The Talking Robots Toolkit

Documentation for the MOLOKO CCG grammar (v6)

Geert-Jan M. Kruijff & Trevor Benjamin
@ the DFKI Language Technology Lab

January 16, 2012

About this document

This documentation describes version 6 of the MOLOKO CCG grammar resource. MOLOKO is provided as part of the core Tarot distribution, under an open-source license.

Citation information

G.J.M. Kruijff & T. Benjamin.
The Talking Robots Toolkit: Documentation for the MOLOKO CCG grammar (v6)
Technical Report TR-xx.yy
DFKI GmbH, Saarbrücken Germany
URL: http://talkingrobots.dfki.de/tarot
DOI: to be provided.

Contact

Geert-Jan M. Kruijff
Language Technology Lab
German Research Center for Artificial Intelligence / DFKI GmbH
Campus D3_2, Saarbrücken
Germany

Email: gj@dfki.de

URL: http://talkingrobots.dfki.de

Tarot: The MOLOKO resource

About this resource

MOLOKO is a natural language grammar. It is defined using Combinatory Categorial Grammar (CCG) for its syntax, and a modal logic-based formalism for its semantics. MOLOKO is intended for use with an OpenCCG parser and/or realizer, i.e. you can use the MOLOKO in both "directions" for understanding and production.

As a CCG-style grammar, MOLOKO follows a lexicalized design. For each word in a lexicon, MOLOKO specifies a set of one or more families it belongs to. A family specifies a syntactic category, a semantic structure, and the interface between the syntactic and semantic structure. A word's family thus determines how it can figure in forming larger expressions (syntactic category), and how it helps to compose meaning (semantic structure).

The MOLOKO grammar is provided as a collection of files and scripts. The files define individual grammar components (lexical families, lexical entries; see below) using the DotCCG format, for use with the VisCCG editor. The scripts transform the DotCCG files into XML format, for use with the OpenCCG parser / realizer (for links see References below).

MOLOKO explicitly separates semantic structure, syntactic categories, and words (word forms). Although MOLOKO has been originally designed for English, this modularization makes it possible to re-use semantic structures and (some) syntactic categories for formulating grammars for languages other than English.

As for how MOLOKO got to its name: When we developed the first version of this grammar, we were working on robots which could among other things observe simple toys, with the eventual goal of manipulating them; See the CoSy project, http://www.cognitivesystems.org. At the time, the Anglo-Irish band Moloko had just released their album "Things to make and do" – which appropriately described what we tried to make our robot do. Furthermore, one of the kinds of toys the robot was particularly apt at observing was cows (courtesy of computer vision) – "moloko" is Russian for (cow) milk. Hence the name, which stuck ever since (even when nowadays, our robots are (also) exploring disaster sites, or assisting people in office environments, or playing games with kids ...)

Using this resource

The MOLOKO grammar is distributed with the LGPL license (like OpenCCG). This license covers academic and commercial use of parts or all of the resources in the MOLOKO grammar.

If you use part or all of the resources of the MOLOKO grammar in your project, we would appreciate the inclusion of the following reference in any work describing this use:

> G.J.M. Kruijff, P. Lison, T. Benjamin, H. Jacobsson, H, Zender, and I. Kruijff-Korbayová. "Situated Dialogue Processing for Human-Robot Interaction." In: H.I. Christensen, G.J.M. Kruijff, and J.L. Wyatt (Eds.). Cognitive Systems. Chapter 8, pp. 311-364, Cognitive Systems Monographs (COSMOS), Vol. 8, ISBN

978-3-642-11693-3, Springer Verlag, Berlin/Heidelberg, Germany, 2010.
http://www.cognitivesystems.org/cosybook/index.asp

The MOLOKO grammar can be used stand-alone with the OpenCCG `tccg` environment, using the OpenCCG API, or with the parsing- and/or realization-functionality available in Tarot (based on OpenCCG).

About this documentation

The goal of this documentation is to describe the details of the MOLOKO grammar implementation. It assumes familiarity with CCG, and the availability of a working OpenCCG `tccg` environment. For more about CCG and OpenCCG, see the References section.

References

For more about CCG, see the following articles:

- J. Baldridge. Categorial Grammar. To appear in Kiss, Tibor and Alexiadou, Artemis (eds.) Handbook of Syntax. Berlin: de Gruyter. To appear in 2011. http://www.jasonbaldridge.com/papers/baldridge_cg_handbook_syntax.pdf
- J. Baldridge and G.J.M. Kruijff. 2003. "Multi-Modal Combinatory Categorial Grammar." In Proceedings of EACL 2003. http://www.aclweb.org/anthology-new/E/E03/E03-1036.pdf
- J. Baldridge and G.J.M. Kruijff. 2002. "Coupling CCG with Hybrid Logic Dependency Semantics." In Proceedings of ACL 2002. http://www.aclweb.org/anthology-new/P/P02/P02-1041.pdf
- G.J.M. Kruijff. A Categorial-Modal Logical Architecture of Informativity. PhD thesis, Charles University. 2001. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.120.5403&rep=rep1&type=pdf

OpenCCG uses the multi-modal form of CCG described in (Baldridge & Kruijff 2003), with the semantics as per (Kruijff,2001; Baldridge & Kruijff 2002). For the original formulation of CCG, see the work by M. Steedman, notably The Syntactic Process (MIT Press: 2001).

For the OpenCCG API and documentation, including a guide to developing CCG grammars in the OpenCCG XML format, see the OpenCCG website at Sourceforge: http://openccg.sf.net.

The MOLOKO grammar is formulated in the DotCCG format, which is more readable than the native OpenCCG XML format. DotCCG grammars can be edited with the VisCCG editor. Both DotCCG and VisCCG are described here: http://comp.ling.utexas.edu/wiki/doku.php/openccg/ccggui_tut. The MOLOKO grammar distribution includes scripts for transforming a DotCCG grammar into the OpenCCG XML format.

1 Introduction

The documentation for the MOLOKO grammar provides an overview of the grammar, along with a technical specification of its rules and categories. As semantics are the heart of any CCG grammar, we begin by describing and motivating its key semantic structures.

The next three sections discuss a number of novel design features. We start with nominal constructions. We describe how the MOLOKO grammar allows adjectives, prepositions and adverbs to be modified and coordinated. Essentially, this involves giving, for example, adjectives only one lexical entry (adj_M) and then generating the others (n_T/n_T) via rules.

Next we deal with verbal constructions. We allow each verb to lexically specify which event modifiers (e.g. time, place, manner) can and cannot modify it. This allows a substantial decrease in the number of lexical entries for each verb. Moving on to clauses, we show how clause-level constructions can be added to the grammar to handle mood in an incremental fashion. It also overviews question semantics. In addition to the (general) verb-oriented structures, we look in detail at how the various copula and copula-like uses of the **be**-verb are treated. It includes a large collection of sample questions.

We have designed MOLOKO such that it can (though need not) be used with an incremental parser. We discuss how the grammar allows for creating semantically integrated, incremental parses. This is illustrated through the use of two detailed step-by-step examples.

The final two sections provide a technical account of the grammar itself. We outline the families, syntactic features, rules, dictionary macros, etc. which form the grammar signature. The final section gives a very brief practical guide to exploring, using and modifying the grammar.

2 Semantics

MOLOKO's semantics can divided into three broad categories: entities, events and modifiers. These correspond to the three propositional acts of reference, predication and modification. Of course, these categories must be treated in the broadest, most general of terms. Entities (T) correspond to not only to concrete objects, but any facet of experience which can be construed, or reified as a thing. Events (E) cover all types of dynamic processes, as well as ascriptions and states which endure over time. Finally, modifiers can be applied to any of the three basic categories (T, E, M) and they need not be inherit properties like size and color for T, or location, time and polarity for E–they also include evaluations, judgements and many other types of textual and interpersonal relationships.

2.1 The Semantic/Ontological Hierarchy

In order for the grammar to be of any use, the three broad semantic categories, T, E and M, require much finer levels of granularity. The MOLOKO grammar includes

a richly sorted ontological type hierarchy for each. Consider the following semantic representation for the utterance *quickly give the red ball to GJ*:

$$
\begin{aligned}
@_{g1:action-non-motion}(&\textbf{give} \\
&\wedge\ \langle Mood\rangle(imp) \\
&\wedge\ \langle Actor\rangle(a1 : animate \wedge\ \textbf{addressee}) \\
&\wedge\ \langle Patient\rangle(b1 : thing \wedge\ \textbf{ball}) \\
&\wedge\ \langle Delimination\rangle(unique) \\
&\wedge\ \langle Quantification\rangle(specific) \\
&\wedge\ \langle Num\rangle(sg) \\
&\wedge\ \langle Modifier\rangle(r1 : color \wedge\ \textbf{red})) \\
&\wedge\ \langle Recipient\rangle(g2 : person \wedge\ \textbf{GJ})) \\
&\wedge\ \langle Modifier\rangle(r1 : manner \wedge\ \textbf{quickly})))
\end{aligned}
$$

At the top level, we see that the event *give* (g1) is of type *action-non-motion*. At the next level, the three entities, i.e. the event participants (see below), *addressee, ball, GJ* are of type *animate*, *thing* and *person* respectively. Finally, the event modifier *quickly* is of type *manner* and the entity modifier *red* is of type *color*.

An important note concerning this semantic type hierarchy is that, for the most part, it is external to the grammar itself. What that means is the semantic sort given to a word does not dictate its treatment within the grammar proper. For example, the fact that kick is specified to represent an 'action-motion' event does not dictate its combinatorial possibilities. This is done, instead, by specifying the appropriate lexical family (here a 'dative' di-transtive). This is done primarily to allow easy, modular extension to the semantic hierarchy. So for example, if finer grain distinctions are needed amongst verb sorts, the grammar need not be (drastically) modified.

2.2 Semantic Relations: Modifiers versus Dependents

Given the quite broad definition of modifiers given above, one could view nearly any relationship as a modification. For example, a determiner like *the* modifies the entity of its head noun by delimiting and quantifying it. Likewise, a negation marker, as in "I did not put it there," can be seen as an event modifier (negative propositional semantics) and a 'intention' modifier (specifying that, counter some expectation, the speaker claims they *didn't* do it). Essentially then, whenever a word or construction adds an extra semantic dependency relation or feature to another independent semantic head, it can be viewed as a modifier. This stands in contrast to head-dependent relations, where a word fills in a missing slot within the semantic head. This is the case, for example, with verbs and their participants, prepositions and their anchors etc.

Although not much rides on this distinction between modifier and dependent, there are two cases in which this has greatly impacted the structure of the MOLOKO grammar. The first concerns participant or argument roles in event semantics and the second the treatment of the major open class modifiers(adjectives, prepositions,

and adverbs). These two special cases will be discussed in sections 2.4.1 and 3 respectively.

For a detailed discussion of modifiers versus dependents, and a dependency-based view on grammatical structure, see (Kruijff 2001) (cf. *References*, p.v).

2.3 Entity Semantics

In addition to their propositional head and its corresponding ontological sort, all entities are given the following levels of semantic structure:

1. Number
2. Specification
3. Delimitation

Where appropriate, entities also receive this additional semantic structure

1. Ownership and Compounding
2. Groups

We will also discuss pronouns in this section.

2.3.1 Number/Specification and Delimitation

The MOLOKO grammar models the semantic impact of "determiners" using two features: *Quantification*, and *Delimitation*. The Quantification feature is used to indicate a qualitative quantification of the entity type in terms of specificity, consistent with the singular/plural distinction defined by the Number feature. Typical values are *specific* (e.g. as in "the box") or *unspecific* (e.g. as in "some box"). The Delimitation feature is used to determine the identifiability of the entity/entities in context, with typical values being *existential* (e.g. "a box") or *unique* (e.g. "the box").

The reason for using Quantification and Delimitation to model the determination (information status) of entities in context is that this (arguably) provides a more general model for the interplay of syntactic structure with (semantic) information structure. We can use these features to aid in modeling information status regardless of whether we are looking at determiners, or at other morphosyntactic aspects, or even at the impact of intonation on presenting information status. Fixing this to a "definiteness" feature immediately derived from the specific determiner used would limit the application; (and disregards the fact that many languages do not even have or use determiners like English does).

The Number feature in the MOLOKO semantic structures represents semantic number.

For the sake of incrementality (see §7), all of this information is encoded in determiners and not in the nouns themselves. Thus, the full np *a cat* is formed by combining $@_{c1:animate}$(**cat**) with :

$$@_{x1:entity}(\\ \wedge\ \langle Delimination\rangle(existential)\\ \wedge\ \langle Quantification\rangle(specific)\\ \wedge\ \langle Number\rangle(sg)\ )$$

2.3.2 Ownership and Compounding

The semantic relationships which exist between the two objects underlying compounding and possessive constructions are rich and varied, and must be handled grammar externally. In each case, we have included only a single dependency relation to mark this connection: $\langle Owner\rangle$ and $\langle Compound\rangle$ respectively. We see both of these relations contributing to the semantic structure of *the edge of GJ's coffee mug*:

$$@_{e1:location}(\textbf{edge}\\ \wedge\ \langle Delimination\rangle(unique)\\ \wedge\ \langle Quantification\rangle(specific)\\ \wedge\ \langle Num\rangle(sg)\\ \wedge\ \langle Owner\rangle(t2 : thing \wedge\ \textbf{mug}\\ \wedge\ \langle Delimination\rangle(unique)\\ \wedge\ \langle Quantification\rangle(specific)\\ \wedge\ \langle Num\rangle(sg)\\ \wedge\ \langle Compound\rangle(c2 : thing \wedge\ \textbf{coffee})\\ \wedge\ \langle Owner\rangle(t2 : person \wedge\ \textbf{GJ})))$$

2.3.3 Groups

We handle the semantics of expressions like *some of GJ's money* and *the first of the three balls*, i.e. subsets or groups, using the following structure:

```
@b1:entity(prop ^
           ...
           <Subgroup>(s1:entity ^ subgroup ^
                                        ...))
```

So, for instance, *some of the balls* receives:

```
@b1:thing(ball ^
           <Delimitation>unique ^
           <Num>pl ^
           <Quantification>unspecific ^
```

```
            <Subgroup>(s1:entity ^ subgroup ^
                       <Delimitation>existential ^
                       <Num>pl ^
                       <Quantification>unspecific))
```

It is important that *ball* is the semantic head of this entity. Otherwise, it would be impossible for the grammar to identify its semantic sort. This is required, e.g., in semantic subcategorization.

Note how this allows treatment of the difference in semantics between *I want three of the books* and I want three books:

```
@b1:thing(book ^
          <Delimitation>unique ^
          <Num>pl ^
          <Quantification>unspecific ^
          <Subgroup>(s1:entity ^ subgroup ^
                     <Delimitation>existential ^
                     <Num>pl ^
                     <Quantification>specific ^
                     <Modifier>(t1:number-cardinal ^ three)))

@b1:thing(book ^
          <Delimitation>variable ^
          <Quantification>specific ^
          <Modifier>(t1:number-cardinal ^ three))
```

2.3.4 Pronouns

Here are some illustrations of the various forms and functions of pronouns:

1. *I am happy, they brought GJ the ball*
2. *give it to me, the robot picked it up and put it with them*
3. *my ball, what is your name*
4. *mine are over there, the girl already ate hers*

The examples in (1) and (2) illustrate full nominal uses, nominative and non-nominative respectively. All of these uses receive the same semantics: the nominative, singular form as **semantichead** with the number specified by the $\langle Num \rangle$ feature. These entities do not receive $\langle Delimination \rangle$ and $\langle Quantification \rangle$ [1]. Thus, the 1st person singular pronoun in any of these uses receives: $@_{i1:person}(\mathbf{I} \wedge \langle Num \rangle(sg))$. Note that *you* is ambiguous between singular and plural and thus receives two readings.

The examples in (3) are ownership marking determiners (possessive pronouns), and (4) shows owned entities. Both make use of the $\langle Owner \rangle$ role discussed above. Here are *their ball* and *theirs*:

[1] though they should.

$@_{b1:thing}$(**ball**
$\wedge$ $\langle Delimination \rangle (unique)$
$\wedge$ $\langle Quantification \rangle (specific)$
$\wedge$ $\langle Number \rangle (sg)$
$\wedge$ $\langle Owner \rangle (t1 : person \wedge$ **they** $\wedge$ $\langle Number \rangle (pl))$

$@_{c1:entity}$(**context**
$\wedge$ $\langle Delimination \rangle (unique)$
$\wedge$ $\langle Quantification \rangle (specific)$
$\wedge$ $\langle Number \rangle (sg)$
$\wedge$ $\langle Owner \rangle (t1 : person \wedge$ **they** $\wedge$ $\langle Number \rangle (pl))$

We need to make two comments about *theirs*. First, this form is ambiguous between the singular reading given and a variable, unspecific plural reading. Second, it makes use of a contextualized semantic head (see §2.6.3).

2.4 Event Semantics

In addition their propositional head and its corresponding ontological sort, events are given the following levels of semantic structure:

1. Participant roles
2. Tense, aspect, and modality (TAM), polarity, and voice
3. Mood
4. Modality

2.4.1 Participant Roles

Recall the semantic representation for *quickly give the red ball to GJ* given in §2.1 above. The two expressed entities (the red ball and GJ) are, together with the unexpressed 'addressee', participants in the event represented by this sentence. They play the roles of the $\langle Patient \rangle$, $\langle Recipient \rangle$, and $\langle Actor \rangle$ respectively. In terms of §2.2 above, they are dependents of the event. The word *quickly*, however, is not. It plays a modifying role in the event.

Although here the distinction is clear, in many cases it is not at all obvious how to classify a given constituent/semantic object. The standard syntactic and semantic tests (e.g. optional vs. obilgatory, core vs. periphery, etc.) are not, in practice, reliable or arguably, in theory, strongly defendable. Consequently, when it comes to making discrete decisions within a grammar there are many different views on where to draw

the participant/modifier line. We have decided upon a relatively 'modifier-friendly' approach, i.e. only a few constituents/semantic objects are treated as participants. The flip side of this is that we have had to implement a means of controlling the types of modifications that a given event (typically verb) allows. For example, the verb *give* allows modifiers of place and time but doesn't allow dynamic modifiers like whereto, and wherefrom (see §4 below).

In addition, we have chosen to only include a small number of different participant role types or labels: ⟨*Actor*⟩, ⟨*Patient*⟩, ⟨*Recipient*⟩, ⟨*Result*⟩, ⟨*Event*⟩ for standard verbs, and a few extras for the *be*-copula (see §6). Again, the idea is that the specifics of participant role interpretation, i.e. what it 'means' to kick or be kicked, to run, or to be 'put on a table', are handled outside the grammar. As the grammar was designed for use in embodied agents, this is of course fully intentional. Note, however, that because of this decision, we do not allow much in the way of semantics internal inferencing (c.f. a semantics employing rich frame-based role structure). In our case, these types of inferences must be handled outside the grammar.

A final note is that events specify only the number and (top-level) semantic type (E, T or M) of their participants. Only in a few cases do they subcategorize based on finer grained semantic levels. For example, the verb *put* specifies that it's result is a dynamic whereto location but di-transtive recipients are not constrained to animate referents. The grammar was built with the expectation that the majority of these 'appropriateness' restrictions are handled outside the grammar.

