mary, I see you"

```
((voc |Mary|) (I.pro ((pres see.v) you.pro)))
```

(b) "I don't think I understand, Susan"

```
((I.pro ((pres do.aux-s) not
         (think.v (tht (I.pro ((pres understand.v) {ref}.pro))))))
 (voc |Susan|))
```

(c) "My ill feelings towards you, Lex, are endless"

```
((My.d (n+preds (ill.a (plur feeling.n))
                (towards.p you.pro)))
 (voc |Lex|) ((pres be.v) endless.a))
```

(d) "You in the yellow shirt, call 911!"

```
((voc (np+preds You.pro
                (in.p (the.d (yellow.a shirt.n)))))
 ({you}.pro ((pres call.v) |911|)) !)
```

(e) "John, you rascal, where have you been?"

```
((voc |John|) (voc (np+preds you.pro rascal.n))
 (sub (at.p (what.d place.n)) ((pres perf) you.pro (be.v *h))) ?)
```

(f) "You rascal where have you been, John?"

```
((voc (np+preds you.pro rascal.n))
 (sub (at.p (what.d place.n)) ((pres perf) you.pro (be.v *h)))
 (voc |John|) ?))
```

(g) "Mr. President, we must call evil by its name"

```
((voc |Mr. President|)
 (we.pro ((pres must.adv-s) (call.v (k evil.n)
                                    (by.p-arg (its.d (name-of.n *s)))))))
```

(h) "Why are ye fearful, O ye of little faith?"

```
((Why.adv-s ((pres be.v) ye.pro fearful.a))
 (voc-O (np+preds ye.pro
                  (of.p (little.a faith.n))))) ?)
```

Items (a) and (b) show basic usages of the `voc` operator. Item (c) shows how vocatives they can appear in the middle of sentences – they are simply placed where they occur with no additional bracketing. Item (d) demonstrates a complex vocative, where the vocative is more than simply a name. Items (e) and (f) show how a sentence can have multiple vocatives, even one right after another.

The vocative use of 'O' needs special attention. 'O' can be used to preface a vocative phrase in poetic speech. One needs to be careful not to conflate it with the injection 'Oh' (the spellings of the two can be interchanged). For vocatives with this 'O' preface, mark it by using `voc-O` instead of `voc` as in item (h).

28 Idioms

Idioms should be annotated as if they are literal. We treat idioms compositionally – such that the idioms are intepreted as having different idiomatic word senses than the non-idiomatic counterparts. This analysis allows us to account for variation of constructions in idioms (e.g. passivization in *"Strings were pulled to get this position"*). Paul Kay, Ivan Sag, and Dan

Flickinger published a document in 2015 with a linguistic analysis using such an approach. This approach reduces the problem of identifying idioms to a word sense disambiguation problem. Below are example annotations of idioms. As you can see, nothing is added to the ULF for the idiom.

(a) "John kicked the bucket"
```
(|John| ((past kick.v) (the.d bucket.n)))
```

(b) "Losing my job was a blessing in disguise"
```
((ka (lose.v (my.d job.n)))
 ((past be.v) (= (a.d (n+preds blessing.n (in.p (k disguise.n)))))))
```

(c) "He's spilling the beans as we speak!"
```
((He.pro ((pres prog) (spill.v (the.d (plur bean.n)))))
 (as.ps (we.pro (pres speak.v))))
```

(d) "Strings were pulled to get this position"
```
((k (plur String.n)) ((past (pasv pull.v))
                      (adv-a ({for}.p (to (get.v (this.d position.n)))))))
```

28.1 Exclamatory/Emphatic Wh-words

Exclamatory wh-words are a semantically curious use of the words "what" and "how" for expressing emotional emphasis. For example, "what" in "What a beautiful car!" is simply expressing emphasis of the evaluation. To handle these cases, we will append `-em` before to extension to indicate the special sense of "what" and "how" (e.g. `what-em.d`, `how-em.adv-a`).[8][9]

(a) "What a beautiful car that is!"
```
(sub (= (What-em.d (= (a.d (beautiful.a car.n)))))
     (that.pro ((pres be.v) *h)))
```

(b) "What beautiful cars these are"
```
(sub (= (What-em.d (beautiful.a (plur car.n))))
     (these.pro ((pres be.v) *h)))
```

(c) "What a strong person he is"
```
(sub (= (What-em.d (= (a.d (strong.a person.n)))))
     (he.pro ((pres be.v) *h)))
```

[8]Based on the similarity of these constructions to "such" in "That is such a beautiful car" one might be tempted to these uses of "what" by marking it as an adjective (as we do with "such"), but there turn out to be sentences such as "What a beautiful car you {bought, have, splurged on, ...}" which require that "what" be treated as a determiner for type-coherence. Furthermore, this method doesn't distinguish the exclamatory "how" from the question form since both are adverbs.

[9]It may be possible to come up with a grammatical distinction between the question and emphatic wh-words by thinking about further examples of premodified indefinite NPs. Considering, for example, questions like

"How big a house does he have?",

"In how deep a financial hole is he?".

They are of form `S[wh] -> XP[wh] S[inv]/XP` examples, with `XP` in `{AP, PP}`, and it's noteworthy that we don't get anything like

*"What beautiful a car did he buy?",

*"What a beautiful car did he buy?"

casting doubt on any parallels between usage of "how" and "what" in emphasis and questions.