Many of the participant roles are based on Prague School functionalist views (Sgall, Hajičová, Panevová). They are described in detail in (Kruijff 2001) (cf. *References*, p.v).

2.4.2 Tense/Aspect/Polarity

Events are currently categorized along the traditional lines of tense (past, present, future), grammatical (as compared to lexical) aspect (the continuous and perfect dimensions), polarity (positive, negative) and voice (active, passive) using semantic features.

Tense is always marked, whereas aspect, polarity and voice rely on the idea of unspecified defaults: only the 'marked' forms are marked. For polarity, if no negative particle/auxillary occurs the event is 'implicitly' postive but does not receive a feature marking this. The same is true for voice - if a passive construction is not used, the event is 'implicitly' active- and for aspect-if the event is not marked for progressive or for perfective, it simply receives no aspectual feature, instead receiving the default (imperfective, non-progressive) interpretation.

So, for example, here are *it got taken*[2] (passive, non-continuous, imperfective, past, positive) and *she isn't coming* (active, continuous, imperfective, present, negative):

```
@t1:action-non-motion(take ^
                      <Mood>ind ^
                      <Tense>past ^
```

[2]Both *get* and *be* passives receive the same semantic treatment. The *Actor* has been filled using a contextualized semantic object (see §2.6.3 below).

```
                    <Voice>passive ^
                    <Actor>(c1:entity ^ context) ^
                    <Patient>(i1:thing ^ it ^ <Num>sg) )

@c1:action-motion(come ^
                  <Aspect>continuous ^
                  <Mood>ind ^
                  <Polarity>neg ^
                  <Tense>pres ^
                  <Actor>(s1:person ^ she ^ <Num>sg) )
```

2.4.3 Mood

Mood–the classification of utterances into imperatives, interrogatives and indicatives– is not a property of events, and hence not a component of what is typically referred to as 'propositional semantics' (or ideational meaning). Nevertheless, as the various layers of meaning, e.g. ideational, interpersonal, textual etc. have not been separated out in the MOLOKO grammar, this interpersonal feature is attached as a feature to the event. For open interrogatives, a dependency relation $\langle Wh - Rest \rangle$ is added at this level. It specifies the nature and scope of the question. This will be discussed in detail in §5.

2.4.4 Modality

Depending on what kind of word is contributing this meaning, the encoding of modality is handled in one of two ways. For pure modal-auxiliary verbs (*can, should, must* etc.) this is marked by adding a $\langle Modifier \rangle$ dependency relation with appropriate contents to the main event. Here is semantic representation for *can you walk*.

$$@_{w1:action-motion}(\textbf{walk} \wedge \langle Mood \rangle(int) \wedge \langle Actor \rangle(y1 : animate \wedge \textbf{you} \wedge \langle Num \rangle(sg)) \wedge \langle Modifier \rangle(c1 : ability \wedge \textbf{can})))$$

Most modal or modal-like words (*want, need, try, have(to), continue, keep, try, stop, be able, be willing, etc.*), however, have been handled as main events with their scoped over event occupying the role $\langle Event \rangle$ (note the co-indexing of I).

$$@_{w1:????}(\textbf{want} \wedge \langle Mood \rangle(ind) \wedge \langle Actor \rangle(i1 : animate \wedge \textbf{I} \wedge \langle Num \rangle(sg)) \wedge \langle Event \rangle(w1 : action - motion \wedge \textbf{walk} \wedge \langle Actor \rangle(i1 : animate))$$

This was done for two reasons. First, whereas no other modal can scope over a pure modal, they can scope over semi-modals(e.g. ** I want to can , * I tried to*

should but *I can keep running*. This can consequently lead to major differences in propositional meaning based on changes in word order, i.e. *I wanted to keep running* vs. *I kept wanting to run*). Second, many of these verbs also have object-controlled readings which require an $\langle Actor \rangle$, thus further motivating this treatment. e.g. *I want him to walk*.

$$
\begin{aligned}
@_{w1:????}(&\textbf{want} \\
&\wedge\ \langle Mood \rangle(ind) \\
&\wedge\ \langle Actor \rangle(i1 : animate \wedge\ \textbf{I} \wedge \langle Num \rangle(sg)) \\
&\wedge\ \langle Patient \rangle(h1 : animate \wedge\ \textbf{him} \wedge \langle Num \rangle(sg)) \\
&\wedge\ \langle Event \rangle(w1 : action - motion \wedge\ \textbf{walk} \\
&\wedge\ \langle Actor \rangle(h1 : animate))
\end{aligned}
$$

Note that in each of these examples the modality has, like all semantic objects, been ontologically subcategorized.

2.5 Modifier Semantics

Consider some examples of modifiers.

1. $\langle Modifier \rangle$(i1:whereto $\wedge$ **into** $\wedge$ $\langle Anchor \rangle$(m1:entity $\wedge$ **mug**))
2. $\langle Modifier \rangle$(y1:time-point $\wedge$ **now**)))
3. $\langle Modifier \rangle$(o1:time-point $\wedge$ **on** $\langle Anchor \rangle$(t1:day $\wedge$ **Tuesday**)))
4. $\langle Modifier \rangle$(w1:instrumental $\wedge$ **with** $\langle Anchor \rangle$(t1:thing $\wedge$ **ball**)))
5. $\langle Modifier \rangle$(q1:manner $\wedge$ **quickly**)))
6. $\langle Modifier \rangle$(a1:frequency $\wedge$ **always**)))

7. $\langle Modifier \rangle$(r1:color $\wedge$ **red**)))
8. $\langle Modifier \rangle$(b1:size $\wedge$ **big**)))
9. $\langle Modifier \rangle$(f1:number-cardinal $\wedge$ **five**)))

10. $\langle Modifier \rangle$(i1:location $\wedge$ **on** $\wedge$ $\langle Anchor \rangle$(t1:entity $\wedge$ **table**))
11. $\langle Modifier \rangle$(w1:accompaniment $\wedge$ **with** $\langle Anchor \rangle$(g1:person $\wedge$ **GJ**)))

12. $\langle Modifier \rangle$(r1:degree $\wedge$ **really**)))
13. $\langle Modifier \rangle$(m1:degree $\wedge$ **much**)))

Most modifiers in MOLOKO are subsumed under one dependency relation named ⟨*Modifier*⟩. Functional subcategorization has been moved to the semantic/ontological sort of the propositional head of the modifier. This is true for event modifiers (1-6), entity modifiers (7-9), event/entity modifiers (10-11) and modifier modifiers (12-13). This is preferable to specifically labeled relations (⟨*Location*⟩ , ⟨*Time*⟩ , ⟨*Property*⟩ , ⟨*Instrument*⟩ etc.) for a number of reasons. First, it provides a uniform treatment of a wide variety of phenomena. Second, it is easily extendable: all that is required is the modification of the semantic/ontological hierarchy. Third, it allows for the hierarchic grouping of relations.

In addition to their propositional head and sort, when appropriate modifiers receive a ⟨*Degree*⟩ value of comparative or superlative. The *default* value is the base value.

2.6 Miscellaneous Semantic Issues

2.6.1 Quantifier and Modifier Scope

The scope of quantifiers and modifiers is given only a basic treatment in the MOLOKO grammar. In general, these operators scope directly over the appropriate semantic object, i.e they attach to it. Thus, clause level negation attaches directly to the main event, nominal quantifiers attach to entities, etc. Moreover, the differences in scope and/or information structure associated with the ordering of such operators is not treated either, e.g. *I am normally happy* and *normally I am happy* receive the same semantic representation.

For more detailed accounts of quantifier scope phenomena and their treatment in CCG, see recent work by M. Steedman; http://homepages.inf.ed.ac.uk/steedman/papers.html.

2.6.2 Proximity

Both the determiner and deictic uses of *this* and *that*, as well as *here* and *there* are handled by contributing the semantic feature ⟨*Proximity*⟩ with values *proximal* and *distal* respectively. Due to issues with DotCCG, we have three separate sets of values corresponding to entities, modifiers and events.

The use of these features is described in detail by Kelleher & Kruijff in various papers, e.g. http://dl.acm.org/citation.cfm?id=1220269.

2.6.3 Contextualized Semantic Objects

The MOLOKO grammar has been designed for use in situated interaction. If it is to handle language in such environments, clearly it must be sensitive to the relevant phenomena. In this section, we will mention a few related cases, all of which have been handled by 'filling in' a semantic role with the head **context**.

To begin, people often 'leave out' linguistic material which traditional grammars mark as necessary. This is particularly true in both situated and highly interactional

settings. To handle such grammatically fragmentary but interactionaly complete utterances, we have added a number readings which 'contextualize' the relevant compliment slots.

Consider this simple hypothetical exchange.

- 1 User: "go get me a coffee"
- 2 Robot: "sure"

Robot goes to kitchen and returns

- 3 User: "hi robot did you get that coffee for me"
- 4 Robot: "um I tried to but I couldn't"
- 5 Robot: "I didn't see any mugs"
- 6 User: "didn't you"
- 7 Robot: "no"
- 8 User: "well try to do it again please"
- 9 Robot: "ok I will"

The event **Robot get coffee for User** is first introduced as an imperative request by the user at line 1. Beginning at line 3, it is re-envoked and then tossed back in forth between the two participants in lines 4, 8 and 9, i.e. it is 'interpersonally argued' with only the mood, tense, aspect, polarity, and other modality like features changing from line to line. Similarly with **Robot didn't see mugs** in 5 and 6 (and possibly 7). What is crucially important here is that many of the resulting utterances do not contain full clauses, i.e. the verbal compliments corresponding to this event are not expressed.

As an illustration of how we have handled this, here are the semantic structures corresponding to the various 'minor clauses' used in the dialogue above: [3]

I tried to but I couldn't

```
@b1:event(but ^
         <First>(t1:modal ^ try ^
                 <Mood>ind ^
                 <Tense>past ^
                 <Actor>(i1:person ^ I ^ <Num>sg) ^
```

[3] One important exception to this pattern is in the case of modal-like verbs whose event comp either cannot or has not been marked by an infinite *to*. Compare e.g., *I started to* and *I started*. In the latter case, the semantic structure does not receive an $\langle Event \rangle$ compliment and hence does not receive a **context** head.

```
                <Event>(c1:event ^ context) ^
                <Subject>i1:person) ^
        <Next>(c2:event ^ context ^
               <Mood>ind ^
               <Polarity>neg ^
               <Modifier>(c3:modal ^ could) ^
               <Subject>(i2:person ^ I ^<Num>sg)))
```

I will

```
@c1:event(context ^
          <Mood>ind ^
          <Tense>fut ^
          <Modifier>(w1:modal ^ will) ^
          <Subject>(i1:person ^ I ^ <Num>sg))
```

didn't you

```
@c1:event(context ^
          <Mood>int ^
          <Tense>past ^
          <Subject>(y1:person ^ you ^ <Num>sg))
```

In addition to those cases where an event compliment is fully unexpressed, it can also be referred to using a pronoun, as in *try to do it again please*. This has also been handled using a **context** head:

```
  @t1:modal(try ^
          <Mood>imp ^
          <Actor>(a1:entity ^ addressee) ^
          <Event>(d1:event ^ do ^
                  <Actor>a1:entity ^
                  <Event>(c1:event ^ context)) ^
          <Modifier>(a2:m-frequency ^ again) ^
          <Modifier>(p1:m-comment ^ please) ^
          <Subject>a1:entity)
```

Similarly, there are number of constructions in which **entities** also have either unexpressed (elided) or pronoun/deictic contextualized semantic heads. We have already seen such usage for ownership pronouns (§2.3.4) and the ⟨*Actor*⟩ in passive constructions (§2.4.2). Here are a few additional examples: *this*, *these two* and *the green one*.

```
@c1:entity(context ^
           <Delimitation>unique ^
           <Num>sg ^
           <Proximity>proximal ^
           <Quantification>specific)
```

```
@c1:entity(context ^
           <Delimitation>unique ^
           <Num>pl ^
           <Proximity>proximal ^
           <Quantification>specific ^
           <Modifier>(t1:number-cardinal ^ two))

 @c1:entity(context ^
           <Delimitation>unique ^
           <Num>sg ^
           <Quantification>specific ^
           <Modifier>(g1:q-color ^ green))
```

The same holds for **locational** and **temporal** deictics. Consider the readings for *put it there, the table in here* and *I haven't seen it since then.* [4]

```
@p1:action-non-motion(put ^
                      <Mood>imp ^
                      <Actor>(a1:entity ^ addressee) ^
                      <Patient>(i1:thing ^ it ^ <Num>sg) ^
                      <Result>(c1:m-whereto ^ context ^
                               <Proximity>m-distal) ^
                      <Subject>a1:entity)

@t1:thing(table ^
          <Delimitation>unique ^
          <Num>sg ^
          <Quantification>specific ^
          <Modifier>(i1:m-location ^ in ^
                     <Anchor>(c1:e-location ^ context ^
                              <Delimitation>unique ^
                              <Num>sg ^
                              <Proximity>proximal ^
                              <Quantification>specific )))

@s1:perception(see ^
               <Polarity>neg ^  <Mood>ind ^
               <Aspect>perfect ^  <Tense>pres ^
               <Actor>(i1:person ^ I ^  <Num>sg) ^
               <Modifier>(s2:m-time-point ^ since ^
                          <Anchor>(c1:e-time-unit ^ context ^
                                   <Delimitation>unique ^
                                   <Num>sg ^
                                   <Quantification>specific) ^
```

[4] *then* and *now* could be marked for and hence distinguished by ⟨*Proximity*⟩

```
                <Patient>(i2:thing ^ it ^ <Num>sg) ^
                <Subject>i1:person)
```

2.6.4 Role-defined entities and modifiers

We have treated relative clauses by adding a specific modifier relation $\langle Role - In \rangle$ to the entity which contains the semantics of the clause, i.e. the event that this entity plays a role in. The restricted entity is co-indexed with the appropriate event role. Here is *ball that I took* and *ball that I wanted to take*.

```
@b1:thing(ball ^
          <Role-in>(t1:action-non-motion ^ take ^
                    <Tense>past ^
                    <Actor>(i1:person ^ I ^
                            <Num>sg) ^
                    <Patient>b1:thing ^
                    <Subject>i1:person)

@b1:thing(ball ^
          <Role-in>(w1:cognition ^ want ^
                    <Tense>past ^
                    <Actor>(i1:person ^ I ^
                            <Num>sg) ^
                    <Event>(t1:action-non-motion ^ take ^
                            <Actor>i1:person ^
                            <Patient>b1:thing) ^
                    <Subject>i1:person)
```

Note that these 'minor clauses' have no mood marking. They do however have a $\langle Subject \rangle$. This is to insure full semantic integration during their incremental parsing.

They are not fully functioning (and hence turned off in the grammar) but we are near to having similar readings for modifiers like *where I wanted to walk*:

```
@w1:m-location(where ^
               <Scope>(w2:action-motion ^ walk ^
                       <Actor>(i1:person ^ I ^
                               <Num>sg) ^
                       <Role-in>(w3:cognition ^ want ^
                                 <Tense>past ^
                                 <Actor>i1:person ^
                                 <Event>w2:action-motion ^
                                 <Subject>i1:person)))
```

This could then allow sentences like *put it where I told you to put it*, *I went where you wanted me to go*, *I don't know where it is*, *I saw what you picked up*, etc.

2.7 Location

To Do: e, q, m location: q and m should be collapsed, in fact all should
static vs. dynamic
chains of locations: the cup is in the box in the room vs. the cup is in the room, in the box. Also chains of dynamics. go out the door around the corner up the stairs ....
All handled by forward projecting rules 8.9. The correct ordering must be sorted out grammar externally
specifying location (prepositions on events, entities)
questioning location: where, which place, which room

2.8 Time

To Do: discuss e, q, m time
distinguish between sequence, point, interval
describe different options (after you came, after that day, after then, afterwards)
questioning time: when, how long
I walked for five minutes

```
@w1:action-motion(walk ^
                  <Mood>ind ^ <Tense>past ^
                  <Actor>(i1:person ^ I ^ <Num>sg) ^
                  <Modifier>(f1:m-time-interval ^ for ^
                            <Anchor>(m1:e-time-unit ^ minute ^
                                     <Delimitation>variable ^
                                     <Quantification>specific ^
                                     <Modifier>(f2:number-cardinal ^ five)))
```

I came after you went

```
@c1:action-motion(come ^
                  <Tense>past ^ <Mood>int
                  <Actor>(i1:person ^ I ^ <Num>sg) ^
                  <Modifier>(a1:m-time-sequence ^ after ^
                            <Event>(g1:action-motion ^ go ^
                                    <Tense>past ^
                                    <Actor>(y1:person ^ you ^ <Num>sg) ^
                                    <Subject>y1:person)) ^
                  <Subject>i1:person)
```

how_long have you been sitting there

```
@s1:action-non-motion(sit ^
                      <Aspect>perfect ^ <Aspect>continuous ^
                      <Mood>int ^ <Tense>pres ^
                      <Actor>(y1:person ^ you ^ <Num>sg) ^
                      <Modifier>(c1:m-location ^ context ^
```

```
                                    <Proximity>m-distal) ^
                          <Wh-Restr>(h1:m-time-interval ^ how_long ^
                                    <Scope>s1:action-non-motion))
```

when did you come in

```
@c1:action-motion(come ^
                  <Mood>int ^ <Tense>past ^
                  <Actor>(y1:person ^ you ^ <Num>sg) ^
                  <Modifier>(i1:m-direction ^ in) ^
                  <Wh-Restr>(w1:m-time-point ^ when ^
                            <Scope>c1:action-motion))
```

3 Dual-Relation Words

In §2.2 we noted the distinction between semantic modifiers and semantic dependents. The open class modifiers, i.e. adjectives, prepositions, and to a lesser degree adverbs, are special in that they need to be able to play both of these roles, i.e. they are dual-relation-words. Consider the following sentence pairs:

- I wrote the letter on the table
- I put the picture on the table (Caused Motion)
- I wanted the bigger picture
- I made the picture bigger (Resultant Verb)

In the first sentence of each pair, the underlined constituents are acting as a modifier. The sentences would be semantically and syntactically complete without them:

- I wrote the letter
- I wanted the picture

Compare this second set of sentences. Here, the same words are somehow 'essential' to the sentences. Removing them leads to syntactic (and semantic) incompleteness.

- I put the letter xxx
- I wanted the picture xxx

To handle the dual functionality of such words they must be able to behave in two grammatically distinct ways. We will first consider how this is handled in traditional CCG and outline its limitations. We will then describe how these are dealt with in MOLOKO.

3.1 Dual-Relation words in traditional CCG

In traditional CCG, the dual functionality of such words is handled by assigning them two entirely separate lexical families. So, for example, the adjective *bigger* would receive:

1. adj_{M} : $@_{b1:size}(\ \wedge$ **big** $\wedge\ \langle Degree\rangle$(comparative))
2. $\text{n}_{\text{T}}/\text{n}_{\text{T}}$:
   $@_{t1:entity}(\ \langle Modifier\rangle$(b1:size $\wedge$ **big** $\wedge\ \langle Degree\rangle$(comparative)))

In the first case, *bigger* is given an atomic category adj and thus provides it with its own semantic head *M*. This variable can then be selected for, and used to fill a dependency role, allowing *bigger* to act as a dependent. Consider again the sentence *I made it bigger* mentioned earlier. At the point in the incremental parsing of this sentence, just before the word *bigger* is reached, we have this partial parse

parse: s/adj

$$@_{m1:action-non-motion}(\textbf{make} \wedge\ \langle Mood\rangle(ind) \wedge\ \langle Actor\rangle(i1 : person \wedge\ \textbf{I} \wedge\ \langle Num\rangle(sg)) \wedge\ \langle Patient\rangle(b1 : thing \wedge\ \textbf{it} \wedge\ \langle Num\rangle(sg)) \wedge\ \langle Result\rangle x1 : quality)$$

In order for this sentence to receive a complete parse, syntactically it requires an adjective (atomic cat adj) to its right, an adjective whose nominal variable would semantically fill its currently empty $\langle Result\rangle$ role. When the word "bigger" is encountered next, lexical reading 1) above is employed, and the variable *M* fills this missing role.

As for the second reading above, "bigger" simply adds its semantic content to the entity being modified (in this case *T*, the entity designated by the noun). This is much clearer if we consider how CCG, as a *dependency* grammar, treats modifiers. Syntactically, modifiers are treated as identity functions. They take an argument of category *X* and return the same category *X*. This is done by giving modifiers a complex category, one which specifies both the modified entity's syntactic category and its combinatorial possible locations (via its slash direction and mode). The intuition here, is that despite their semantic differences, syntactically "dog" behaves identically to "big dog," or "big scary dog" or even "big scary dog running right towards you." It is instead on the semantic side that modifiers do their work. They take the nominal variable associated with their modified entity and attach some new semantic content to it, that is they modify it.

Unfortunately, by creating two separate readings –one for each relation– we are forced into a few unexpected limitations. First, consider the following two sentences, where the examples from §2.2 have been slightly altered by the addition of the word "much:"

- I made it much bigger
- I wanted the much bigger picture

Cases like these can be handled quite simply by treating the word "much" as an adjective pre-modifier (Note: of type *degree*)

adj_M/adj_M : $@_{m1:modifier}$($\langle Modifier \rangle$(m2:degree $\wedge$ **much**))

This then combines with "bigger" creating "much bigger," a new adjectival construction whose nominal variable M is given the semantic content of "much:"

$$@_{b1:size}(\textbf{big} \wedge \langle Degree \rangle(comparative) \wedge \langle Modifier \rangle(m2 : degree \wedge \textbf{much}))$$

This can then act as a dependent, e.g. filling the $\langle Resultant \rangle$ role in the semantics for "made," in exactly the same way as a simple, unmodified adjective.

It is in the second sentence that we encounter a problem. As we saw above, modifiers require a nominal variable to attach their modifying semantic content to. This is true of any linguistic unit acting as modifier, whether it is a single word like "big" or "quickly," or a larger expression like "on the table," "that I told you about" or "when you come back." It is also true regardless of the type of the modified entity. This was the case in §2.2, and also with "much" directly above. Because the atomic category version of adjectives already has a nominal variable (to be selected for as a dependent), it can easily be modified.

However, this is not the case for the modifier reading of adjectives, prepositions and adverbs. Currently, complex categories do not receive their own nominal variables in OpenCCG. So in the case of "bigger" above, the nominal variable T can be used to refer to the modified entity, but unlike the dependent-version in 1), there is no separate nominal variable refers to the adjective as a whole. Consequently, there is no way to modify an adjective operating in this way; or more generally, it is impossible to modify a modifier.

This is a significant problem. It rules out the possibility of parsing, or generating sentences like these:

- Adverb Modifiers: go to the kitchen very slowly, please don't do it so quickly
- Adjective Modifiers: I want the really big one
- Preposition (pp) Modifiers: walk over to GJ, walk up to the table

A second highly similar problem is that without semantic heads, modifiers performing the modifying role cannot be conjoined, i.e. sentences like "the green and yellow ones," "put it on the table quickly but carefully" and "I walked in the room and around the desk" cannot be handled.

3.2 Dual-Relation Words in MOLOKO

These limitations are addressed in the MOLOKO grammar through the use a single reading for dual-relation words in combination with type-changing rules. Adjectives, prepositions and adverbs are given only an atomic category reading, and consequently always have a nominal variable, allowing them to serve as dependents (including being modified). To perform their modifying function, these basic categories can be expanded into their complex combinatorial form. For example, MOLOKO contains this type-changing rule for adjectives:

$$\text{adj}_M \Longrightarrow n_T/n_T : @_{t1:entity}(\ \langle Modifier\rangle(\text{m1})\ )\ )$$

The result is the familiar pre-nominal adjective reading, but instead of the $\langle Modifier\rangle$ relation receiving its contents off-line (directly from the lexicon), this rule dynamically copies the contents of the atomic version's nominal variable M into the open dependency relation. In some sense it turns modification into some form of bizarre dependency. What is crucial here is that this rule acts on any linguistic constituent with category *adj* encountered during parsing, including already modified *adj*'s.

Thus, while parsing "I want the really big one," the rule can operate either on "big" or on "really big"[5]

By continually using adjectives as an example, we have masked a very important fact: most modifiers have a variety of combinatorial possibilities. Consider the adverb "normally." It can placed in a variety of locations within the clause:

- *normally* you can get it from GJ
- you *normally* can get it from GJ
- you can *normally* get it from GJ
- you can get it from GJ *normally*

These different syntactic permutations are handled by giving "normally" multiple readings, each with a different syntactic form. In the standard CCG approach, this is done by giving these words multiple lexical entries. Of course this leads to the same problems mentioned above. In the MOLOKO grammar, we handle this variability by creating a type-changing rule for each syntactic possibility. So for example, here are two adverb type-changing rules:

- $\text{adv}_M \Longrightarrow s_E/s_E : @_{e1:event}(\ \langle Modifier\rangle(\text{m1})\ )\ )$
- $\text{adv}_M \Longrightarrow s_E \setminus s_E : @_{e1:event}(\ \langle Modifier\rangle(\text{m1})\ )\ )$

However, this still leaves one major issue unresolved: different words within the same syntactic class can behave differently. Compare the adverb "normally" to the adverb "too" (with the meaning of "also"):

[5] In actuality, it operates on "really" alone. See the second example in the section on Incrementality, §7.

- * too you can get it from GJ
- you too can get it from GJ
- ? you can too get it from GJ
- you can get it from GJ too

To account for these idiosyncracies, each word must somehow specify its behavior. Again, in standard CCG this can be done easily by assigning a word to only its appropriate families. In MOLOKO, this is done instead by 'naming' each type-changing rule and for each word, specifying lexically which rules can apply to it. We could have created a binary syntactic feature for each rule, and thus basically would specify + values for each feature, and too would specify - for the rules it wishes to block.

However, to avoid this explosion of features, all of this information is instead contained in a single syntactic feature called *cc-type* (complex-category type). Here is a more detailed look at some of the rules used for adverbs:

- $\text{adv}_{\text{M},cc-type=pre-s} \Longrightarrow$
  $\text{s}_\text{E}/\text{s}_\text{E}$: $@_{e1:event}(\ \langle Modifier \rangle(\text{m1})\ )\ )$
- $\text{adv}_{\text{M},cc-type=pre-s} \Longrightarrow$
  $\text{s}_\text{E} \setminus \text{s}_\text{E}$: $@_{e1:event}(\ \langle Modifier \rangle(\text{m1})\ )\ )$

These individual, rule-specific feature values are then grouped (via multiple inheritance) in the syntactic feature hierarchy. This creates a set of syntactic classes, each specifying exactly which rules can apply (and can't apply) to words belonging to this class, and hence enforces the appropriate combinatorially behaviors.

Each of the Dual-Relation parts of speech (adj, adv and prep) has its own set of syntactic classes (e.g. adv1, adv2, adv3, etc), and each word from these POS belongs to one of these classes. This is specified lexically via macros.

4 Event Modifier Restriction

We noted earlier that the MOLOKO is quite promiscuous when it comes to semantic or pragmatic un-acceptability, i.e. that these notions must be handled outside of the grammar. There is, however, one major exception to this design principle: verbs are able to lexically specify what kinds of event modifiers they can receive. This is the counterpart to treating most constituents as modifiers instead of as dependencies (see §2.4.1). In this section, first, we will give some examples illustrating the utility of this design feature, and then go into detail on how this was implemented.

4.1 Motivation

Consider first this pair of sentences:

1. I played in the room

2. I walked in the room

Despite their similarity, these two sentences differ in that the second allows a reading which the first does not. In both cases, "in the room" can be a static locational modifier, specifying the place where the event occurred.[6] However, only in (2) can "in the room" specify the dynamic goal of the event, i.e., walking *into* the room. There is a case for arguing that this difference in modificational behaviour is inherit in the verb and should thus be lexically specified (c.f. Goldberg). As another example, consider

- * I am a ball to the door

The unacceptability of this sentence can be accommodated by lexically specifying that the copula verb (be) blocks the class of dynamic modifiers. Similarly, in the sentence

- * I want you to play in the room.

we can easily block the reading where "in the room" modifies "want," i.e., the reading where it was the wanting that occurred within the room, instead of the true reading where it is the playing in the room that is wanted.

Another, perhaps less obvious, application of this feature is the blocking of certain questions. It can be used to correctly allow the first example to have two readings, the second example one reading, and the third none.

1. where did you walk
2. where did you play
3. * where did you want

It is clear that we cannot and should not attempt to grammatically specify every allowable type of event modification. For one, the type of information that underlies these differences certainly falls under the realm of world knowledge. More importantly, for nearly any restriction we could specify, we could most likely find a set of contexts where these restrictions would no longer apply. Language is after all a tool used by humans who have a remarkable gift for construing situations in novel and unexpected ways. However, practically, the ability to grammar-internally rule out some bizarre or highly unlikely readings is a powerful one. It can alleviate a great deal of burden on parse pruning by limiting what readings we are willing to even consider within the context the task for which the grammar is being employed.

4.2 Implementation

In fact, the restriction of event modifiers is handled almost identically to the way the the combinatorial behavior of dual-relation words is handled (see §3.2 above). It is done through the use of

[6] For the example in (2), imagine a person pacing.

1. a pair syntactic features which are attached lexically to modifiers and verbs

2. the inclusion of these features within the syntactic category responsible for attaching the modifier to the modified entity. This includes the type-changing rules used to handle the modifier function of adverbs and prepositions and also subordinate clause modifiers.[7]

4.2.1 Syntactic Features

The first syntactic feature, m-type (modifier type), is attached to the event modifier and essentially parrots the modifier's semantic sort. For example, the preposition "into," of semantic type *m-whereto* is assigned a mod-type value of *s-whereto*, etc. A new feature with the prefix 's-' (for syntactic) is used here because it appears OpenCCG does not like using the same feature for multiple purposes.

The second feature, *m-class*, is attached to the verb, and specifies exactly which types of modifiers can attach to it. Just like *complexcat*-class, this is done in the syntactic type hierarchy by grouping the *mod-type* values (via multiple inheritance). For example, consider this type definition:

m-class-4 [s-frequency s-probability s-comment s-time]

This class is very restrictive, allowing only frequency ("always"), probability ("certainly"), time ("in five minutes") and comment ("please") modifiers, blocking all others (location, dynamic, manner, etc.) Both of these features are specified lexically via macros. For a full list of the current mod-types and mod-classes see the file `types-feature.ccg`.

4.2.2 Syntactic Category Modification

In the case of prepositions and adverbs (i.e. the dual-relation words able to serve as event modifiers), we simply expanded the type-changing rules to enforce these restrictions. For each of the n different syntactic variants, we added a rule for each of the m different m-types. Thus, instead of their being only n rules, there are now n x m rules, each specifying a particular syntactic variant and a specific modifier type. Here is an example for event modifying prepositions:

$$\mathrm{pp}_{\mathrm{M},cc-type=post-s,m-type=m-whereto}\$_1$$
$$\Longrightarrow \mathrm{s}_{\mathrm{E}}/\mathrm{s}_{\mathrm{E},m-class:s-whereto}\$_1 : @_{e1:event}(\ \langle Modifier\rangle(\mathrm{m1})\ )\ )$$

[7] Although event modifier restriction is surely a matter of semantics and not syntax, there are two reasons why it is better handled via feature. First, OpenCCG much better at accessing syntactic features than semantic ones. In fact, once semantic content has been added the semantic representation of a parse, it is no longer 'visible' to the parsing at all. Second, as noted in above, one of the design features for MOLOKO was to keep the semantic sorts as grammar external as possible. Using the event-type to control modification would violate this.

This rule handles dynamic "whereto" prepositions that modify a sentence on its right, e.g., the preposition "into" in the sentence "I went into the room." Note also that the modified sentence is marked with the feature *m-class:s-whereto*. This ensures that only those verbs which belong to a mod-class which allow (can unify with) whereto modifiers can actually be modified by this.

For other event modifier categories, such as subordinate clause markers ("when," "while," etc), again, the appropriate modifier type was assigned to the modified entity's *mod-class* value.

It is important to remember, that because the semantic classes used for modifiers have actually been built into the grammar signature (by creating features and rules), any further modification to this portion of the semantic type hierarchy would lead to potential inconsistencies within the grammar. Thus, any such changes would need to be reflected within the grammar itself as well. See `types-feature.ccg` for details.

5 Mood

The interpersonal feature ⟨*Mood*⟩ and its syntactic encoding have been given a quite detailed and slightly unorthodox treatment in the MOLOKO grammar. The feature is not uniformly specified by the first/main verb, but has instead been distributed throughout the grammar, with each mood value receiving a different treatment. Before detailing how this has been done, we briefly describe the motivation behind this choice.

5.1 Motivation: Mood in English

At least in English, the syntactic marking of mood is best thought of as clausal-level feature specified by the presence of and/or sequential ordering of the following core clausal elements:

1. the grammatical subject
2. the main verb
3. the auxiliary verb
4. a question ("wh*") word

For example in Construction Grammar, the mood of a clause is specified by a number of Clause-Level Word-Order Constructions. Consider these simple clauses:

1. I picked up the ball.
2. pick up the ball.
3. did you pick up the ball
4. what did you pick up OR who picked up the ball

Table 1: Identifying Mood

Mood	Initial Element	Example
Indicative	Noun-Phrase (in nominal case)	I
Imperative	Verb (in base form)	pick
Y/N-Interrogative	Auxiliary Verb(in finite form)	did
Wh-Interrogative	Wh-Word	what

Now consider the mood and initial core element of each clause in this table. Quite remarkably, in all four cases, we can identify the mood of the clause by considering only its first core element. In other words, the first core element of a clause serves as a cue, projecting the mood of the clause to come. This is important for increasing the effectiveness of incremental processing in two ways.

First, when the mood of a clause is known, a lot is known about its core behavior both syntactically and functionally. This is much like the case of verbs and their arguments: when we encounter the verb *put*, we know that a np (the patient) and a prepositional phrase (the result) are to follow. In the case of mood, an indicative clause, for example, consists of the subject-np followed by a fine main verb (either an auxilliary or a lexical verb and its arguments). Thus, when we encounter an initial (nominative) np , if it is the subject-np of an indicative clause, then necessarily the next expected (core) element is the 'verb phrase'. This fits with the general early projection principle (see §7.2).

Secondly, the mood of a clause is highly connected to the discourse function it may be playing (assertion, command, question, etc). Thus, this information should be integrated into the semantic representation as early as possible, allowing the possibility for earlier expectation/predication at the discourse level, and for discourse expectation-based pruning.

5.2 Implementing Mood

A number of organizational choices were required to allow for the early mood projection described above. First, as the cue elements described above are non-homogenous, the control of mood had to be distributed throughout the grammar instead of residing solely with the main verb or auxiliary. This also lead to some rather unorthodox treatments. The specifics will be described below.

Second, because mood control has been moved away from the verbs themselves, they had to be blocked from creating their own mood-less clauses. Thus, the subject-np of verbs has been "shut-off" or made inert. As an example of this, consider the verb *put*. Here is what we could consider the *indicative past-tense* entry in standard CCG:

```
s \np /pp /np :
@p1:action(put ^
          <Mood>ind ^
```

```
          <Tense>past ^
          <Actor>x1:entity ^
          <Patient>x2:entity ^
          <Result>x3:m-whereto)
```

Compare this to the corresponding MOLOKO entry.

```
s \!np /pp /np :
@p1:action(put ^
          <Tense>past ^
          <Actor>x1:entity ^
          <Patient>x2:entity ^
          <Result>x3:m-whereto)
```

Instead of \np we have \!np. Whereas in the first case this lexical item could combine with a preceding np to form an indicative clause, this is not possible in MOLOKO. Instead, this verb is selected for syntactically and semantically as a compliment by one of the mood-controlling constructions (which will also add the appropriate ⟨*Mood*⟩ value). In essence, the MOLOKO grammar does not follow the standard treatment of verbs as the (only) syntactic and semantic head of the clause.

Third, a 'semantic slot' was added to house the grammatical subject. This slot was attached to the top-level (i.e. clausal) event via a dependency relation named ⟨*Subject*⟩.[8] Despite its seemingly ad-hoc nature, this relation actually serves a number of important functions, though not all at the ideational (propositional) level.

First, in terms of Information Structure, the subject is nearly always the Primary Topic, both in active and passive clauses.

Second, at the interpersonal level, the subject along with the finite (see below) plays an important role in arguing the current proposition "under negotiation" (see Mood analysis in Systemic Functional Grammar), e.g. "I am hungry, No you are not, Yes I am, Well you shouldn't be, Oh I shouldn't, should I? How could you possibly be?"

Third, the subject serves as the semantic and syntactic controller of chained clauses etc. "he walks into the room, picks up a ball and gives it to GJ" (see §8.1 and also the the example of imperatives below). In all three cases, explicitly marking that this entity is playing all of these roles could very useful. Finally, and arguably most importantly this slot allows for the semantic integration of what would otherwise be two chunks (the projected clause/event and the np/entity), satisfying the *one semantic chunk* principle (see §7.1).

5.2.1 Indicatives

Currently, three types of indicative clauses are handled by the grammar:

1. Standard: *he put the ball on the table*
2. Fronted/Topicalized np: *the ball he put on the table*

[8] This has intentionally been removed from all of the semantic representations presented so far.