(d) "What smart kids you are"
```
(sub (= (What-em.d (smart.a (plur kid.n))))
     (you.pro ((pres be.v) *h)))
```

(e) "What a mess he made!"
```
(sub (What-em.d (= (a.d mess.n)))
     (he.pro ((past make.v) *h)))
```

(f) "What an beautiful car!"
```
(sub (= (What-em.d (= (a.d (beautiful.a car.n)))))
     ({that}.pro ((pres {be}.v) *h)))
```

(g) "What an idea!"
```
(sub (= (What-em.d (= (an.d idea.n))))
     ({that}.pro ((pres {be}.v) *h)))
```

(h) "What a charming actress!"
```
(sub (= (What-em.d (= (a.d (charming.a actress.n)))))
     ({she}.pro ((pres {be}.v) *h)))
```

(i) "What a bunch of beautiful pictures!"
```
(sub (= (What-em.d (= (a.d (n+preds bunch.n
                                    (of.p (k (beautiful.a
                                              (plur picture.n)))))))))
     ({those}.pro ((pres {be}.v) *h)))
```

(j) "What a beautiful car you bought!"
```
(sub (What-em.d (= (a.d (beautiful.a car.n))))
     (you.pro ((past buy.v) *h)))
```

As you can see in items (f-h), the pronoun and copula can often be omitted. Please insert the most appropriate pronoun and in these cases (often "that" or "those"). Item (i) shows an example of a use with a collection noun (e.g. bunch, couple, handful, etc.). Item (j) is an example where the emphatic "what"-phrase is a term argument to a verb. Hence it isn't wrapped by `(= ..)`.

Notice the similarity of these annotations to question sentences such as *"Which character were you?"*: `((sub (= (Which.d character.n)) ((past be.v) you.pro *h)) ?)`

Exclamatory/emphatic "how" is handled in much the same way and has the same issues of omitted verb phrases (e.g. "How strange {that is}).

(k) "How studious he is!"
```
(sub (How-em.adv-a studious.a) (he.pro ((pres be.v) *h)))
```

(l) "How curious they are!"
```
(sub (How-em.adv-a curious.a) (they.pro ((pres be.v) *h)))
```

(m) "How strange!"
```
(sub (How-em.adv-a strange.a) ({that}.pro ((pres {be}.v) *h)))
```

(n) "How I used to enjoy this!"
```
(sub How-em.adv-a (I.pro (((past use.v) (to (enjoy.v this.pro))) *h)))
```

To help understand the distinction between the question and emphatic wh-words, here are a few contrasting pairs of sentences in more complex scenarios.

(o1) "You should see what beautiful car he bought"

```
(You.pro ((pres should.aux-v) (see.v
          (ans-to (sub (what.d (beautiful.a car.n))
                       (he.pro ((past buy.v) *h)))))))
```

(o2) "You should see what a beautiful car he bought"

```
(You.pro ((pres should.aux-v) (see.v
          (ans-to (sub (What-em.d (= (a.d (beautiful.a car.n))))
                       (he.pro ((past buy.v) *h)))))))
```

(p1) "You should see what model of car he bought"

```
(You.pro ((pres should.aux-v) (see.v
          (ans-to (sub (what.d (n+preds (model-of.n (k car.n))))
                       (he.pro ((past buy.v) *h)))))))
```

(p2) ?"You should see what a model of car he bought"

```
(You.pro ((pres should.aux-v) (see.v
          (ans-to (sub (what-em.d (= (a.d (n+preds (model-of.n (k car.n))))))
                    (he.pro ((past buy.v) *h)))))))
```

(q1) "I know in how deep a financial hole he now is, because of his risky investments"

```
(I.pro ((pres know.v) (ans-to
        (sub (in.p (sub (how.adv-a deep.a)
                        (a.d (*h (financial.a hole.n)))))
             (he.pro now.adv-e ((pres be.v) *h)
                     (adv-s (because_of.p
                             (his.d (risky.a (plur investment.n))))))))))
```

(q2) "In how deep a financial hole he now is, because of his risky investments!"

```
(sub (In.p (sub (how-em.adv-a deep.a)
                (a.d (*h (financial.a hole.n)))))
     (he.pro now.adv-e ((pres be.v) *h)
             (adv-s (because_of.p
                     (his.d (risky.a (plur investment.n)))))))
```

29 Adjectives with Complements

While most adjectives are monadic, there are certain classes of adjectives that take arguments. A prime example of this is adjectival derivations of verbs, e.g. "frightened".

"John is frightened of spiders"

```
(|John| ((pres be.v) (frightened.a (of.p-arg (k (plur spider.n))))))
```

Infinitive complements.
Some adjectives take infinitive complements rather than prepositionally marked ones which we found to be particularly tricky to analyze. We list some here and how we analyze them for reference during annotation.

(a) *was supposed to*
"I was supposed to go home"
`(I.pro ((past be.v) (supposed.a (to (go.v (k home.n))))))`

(b) *was obligated to*
"I was obligated to stay"
`(I.pro ((past be.v) (obligated.a (to stay.v))))`

(c) *was destined to*
"I was destined to fail"
`(I.pro ((past be.v) (destined.a (to fail.v))))`

(d) *was apt to*
"I was apt to agree"
`(I.pro ((past be.v) (apt.a (to agree.v))))`

(e) *was able to*
"I was able to finish in time"
`(I.pro ((past be.v) (able.a (to ((finish.v {ref}.pro) (adv-e (in.p (k time.n))))))))`

Notice that some of these can become passive in the right context. "I was destined by my circumstances to ..."

29.1 Special case "used to"

"Used to" is a particularly tricky case with a lot of variants and exceptional uses.

- **Basic example**
  I used to sleep 8 hours a night.
- **Tense disappears with 'did' auxiliary**
  Did you use to work here?
  We didn't use to earn much.
- **We can include negation in between**
  They used not to allow shops to be open on Sundays
- **Different meanings in difference contexts**
  I am used to doing something
  I used to do something
  The first example involves a gerund (an -ing construction) as argument, not an infinitive. (A noun phrase could replace the gerund: *"I am used to a busy work schedule".*) Also we can't make the variants listed above where 'used' and 'to' are split apart, or a tenseless 'use' while preserving the meaning.