3. Dropped/contextualized subject: *put it on the table*

Standard indicative clauses (1) have the following structure: subject-np + finite-verb. In essence, this has been handled by extending the standard syntactic type raising rule used for subject-verb integration, i.e.

$$np_T \Longrightarrow s_E \ /(s_E \setminus np_T)$$

has become

$$np_T \Longrightarrow s_E/(s_E \setminus_! np_T) : @_{e1:event}(\wedge \langle Mood \rangle(ind) \wedge \langle Subject \rangle(t1 : entity))$$

Syntactically, this rule performs a number of functions. First, it incrementally creates the proper clausal expectations: the clause initial np becomes a s missing its verb (i.e. $s \setminus_! np$). Second, it properly distributes syntactic features among the various elements (e.g. the number and person of the np are inherited by the vp, forcing agreement. see §8.5 for details). Semantically, we see first that the expected event E has been built into the semantic structure explicitly, although of course in a fully general way (i.e. there are no expectations concerning its sub-sort, proposition, tense, thematic-roles, etc.). Second, this event is marked as indicative. Third, T, the index of the initial np fills the $\langle Subject \rangle$ role of this event. It is also co-indexed with the verb's own subject-np (i.e. its $\setminus_!$ np) meaning that it will also play whatever thematic role the verb creates for it (e.g. $\langle Actor \rangle$ or $\langle Cop - Restr \rangle$). Thus, for example, this rule will combine with the pronoun *I* to create the following parse:

parse: $s/(s \setminus_! np)$

$$@_{x1:event}(\wedge \langle Mood \rangle(ind) \wedge \langle Subject \rangle(i1 : person \wedge \mathbf{I} \wedge \langle Num \rangle(sg)))$$

Again, although syntactic features are not shown the verb compliment is restricted to be in 1st person singular form.

Fronted NP clauses (2) have the following structure: fronted-np + subject-np + finite-verb-with-'missing'-np. This has also been handled by the following type-changing rule:

$$np_T \Longrightarrow s_E \ /(s_E \setminus_! np_S/np_T) \ /np_S : @_{e1:event}(\wedge \langle Mood \rangle(ind) \wedge \langle Fronted \rangle(t1 : entity)) \wedge \langle Subject \rangle(s1 : entity))$$

Note the ordering and co-indexing of the nps. The initial np, the one that undergoes this type-change, fills a construction specific role $\langle Fronted \rangle$ and co-indexes with the 'missing' np. The second np, the first complement of this construction, is the clausal

subject and behaves identically to the case above. Thus, the clause *the ball I haven't seen* (as in *the mug is over on the table, but the ball I haven't seen* receives this parse:

```
@s1:perception(see ^
               <Mood>ind ^
               <Polarity>neg ^
               <Tense>past ^
               <Aspect>perfect ^
               <Actor>(i1:person ^ I ^
                       <Num>sg) ^
               <Patient>(b1:thing ^ ball ^
                         <Delimination>unique ^
                         <Num>sg ^
                         <Quantification>specific) ^
               <Fronted>b1:thing ^
               <Subject>i1:person)
```

Finally, dropped-subject indicatives have this structure: finite-verb. As this structure is nearly identical to standard imperative clauses, I will delay discussing how it was syntactically handled until then. Semantically, this has been treated as a case of a contextualized semantic object (see §2.6.3, in this case, the entity filling the ⟨*Subject*⟩ role. Thus, the clause *walks into the office* receives this parse:

```
@w1:action-motion(walk ^
                  <Mood>ind ^
                  <Tense>pres ^
                  <Actor>(c1:entity ^ context) ^
                  <Modifier>(i1:m-whereto ^ into ^
                             <Anchor>(o1:e-location ^ office ^
                                      <Delimination>unique ^
                                      <Num>sg ^
                                      <Quantification>specific)) ^
                  <Subject>c1:entity)
```

5.2.2 Imperatives

Currently, three types of imperative clauses are handled by the grammar:

1. Standard Addressee positive: *put the ball on the table, be quiet*
2. Addressee negative: *don't put the ball on the table, don't be so loud*
3. Speaker + Addressee: *let's put it on the table, let's be quieter*

The first standard imperative has this structure: base-form verb.[9] We have handled this via a series subject-removing rules.[10]As an example of such a rule, here is

[9]In fact, they must be of the verbal form 'vf-to-imp'. This allows verbs to be specified lexically whether or not they allow imperative readings. This is discussed in §8.5.

[10]The need for having multiple rules will be discussed in 8.5.

transitive clause version:

$$s_E \ /np_X \ \backslash_! \ np_S \Longrightarrow s_E \ /np_X \ : @_{e1:event}(\wedge \ \langle Mood \rangle(imp) \wedge \ \langle Subject \rangle(a1 : entity \wedge \ \textbf{addressee}))$$

It is crucial that the verb be in base form: only *walk*, but not *walks* or *walked* can be used in imperative clauses. Dropped-subject indicatives (see above) are handled in the same manner, except in those cases the verbal form is constrained to being finite.

An alternate way of handling these two moods would be to do so lexically: each verb would have a subject-less entry which would receive, in the case of imperatives, the feature $\langle Mood \rangle$(imp) and vform value *vf-base*. Although this seems a priori more attractive, there are (at least) two reasons for preferring the rule approach instead. Both of these can be illustrated by considering MOLOKO's semantic representation for *go to the office and wait there*:

```
@a1:event(and ^
          <Mood>imp ^
          <Subject>(a2:entity ^ addressee) ^
          <First>(g1:action-motion ^ go ^
                  <Actor>a2:entity ^
                  <Modifier>(t1:m-whereto ^ to ^
                             <Anchor>(o1:e-place ^ office ^
                                      <Delimitation>unique ^
                                      <Num>sg ^
                                      <Quantification>specific))) ^
          <Next>(w1:action-non-motion ^ wait ^
                 <Actor>a2:entity ^
                 <Modifier>(c1:m-location ^ context ^
                            <Proximity>m-distal)))
```

First, although imperative clauses do not have syntactic (i.e. 'surface realized') subjects, they still have an underlying semantic subject. In CCG, this can only be accessed (and hence controlled) by reference to the encoding np. Consequently, the proper assignment of the referent **a2** (the 'addressee') to both $\langle Actor \rangle$ roles is possible only by first coordinating these two clauses (co-indexing their individual subjects) and only then applying the imperative rules to this resulting clause. Second, a lexical treatment would require treating this type of utterance as the coordination of two 'full' sentences, resulting in two separate $\langle Subject \rangle$ roles and mood values. Instead, by allowing a single rule application, we receive this much more elegant representation, i.e. a single $\langle Subject \rangle$ and a single mood value 'scoping over' the entire clausal chain (see §8.1 for more details).

Negative imperatives have the structure: *don't* + base-form-vp. Because of the lexical item *don't* this was able to be handled lexically. Thus, *don't* receives a lexical entry of this form:

$$s_E \ /(s_E \setminus_! np_s) : @_{e1:event}(\\ \wedge \ \langle Mood \rangle(imp) \\ \wedge \ \langle Polarity \rangle(neg) \\ \wedge \ \langle Subject \rangle(a1 : entity \wedge \ \textbf{addressee}))$$

Similarly Speaker+Addressee imperatives have the structure: *let's* + base-form-vp, allowing a simple lexical entry for *let's* of the form:

$$s_E \ /(s_E \setminus_! np_s) : @_{e1:event}(\\ \wedge \ \langle Mood \rangle(imp) \\ \wedge \ \langle Subject \rangle(a1 : entity \wedge \ \textbf{speaker} + \textbf{addressee}))$$

5.2.3 Closed Interrogatives

Closed interrogative clauses are of the form: aux-verb + subject-np + main-verb. These have been handled by giving all relevant verbs (copula, auxiliary and modal) a lexical entry corresponding to this clause type. Here is the closed-int entry for the progressive auxiliary *be* (as in *I am sleeping*) and the negative modal auxiliary *can't* (see §2.4.4 for the semantic treatment of modality):

$$s_E \ /(s_E \setminus_! np_s) \ /np_s : @_{e1:event}(\\ \wedge \ \langle Mood \rangle(int) \\ \wedge \ \langle Aspect \rangle(continuous) \\ \wedge \ \langle Tense \rangle(pres) \\ \wedge \ \langle Subject \rangle(s1 : entity))$$

$$s_E \ /(s_E \setminus_! np_s) \ /np_s : @_{e1:event}(\\ \wedge \ \langle Mood \rangle(int) \\ \wedge \ \langle Polarity \rangle(neg) \\ \wedge \ \langle Tense \rangle(pres) \\ \wedge \ \langle Modifier \rangle(c1 : ability \wedge \ \textbf{can})) \\ \wedge \ \langle Subject \rangle(s1 : entity))$$

In addition to these semantic differences, the vp compliment is restricted according to the lexical requirements of the particular auxiliary verb (*be* selects for the ing form and *can't* selects for the base form)

5.2.4 Open Interrogatives

Open interrogative clauses are the most varied semantically and syntactically among the mood types. Consequently, the particulars of each type of question will not be

discussed until §8.10. In this section, we will reserve ourselves to briefly discussing some of the general principles. To begin, table 2 offers a few illustrating examples. Where relevant, long distance dependencies/extracted constituents are marked with an x.

Table 2: Open Interrogative Examples

Question-Item	Auxilliary	Subject	Verbal Group
who		"	picked up the ball
who		"	wants some coffee
what	did	the robot	see x
where	can	I	put it x
what	are	you	doing x over there
which ball	does	he	want x on the table
what color ball	does	he	want me to have x
how many balls	should	I	pick up x for you
how big	should	I	make it x
where	are	you	going
where	did	he	want me to sit
how quickly	did	he	walk

The initial core element of all of these clauses is a question item. This can be as simple as a single word (e.g. *where* or *who*) or involve a head word plus restricting compliments (e.g. *what color ball* or *how big*). Next, in all but the cases where the subject is being questioned, this is followed first by an auxiliary and then by the subject-np. Finally, in all cases the verbal group is the final element. Typically, it is 'missing' the compliment corresponding to the question item, except in those cases where it is a modifier which is being questioned. All of these examples have been handled by creating an appropriate lexical entry for the corresponding question item head. Thus, it is the 'wh-word' which controls the building of these clauses. Semantically, all open interrogative clauses have been given a dependency relation $\langle Wh - Restr \rangle$ (abbr. of restrictor) which specifies the nature and the scope of the questioned item. As a first example, consider the entry for *what* used in the question *what did the robot see*.

$$\begin{aligned} s_E \ /(s_E \setminus_! np_s/np_w) \ /np_s \ /aux_A : @_{e1:event}(& \\ & \wedge \ \langle Mood \rangle(int) \\ & \wedge \ \langle Subject \rangle(s1 : entity) \\ & \wedge \ \langle Wh - Restr \rangle(w1 : entity \wedge \ \textbf{what})) \end{aligned}$$

This construction has three complements, expected in the following order: an auxiliary, the subject-np and finally a verb 'missing' a np. The various syntactic and semantic features of the clause (agreement, tense, etc) are properly selected from and imposed by the subject and the auxiliary elements. In this case, the the $\langle Wh-Restr \rangle$

is quite simple: it is an entity co-indexed with the missing np, signifying that what is being questioned is an entity which is playing some thematic role in the event being built. As a slightly more complex example, consider the parse for *which ball does he want on the table*.[11]

```
@w1:cognition(want ^
              <Mood>int ^
              <Tense>pres ^
              <Actor>(h1:person ^ he ^
                      <Num>sg) ^
              <Patient>(b1:thing ^ ball) ^
              <Result>(o1:m-location ^ on ^
                       <Anchor>(t1:thing ^ table ^
                                <Delimination>unique ^
                                <Num>sg ^
                                <Quantification>specific)) ^
              <Subject>h1:person ^
              <Wh-Restr>(w2:specifier ^ which ^
                         <Scope>b1:thing))
```

The entry for the question word *which* is identical to the one above, except that it has a fourth complement: a noun specifying the class of the entity being questioned. The index corresponding to this noun fills a dependency relation which gives the scope of the $\langle Wh-Restr \rangle$. Finally, it is this index, not the index of the $\langle Wh-Restr \rangle$ as a whole, which is co-indexed with a thematic role in the event. *which* also allows a contextualized semantic head reading, e.g. *which does he want on the table* would be identical except $\langle Scope \rangle$(c1:entity $\wedge$ **context**). Note that this contrasts with *what does he want on the table*.

Some $\langle Wh-Restr \rangle$ receive two levels of $\langle Scope \rangle$. For example, here is one of the semantic representations for *what color ball do you want*:

```
@w1:cognition(want ^
              <Mood>int ^
              <Tense>pres ^
              <Actor>(y1:person ^ you ^
                      <Num>sg) ^
              <Patient>(b1:thing ^ ball ^
                        <Delimitation>unique ^
                        <Num>sg ^
                        <Quantification>specific) ^
              <Subject>y1:person ^
              <Wh-Restr>(w2:specifier ^ what ^
                         <Scope>(c1:quality ^ color ^
                                 <Scope>b1:thing)))
```

As one final example, consider the parse for *where did he want me to sit*:

[11] *what ball does he want on the table* receives an identical representation.

```
@w1:cognition(want ^
                <Mood>int ^
                <Tense>past ^
                <Actor>(h1:person ^ he ^
                        <Num>sg) ^
                <Event>(s1:action-non-motion ^ sleep ^
                        <Actor>i1:person) ^
                <Patient>(i1:person ^ I ^
                          <Num>sg) ^
                <Subject>h1:person ^
                <Wh-Restr>(w2:m-location ^ where ^
                           <Scope>s1:action-non-motion))
```

The syntax required to handle this is complex and will be discussed in §8.10.2. Semantically, notice that there is no 'missing' element: the $\langle Wh-Restr \rangle$ is not 'taking the place' of anything. Instead, it is a functional operator which scopes over one of the thematic roles of the event, in this case the $\langle Event \rangle$ role of the verb *want*. What is crucially important in examples like this is that the wh-word itself is able to semantically restrict what kinds of events(verbs) it is allowed to scope over, i.e. to question. This is done using the modifier classes described in §4. In this case, the static locational entry for *where* specifies that it can only question events which allow this type of modifier. Consequently, * *where did he want me to believe* would not parse.

6 The *be* Verb

This section discusses MOLOKO's treatment of the *be* verb. To begin, here are a few examples which can be used to isolate its broad functions:

1. *the cat was sitting on the table*
2. *the cat was picked up*
3. *the cat was willing to sit next to me*
4. *the cat was on the table*
5. *the cat was happy*
6. *this is a cat*
7. *a cat is a kind of animal*
8. *there was a cat*
9. *on the table was a cat*

Examples (1) and (2) are clear cases of auxiliary verbs, the *-ing* **continuous/progressive** and **passive** respectively. We have treated the usage in (3), with what are sometimes call predicatival or adjectival verbs, as a auxiliary as well. Thus, we treat *willing* as the main verb/event:[12]

```
@w1:modal(willing ^
          <Mood>ind ^
          <Tense>past ^
          <Actor>(c1:animate ^ cat )^
          <Event>(s1:action-non-motion ^ sit ^
                  <Actor>c1:animate ^
                  <Modifier>(n1:m-location ^ next ^
                             <Anchor>(i1:person ^ I ) ^
          <Subject>m1:person)
```

For examples (1)-(3), see §2.4.2 for semantics and §8.4.4 for families.

Examples (4)-(7) are various instances of the **copula/ascriptive** usage of the verb. The last two are **presentational/existential**. We will discuss the semantics of each of these in turn. For their families, see §8.4

6.1 Ascription

The ascriptive use of *be* has this basic semantic structure:

$$@_{b1:ascription}(\textbf{be} \wedge \langle Cop-Restr\rangle(X) \wedge \langle Cop-Scope\rangle(Y))$$

The $\langle Cop-Restr\rangle$ is the entity which is being ascribed, and the $\langle Cop-Scope\rangle$ is what is being ascribed of (or predicated over) it. Here are the three main classes of $\langle Cop-Scope\rangle$:

1. Nominal: *it is a ball* / *GJ* / *me* / *a kind of coffee* / *the color of the mug*
2. Adjectival: *it is blue* / *bigger than the mug* / *happy* / *off* / *correct* / *ok*
3. Prepositional: *it is on the table* / *here* / *with me* / *for me*

Class 1 includes but does not distinguish between **category description**[13] and **identificational** uses. Class 2 allows the ascription of any **quality**, physical, attitudinal or other. Class 3 handles **static location, accompaniment** and **benefactor**.

In addition to the Indicative uses illustrated above, it can also be used for Imperatives, both both positive (*be happy*) and negative (*don't be so sad*).

[12] in this and all the other examples in this section, we have removed all semantic features ($\langle Num\rangle$, $\langle Degree\rangle$, etc.)

[13] Note that we currently do not have a proper treatment of generic entities. However, it can be defined as in example (5) above.

```
@b1:ascription(be ^
               <Mood>imp ^
               <Cop-Restr>(a1:entity ^ addressee) ^
               <Cop-Scope>(h1:q-attitude ^ happy) ^
               <Subject>a1:entity)

@b1:ascription(be ^
               <Mood>imp ^
               <Polarity>neg ^
               <Cop-Restr>(a1:entity ^ addressee) ^
               <Cop-Scope>(s1:q-attitude ^ sad) ^
               <Subject>a1:entity)
```

It can also be used for Closed (Y/N) Interrogatives, again positive (*is it a ball*) and negative (*wasn't it here earlier*).

```
@b1:ascription(be ^
               <Mood>int ^
               <Tense>pres ^
               <Cop-Restr>(i1:thing ^ it ) ^
               <Cop-Scope>(b2:thing ^ ball )
               <Subject>i1:thing)

@b1:ascription(be ^
               <Mood>int ^
               <Polarity>neg ^
               <Tense>past ^
               <Cop-Restr>(i1:thing ^ it ) ^
               <Cop-Scope>(c1:m-location ^ context ) ^
               <Modifier>(e1:m-time ^ early ) ^
               <Subject>i1:thing)
```

Finally, it can be used in Open (Wh) Interrogatives to question both the $\langle Cop - Scope \rangle$ role and the $\langle Cop - Restr \rangle$. Like all events, it can also be questioned for modifiers. These will be handled in turn.