Given these issues, there seem to be two different types of 'used to' in the examples above. First the version that takes an infinitives (without the -ing) is annotated as a verb.

```
(I.pro ((past use.v) (to (do.v something.pro))))
```

The other one seems to be an adjective reading with a gerund, `gd` second argument.

```
(I.pro ((pres be.v) (used_to.a (gd (do.v something.pro)))))
```

This is further supported by the fact that when we the copula for the second variant, we

don't lose the apparent 'tense' marking on 'used': "Were you used to doing something". Furthermore, we can't add a negation in between 'used' and 'to' for the adjectival reading: *"They were used not to going shopping on Sundays."

Of course also beware of the passive form of 'use', which can look a lot like the adjectival version, but without the gerund.

"I was used to confuse John" (say I look a lot like another person John knows)
– sim. "I was used for confusing John"
– sim. "I was used in order to confuse John"

```
(I.pro ((past (pasv use.v)) (adv-a ({for}.p (to (confuse.v |John|))))))
```

30 Interjections

Interjections include semantically distinct subcategories that we will handle separately: evaluations (*Cool!*, *Nice day*), expletives (*ow!*, *damn!*), *yes-no*, and discourse level connectives (*but*, *for*, *so*).

30.1 Evaluative Interjections

Evaluative interjections are really sentences with an implicit reference sentence that is being evaluated. They are very commonly phrasal utterances, so rather than always supplying an the implicit sentence in every case, we introduce the macro `pu`, for *phrasal utterance*, which marks a coherent phrase (one that is an acceptable and interpretable utterance to a speaker) that is acting as a sentence.

(a) "Cool!"

```
(pu Cool.a)
```

(b) "Nice day"

```
(pu (Nice.a day.n))
```

(c) "Great"

```
(pu Great.a)
```

(d) "Fantastic"

```
(pu Fantastic.a)
```

(e) "Most definitely"

```
(pu (adv-s ((mod-a most.a) definite.a)))
```

Since we consider these evaluative interjections full sentences on their own, they will be completely separated from sentences following them.

(f) "Fantastic, that works for me"

```
((pu Fantastic.a)
 (that.pro ((pres work.v) (for.p-arg me.pro))))
```

30.2 Expletives & Non-compositional Utterances

Expletives are words that express emotional intensity, often using curse words. When used alone – say as a reaction – they will be annotated with a `.x` extension. It will represent a sentence on its own. Multi-word expletives can each use a `.x` extension an simply bracketed together. We'll include in this category phrases that are technically not expletives, but which are similarly non-propositional or have non-compositional meanings without being fillers such as *"huh?"*, *"Oh!"*, *"Shh"*, and *"Tsk-tsk"*.

(a) "Damn!" – `Damn.x`

(b) "Bloody hell!" – `(Bloody.x hell.x)`

(c) "Ow!" – `Ow.x`

(d) "Huh?" – `Huh?.x`

(e) "Oh, what a nice surprise!"

```
(Oh.x (sub (= (What-em.d (= (a.d (nice.a surprise.n)))))
           ({this}.pro ((pres {be}.v) *h))))
```

(f) "Shh, this is a library"

```
((Shh.x !) (this.pro ((pres be.v) (= (a.d library.n)))))
```

(g) "Tsk-tsk, I did warn you"

```
(Tsk-tsk.x (I.pro ((past do.aux-v) (warn.v you.pro))))
```

Notice that in item (f), we can still use the imperative operator `!` with the interjections. This is because they act as sentences. Also, *"Shh"* has an imperative meaning: *you be quiet!*.

Expletives can also be used in attributive adjective position. For these uses mark them as adjectives since they are performing a syntactic functionality. The meaning will be resolved in the word sense disambiguation step.

(h) "That wretched spider"

```
(That.d (wretched.a spider.n))
```

30.2.1 Fillers

Hesitation markers and filler words seem similar to the `.x` markings, but we will ignore them in the annotation since they are either disfluencies or turning-holding.

(i) "That was, um, not my intention"

```
(That.pro ((past be.v) not (= (my.d intention.n))))
```

(j) "Er, okay"

```
(pu okay.a)
```

30.3 Yes-no

Any word to express 'yes' or 'no' semantically will be annotated with `.yn` extension. These make a truth-value evaluation over a sentence – thus is a special case of the general evaluative statements. In practice, these are used separately from the evaluated sentence, which is supplied by the context, so we will annotate correspondingly. The annotated yes-no expressions can be thought of as macro'd expressions that hide an implicit referential sentence. Thus yes-no expression will have a sentence meaning. If they come before a sentence, we will annotate them as two separate statements that are grouped together – just like two statements joined by semi-colons.

(a) "Yes" – `Yes.yn`

(b) "Uh-huh, that's the plan"

```
(Uh-huh.yn (that.pro ((pres be.v) (= (the.d plan.n)))))
```

(c) "Definitely yes"

```
(Definitely.adv-s yes.yn)
```

(d) "Yes, definitely"

```
(Yes.yn (pu definitely.adv-s))
```

(e) "Surprisingly, no"

```
(Surprisingly.adv-s no.yn)
```

Here is a small sample of words in this category: *yes, no, yeah, uh-huh, uh-uh, sure, nope, nah.*

30.4 Miscellaneous Interjections

Annotating "please"
"Please" has a fully pragmatic reading, syntactically acts as an adverb, and operates over an full proposition (though it may have a pragmatic focus on different specific parts of the sentence) so it will be annotated as `please.adv-s`.

Phrasal Question Answers
Often times phrasal utterances are used to answer questions:
"Where were you over the weekend?"
"at home"
We will use the `pu` operator that was introduced in §30.1. In fact, many of the examples in that section could be used in this context.