Questioning the $\langle Cop - Restr \rangle$ An entity can be given with a prompt for 'filling in' some 'property' specified by the Wh-Word. Here are some examples sorted by the classes specified above:

1. Nominal

what is this thing on the table	it is a ball
who is it	it is me (see below)
who is that guy	it is my dad
what is a cat	a cat is a kind of animal

which ball *is mine* yours is the one over there

```
@b1:ascription(be ^
                <Mood>int ^
                <Tense>pres ^
                <Cop-Restr>(i1:entity ^ it)
                <Cop-Scope>(w1:entity ^ what) ^
                <Subject>c1:entity ^
                <Wh-Restr>w1:entity)
```

2. Adjectival

what shape is it	it is round
what color is it	it is red (see below)
what size is it	it is small
how are you	I am good[14] (see below)
how big is it	it is really big[15]
how cold is it	it is ok

```
@b1:ascription(be ^
                <Mood>int ^
                <Tense>pres ^
                <Cop-Restr>(y1:person ^ you)
                <Cop-Scope>(h1:quality ^ how) ^
                <Subject>y1:person ^
                <Wh-Restr>h1:quality)
```

```
@b1:ascription(be ^
                <Mood>int ^
                <Tense>pres ^
                <Cop-Restr>(i1:thing ^ it ) ^
                <Cop-Scope>(c1:quality ^ color) ^
                <Subject>i1:thing ^
                <Wh-Restr>(w1:specifier ^ what ^
                           <Scope>c1:quality))
```

1. Prepositional

where is the big green mug	it is on your table (see below)
what is it under	it is under the table (see below)
which room is it in	it is in GJ's office
who is it for	it is for me
which cat is this milk for	it is for mine (see below)
who is he with	he is with GJ

```
@b1:ascription(be ^
               <Mood>int ^
               <Tense>pres ^
               <Cop-Restr>(m1:thing ^ mug ^
                           <Modifier>(b2:q-size ^ big) ^
                           <Modifier>(g1:q-color ^ green)) ^
               <Cop-Scope>(w1:m-location ^ where) ^
               <Subject>m1:thing ^
               <Wh-Restr>w1:m-location)

 @b1:ascription(be ^
               <Mood>int ^
               <Tense>pres ^
               <Cop-Restr>(i1:thing ^ it )
               <Cop-Scope>(u1:m-location ^ under ^
                           <Anchor>w1:physical) ^
               <Subject>i1:thing ^
               <Wh-Restr>(w1:physical ^ what))

 @b1:ascription(be ^
               <Mood>int ^
               <Tense>pres ^
               <Cop-Restr>(m1:thing ^ milk )
               <Cop-Scope>(f1:m-benefactor ^ for ^
                           <Anchor>(c2:animate ^ cat ^ )
               <Subject>c1:thing ^
               <Wh-Restr>(w1:specifier ^ which ^
                         <Scope>c2:animate))
```

We need to note the following on questioning **physical properties** (color, size, shape, etc.) In addition to *what color/size/shape is the mug*, there is the option of *what is the color/size/shape of the mug*. The former is treated as an 'adjectival' questioning, scoping over $\langle Cop - Scope \rangle$*(q:quality)* (see above). The latter is treated very naively, scoping over the whole $\langle Cop - Scope \rangle$ AND as some sort of entity. We hope to unify these two readings in a later version of MOLOKO.

```
@b1:ascription(be ^
               <Mood>int ^
               <Tense>pres ^
               <Cop-Restr>(c1:xxxx ^ color ^
                           <Delimitation>unique ^
                           <Num>sg ^
                           <Quantification>specific ^
                           <Owner>(b2:thing ^ ball ^
```

```
                                <Delimitation>unique ^
                                <Num>sg ^
                                <Quantification>specific)) ^
                    <Cop-Scope>(w1:xxxx ^ what) ^
                    <Subject>c1:entity ^
                    <Wh-Restr>w1:xxxx)
```

Questioning the $\langle Cop - Restr \rangle$ A 'property' can also be given with a prompt for 'filling in' what entity(s) this applies to. So, for example, *what is big* or *what is on the table*. Of course all of the other Wh-words are available: *who is nicer*,[16] *which coffee is black*, *how_many balls are in here*, etc. For example, here is *who was in there*:

```
  @b1:ascription(be ^
                    <Mood>int ^
                    <Tense>past ^
                    <Cop-Restr>(w1:animate ^ who) ^
                    <Cop-Scope>(i1:m-location ^ in ^
                                <Anchor>(c1:e-location ^ context )
                    <Subject>w1:animate ^
                    <Wh-Restr>w1:animate)
```

The $\langle Cop - Restr \rangle$ for class 1 (nominals) is 'shut off' from questioning, i.e. *what is this* does not expect an answer like *a ball is this*. There maybe a sub-class of cases where such questioning would make sense, but this is for future work.

Questioning the Ascription Itself The time and place of the ascription can be questioned. For example, *where was he happy* and *when was he here*:

```
@b1:ascription(be ^
                    <Mood>int ^
                    <Tense>past ^
                    <Cop-Restr>(h1:person ^ he ^) ^
                    <Cop-Scope>(h2:q-attitude ^ happy) ^
                    <Subject>h1:person ^
                    <Wh-Restr>(w1:m-location ^ where ^
                               <Scope>b1:ascription))

  @b1:ascription(be ^
                    <Mood>int ^
                    <Tense>past ^
                    <Cop-Restr>(h1:person ^ he) ^
                    <Cop-Scope>(c1:m-location ^ context ^
```

[16] we have not yet handled **constrained option** questions, like *who is nicer, me or him*

```
                    <Proximity>m-proximal) ^
          <Subject>h1:person ^
          <Wh-Restr>(w1:m-time-point ^ when ^
                    <Scope>b1:ascription))
```

6.2 Presentation

The *be* verb can also be used to present entities 'on the scene' and/or to assert/deny their existence. Here are some indicative examples:

1. *there is a mug on the table*
2. *there are two mugs and a book*
3. *there were some good books yesterday*
4. *there has never been a table there*

Here, for example, is the first:

```
@b1:presentational(be ^
                   <Mood>ind ^
                   <Tense>pres ^
                   <Presented>(m1:thing ^ mug)
                   <Modifier>(o1:m-location ^ on ^
                             <Anchor>(t1:thing ^ table ^))
                   <Subject>(t2:dummy ^ there) )
```

The entity is placed in the ⟨*Presented*⟩ role, and additional modifiers (location, time, etc) modify the event itself.[17] The ⟨*Subject*⟩ is filled by the 'dummy' word *there*. This does not contribute to the semantics, but is required for proper generation and for handling the syntax of questions.

In addition, there is a special indicative construction exemplified by *in the room were some cats and a dog*:

```
@b1:presentational(be ^
                   <Mood>ind ^
                   <Tense>past ^
                   <Modifier>(i1:m-location ^ in ^
                              <Anchor>(r1:e-place ^ room) ^
                   <Presented>(a1:entity ^ and ^
                              <First>(c1:animate ^ cat)
                               <Next>(d1:animate ^ dog ))
```

[17] in fact, this example is treated as ambiguous. It also has a reading where the mug is modified. If there is any difference in meaning, we certainly can't figure it out. Note that the possibility of other modifiers (time, etc) and the possibility of questions like *where were there balls* suggests that this location should be able to modify the event.

Note that this leads to two readings for *there is my ball*, with with a 'real' locational *there* and one with a dummy.

There is no imperative form for the presentational, but it can be used for both Open and Closed Interrogatives. For example, *were there any dogs there* and *what was there yesterday*[18]:

```
@b1:presentational(be ^
                   <Mood>int ^
                   <Tense>past ^
                   <Modifier>(c1:m-location ^ context ^
                              <Proximity>m-distal) ^
                   <Presented>(d1:animate ^ dog) ^
                   <Subject>(t1:dummy ^ there))

@b1:presentational(be ^
                   <Mood>int ^
                   <Tense>past ^
                   <Modifier>(y1:m-time-point ^ yesterday) ^
                   <Presented>(w1:entity ^ what) ^
                   <Subject>(t1:dummy ^ there) ^
                   <Wh-Restr>w1:entity)
```

7 Incrementality

A driving factor motivating the development of MOLOKO was the need for incrementality in utterance parsing. This is essential for its use as a component in the broader context of incremental situated dialogue processing. At each step in the flow of interaction, we want to be able to get as much meaning out of our linguistic representation as possible. In our context, these steps boil down to words and thus what we want are word-by-word linguistic representations which are both integrated with what came before, and projective (or predicative) of what could come next.[19] These two dimensions of incrementality, *back-integration* and *forward-projection* , are discussed below in turn. Following this, a number of illustrative parses will be given, discussing step-by-step how incrementality is achieved. This will hopefully give a taste of how these principles have impacted the design of the grammar.

7.1 Back-Integration

The integration of the 'now' with the 'before' can be separated into issues of syntactic integration and issues of semantic integration. The first are concerned with parsability. We never want the parser to complain because it 'doesn't know yet' if

[18] this is also ambiguous between *there* as dummy subject or as the $\langle Cop - Scope \rangle$ of ascription.

[19] As the MOLOKO grammar is a 'pure' generative grammar, it cannot make any predictions beyond what could possibly come. To get at what is more (or most) likely to come, the possible readings outputted by MOLOKO must be augmented with other sources of knowledge (statistical expectations, predictions/preferences based on some element of physical or discourse/interactional context etc)

a particular reading could be possible. Instead, the grammar should be designed so that at each point, all the possible readings can parse. The second is concerned with the structure of the current semantic representation. We do not want multiple semantic chunks which must 'wait' to be combined. Instead, whenever possible, we want semantic readings which consist of only a single semantic structure, with each sub-structure connected in a way licensed by the grammar.

7.2 Forward-Projection

MOLOKO was designed to optimize the following maxim: *words should tell you what they expect as soon as they can.* In other words, they should project early. Translated into a grammar design principle, whenever possible, the first word in a construction projects its dependencies. This provides the maximal amount of information in terms of what could come next both syntactically (in terms of 'rightward' dependents) and semantically (in terms of 'unfilled roles'). We have already seen how this impacted the treatment of mood. Note, that again we see a crucial difference between dependents and modifiers. Modifiers are not projected, and when encountered may lead to an increase in the number of parses instead of the typical decrease in parses as more 'information' becomes incrementally available (see e.g. step 6 below).

7.3 Some Illustrations

7.3.1 Example 1: *the man put a ball on the table*

First, consider the incremental parsing for this straightforward indicative sentence:

```
Step 1:  the

Parse 1: np/^n :
  @x1(
      <Delimination>unique ^
      <Num>sg ^
      <Quantification>specific)

Parse 2: s/(s\!np/np)/^np/^n :
  @x1:event(
            <Mood>ind ^
            <Fronted>(x2 ^
                      <Delimination>unique ^
                      <Num>sg ^
                      <Quantification>specific) ^
            <Subject>x3)

Parse 3: s/(s\!np)/^n :
  @x1:event(
            <Mood>ind ^
```

```
            <Subject>(x2 ^
                      <Delimination>unique ^
                      <Num>sg ^
                      <Quantification>specific))
```

In addition to these 3 parses, there are those corresponding to the plural readings of *the*.[20] Clearly, as this is the first word in the utterance there is nothing to back-integrate into.[21] However, from even this first word there is a massive amount of projection. First, at the immediate level, *the* begins building a np encoding an entity which we know is unique and specific. It projects an n_T, with T expected to add the proposition(the 'semantic head' of this modification relation). Moving upwards, the np combines with the two indicative clause rules to begin building a clause/event with T filling the role of $\langle Subject \rangle$ or $\langle Fronted \rangle$. The appropriate clausal elements (verb-phrase and subject then verb-phrase-'missing'-a-rightward-np-dependent) are projected as well, though critically, after this nominal.

```
Step 2: the man

Parse 1: np:
  @m1:person(man ^
      <Delimination>unique ^
      <Num>sg ^
      <Quantification>specific)

Parse 2: s/(s\!np/np)/^np:
  @x1:event(
            <Mood>ind ^
            <Fronted>(m1:person ^ man ^
                      <Delimination>unique ^
                      <Num>sg ^
                      <Quantification>specific) ^
            <Subject>x3)

Parse 3: s/(s\!np) :
  @x1:event(
            <Mood>ind ^
            <Subject>(m1:person ^ man ^
                      <Delimination>unique ^
                      <Num>sg ^
                      <Quantification>specific))
```

First, because *man* is singular, all three of the plural readings are lost. In the remaining three singular readings, we see that *man* has satisfied the projected noun

[20] Although we can implement the 'filling-in' of underspecifed values at the level of syntactic features and semantic sorts, this cannot be done properly at the level of semantic features.

[21] This of course says nothing about incremental discourse parsing, which is handled in a separate module outside of MOLOKO.

and added its prop to the entity, back-integrating perfectly both syntactically and semantically. Note that at this point because *man* has no dependents it make no projections and we have a potentially complete np and its corresponding entity. This, of course, does not prohibit further rightward modification (the man on the table, the man that I told you about, etc) but as this is optional, it does not figure into the parse of the utterance.[22]

```
Step 3: the man put

Parse: s/pp/np :
  @p1:action(put ^
             <Mood>ind ^
             <Tense>past ^
             <Actor>(m1:person ^ man ^
                     <Delimination>unique ^
                     <Num>sg ^
                     <Quantification>specific) ^
             <Patient>x1:entity ^
             <Result>x2:m-whereto ^
             <Subject>m1:person)
```

Note that all but the standard indicative parses have been lost, in both cases because *put* did not fit with what was expected: nothing for the np parse and a second np (i.e. the ⟨*Subject*⟩) for the fronted parse. The verb *put* has exerted its control over the entire clause, massively increasing the syntactic and semantic complexity of the parse.[23] It has filled in the proposition of the event and added the semantic ⟨*Tense*⟩ feature. It has also added its participant roles. The ⟨*Actor*⟩ role has been filled by the man (m1) via co-indexing, and the ⟨*Patient*⟩ and ⟨*Result*⟩ slots will be filled by the semantic objects corresponding to the projected syntactic complements /np and /pp correspondingly. Note that the semantic sorts of these semantic objects are available (entity and whereto). At this point then, we have some very useful information about what we expect to come in the remainder of the utterance. This is, of course, still provisional. We may end up receiving less information (a fragment) or more likely, more information (additional modifications *the man put it on the table yesterday when you weren't here*, etc)

```
Step 4: the man put a

Parse: s/pp/^n :
  @p1:action(put ^
```

[22] In this way, CCG is quite different from Phrase Structure-based formalisms, where we would in fact get a number of 'np-bar' readings at this point, projecting these possible modifications. If we want somehow to model this 'linguistic knowledge' about the possibility of the rightward modification of nps, it would have to be handled somehow.

[23] These properties are no doubt what have lead to the traditional treatment of the main verb as *the* clausal head, both syntactically and semantically (including mood). In §5 we motivate our decision to consider clausal constructions as an additional 'major contributor' to the clause.

```
            <Mood>ind ^
            <Tense>past ^
            <Actor>(m1:person ^ man ^
                    <Delimination>unique ^
                    <Num>sg ^
                    <Quantification>specific) ^
            <Patient>(x1:entity ^
                      <Delimination>existential ^
                      <Num>sg ^
                      <Quantification>specific) ^
            <Result>x2:m-whereto ^
            <Subject>m1:person)
```

The determiner *a* begins building the np which was projected in step 3. It is integrated syntactically (/np is 'replaced' with $/_{\wedge}$n) and semantically (the entity filling the $\langle Patient \rangle$ role is now marked to be existential, specific and singular). The only change in projection is that we are now expect a $\text{n}_{\tau sg}$ instead of a $\text{np}_{\tau sg-or-pl}$.

```
Step 5: the man put a ball

s/pp .....
```

Nothing much interesting happens here. The patient entity is (provisionally) full, and the amount of projected material decreases, with now the pp expected 'next'.

```
Step 6: the man put a ball on

Parse 1: s/pp/^np :
  @p1:action(put ^
             <Mood>ind ^
             <Tense>past ^
             <Actor>(m1:person ^ man ^ ...)
             <Patient>(b1:thing ^ ball ^
                       <Delimination>existential ^
                       <Num>sg ^
                       <Quantification>specific ^
                       <Modifier>(o1:m-location ^ on ^
                                  <Anchor>x1)) ^
             <Result>x2:m-whereto ^
             <Subject>m1:person)
Parse 2: s/^np :
  @p1:action(put ^
             <Mood>ind ^ \id{T}
             <Tense>past ^
             <Actor>(m1:person ^ man ^ ...)
             <Patient>(b1:thing ^ ball ^
```

```
                    <Delimination>existential ^
                    <Num>sg ^
                    <Quantification>specific) ^
          <Result>(o1:m-whereto ^ on ^
                   <Anchor>x1) ^
          <Subject>m1:person)
```

Encountering the word *on* leads to two parses. This is due to a combination of factors. The preposition *on* is lexically ambiguous between a dynamic whereto and a static locational reading and also between its function as dependent and modifier. The $\langle Result \rangle$ role requires a whereto dependent ($pp_{M:m-whereto}/_{\wedge}np_{A}$) and the immediately preceding nominal (i.e. ball) is allowed to be locationally post-modified($pp_{M:m-location}/_{\wedge}np_{A}$ which is transformed via rule into $n_{T}/_{\wedge}np_{A} \setminus_{\star} n_{T}$, see next illustration for more details) . These combinatorial possibilities result in parse 1 and 2 respectively. In the first case, the projected pp_{R} comp is saturated with R filling *put*'s $\langle Result \rangle$ role. In the latter, this expectation has been is temporally 'pushed back' in priority by the presence of this modifier. This reading is available for sentences like *the man put the ball on the table into the box*. In both cases, what is expected next is the np_{A} corresponding to *on*'s $\langle Anchor \rangle$ role

Step 7 and 8: the man put a ball on the table

```
Parse 1: s/pp :
  @p1:action(put ^
             <Mood>ind ^
             <Tense>past ^
             <Actor>(m1:person ^ man ^ ...)
             <Patient>(b1:thing ^ ball ^
                       <Delimination>existential ^
                       <Num>sg ^
                       <Quantification>specific ^
                       <Modifier>(o1:m-location ^ on ^
                                  <Anchor>(t1:thing ^ table ^
                                           <Delimination>unique ^
                                           <Num>sg ^
                                           <Quantification>specific))) ^
             <Result>x1:m-whereto ^
             <Subject>m1:person)
Parse 2: s :
  @p1:action(put ^
             <Mood>ind ^
             <Tense>past ^
             <Actor>(m1:person ^ man ^ ...)
             <Patient>(b1:thing ^ ball ^
                       <Delimination>existential ^
                       <Num>sg ^
```

```
                <Quantification>specific) ^
        <Result>(o1:m-whereto ^ on ^
                <Anchor>(t1:thing ^ table ^
                        <Delimination>unique ^
                        <Num>sg ^
                        <Quantification>specific)) ^
        <Subject>m1:person)
```

The words *the* and *table* combine to create an np_A with A satisfying this $\langle Anchor \rangle$ role. In the first parse, this finishes the projected dependencies resulting in a (potentially) complete s reading. In the latter case, we essentially return to the parse in step 5, though this time with a modified patient.

7.3.2 Example 2: *this big ball on the table*

As a second example, we consider a parse involving the pre- and post modification of a noun. As clause level projection has already been discussed, we focus only on the parses which relate to the local building of this np

```
step 1: this

Parse 1: np/^n :
  @x1(
      <Delimination>unique ^
      <Num>sg ^
      <Proximity>proximal ^
      <Quantification>specific)

Parse 2: np :
  @c1:entity(context ^
             <Delimination>unique ^
             <Num>sg ^
             <Proximity>proximal ^
             <Quantification>specific)

Parse 3: s :
  @c1:event(context ^
            <Proximity>proximal)
```

The word *this* has a number of readings. The first is its determiner reading (c.f. step 1 parse 1 above). The second and third are its context entity and event readings (see §2.6.3).

```
step 2: the big

Parse 1: np/^n :
  @x1:entity(
```

```
            <Delimination>unique ^
            <Num>sg ^
            <Proximity>proximal ^
            <Quantification>specific ^
            <Modifier>(b1:q-size ^ big))
```

We can see that both in terms of back-integration and forward-projection this reading is behaving as we want it. The projected entity is further specified (modified) and the parse still projects a noun along with its associated nominal variable T and proposition.