(a) "at home" – `(pu (at.p (k home.n)))`

31 Greetings

Single word greetings are annotated with a `.gr` extension.

(a) hi – `hi.gr`

(b) hello – `hello.gr`

(c) goodnight – `goodnight.gr`

(d) farewell – `farewell.gr`

Multi-word greetings implicitly bidding the listener some predicate are annotated with a greeting-forming operator `gr`, which takes one predicate argument. The greeting forming operator `(gr <pred>)` is synonymous with "I bid you <pred>"

(e) Good night – `(gr (Good.a night.n))`

Most other greetings are conventionalized sentences, so we'll annotate them literally, similar to idioms. These will often be phrasal utterances.

(f) "See you tomorrow"

```
(pu (See.v you.pro tomorrow.adv-e))
```

(g) "How's it going?"

```
((sub How.adv-s ((pres prog) it.pro (go.v *h))) ?)
```

(h) "Pleased to meet you"

```
(pu (pleased.a (to (meet.v you.pro))))
```

NB: There are some greetings that use implicit it-extraposition construction, so we can't annotate them yet: *"{it is} Nice to meet you"*, *"{it is} Good to see you"*.

32 Singular Plurals

Some nouns act as plurals even though they are singular objects: *pants*, *glasses*, *scissors*, etc. Given that there is not just thing as a 'pant', it seems strange to annotate 'pants' as `(plur pant.n)`. However, there are contexts in which 'pant' does occur, e.g. as a nominal modifier: *pant salesman* and *scissor maker* rather than **pants salesman* or **scissors maker*. Furthermore, there are instances where its ambiguous whether the word is a singular plural or homonym that is used plurally. For example, *"I would like my glasses back"* could refer to a pair of eye-glasses or some plurality of water-glasses. Given these observations, we will annotate plurals just as they appear syntactically:

glasses - `(plur glass.n)`; pants - `(plur pant.n)`; scissors - `(plur scissor.n)`.

33 Counterfactuals & Conditionals

Counterfactuals in English appear through syntactic past tense in a phenomenon called "fake tense" (for example items (a) and (b) below), with the exception of explicit subjective mood, which uses 'were' regardless of the number (for example item (c)). Counterfactuality will be marked with a `cf` operator, which is supplied instead of the tense operator. That is, counterfactual sentences will not be marked with tense.

(a) "I wish I was rich"

```
(I.pro ((pres wish.v) (tht (I.pro ((cf be.v) rich.a)))))
```

(b) "I wish I believed you"

```
(I.pro ((pres wish.v) (tht (I.pro ((cf believe.v) you.pro)))))
```

(c) "I wish I were rich"

```
(I.pro ((pres wish.v) (tht (I.pro ((cf were.v) rich.a)))))
```

(d) "If I was rich, I would own a boat."

```
((if.ps (I.pro ((cf be.v) rich.a)))
 (I.pro ((cf will.aux-s) (own.v (a.d boat.n)))))
```

(e) "Were he to leave, the project would collapse"

```
(((cf be-to.aux-s) he.pro leave.v)
 ((the.d project.n) ((cf will.aux-s) collapse.v)))
```

(f) "Had I forseen this, I would never have participated"

```
(((cf perf) I.pro (forsee.v this.pro))
 (I.pro ((cf will.aux-s) never.adv-e (perf participate.v))))
```

(g) "If I had been rich, I would own a boat"

```
((If.ps (I.pro ((cf perf) (be.v rich.a))))
 (I.pro ((cf will.aux-s) (own.v (a.d boat.n)))))
```

(h) "If I had been rich, then I would have owned a boat"

```
((If.ps (I.pro ((cf perf) (be.v rich.a))))
 (then.adv-s (I.pro ((cf will.aux-s) (perf (own.v (a.d boat.n)))))))
```

Using the `cf` operator easily generalizes to cases where the counterfactual is marked through a tense on the verb or auxiliary such as examples items (b) and (f) (rather than say 'were'). Since counterfactuals are formed using syntactic tense, counterfactuals about the past are handled through perfect aspect, see items (f) to (h). As far as the annotation is concerned, simply mark the counterfactual with `cf` and otherwise annotate the perfect as usual.

Since the semantics of perfect is that the verb phrase is completed, it enables a past tense reading of the embedded verb phrase. Also, not all counterfactual constructions will be strictly counterfactual. There are phenomena that are in some sense "future counterfactuals" which are counterfactual constructions on future events which indicate that the event is unlikely to occur. Since this is the future, it cannot be strictly counterfactual. Here's an example for clarity.

If you gave me $5, I would buy a soda

This turns out to have to do with the verb class distinction, which you don't need to fully understand. But basically, telic verbs (those that have an defined end—*give*, *stop*, *finish something*) lead to these "future counterfactual" readings. Statives on the other hand get strict counterfactual readings. This relates to the fact that in non-counterfactual conditionals telic verbs get a future evaluation, but not static verbs, e.g.

If you give me $5, I will buy a soda (future eval of antecedent)

If you are tall, I will buy a soda (present eval of antecedent)

It turns out you can turn strict counterfactuals to "future counterfactuals" and vice-versa using temporal adverbials (though it can be awkward).

If I were tall ***tomorrow****, I would be able to touch the ceiling.*

If you gave me $5 ***yesterday****, I would buy a soda*

So we'll annotate these two case in the same way. The difference in semantic interpretation will have to be managed along with the full temporal context.

Items (e) and (f) show cases of an implicit "if" via auxiliary inversion. For these, the inversion captures the full meaning of the counterfactual conditional antecedent, so no additional annotation is necessary. Please keep in mind that the meaning of this inversion is the same as regular "if"-conditional antecedents. That is, this is postprocessed with the following rule.

`((cf AUX) NP VP)` ≡ `(if.ps (NP ((cf AUX) VP)))`

Item (h) shows how to handle 'then' in if-then statements. Simply annotate them as `then.adv-s` which scope over the consequent. It helps mark the consequent, but in postprocessing will be reduced into the semantics of the 'if' conditional. Finally, 'would' in these counterfactual constructions are annotated with `(cf will.aux-s)` due to the correspondence in the sentences below:

If you give me $5 I will go buy a cake

If you gave me $5 I would go buy a cake

If I am rich, I will buy a boat

If I was rich, I would buy a boat

Note that the second and fourth sentences are the counterfactual variants of the first and third. Thus is it consistent to annotate the change in the consequent *will* → *would* using the same operator as the the change in the antecedents *give* → *gave* and *am* → *was*.

34 Miscellaneous Issues

34.1 Discussion on Annotating Prepositions

By and large, prepositions will be annotated in one of the following two contexts:

```
(adv-* (*.p something.pro))
(*.p-arg something.pro)
```

Selecting between `*.p-arg` *and* `(adv-* (*.p ..))`

`*.p-arg` marks arguments of a verb that are denoted with a preposition. This will mean that the preposition will not have its typical meaning (as used with other verbs). Additionally, the supplied argument is in fact an argument and not a modifier of the verb, e.g.

```
(believe.v (in.p-arg someone.pro))
```

Notice that "in" here doesn't mean the same thing as "Saw him in the building" or "I found him in the mall". The adverbial readings of prepositions are applicable in a wide variety of contexts so please consider other verbs to see if the preposition preserves meaning when selecting between the two preposition options. Also, if the adverbial prepositional phrases are dropped, the full meaning of the verb is preserved.

`*.p-arg` can be used with verbs or verb derived words only! This is because it marks an argument in relation to the verb. So we can't do

```
(man.n (of.p-arg (the.d (plur person.n))))
```

but we can do

```
(sale.n (of.p-arg (the.d car.n)))
(collapse.n (of.p-arg (the.d building.n)))
```

On rare occasion prepositions will be used bare because of one of the following contexts:

- Prepositional phrase after a copula acting directly as a predicate.

  ```
  (I.pro ((pres be.v) (in.p (the.d forest.n))))
  ```

- Post-nominal modification.

  ```
  (n+preds day.n (of.p (the.d week.n)))
  ```

- Predicate complement.

  ```
  (He.pro ((past force.v) me.pro (onto.p (the.d street.n))))
  ```

34.2 -ing VPs as Action Adverbials

There remains a reading of -ing VPs that have not gotten proper mention in the guidelines so far. We have so far discussed -ing marking progressives (§11) and kinds of actions (§6.1). This -ing can also mark verb phrases as action adverbials (it was briefly mentioned in §13.5). For example

I took a stroll last night, looking at the stars.

"looking at the stars" is a verb phrase that describes a concurrent action taken during the main verb phrase. This is annotated by `adv-a` and a tense-less verb phrase, e.g.

```
(I.pro ((past take.v) (a.d stroll.n)
        (adv-e (last.a night.n))
        (adv-a (look.v (at.p-arg (the.d (plur star.n))))))
```

The tense of the verb phrase this verb phrase is tense-less because the tense is dependent on the modified verb phrase:

I took a stroll last night, looking at the stars.

I will take a stroll tonight, looking at the stars.

As with most clauses, this can be topicalized as well: *"Looking at the stars, I took a stroll."* There is a corresponding noun version of this phenomenon, which has come up a few times earlier in the guidelines.

the dog wagging its tail was my favorite

```
((The.d (n+preds dog.n (wag.v (its.d tail.n))))
 ((past be.v) (= (my.d (favorite.a {ref}.n)))))
```

Notice that just like in the VP case, this the subject of the -ing VP is shared with the argument to the modified predicate.

34.3 Unacceptable Fragments

For fragments do not make coherent statements are not annotated.

- "So, why did"
- "It depends on the"
- "when you touch it"

35 Conclusion

Now that you have gone through this tutorial, you should be equipped to annotate most sentences with the corresponding ULF. If while annotating, you see a situation that has not been covered in this tutorial, 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)

§13.1 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:

- *nnp - shifts noun phrase to noun predicate modifier
- mod-n - shifts any predicate to a noun predicate modifier
- mod-a - shifts any predicate to an adjective predicate modifier
- adv-a - shifts any predicate to monadic verb 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.

```
(N1' N2')   →   ((mod-n N1') N2')
(NP' N')    →   ((nnp NP') N')
(A' N')     →   ((mod-n A') N')
(A1' A2')   →   ((mod-a A1') A2')
(N' A')     →   ((mod-a N') A')
```

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

```
(n+preds N' Pred1 Pred2 ... PredN) ≡
   (:l x ((x N') and (x Pred1) (x Pred2) ... (x PredN)))
(np+preds NP' Pred1 Pred2 ... PredN) ≡
   (the.d (:l x ((x = NP) and (x Pred1) (x Pred2) ... (x PredN))))
```

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

These macros are introduced in §13.7 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

(sub C S[*h]) ≡ S[*h←C]

Example

```
(sub |Juliet| (|Romeo| ((pres love.v) *h)))
→ (|Romeo| ((pres love.v) |Juliet|))
```

sub is introduced in §12 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 λ symbol, the lambda bound variable, and the formula, respectively.