In order to understand exactly how this reading is built, we must look a bit more carefully at how adjectives like *big* work. As discussed in () all adjectives begin as atomic categories. So, we have:

adj[M] : $@_{b1:size}$ (**big**)

This is then 'transformed' into a complex category, capable of modification, via the following type-changing rule:

adj[M] $\Longrightarrow$ n_T/n_T : $@_{t1:entity}$($\langle Modifier \rangle$(m1)))

This result is thus

n_T/n_T : $@_{t1:entity}$($\langle Modifier \rangle$(b1:size $\wedge$ **big**))

which via simple forward composition combines with step 1's $np/_{hat}n$ resulting in the step 2 above.

```
Step 3: the big ball

Parse 1: np :
  @b1:thing(ball ^
            <Delimination>unique ^
            <Num>sg ^
            <Proximity>proximal ^
            <Quantification>specific ^
            <Modifier>(b2:q-size ^ big))
```

The readings here are parallel to those in step 2 of example 1. Again, the important thing is that we have a (potentially) complete np and entity.

```
Step 4: the big ball on

Parse 1: np/^np :
  @b1:thing(ball ^
            <Delimination>unique ^
            <Num>sg ^
```

```
<Proximity>proximal ^
<Quantification>specific ^
<Modifier>(b2:q-size ^ big) ^
<Modifier>(o1:m-location ^ on ^
          <Anchor>x1))
```

Before investigating exactly how the complete reading from 3 was 'opened up' to allow further modification, we should notice that this reading is perfectly incremental. The preposition *on* has attached itself to the entity as a locational modifier, and has appended its own syntactic projection creating the complex cat $np/_{hat}np$. We have thus gone from a 'complete' reading back to a syntactically and semantically 'unfinished' reading.

To explain what has happened here, it is best to look at the derivational history of this reading. The word in parentheses specify where the derivational step comes from. This can be a lexical entry (lex), a type changing grammatical rule (gram) or a composition rule, e.g. (>) or (>B).

```
(lex)            this :- np/^n
(lex)            big :- adj
(gram)           tcr1: adj$1 => n/^n$1
```

The $\$_1$ here represents any potential dependencies that the adj may bring along with it. We will see this in action a few steps down with *on*

```
(tcr1)          big :- n/^n
(lex)           ball :- n
(>)             big ball :- n
```

Note that at this point, we have not actually integrated *the* into the derivation yet. This does not violate incrementality, however, because we are actually 'in the middle' of a parse step. Incrementality is imposed on the output of a parse step, not on the process itself

```
(lex)          on :- pp/^np
(gram)         tcr2: pp$1 => n$1\*n
(tcr2)         on :- n/np\*n
```

First, we have the lexical entry for *on*, which as discussed above is $pp/_{\wedge}np$, i.e. the atomic category plus its nominal dependency. Then, the rule named tcr2 comes into play, transforming the basic category into the complex category $n/np \backslash_{\star} n$. The $\$_1$works here to take all of preposition's syntactic dependencies and place them in the correct location in the new complex category. In this case, as this is a post modifier, this is directly after the modifee (hence the ordering $/np \backslash_{\star} n$).[24] This category should understood as: 'I want an n directly (*) before me (this is the modifee), followed by a np (*on*'s dependent), and then I'll give you back another n (corresponding to the resulting now modified entity)

[24]Remember due to the currying of CCG categories, the order in syntactic categories is the reverse of their linear realization.

```
(<)              big ball on :- n/np
```

This first requirement is met by *big ball*, leaving the np.

```
(>B)             the big ball on :- np/np
```

Finally, *the* comes back into play: $np/_{hat}n + n/np = np/np$ via >B

```
Step 5 and 6 : this big ball on the table

Parse 3: np :
  @b1:thing(ball ^
            <Delimination>unique ^
            <Num>sg ^
            <Proximity>proximal ^
            <Quantification>specific ^
            <Modifier>(b2:q-size ^ big) ^
            <Modifier>(o1:m-location ^ on ^
                       <Anchor>(t1:thing ^ table ^
                                <Delimination>unique ^
                                <Num>sg ^
                                <Quantification>specific)))
```

The ⟨*Anchor*⟩ role and /np dependent are filled by *the table* resulting in yet another (potentially) complete np / entity.

8 Families

The MOLOKO grammar is written in the DotCCG language. Due to the power of its definitional macro function, a large amount of redundancy has been removed from the underlying families. We have aimed to reduce the number of families and move a lot of the variability to the level of the 'dictionary'. Consequently, in what follows, we have included a list of the various 'dictionary forms' used in addition to the traditional families and rules. §8.13 discusses the use of 'higher order' dictionary macros to group multiple uses of a single word together.

Each dictionary form has an additional form _XXXXX which allows for alternate word-forms (i.e. separate word-forms which map to the same predicate), e.g.

noun(mom, person,) _noun(mom, mamma, person,)
_noun(mom, mommy, person,)

Examples of each family and dictionary form are given. For a discussion of the organization of the grammar files themselves, see §9

8.1 Coordination

Coordination families are defined locally, i.e. noun and np coordination is with the other nominal families, adjective coordination with adjectives, etc. [25] The general form for such a family is:

Coord – X– *and, but, then*

$\vdash \quad X_{R\,COM=yes} / X_{N} \setminus_{*} X_{F\,COM=no} \quad :$
$@_{R:}(* \wedge \langle First \rangle(F) \wedge \langle Next \rangle(N))$

The feature COM is used to guarantee only a single reading for multiple coordination (*ball and cup and mug*), in particular the left-branching or 'staircase' reading. The $\langle First \rangle$-$\langle Next \rangle$ semantics is also used for prepositional chains, and for discourse markers (§8.11).

Naturally, individual families have control over which syntactic features are enforced on the conjuncts and which are inherited by their 'result'. For example, verb coordination specifies that the conjuncts share vform and agreement features and that these are inherited by the result. Thus, e.g. *I am blue and want a drink* parses, but * *I am blue and wants a drink* doesn't, similarly *I am sitting and looking at the table* but * *I am sitting and looked at the table*. As another example, sentential coordination specifies that each of the conjuncts are either indicatives or interrogatives. This forces imperatives to combine using verbal coordination, resulting in the nice single mood representation discussed in §5.

Dictionary Forms:

- COORD (form, pos, class)
  coord (but, adj, quality)

  COORD+ (form, pos, class, features)
  coord+ (and, n, entity, s-pl pl)

  - creates an entry for the specified part-of-speech. The + version allows features to be added to the result.

8.2 Nouns

Syntactic Features:

- NUM: *s-sg s-pl* {*s-pl-sp s-pl-unsp*} *s-mass*
  s-pl-sp and *s-pl-unsp* are used to mark, e.g. *the balls* and the two balls respectively. See determiners below.
- PERS: *non-3rd* {*1st 2nd* } *3rd*

[25] Adjectival, adverbial and prepositional coordination are possible in MOLOKO, see §3.

- CASE: *nom acc acc-loc*
  acc-loc is used for prepositional compliments (see preps below)
- NFORM: *nf-real* {*full pro nf-ctxt* } *nf-dummy* {*dummy-there* }
  First, specifies if the noun is 'real', i.e. referential, or 'dummy', i.e. purely grammatical e.g. *there is a ball on the table*. Real are further subdivided into full (lexical) (*the ball, GJ*), pronominal or contextualized (*the green)* .
- CC-TYPE: *compound-1st compound-head .... n-all n-1 ...*
  allows the lexical specification of a nouns behavior in the noun-noun compound construction (see rules below). *compound-1st* and *compound-2nd* are atomic values and the rest define syntactic classes like those outline in section X above.

Families:

1. **Noun** *ball, men, library, water* $\vdash$ n_T :
   $@_{t1:entity}(*)$
2. **Noun + of − np** *edge, corners, side* $\vdash$ $n_T/obl_{T\,of}$:
   $@_{t1:entity}(* \wedge \langle Owner \rangle$(o1:entity))
3. **Context − n + modifier** *green, second, big* $\vdash$ n_T :
   $@_{t1:entity}($ **context** $\wedge \langle Modifier \rangle$(m1:modifier $*$))
4. **Owned − np** *mine, yours, hers* $\vdash$ $np_{T\,s-sg,3rd,full}$:
   $@_{t1:entity}($ **context** $\wedge \langle Spec \rangle$(sg, specific, unique) $\wedge \langle Owner \rangle$(o1:entity $*$))
5. **Owner − pro** *my, your, her* $\vdash$ $np_{T\,3rd}/np_{T\,s-sg}$:
   $@_{t1:entity}(\langle Spec \rangle$(sg, specific, unique) $\wedge \langle Owner \rangle$(o1:entity $*$))
6. **event − np** *it, this , that* $\vdash$ s_E :
   $@_{e1:}(*)$

For 1 - 3, there is a corresponding 'bare-np' family, i.e. replace n_T with np_T
For 4 - 5, there is also a plural entry, i.e. replace syntactic and sg features with pl

Rules:
Rules 1 and 2 act like determiners, specifying certain special types of nouns and changing them to nps. As these words begin lexically as nouns, they are able to do what all nouns can do (be modified, act as parts of compounds, etc). Rule 3 handles the noun-noun compound construction

1. any plural, unspecific noun into a generic plural np e.g. *balls are round*
   $n_{T\,s-pl-unsp} \implies np_{T\,3rd}$: $@_{t1:entity}(\langle Spec \rangle$(variable, unspecific))
2. nouns marked as mass into a full np e.g. *I want coffee*
   $n_{T\,s-mass} \implies np_{T\,3rd,sg}$: $@_{t1:entity}(\langle Spec \rangle$(variable, uncountable))

3. A noun's ability to function as 1st part or head can restricted lexically via the cc-type feature. Setting the 'resulting' n to *cc-none* blocks the recursion of this rule

   $n_{c\,full,cc-type=compound-1st} \implies$
   $n_{T\,cc-type=cc-none} /\!* n_{\mathbf{T_{cc-type=compound-head}}}$: $@_{t1:entity}(\ \langle Compound \rangle(\text{c:entity})\ )$

Dictionary Forms:

- NOUN (sg-form, class, args)
  noun (ball, thing,)

  NOUN-IRR (sg-form, pl-form, class, args)
  noun-irr (man, men, person,)

  NOUN+OF-NP (sg-form, class, args)
  noun+of-np (edge, e-location)

  NOUN-IRR+OF-NP (sg-form, pl-form, class, args)
  noun-irr+of-np ()

  - these create a $n_{T\,s-sg,full}$ and$n_{T\,s-pl,full}$, each with $@_{T:class}$(**form**). All specification, including *semantic* number is added by the determiner.

- NAME (form, class, args)
  name (GJ, person)

  - creates a $np_{T\,s-sg,3rd,full}$ with $@_{T:class}$(**form**)

- NOUN-MASS (form, class)
  noun-mass (water, thing)

  - creates a $n_{T\,s-mass,3rd,full}$ with $@_{T:class}$(**form**).
  The latter is transformed into the np form via rule (see rules)

- PRONOUN (pred, pers, num, nom-form, acc-form, owner, owned, class,)
  pronoun (I, sg, 1st, I, me, my, mine, person)

  - creates 6 forms: $np_{T\,num,pers,pro,nom}$ and $np_{T\,num,pers,pro,acc}$ and a sg and pl for owner and owned (see families)

- CONTEXT-N (form, class, args)
  context-n (one, entity, s-sg)

  CONTEXT-NP (form, class, args)
  context-np (this, entity, sg s-sg proximal unique specific)

  CONTEXT-N+MODIFIER (form, class, args)
  context-n+modifier (green,q-color, s-sg)

  - these create $n_{T\,nf-ctxt}$ and $np_{T\,3rd,nf-ctxt}$, each with $@_{T:class}$(**context**)

- CONTEXT-S (form, args)
  context-s (that, distal)

  - creates $s_{E\,fin-deictic,m-class-none}$ with $@_{E:}$(**context**)

8.3 Determiners

Families:

1. **Det** *a, the, these* $\vdash$ $np_{T\,3rd}/n_T$:
   no-semantics
2. **SDet** *every, a_few, more* $\vdash$ $np_{T\,3rd}/n_T$:
   $@_{T:entity}(\ \langle Modifier\rangle(*)\ )$
3. **Un** $-$ **to** $-$ **Spec** $-$ **Det** *three, four* $\vdash$ $n_{T\,s-pl-sp}/n_{T\,s-pl-unsp}$:
   $@_{T:entity}(\ \langle Modifier\rangle(*)\ )$
4. **Group** $-$ **np** *some, any* $\vdash$ $np_T/obl_{s\,of}$:
   $@_{T:entity}(\ \mathbf{group} \wedge \langle Set\rangle(\text{s:entity})\ )$
5. **SGroup** $-$ **np** *some, any* $\vdash$ $np_T/obl_{s\,of}$:
   $@_{T:entity}(\ \mathbf{group} \wedge \langle Modifier\rangle(*) \wedge \langle Set\rangle(\text{s:entity})\ )$
6. **SGroup** $-$ **n** *first, second* $\vdash$ $n_T/obl_{s\,of}$:
   $@_{T:entity}(\ \mathbf{group} \wedge \langle Modifier\rangle(*) \wedge \langle Set\rangle(\text{s:entity}\ )\ )$
7. **Det** $-$ **poss** $-$ **s** *'s* $\vdash$ $np_{T\,3rd}/n_T \setminus_* np_O$:
   $@_{T:entity}(\ \langle Owner\rangle(\text{o:entity}\ )\ )$
8. **Numer** $-$ **id** *101, three* $\vdash$ nid_N :
   $@_{N:number-id}(*)$

Note: 3 does not result in an np, another determiner is required, e.g. *the three balls*
See §2.3.3 for a discussion of groups.
7 is used with the rule below to create a 'post-determiner' for *floor 3*, *office 101*, etc.
Possessive pronouns (e.g. *my, your, his*) are handled in the noun family **Owner** $-$ **pro**

Rules:

1. turns a number-id (nid, see above) into a post-determiner, creating a specific, singular and unique entity which is 'identified' by this number: e.g. *go to floor 3*
   $nid_N \implies$
   $np_{T\,3rd} \setminus_* n_{T\,s-sg}$:
   $@_{t1:entity}(\ \langle Spec\rangle(\text{unique, specific, sg}) \wedge \langle Modifier\rangle(\text{N:number-id})\ )$ [26]

Dictionary Forms:

[26] this rule is not incremental. It should be replaced by a rule which turns a n into an np $/_*$ nid

- DET (form, num, args)
  det (a, sg, existential specific)

  UN-TO-SPEC-DET (form, class)
  un-to-spec-det (two, number-cardinal)

  SDET (form, num, class, args)
  sdet (three, pl, number-cardinal, existential specific)

  - specifies the syn-num of the n compliment, adds specification to T through features and/or $\langle Modifier \rangle$

- GROUP-NP (form, args)
  group-np (any, variable specific)

  SGROUP-NP (form, class, args)
  sgroup-np (three, number-cardinal, existential specific)

  SGROUP-N (form, class, args)
  sgroup-n (third, number-ordinal, sg s-sg)

  - creates a group with head n or np, subset specified by feats and/or $\langle Modifier \rangle$

- NUMBER-ID (form)
  number-id (three)

  - creates an entry, e.g. $@_{t1:number-id}$(**three**)

8.4 Verbs

Syntactic Features:

- NUM AND PERS: *see Nouns*

- MOOD: *s-major*{ *s-ind s-imp s-int s-ind-ell* } *s-minor*
  s-ind-ell is given to indicatives with contextualized (ellighted/dropped) subjects
  s-minor is given to clauses which are selected for. In this grammar, this includes all lexical verbs (see Mood).

- POL: *s-pos s-prov-pos s-neg*
  p-prov-pos is for verbs which are provisionally positive, i.e. they are able to negated (e.g. *walks* is *s-pos* but *can* is *s-prov-pos*)

- VFORM: *fin*{ *fin-clause*{ *fin-full fin-ctxt*} *fin-deictic*} *vf-base vf-to-imp inf ing pp vf-be*
  pp is past-particple: *I haven't seen him*
  vf-be is for 'adjectival verbs' *I am able to go*
  vf-to-imp is the same form as *vf-base*, but exists to allow lexical control of imperative.
  fin-deictic is for pronominal verbs *I said it*
  fin-ctxt is for subj+finite clauses *I did, I should, he couldn't*
  fin-full is for all other finite clauses *I am hungry, I walked*

The separation of *fin* into two levels is necessary because general sentential complements allow deictics, but the indicative mood rules requires a clausal compliment (i.e. ** I this*)

- FIN: *be do can should will could would must have*
  this feature is currently not being used, but it will allow for handling tag questions *I can do it can't I*

- MCLASS: *s-manner s-instrumental, ... m-class-1, m-class-2, ...*
  the *s-....* values are grouped into classes *m-class-x*. These classes are lexically selected and specify the types of modifiers which can modify this clause (see §4 for description, see `types-feature.ccg` for full list of values)

8.4.1 Basic Verbs

Basic Families:
All standard verb family entries receive:
$\vdash \quad s_{E,num:num,pers:pers,mood=s-minor} \backslash_! np_{S,num:num,pers:pers,case=nom} \quad \ldots \quad :$
$@_{E:event}(* \wedge \langle Actor \rangle (S{:}entity) \ . \ . \ . \ )$

In the following descriptions, only the additional structure (i.e. that of the verbs compliments) will be given Also, in the remainder of this document *num:num, pers:pers* will be abbreviated as *agr*

1. **iv** *I walked* $\vdash \quad - \quad :$

   -

2. **tv** *I saw the ball* $\vdash \quad /np_{P\,acc} \quad :$

   $\wedge \langle Patient \rangle (P{:}entity)$

3. **v + at − np** *I looked at the ball* $\vdash \quad /obl_{P\,at} \quad :$

   $\wedge \langle Patient \rangle (P{:}entity)$

   also families for at-np, to-np, with-np, for-np

4. **v + np + prt** *I pick up the ball / picked it up* $\vdash \quad /np_{P\,acc}/prt_R \quad :$

   $\wedge \langle Patient \rangle (P{:}entity) \wedge \langle Particle \rangle (R) \ \vdash \quad /prt_R/np_{P\,acc} \quad :$
   $\wedge \langle Patient \rangle (P{:}entity) \wedge \langle Particle \rangle (R)$

5. **v + adj** *I feel happy* $\vdash \quad /adj_R \quad :$

   $\wedge \langle Result \rangle (R{:}quality)$

6. **v + pp − loc** *it goes on the table* $\vdash \quad /pp_R \quad :$

   $\wedge \langle Result \rangle (R{:}m\text{-}location)$

   also family for pp-whereto

7. **v + np + pp − whereto** *I put it on the table* $\vdash$ $/pp_{R}/np_{P\,acc}$:
   $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Result \rangle$(R:m-whereto)

   also family for pp-loc (*I want it on the table*)

8. **v + np + adj** *I made it bigger* $\vdash$ $/adj_{R}/np_{P\,acc}$:
   $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Result \rangle$(R:property)

9. **dtv** *I gave him the ball* $\vdash$ $np_{P\,acc}/np_{R\,acc}$:
   $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Recipient \rangle$(R:entity)

10. **dtv − to** *I gave it to him* $\vdash$ $obl_{R\,to}/np_{P\,acc}$:
    $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Recipient \rangle$(R:entity)

11. **v + sent − ind** *I thought I was hungry* $\vdash$ $/s_{V\,fins-ind}$:
    $\wedge \langle Event \rangle$(V:event)

12. **v + deictic − event** *I did it* $\vdash$ $/s_{V\,fin-deictic}$:
    $\wedge \langle Event \rangle$(V:event)

13. **v + verb − inf** *I need to eat* $\vdash$ $/(s_{V\,inf} \setminus_{!} np_{S})$:
    $\wedge \langle Event \rangle$(V:event)

    also family for verb-ing (*I kept going*)

14. **v + np + verb − inf** *I wanted him to eat* $\vdash$ $/(s_{V\,inf} \setminus_{!} np_{P})/np_{P}$:
    $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Event \rangle$(V:event)

    also family for verb-ing (*I kept it going*) and verb-base (*I helped him eat*)

15. **v + instrumental − np + verb − inf** *you use it to make coffee* $\vdash$ $/(s_{V\,inf} \setminus_{!} np_{P})/np_{P}$
    :
    $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Event \rangle$(V:event $\langle Modifier \rangle$(m-instrumental $\wedge$ **with**
    $\wedge \langle Anchor \rangle$(P)))

16. **v + np + from − np − result** *I made it from plastic, it is made from plastic* [27] $\vdash$ $/obl_{R\,from}/np_{P\,acc}$:
    $\wedge \langle Patient \rangle$(P:entity) $\wedge \langle Result \rangle$(R:entity)

    also family for of+np and out_of-np (*I helped him eat*)

Special Families:

[27] this is actually as passivization of this family

1. **imp – do** *don't be so loud* $\vdash$ $s_{E\,s-imp}/(s_{E\,vf-base} \backslash_! np_s)$:
   $@_{E:event}(\langle Mood \rangle(\text{imp}) \wedge \langle Subject \rangle(\text{S:entity } \textbf{addressee}))$
2. **imp – lets** *let's go get some coffee* $\vdash$ $s_{E\,s-imp}/(s_{E\,vf-base} \backslash_! np_s)$:
   $@_{E:event}(\langle Mood \rangle(\text{imp}) \wedge \langle Subject \rangle(\text{S:entity } \textbf{addressee} + \textbf{speaker}))$
3. Thanks and Thank-you (see verbs.ccg)

Dictionary Forms:

For each of these dictionary forms, there is a corresponding version xxxx-no-imp which 'blocks' the imperative form of this verb, i.e. it does not create a *vf-to-imp* entry.

- VERB (stem, ing, pasttense, pastpart, modifier-class, class, families)
  verb (give, giving, gave, given, m-class-2, action-non-motion, tv dtv dtv-to)

  - for each family specified, creates an entry for each form
  (i.e. a past tense form, past-particple form, etc)

- VERB-REG (stem, modifier-class, class, families)
  verb-reg (want, m-class-3, xxxxx, tv v+verb-inf v+np+verb-inf v+np+pp-loc v+np+adj)

  - works identically, but for verbs that are perfectly regular
  (i.e. stem, stem+ing, stem+ed, stem+ed)

- ADJECTIVAL-VERB (stem, modifier-class, class)
  adjectival-verb (able, m-class-3, ability)

  - creates one form which is selected by the auxilliary *be* verb

8.4.2 Copula *be*

Recall that the copula *be* has 3 basic argument structures (see §6):

1. *it is a ball* : an np argument which agrees in number and person with the subject np, blocking readings like *they are a ball*.
2. *it is on the table ball* : a pp argument, which must have a semantic type which can modify entities.[28]
3. *it is blue* : an adj argument.

Each of these has an entry for the minor form, y/n interrogative form, minor form and compliment-questioning form .

[28] currently, m-location *I am in the room*, m-accompaniment *I am with you*, and m-benefactor *this is for you*. There is a separate entry for each.

1. Minor Entries (used in indicatives and questioning the $\langle Cop - Restr \rangle$) each with:
   $@_{E:event}(* \langle Cop - Restr \rangle$(S:entity) $\langle Cop - Scope \rangle$(X))
   $\vdash$ $s_{E\,agr,s-minor} \backslash_! np_{s\,agr,nom}/adj_X$:
   $\vdash$ $s_{E\,agr,s-minor} \backslash_! np_{s\,agr,nom}/pp$:
   $\vdash$ $s_{E\,agr,s-minor} \backslash_! np_{s\,num:num,pers:pers,nom,questionable=no}/np_{num:num}$:
   [29]

2. Yes/No Interrogative Entries each with:
   $@_{E:event}(* \langle Mood \rangle$(int) $\wedge \langle Subject \rangle$(S:entity)
   $\wedge \langle Cop - Restr \rangle$(S:entity) $\langle Cop - Scope \rangle$(X))
   $\vdash$ $s_{E\,agr,s-minor}/adj_X/np_{s\,agr,nom}$:
   $\vdash$ $s_{E\,agr,s-minor}/pp/np_{s\,agr,nom}$:
   $\vdash$ $s_{E\,agr,s-minor}/np_{num:num}/np_{s\,num:num,pers:pers,nom}$:

3. Entries for questioning the $\langle Cop - Scope \rangle$, selected for by the wh-word (see below):
   $@_{E:event}(* \wedge \langle Subject \rangle$(S:entity)
   $\wedge \langle Cop - Restr \rangle$(S:entity) $\langle Cop - Scope \rangle$(X))
   $\vdash$ $cop_{E\,agr}/adj_X/np_{s\,agr,nom}$:
   $\vdash$ $cop_{E\,agr}/pp/np_{s\,agr,nom}$:
   $\vdash$ $cop_{E\,agr}/np_{num:num}/np_{s\,num:num,pers:pers,nom}$:

4. Negation: There is a negation entry for each of these entries. See verbs.ccg for details

8.4.3 Presentational *be*

The presentational *be* construction has three entries corresponding to its different uses:

1. Minor Entry *there are some balls* / *what were there* $\vdash$ $s_{E\,s-minor} \backslash_! np_{D\,dummy-there}/np_{s\,agr}$:
   $@_{E:event}(* \wedge \langle Presented \rangle$(S:entity))

2. Y-N Interrogative *were there any balls* $\vdash$ $s_{E\,s-int}/np_{s\,agr} \backslash_! np_{D\,dummy-there}$:
   $@_{E:event}(* \wedge \langle Mood \rangle$(int) $\wedge \langle Subject \rangle$(D:dummy) $\wedge \langle Presented \rangle$(S:entity))

3. Inverted Locational *on the table was a ball* $\vdash$ $s_{E\,s-int}/np_{s\,agr} \setminus M_{pp\,cc-type:post-n}$:
   $@_{E:event}(* \wedge \langle Mood \rangle$(ind) $\wedge \langle Modifier \rangle$(M:m-location) $\wedge \langle Presented \rangle$(S:entity))

It is the presented object which determines agreement, i.e. * *there is some balls*. The subject *there* is a 'dummy' contributing nothing to the propositional semantics of event. It consequently receives no semantic role beyond the standard $\langle Subject \rangle$.

[29] the subject in this entry is blocked from being questioned, i.e. what is this receives only one reading, questioning the $\langle Cop - Scope \rangle$

Note that having both 1 and 3 means that the sentence *there was a ball* correctly receives two readings:

```
@b1:presentational(be ^
                   <Mood>ind ^
                   <Tense>past ^
                   <Modifier>(c1:m-location ^ context ^
                              <Proximity>m-distal) ^
                   <Presented>(b2:thing ^ ball ^
                               <Delimitation>existential ^
                               <Num>sg ^
                               <Quantification>specific))

@b1:presentational(be ^
                   <Mood>ind ^
                   <Tense>past ^
                   <Presented>(b2:thing ^ ball ^
                               <Delimitation>existential ^
                               <Num>sg ^
                               <Quantification>specific) ^
                   <Subject>(t1:dummy ^ there))
```

8.4.4 Auxiliary and Modal Verbs

These are the five entries for the families handling modal verbs :

Modal – vf – base *can, should, would*

1. Indicative *I can walk*
   acts like an adverbial modifier which also "changes" the vform of the verb. $\vdash$ $(s_{E:vform:fin,mclass:mclass,pol:pol} \backslash_! np_{s:agr})/_\wedge(s_{E:vf-base,mclass:mclass,pol:pol} \backslash_! np_{s:agr})$:
   $@_{E:event}(\ \langle Modifier\rangle(*)\ )$

2. Indicative Contextual-Event *I can*
   the main event is set to **context**. m-class restricted to avoid crazy post mod $\vdash$ $s_{E:fin-ctxt,m-class-3} \backslash_! np_{s:agr}$:
   $@_{E:event}($ **context** $\wedge \langle Modifier\rangle(*)\ )$

3. Yes/No Interrogative *can you walk* $\vdash$ $s_{E:s-int}\ /(s_{E:vf-base} \backslash_! np_{s:agr})\ /np_{s:agr}$:
   $@_{E:event}(\ \langle Modifier\rangle(*) \wedge \langle Mood\rangle(\text{int}) \wedge \langle Subject\rangle(\text{S:entity})\ )$

4. Yes/No Interrogative Contextual-Event *can you*
   the main event is set to **context**. m-class restricted to avoid crazy post mod $\vdash$ $s_{E:s-int,fin-ctxt,m-class-3}\ /np_{s:agr}$:
   $@_{E:event}($ **context** $\wedge \langle Modifier\rangle(*) \wedge \langle Mood\rangle(\text{int}) \wedge \langle Subject\rangle(\text{S:entity}))$

5. Atomic *can*
   this is selected for by wh-words and other UDCs (see below). $\vdash$ $aux_{E:vf-base}$

:
$@_{E:event}(\ \langle Modifier\rangle(*)\ )$

There are only two differences between the treatment of modal verbs and auxiliary verbs (e.g. *be, have, do* etc.). First, whereas modals add their own propositional head using $\langle Modifier\rangle$ (see section X), auxiliaries do not, adding only tense, aspect and voice features lexically (see secton X). Second modals always select a verb in base form whereas auxiliaries can choose different forms (e.g. *am walking, did walk*). Thus, there are 4 other families essentially identical to this except they have no $\langle Modifier\rangle$(*) and instead of having *vf-base* the selected verbal group has *pp, ing, vf-be* and *vf-base*.

One exception is passive auxiliary *be*. This requires more complex structure. There are two indicative and two yes/no interrogative entries. The first is for an unexpressed $\langle Actor\rangle$. This is marked with $@_{S:entity}$(**context**). The second is for an $\langle Actor\rangle$ specified by an oblique *by* phrase. Note that the feature $\langle Voice\rangle$(passive) is added lexically and that the object becomes the $\langle Subject\rangle$

Aux – passive *the ball was picked up , the ball was picked up by GJ*

1. $\vdash\ (s_{E\,vform:fin,mclass:mclass,pol:pol} \backslash_! np_x)/_\wedge(s_{E\,pp,mclass:mclass,pol:pol} \backslash np_{s\,agr}/np_x)$:
   $@_{S:entity}$(**context**)
2. $\vdash\ (s_{E\,vform:fin,mclass:mclass,pol:pol} \backslash_! np_x)/_\wedge obl_{s\,by}/_\wedge(s_{E\,pp,mclass:mclass,pol:pol} \backslash np_{s\,agr}/np_x)$
   :
   no semantics
3. $\vdash\ s_{E\,s-int}\ /(s_{E\,pp} \backslash np_{s\,agr}/np_x)\ /np_{x\,nom}$:
   $@_{E:event}(\ \langle Mood\rangle(\text{int}) \wedge \langle Subject\rangle(\text{X:entity})\ )$
   $@_{S:entity}$(**context**)
4. $\vdash\ s_{E\,s-int}\ /_\wedge obl_{s\,by}\ /(s_{E\,pp} \backslash np_{s\,agr}/np_x)\ /np_{x\,nom}$:
   $@_{E:event}(\ \langle Mood\rangle(\text{int}) \wedge \langle Subject\rangle(\text{X:entity})\ )$

The *get* passive takes only the indicative entries, not the y-n interrogatives, i.e. *did you get hit* not * *got you hit*

Dictionary Forms:

- MODAL (form, class, args)
  modal (can, ability,)

  - creates two forms, a positive form with *s-prov-pos* and a negative form (form + nt) with *s-neg* and $\langle Polarity\rangle$(neg)

All of the auxiliary verbs are individually specified in `dictionary-closed.ccg`

8.5 Mood Rules

These are the Mood Rules described in §5.

1. $np_{S\,nom,agr}$ \$1 $\Longrightarrow$
   $s_{E\,s-ind}$ /($s_{E\,VFORM:fin-clause,s-minor,agr}$ $\backslash_!$ np_S) \$1 :
   $@_{E:event}$($\wedge$ $\langle Mood \rangle$(ind) $\wedge$ $\langle Subject \rangle$(S:entity))

2. $np_{F\,acc}$ \$1 $\Longrightarrow$
   $s_{E\,s-ind}$ /($s_{E\,VFORM:fin-clause,s-minor,agr}$ $\backslash_!$ np_S/np_F) /$np_{S\,nom,agr}$ \$1 :
   $@_{E:event}$($\wedge$ $\langle Mood \rangle$(ind) $\wedge$ $\langle Subject \rangle$(S:entity) $\wedge$ $\langle Fronted \rangle$(F:entity))

3. $s_{E\,agr,vf-to-imp}$ $\backslash_!$ $np_{S\,agr,nom}$ ... $\Longrightarrow$
   $s_{E\,s-imp}$ ... :
   $@_{E:event}$($\wedge$ $\langle Mood \rangle$(imp) $\wedge$ $\langle Subject \rangle$(S:entity $\wedge$ **addressee**)) [30]

4. $s_{E\,agr,fin}$ $\backslash_!$ $np_{S\,agr,nom}$ ... $\Longrightarrow$
   $s_{E\,s-ind-ell}$ ... :
   $@_{E:event}$($\wedge$ $\langle Mood \rangle$(imp) $\wedge$ $\langle Subject \rangle$(S:entity $\wedge$ **context**))

1 and 2 require their verbal compliment to be finite and clausal (i.e. * *I this*) and also of mood *s-minor*
3 and 4 are actually a family of rules, one per argument structure i.e. one for each verb family.
The need for multiple rules is the result of the combination of 1) the manner in which verbal arguments are ordered in (most?) CCG grammars and 2) the specific workings of the \$ operator. Despite 'coming' first, subjects are built as the last complement of verbs, i.e. a transitive verb receives: $s_E \backslash np_S/np_X$, not the more 'natural' (i.e. incrementally iconic): $s_E/np_X \backslash np_S$. If the latter were employed, a single rule of this nature would handle all argument structures:

$s_{E...}$ \$1 $\backslash_!$ $np_{S...}$ $\Longrightarrow$ $s_{E...}$\$1 : $@_{E:event}$...

However, because the \$ operator attaches to the *immediately previous* category, in this analog rule

$s_{E...}$ $\backslash_!$ $np_{S...}$ \$1 $\Longrightarrow$ $s_{E...}$\$1 : $@_{E:event}$...

the \$ would attach to the preceding subject np, i.e. it would not 'stand for' the (potential) compliments of the verb, but of the subject. This is clearly not what we want, hence the multiple rules.

8.6 Open-Class Modifiers

In this section we outline some of the common features of adjectives, prepositions and adverbs. This will allow simplification of these individual sections.

Syntactic Features:

- MOD-TYPE: *x-manner x-instrumental ....*
  this syntactic feature mirrors the semantic sort of the modifier. It is used in modifier restriction (currently only event modifiers)

[30] *vf-to-imp* is the same form as *vf-base*. It was added to allow verbs to lexically determine whether or not they can generate imperative forms

- CC-TYPE: *post-s pre-s pre-vp post-vp post-n pre-n + classes (see below)* these 'atomic' values are collected into classes using multiple inheritance, e.g. *prep-1 [post-s post-vp post-n]* . Modifiers are lexically assigned a class which specifies the rules it can undergo, i.e. it's combinatorial possibilities.

- DEGREE: *s-no-degrees s-degree* {*s-degree-base s-comparative s-superlative*} *s-no-degree* is given to modifiers which cannot be given degrees, e.g. stative adjectives *s-degree-base* is given to the base form of modifiers, which can then be changed, e.g. *big* $\Longrightarrow$ *more big*

Rules:

Recall from §3 that all open-class modifiers receive an initial lexical entry containing an atomic syntactic head with its associated semantic index, e.g. generic prepositions begin as $pp_{M\,mod-type,cc-type}/np_T$. In general then we have $cat_{M\,mod-type,cc-type}$ (+comps).

Each of the values of cc-type corresponds to a complex syntactic category:[31]

1. *pre-n* : n_T/n_T (+comps)
2. *pre-s* : s_E/s_E (+comps)
3. *pre-vp* : $(s_E \backslash_! np_s)\ /(s_E \backslash_! np_s)$ (+comps)
4. *post-n* : n_T (+comps) $\backslash_* n_T$
5. *post-s* : s_E (+comps) $\backslash_* s_E$
6. *post-vp* : $(s_E \backslash_! np_s)$ (+comps) $\backslash_* (s_E \backslash_! np_s)$
7. *pre-cop-comp* : *that is <u>also</u> a ball, I am <u>certainly</u> bigger, he is <u>never</u> here* [32]

Each of the values of mod-type corresponds to one of the 'atomic' values of mclass (see verbs above), e.g. *x-location* (modifier feature) : *s-location* (s feature).

For each category and each combination of cc-type and mclass, there is a rule which changes the atomic lexical cat into a complex modifying cat which imposes its restriction on its modifiee , e.g.

$cat_{M\,x-location,post-s}\ \$_1 \Longrightarrow$
$s_E\ \$_1\ \backslash_*\ s_E\ _{mclass:s-location}$: $@_{E:event}(\ \langle Modifier\rangle$M:m-location)

The use of $\$_1$ ensures that any compliments, lexically specified or 'picked up' in some other way, are carried through after the change.

Because all of the atomic-to-complex modifier rules follow the same basic pattern, we will not list them individually.

[31] Note that the placement of the comps is dependent on the positioning of the modifier : pre vs post modifee. This is essential for the proper incremental parsing of modification.