A.4 Relativizers (that.rel and who.rel)

Definitions

S_{emb}[that.rel] ≡ (:l *r S_{emb}[that.rel←*r])

S_{emb}[who.rel] ≡ (:l *r S_{emb}[who.rel←(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

```
(the.d (n+preds table.n
               (sub that.rel
                    (|John| ((past know.v) (tht (I.pro ((past like.v) *h))))))))
```

Which when n+preds and sub are expanded results in[10]

```
(the.d (:l x ((x table.n) and.cc
              (x (|John| ((past know.v)
                          (tht (I.pro ((past like.v) that.rel)))))))))))
```

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

```
(the.d (:l x ((x table.n) and.cc
              (x (:l *r (|John| ((past know.v)
                                 (tht (I.pro ((past like.v) *r)))))))))))
```

[10] If you're not sure how these expansions work step-by-step, take a look at Appendix Section A.4.1 for a

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.

```
*(the.d (:l x ((x table.n) and.cc
               (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

`(sub C[that.rel] F)` ≡ `(:l *r (sub C[that.rel←*r] F))`

`(that.rel VP)` ≡ `(:l *r (*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 `(:l y ((y person.n) and (y = *r)))` rather than `*r`.

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 car.n (sub that.rel (you.pro ((past buy.v) *h))))`,

now we can expand `n+preds` into the lambda expression

`(:l x ((x car.n) (x (sub that.rel (you.pro ((past buy.v) *h))))))`,

then `sub` moves the relativizer into the relative clause

`(:l x ((x car.n) (x (you.pro ((past buy.v) that.rel)))))`,

then we interpret the relativizer `that.rel` to `*r` in a lambda expression

`(:l x ((x car.n) and.cc (x (:l *r (you.pro (past buy.v) *r)))))`,

via lambda-conversion becomes

`(:l x ((x car.n) and.cc (you.pro (past buy.v) x)))`,

i.e., 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 car.n (sub that.rel (you.pro ((past buy.v) *h))))`,

now we can expand `n+preds` into the lambda expression

`(:l x ((x car.n) (x (sub that.rel (you.pro ((past buy.v) *h))))))`,

then we used the simplified rule `(sub C[that.rel] F)` ≡ `(:l *r (sub C[that.rel←*r] F))`

`(:l x ((x car.n) and.cc (x (:l *r (sub *r (you.pro (past buy.v) *h))))))`,

then sub moves the *r into the relative clause

```
(:l x ((x car.n) (x (:l *r (you.pro ((past buy.v) *r))))))
```

via lambda-conversion becomes

```
(:l x ((x car.n) and.cc (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 (the.d (:l x ((x 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

```
(np+preds (the.d manager.n)
          (sub who.rel (you.pro ((past meet.v) *h))))
```

via np+preds and sub becomes

```
(the.d (:l y ((y = (the.d manager.n)) and.cc
              (y (you.pro ((past meet.v) who.rel))))))
```

via who.rel becomes

```
(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))))))))))))
```

which after *r lambda-conversion becomes

```
(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 It-clefts

It clefts are annotated by marking the "it" as `it-cleft.pro` and supplying the topicalized argument as the first argument and the relative clause as the second argument to `be.v`. post-processing it-clefts boils down to inserting the first argument into the relativizer position. It turns out that it-clefts allow arbitrary clauses so the relativizer will be replaced directly, rather than expanding to the lambda expression.

Simple definition of it-cleft rule
`(It-cleft.pro (((<tense> be.v) X) R[*.rel])) ⇒ R[*.rel←X]`

Here's an example of this rule application:
`(It-cleft.pro (((past be.v) |Mary|) (who.rel ((past arrive.v) first.adv-a))))`
`⇒ (|Mary| ((past arrive.v) first.adv-a))`

This turns out not to be quite enough to handle all the cases of it-clefts, since it-clefts can include auxiliaries, negations, and sentence adverbials which applies to the mapped sentence as well as question constructions. So we'll first require that all the rules in this section be applied after question inversion is undone and add the following for it-clefts with auxiliaries, negations, and sentence adverbials.

Definition of it-cleft rule including auxiliary handling.
`(It-cleft.pro ((<tense> <aux>) ((be.v X) R[*.rel]))) ⇒ (<aux> R[*.rel←X])`

Here's an example.
`(It-cleft.pro ((past can.aux-s) ((be.v |Mary|) (that.rel (past leave.v)))))`
`⇒ (can.aux-s (|Mary| (past leave.v)))`

Definition of it-cleft rule for sentence-adverbials (and negation).
`(<sent adv> (It-cleft.pro (((<tense> be.v) X) R[*.rel]))) ⇒ (<sent adv> R[*.rel←X])`
where this needs to be applied after the applicable sentence-adverbials are lifted to sentence level.

Here's an example.
```
(Probably.adv-s (It-cleft.pro (((past be.v) me.pro)
                               (who.rel ((past be.v) confused.a)))))
⇒ (Probably.adv-s (me.pro ((past be.v) confused.a)))
```

These mapping doesn't have the tense and auxiliary marking in the same location as the surface ULF annotation. We can write more sophisticated mapping rules so that the surface form is the same as the guidelines. However, this mapping will capture the correct further

postprocessed meaning since the sentence-level auxiliaries and tenses are lifted to sentence level anyway.

A.5.1 Presupposition from it-clefts

You may have noticed that the rules listed so far don't seem to quite capture the meaning of the it-cleft. This is because the it-cleft carries a strong presupposition that the relative clause is satisfied by something. From the example above "It could be Mary that left" we infer that "Someone left" regardless of whether it is in fact Mary that left. Presuppositional meanings must be handled distinctly from the truth-functional meaning so this will need to involve a separate, presuppositional inference rule.

(It-cleft.pro (... A .. R[*.rel]))
$\Rightarrow_{presuppositional}$ ((Some.d <bleached A type>) (λ x R[*.rel$\leftarrow$x]))

A.5.2 Postprocessing for cleaner inferences

There are postprocessing steps that can be taken to get a cleaner representation for making inferences. We use the operators emph and psbl to capture the emphasis of basic it-clefts and emphasized possibility of it-clefts with auxiliaries. psbl takes an additional argument of the modal, negation, or sentence-adverbial that defines the exact modal distinction being made.[11]

Definitions for emph *and* psbl *postprocessing.*
(It-cleft.pro (((<tense> be.v) X) R[*.rel])) $\Rightarrow$ R[*.rel$\leftarrow$(emph x)]
(It-cleft.pro ((<tense> <aux>) ((be.v X) R[*.rel]))) $\Rightarrow$ R[*.rel$\leftarrow$(psbl X <aux>)]
(<sent adv> (It-cleft.pro (((<tense> be.v) X) R[*.rel])))
$\Rightarrow$ R[*.rel$\leftarrow$(psbl X <sent adv>)]

Here are examples that parallel the examples given in the direct handling.

(It-cleft.pro (((past be.v) |Mary|) (who.rel ((past arrive.v) first.adv-a))))
$\Rightarrow$ ((emph |Mary|) ((past arrive.v) first.adv-a))
(It-cleft.pro ((past can.aux-s) ((be.v |Mary|) (that.rel (past leave.v)))))
$\Rightarrow$ ((psbl |Mary| can.aux-s) (past leave.v))
(It-cleft.pro ((past must.aux-s) ((be.v |Mary|) (that.rel (past leave.v)))))
$\Rightarrow$ ((psbl |Mary| must.aux-s) (past leave.v))

From here it's very simple to get both the semantic content as well as the pragmatic and presuppositional inferences.

Semantic content inference rules and examples
S[(emph X)] $\Rightarrow$ S[(emph X)$\leftarrow$X]
e.g. ((emph |Mary|) ((past arrive.v) first.adv-a))
$\Rightarrow$ (|Mary| ((past arrive.v) first.adv-a))
S[(psbl X <op>)] $\Rightarrow$ (<op> S[(psbl X)$\leftarrow$X])
e.g. ((psbl |Mary| can.aux-s) (past leave.v)) $\Rightarrow$ (can.aux-s (|Mary| (past leave.v)))

Presuppositional content inference rules and examples
S[(emph X)] $\Rightarrow_{presupp}$ S[(emph X)$\leftarrow$(Some.d <bleached type of X>)]

[11] emph can be mapped to psbl parameterized with do.aux-s if this turns out to result in simpler handling of this phenomena. (emph X) $\equiv$ (psbl X do.aux-s). Notice that since do.aux-s has a null modal operation, the expansion of (psbl X do.aux-s) sentences can be reduced to the corresponding (emph X) sentences.

e.g. `((emph |Mary|) ((past arrive.v) first.adv-a))`
$\Rightarrow_{presupp}$ `((Some.d person.n) ((past arrive.v) first.adv-a))`
`S[(psbl X)]` $\Rightarrow_{presupp}$ `S[(psbl X)`$\leftarrow$`(Some.d <bleached type of X>)]`
e.g. `((psbl |Mary|) (past leave.v))`
$\Rightarrow_{presupp}$ `((Some.d person.n) (past leave.v))`

This postprocessing as described actually loses some information for the auxiliary case since the meaning can depend on the exact meaning of the auxiliary. For example, "It might be John that I saw" and "It must be John that I saw" have a different in degree of certainty. Thus to fully capture this we would need to either introduce a variant of `psbl` for each auxiliary (e.g. `psbl-can`, `psbl-must`, etc.) or make `psbl` a two-argument operator (e.g. `(psbl |John| can.aux-s)`, `(psbl |John| must.aux-s)`. We could even merge this all together and define `(emph X)` $\equiv$ `(psbl X do.aux-s)`. By the semantics of `do.aux-s` having a null modal effect, the meanings would be equivalent.

A.5.3 Negation

It-cleft negations need to be handled specially because they carry the same presuppositional inferences but different semantic information.

It was John that I saw
$\rightarrow_{presupp}$ *I saw someone* AND $\rightarrow$ *I saw John*
It was not John that I saw
$\rightarrow_{presupp}$ *I saw someone* AND $\rightarrow$ *I did not see John*
It was John that I didn't see
$\rightarrow_{presupp}$ *I didn't see someone* AND $\rightarrow$ *I did not see John*

It turns out this will be very similar to the handling of it-clefts with auxiliaries. In order to still get the right presuppositional inferences, we need to introduce `emph-not` and `psbl-not` operators that get introduced if there's a negation directly after the copula of the it-cleft construction. If we have argument taking `emph` and `psbl` as suggested in the previous section, they could simply take the negation as an argument.

"It was not John that I saw"
`(I.pro ((past see.v) (emph-not |John|)))`
$\Rightarrow$ `(not (I.pro ((past see.v) |John|)))` AND
$\Rightarrow_{presupp}$ `(I.pro ((past see.v) (some.d person.n)))`
"It was John that I didn't see"
`(I.pro ((past do.aux-s) (see.v (emph |John|))))`
$\Rightarrow$ `(I.pro ((past do.aux-s) not (see.v |John|)))` AND
$\Rightarrow_{presupp}$ `(I.pro ((past do.aux-s) not (see.v (some.d person.n))))`

A.5.4 Arbitrarily complex modality

The modal/sentence-level operators that focus on the topicalized portion of the it-cleft construction can be arbitrarily complex, e.g.

"It conceivably but not very likely was me who fell asleep"

Given our parameterized `psbl` operator, this is straightforward to handle.

```
(It-cleft.pro ((conceivably.adv-s but.cc (adv-s (not (very.mod-a likely.a))))
               (((past be.v) me.pro) (who.rel ((past fall.v) asleep.a)))))
→ ((psbl me.pro (conceivably.adv-s but.cc (adv-s (not (very.mod-a likely.a)))))
   ((past fall.v) asleep.a))
```

A.6 Rightshifting (`rep` operator)

The `rep` operator is defined in almost exactly the same way as `sub` but with the arguments reversed.

A.7 Possessives

Prenominal Possessive Rewriting Definitions

`((NP 's) N[(_-of.n *s)])` ≡ `(the.d N[*s←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 20, 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 20 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 ed at Table 2).

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))`

A.8 Temporal Terms

Temporal terms turn out to be very flexible in English so there are some shorthand relations in ULF to simplify the annotation process while capturing the proper meaning.

First, there are the deictic temporal terms, *today*, *yesterday*, etc. that seem to be able to act as preposition-like predicates, pronouns, and adverbs:

"The lunch today was good"

"Today was a good day"

"We had fun today"

In the ULF annotations, we allow these to be annotated as `today.a`, `today.pro`, and `today.adv-e`, respectively. At their core, all of these meanings are based on the pronoun reading with the following definitions:

`today.a` ≡ `(during.p today.pro)`

today.adv-e ≡ (adv-e (during.p today.pro))
≡ (adv-e today.a)

There are still times where adv-e or {during}.p will need to be added since English allows this type-flexibility for all deictic temporal phrases. For example, "The lunch the day before yesterday was good" would need to be annotated as

```
((The.d (n+preds lunch.n
                ({during}.p (the.d (n+preds day.n
                                            (before.p yesterday.pro))))))
 ((past be.v) good.a))
```

The second type of temporal term that gets special treatment in ULF are names for particular days, e.g. *Monday*, *January*, *Labor Day*. These have both normal name and predicate name readings.

"Monday was rainy"

"This Monday was great"

We allow the annotation as either a name or predicate name as appropriate, |Monday| or |Monday|.n. |Monday| is a predicate that is true of all Mondays. |Monday| has an ambiguous interpretation as either (k |Monday|.n) (as in "Monday is my favorite day of the week") and (the.d |Monday|.n) (as in "Monday was rainy"). The latter reading represents the dependence of the meaning on the context.

A.9 Expanding Adverbs that Modify Adjectives and Verbs

Sentence-level adverbs that seem to act locally on adjectives and verbs require a special expansion interpretation. For those that are familiar, the expansion will look similar to non- representation in Montague semantics in terms of a lambda expression and not. The main section of the guidelines that cover this phenomenon is Section 13.4. The expansion rule is as follows:

(ADV-S/E/F ADJ/V) ⇒ (lambda x (ADV-S/E/F (x ADJ/V)))

Let's see some examples of how this works.

"The surprisingly happy man left"

```
((The.d ((surprisingly.adv-s happy.a) man.n)) (past leave.v))
→ ((The.d ((lambda x (surprisingly.adv-s (x happy.a))) man.n))
    (past leave.v))
```

To be fully explicit, this lambda is acting as an attributive adjective to man. So we get

```
→ ((The.d ((attr (lambda x (surprisingly.adv-s (x happy.a)))) man.n))
    (past leave.v)).
```

Since surprisingly.adv-s is now acting at a sentence-level already, it will not be lifted up to the top level. That way we make the right distinction in meaning between this example and the ULF for *"The happy man surprisingly left"*. This alternative would be interpreted as

```
((The.d (happy.a man.n)) surprisingly.adv-s (past leave.v))
```

which with the sentence adverb lifted and the implicit attr introduced looks like

```
→ (surprisingly.adv-s ((The.d ((attr happy.a) man.n)) (past leave.v))).
```

There seems to be an issue regarding how to identify the type of the modified predicates. Since lambdas don't capture the noun/verb/adjective predicate distinctions it is now ambiguous. We will as a rule determine the category of the lambda expression to be the same as the type of the predicate that is being modified.

We now will look at three related sentences which will show examples of using `adv-e`, modification of verbs, and some more subtle restrictions of the sentence-level adverb lifting.

- *"I expected briefly to be confused"*

  `(I.pro ((past expect.v) briefly.adv-e (to (be.v confused.a))))`

  → `(briefly.adv-e (I.pro ((past expect.v) (to (be.v confused.a)))))`

- *"I expected to briefly be confused"*

  `(I.pro ((past expect.v) (to (briefly.adv-e (be.v confused.a)))))`

  → `(I.pro ((past expect.v)`
  `(to (lambda x (briefly.adv-e (x (be.v confused.a)))))))`

- *"I expected to be briefly confused"*

  `(I.pro ((past expect.v) (to (be.v (briefly.adv-e confused.a)))))`

  → `(I.pro ((past expect.v)`
  `(to (be.v (lambda x (briefly.adv-e (x confused.a)))))))`

Notice that all of these representations are different from each other. However, we expect that the second and third examples should have the same meaning. This will be captured by the fact that `be.v` is only acting as a predicate applicator in these examples. In essence, it has a null semantic effect here.

B Deeper Discussion

B.1 Postprocessing Domain Specific Content

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, EL uses a "record type" to write these down

`<record>` ::= (`$` `<record type>` $\texttt{<term>}_1$ $\texttt{<term>}_2$ ... $\texttt{<term>}_n$), `n` geq `1`

For ULF, we simply mark these as domain specific grammars and preserve the string (see Section 24). The ULFs should be further resolved into record types as discussed here. This multi-step approach is necessary because these phenomena have complex grammars but do not appear often enough in general text to be learned from a small dataset. Thus, domain-specific parsers either hand-written or trained on a dedicated dataset can be used to resolve the semantic content of these phenomena.

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 ing of common record types to get a sense of this representation.

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 8.3 on generated determiners.

B.2 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 postprocessed 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, `|_|.n`

`|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.)