[32] see modifiers.ccg for the specific details of these three categories

Modifier Modifiers:

Each open class modifier has two families for handling pre and post modifiers:
$\vdash \quad pos_M/pos_M \quad :$
$@_{M:modifier}(\ \langle Modifier\rangle(\text{X:m-class } *)\ )$
$\vdash \quad pos_M \setminus_* pos_M \quad :$
$@_{M:modifier}(\ \langle Modifier\rangle(\text{X:m-class } *)\ )$

There is a single dictionary form which uses these families to create entries:

MODIFIER (form, pos, side, class)
modifier (really, adj, pre, intensity)

Note that each form requires its own dictionary entry

8.7 Adjectives

Families:

1. **Adj** *big, red, better* $\vdash \quad adj_M \quad :$
   $@_{M:property}(\ *\ )$
2. **Mod – pre – adj – comparative** *much* $\vdash \quad adj_M/adj_{M\,comparative} \quad :$
   $@_{M:modifier}(\ \langle Modifier\rangle(*)\ )$
3. **More – adj** *more* $\vdash \quad adj_{M\,comparative}/adj_{M\,degree-base} \quad :$
   $@_{M:modifier}(\ \langle Degree\rangle(\text{comparative})\ )$
4. **Most – adj** *most* $\vdash \quad adj_{M\,superlative}/adj_{M\,degree-base} \quad :$
   $@_{M:modifier}(\ \langle Degree\rangle(\text{superlative})\ )$
5. **Adj – er – than** *than* $\vdash \quad adj_M/np_A \setminus_* adj_{M\,comparative} \quad :$
   $@_{M:modifier}(\ \langle Modifier\rangle(* \wedge \langle Anchor\rangle(\text{a:entity})\ )\ )$

Rules: see above

Dictionary Forms:

- ADJ-NONE (base, class, args)
  adj-none (wrong, q-attitude,)

  - creates $adj_{M\,s-degree-base}$ with $@_{M:class}(\textbf{form})$

- ADJ-DEG (base, comp, sup, class, args)
  adj-deg (big, bigger, biggest, q-size,)

  - creates an entry for each degree value, e.g. $adj_{M\,s-comparative}$ with $@_{M:class}(\textbf{form} \wedge \langle Degree\rangle(\text{comparative})\ )$

8.8 Adverbs

Families:

1. **Adv** *quickly, now, forward* $\vdash$ adv_M :
   $@_{M:modifier}(*)$
2. **Adv + dep − clause** *when, while, if* $\vdash$ $adv_M/s_{vfin,s-ind}$:
   $@_{M:Modifier}(* \wedge \langle Event \rangle$(V:event))

Rules: see above

Dictionary Forms:

- ADVERB (base, class, cc-class, args)
  adverb (quickly, manner, adv-all,)
  - creates $adv_{Ms-degree-base}$ with $@_{M:class}$(**form**)
- ADVERB-DEG (base, comparative, superlative, class, cc-class, args)
  adverb-deg (soon, sooner, soonest, time, adv-all,)
  - creates an entry for each degree value, e.g. $adv_{Ms-comparative}$ with $@_{M:class}$
- ADVERB+DEP-CLAUSE (base, class, cc-class, args)
  adverb+dep-clause (when, time, adv-1,)
  - see family above.

8.9 Prepositions

Families:

The $\langle Anchor \rangle$ complement sub-categorizes for entities of sort physical (man, table, kitchen) , time-unit (minute, second, hour) or section(right, left, edge, corner, side). This guarantees that *in the office* and *in five minutes* receive disjoint readings and that *I am to your right* parses but |*I am to the kitchen* doesn't.[33] The standard families subcat for physical.
The compliment of prepositions is marked as case *acc-loc*. This is also marked on the pro-locational nps *here* and *there*. Thus, *in here, up there* are permitted but *I took here and here walked* are not

1. **Prep** *in , through, with, for* $\vdash$ $pp_M/np_{Aacc-loc}$:
   $@_{M:modifier}(* \wedge \langle Anchor \rangle$(A:physical)))

[33] this sentence of course does have uses, e.g. when giving feedback indicating the current progress of a route travelled, but this is behind the scope of the current grammar

2. **Prep + of − np−** *right, in_front, on_top* $\vdash$ $pp_M/obl_{A\,of,acc-loc}$:
   $@_{M:modifier}(* \wedge \langle Anchor \rangle$(A:physical)))

   there are similar families for *to* and *from*, i.e. *next to* and *away from*

3. **Prep + no − arg−** *here, there, somewhere* $\vdash$ pp_M :
   $@_{M:modifier}(*)$

4. **Prep − −time − unit** *for five minutes, in two weeks* $\vdash$ $pp_M/np_{A\,acc-loc}$:
   $@_{M:modifier}(* \wedge \langle Anchor \rangle$(A:e-time-unit)))

Rules:

These rules turn any locational/dynamic preposition into a first conjunct. This is required to handle location chains. The rule are projective (they add a new 'complement' to the parse), and because they apply to any such preposition, will always 'fire'. Forcing the conjuncts and the result to have the same cc-type weeds out a lot of unnecessary parses.

1. $pp_{F\,x-dynamic,cc-type:cc-type} \Longrightarrow$
   $pp_{R\,x-dynamic,cc-type:cc-type}/pp_{N\,x-dynamic,cc-type:cc-type}$:
   $@_{R:m-dynamic}($ **list** $\langle First \rangle$(F:m-dynamic) $\wedge$ $\langle Next \rangle$(N:m-dynamic))

2. $pp_{F\,x-location,cc-type:cc-type} \Longrightarrow$
   $pp_{R\,x-location,cc-type:cc-type}/pp_{N\,x-location,cc-type:cc-type}$:
   $@_{R:m-location}($ **list** $\langle First \rangle$(F:m-location) $\wedge$ $\langle Next \rangle$(N:m-location))

Dictionary Forms:

- PRP (form, class, cc-class, args)
  prp (into, whereto, prep-2,)

  - any prep with a physical $\langle Anchor \rangle$

- PRP− (subcat, form, class, cc-class, args)
  prp− (time-unit, for, time-interval, prep-2,)

  - time unit $\langle Anchor \rangle$

- PRP+ (comp, form, class, args)
  prp+ (of-np, right, location, prep-1,)

  - This also handles no-arg entries

8.10 Wh-Words

All wh-word entries control the ordering of the clausal elements (1) , set the syntactic and semantic mood of the clause, and add the semantic role $\langle Wh - Restr \rangle$ which

specifies the nature and the scope of the questioned item (3). In some cases, they also have their own compliments which are used to further build up the $\langle Wh-Restr \rangle$ (2). Thus, all Wh-words have the same top level structure and in what follows we specify only these three components:

$\vdash \quad E_{s\,s-int} + 1(+2) \quad :$
$@_{E:event}(\ \langle Mood \rangle(\text{int}) \wedge \langle Subject \rangle(\text{S}) \wedge \langle Wh-Restr \rangle(\ +\ 3\ )\ )$

There are a wealth of wh-word families each with a potentially large number of entries. We will begin by separating these families into two broad groups: those which question a role (i.e. they 'fill' an extracted argument) and those which question an event itself (they do no fill an argument slot).

8.10.1 Questioning a Role

In this section,we will refer to the questioned role as *item*. Within this sub-set of wh families, we can further divide the entries along two lines. First, the nature of item (subject vs. other) determines a lot about the structure of the entry.

- Each subject entry has: (1) = $/(s_{E\,fin} \backslash_! np_{s\,3rd,s-sg})$ and (3) involves $@_{S:entity}$, .i.e. there is a co-indexing with the subject.
- Each other entry involves a verbal group 'missing' a complement, i.e. $(s_{E\,vform:vform} \backslash_! np_s/item)$. This will be abbreviated to v − minus(*item*)

Second, the nature of the clause (copula/presentational-be vs. auxiliary/modal verb + other-verb) is important. Whereas the copula verb in *where is the ball* controls the syntax and semantics of the clause, the aux in *what is he looking at* and the modal in *where can I put it* do not: they merely add some semantic information to the clause.[34] Note that this distinction is not important for subject questions: there is no need for a separate aux/modal entry corresponding to *who went to bathroom*

- Each copula entry has: (1) = $/(cop_{E\,agr}\ item)$
  Note that in this case, the subject is handled by the copula verb itself. This 1) allows questioning $\langle Anchor \rangle$ in, e.g. *who is he with* [35] and 2) blocks it in *who is on xx the table*.
- There are two aux/modal entries:
  1. has (1) = $/\text{v} - \text{minus}(item)/np_{s\,3rd,s-sg}/aux_{E\,pol:pol,fin:fin,...}$
  2. has (1) = $/\ (s_G\ \$_1\ item)\ /\text{v} - \text{minus}(/\ s_G\$_1)/np_{s\,3rd,s-sg}/aux_{E\,pol:pol,fin:fin,...}$

  The first handles cases where the questioned item occurs in the main clause, e.g. *what did he pick up* <u>*xxx*</u> and the second where it occurs in a subordinate clause, e.g. *what did he want you to pick up* <u>*xxx*</u> . In the latter case, note that this subordinate clause compliment ($/s_G\ \$_1\ item$) must be 'repeated' as

[34] as well as determining various syntactic features, e.g. verbal agreement, vform of the main verb, etc.
[35] although this is non-incremental

a compliment of this wh-construction following the verbal group compliment (v – minus(...))

Families:

1. **Wh – np–** *who, what*

2. **Wh – np – spec–** *which ball, ulwhat ball*

   item = /np_F
   (2) = /$n_{F\,s-sg}$
   (3) = @$_{V:??}$($*$ ∧ ⟨*Scope*⟩(F:entity))

3. **Wh – np – spec – ctxt–** *which*

   item = /np_F
   (2) = no compliments (the semantic head is 'contextualized' as in *which do you want*
   (3) = @$_{V:??}$($*$ ∧ ⟨*Scope*⟩(F:entity **context**))

   There are also two families for questioning the quantity of referents which are countable (i.e. *how_many*) or uncountable (i.e. *how_much*). [36] The only difference is the number feature value of (2). Each of these families contain entries for both nominally specified for contextualized semantic head. The reason why these have been separated into two families in the case np-spec above is that whereas we want the nominally specified specifier reading for *what* (i.e. *what ball did you pick up*) we did not want the contextualized reading, i.e. *what did you pick up* should not receive two readings with differing ⟨*Wh – Restr*⟩.

4. **Wh – np – qclass–** *which color ball*

   item = /np_F
   (2) = /n_F /$qclass_Q$ [37]
   (3) = @$_{V:??}$($*$ ∧ ⟨*Scope*⟩(Q ⟨*Scope*⟩(F:entity)))
   Ê
   This family also includes a set of entries for contextualized entities allowing the more common *which color do you want*.

```
@w1:cognition(want ^
              <Mood>int ^
              <Tense>pres ^
              <Actor>(y1:person ^ you ^  <Num>sg) ^
              <Patient>(c1:entity ^ context) ^
              <Subject>y1:person ^
```

[36] this could and undoubtedly should be handled using a single form of *how* and two separate lexical entries for *much* and *many*

[37] The category qclass, which stands for Quality Class, refers to classes of properties such: *color, size, shape, temperature, etc.*

```
              <Wh-Restr>(w2:specifier ^ which ^
                         <Scope>(c2:quality ^ color ^
                                <Scope>c1:entity)))
```

This corresponds to the situation where, say, the recipient is being asked to choose between a blue, a green and a red ball, an answer like *blue* would be referring to the blue ball. Consequently, it also makes sense to treat the question as referring to an entity with a contextualized semantic head, in this case **ball**. In other words, this reading is a better representation of this question's semantics than:

```
@w1:cognition(want ^
              <Mood>int ^
              <Tense>pres ^
              <Actor>(y1:person ^ you ^  <Num>sg) ^
              <Patient>(c1:entity ^ color) ^
              <Subject>y1:person ^
              <Wh-Restr>(w2:specifier ^ which ^
                         <Scope>c1:entity))
```

5. **Wh − pp−** *where*

   item = $/\mathsf{pp}_{\mathsf{F}}$
   (2) = no compliments
   (3) = $@_{F:modifier}(\,*\,) \vdash \quad :$
   $@_{:}(\ \ )$

6. **Wh − sent−** *what*

   item = $/\mathsf{s}_{\mathsf{F}}$ ..OR.. *item* = $(/\mathsf{s}_{\mathsf{F}\,inf} \backslash_{!}\, \mathsf{np}_{\mathsf{S}})$
   (2) = no compliments
   (3) = $@_{F:event}(\,*\,)$

   The first handles questioning full sentence complements *what did you say* and the second infinitival compliments *what did you want* where *want* is the same as in *I want to play with the ball*, i.e. the question is more or less *what did you want to do*.

7. **Wh − adj−** *how*

   item = $/\mathsf{adj}_{\mathsf{F}}$
   (2) = no compliments
   (3) = $@_{F:quality}(\,*\,)$

8. **Wh − adj − degree** *how big*, *how strong*

   item = $/\mathsf{adj}_{\mathsf{F}}$
   (2) = $/\mathsf{adj}_{\mathsf{F}}$
   (3) = $@_{V:??}(\,* \wedge \langle Scope\rangle(\mathrm{F{:}quality})\,)$

9. **Wh – adj – qclass** *which color, what shape*

   item = /adj_F
   (2) = /$qclass_F$
   (3) = $@_{V:??}(* \wedge \langle Scope \rangle$(F:quality))

   Consider the resultative use of the verb *make* exemplified in *I made it big*. These three families provide three ways of questioning this $\langle Result \rangle$ role, i.e. *how* / *how big* / *what size did you make it*. Of course they also provide alternate ways of questioning the adjectival $\langle Cop - Restr \rangle$ role in the copula verb, i.e. *how* / *how big* / *what size is it*.

8.10.2 Questioning an Event Modifier

The last family of wh-words contains entries which do not question one of the events 'required' roles, but instead some optional aspect, i.e. a modifier. For example, in the questions

- *where are you* **sitting**
- *where are you* **going**
- *when did you* **come** *in*
- *how did you want me to* **walk**

the wh word is questioning the bolded verb/event in terms of its static location, dynamic (where-to) location, time, and its manner respectively.

Wh – sent – modifier *where, how*

(1) = /(s_F *vform:vformmclass:mclass* \\! np_S) /np_S *3rd,s–sg*/aux_F *pol:pol,fin:fin,...*
..OR..
/ (s_F $1) /v – minus(/ s_F *mclass:mclass* $1)/$np_S$ *3rd,s–sg*/aux_E *pol:pol,fin:fin,...*
(2) = no compliments
(3) = $@_{V:modifier}(* \wedge \langle Scope \rangle$(F:event))

The first entry is used for questioning (i.e. scoping over) the main clause (first two examples above), and the 2nd some subordinate clause (third example). The constraint on this clause's *mclass* is lexically specified by the wh-word (see §4).

Dictionary Forms:

- WH-WORD (form, class, families)
  wh-word (which, specificer, Wh-np-spec- Wh-np-qclass- ...)

  Note that like the dictionary forms for verbs, multiple families can be evoked from a single entry.

- WH-WORD+FEATS (form, class, families, feats)
  wh-word+feats (where, Wh-sent-modifier, s-dynamic)

  This parrots wh-word but allows the addition of lexical macros. Note that this is how *where* sets restricts its scoped-over event to mclass = *s-dynamic*

- QUALITY-CLASS (form)
  quality-class (size)

  This creates a single entry of the form qclass $_{QW}$ ith $@_{Q:quality}$. It is currently only used in questions

8.11 Discourse Markers

Currently, discourse markers[38] (DMs) are given quite naive treatment. They are ordered linearly using the standard $\langle First \rangle$ and $\langle Next \rangle$ structure used in coordination. Thus, respectively, *right ok* and *right ok put it over there* receive:

```
@l1:d-units(list ^
            <First>(r1:marker ^ right) ^
            <Next>(o1:marker ^ ok))

@l1:d-units(list ^
            <First>(r1:marker ^ right) ^
            <Next>(l2:d-units ^ list ^
                   <First>(o1:marker ^ ok) ^
                   <Next>(p1:action-non-motion ^ put ^
                          <Mood>imp ^
                          <Actor>(a1:entity ^ addressee) ^
                          <Patient>(i1:thing ^ it ^ <Num>sg) ^
                          <Result>(c1:m-whereto ^ context ^
                                   <Proximity>m-distal ^
                                   <Modifier>(o2:direction ^ over)) ^
                          <Subject>a1:entity)))
```

A required extension is the handling of post positioned DMs, e.g. *it's in the kitchen right* / *isn't it* ... .

Families:

1. DM *yes, no, okay, right* $\vdash$ $\mathrm{du}_{R\,COM=no}$:
   $@_{D:marker}(*)$ [39] $\vdash$ $\mathrm{du}_{R\,COM=yes}/\mathrm{du}_{N\,COM=no}$:
   $@_{R:d-units}(\mathbf{list} \wedge \langle First \rangle(\mathrm{F{:}marker}\ *) \wedge \langle First \rangle(\mathrm{N{:}marker}) \vdash \mathrm{du}_{R\,COM=yes}/\mathrm{s}_{N\,mood:s-major}$:
   $@_{R:d-units}(\mathbf{list} \wedge \langle First \rangle(\mathrm{F{:}marker}\ *) \wedge \langle First \rangle(\mathrm{N{:}event})$

[38] also referred to as cue words/phrases, discourse particles/conenctives
[39] see 8.1 below for a description of COM

There is also a family DM+np which takes an np compliment with role ⟨*Addressee*⟩. It is used in greetings and closings, e.g. *hello Robot, bye GJ.*

Note that the pre-sentential entry restricts to major mood. Also, note that the parse for a sentence preceded by a DM will receive category du and not s

Vocative NPs are handled by giving all names a DM reading. Thus, *Robot can you come get it for me* receives:

```
@l1:d-units(list ^
            <First>(r1:animate ^ Robot) ^
            <Next>(g1:action-non-motion ^ get ^
                   <Mood>int ^
                   <Tense>pres ^
                   <Actor>(y1:person ^ you ^ <Num>sg) ^
                   <Modifier>(c1:modal ^ can) ^
                   <Modifier>(c2:m-manner ^ come) ^
                   <Modifier>(f1:m-benefactor ^ for ^
                              <Anchor>(i1:person ^ I ^  <Num>sg)) ^
                   <Patient>(i2:thing ^ it ^ <Num>sg) ^
                   <Subject>y1:person))
```

Dictionary Forms:

- DIS-MARKER (form, class)
  dis-marker (yes, alignment)

  DIS-MARKER+NP (form, class)
  dis-marker+np (hi, greeting)

  - these create three items corresponding to the three entries above.

8.12 Other Minor Families

Families:

1. **NP − marker** *I gave it to him* ⊢ $\text{obl}_{\text{T}MARK=*}/\text{np}_{\text{T}acc}$:

2. **Infinitive − to** *I want to help you* ⊢ $(\text{s}_{\text{E}inf} \backslash_! \text{np}_\text{s})/(\text{s}_{\text{E}vf-base} \backslash_! \text{np}_\text{s})$:
   ⊢ $\text{s}_{\text{E}inf} \backslash_! \text{np}_\text{s}$:
   $@_{E:event}$(**context**)

3. **For − verb − ing** *thank you for helping me* ⊢ $(\text{s}_{\text{E}for-ing} \backslash_! \text{np}_\text{s})/(\text{s}_{\text{E}ing} \backslash_! \text{np}_\text{s})$:

1 marks oblique entity compliments. 2 and 3 mark verbs.
The second entry in 2 contextualizes event compliments, as in *I wanted to*:

```
@w1:cognition(want ^
                <Mood>ind ^
                <Tense>past ^
                <Actor>(i1:person ^ I ^ <Num>sg) ^
                <Event>(c1:event ^ context) ^
                <Subject>i1:person)
```

8.13 Higher Order Dictionary Forms

MOLOKO has made use of DotCCG **def** macros by creating 'higher order' dictionary forms to group the various, sometimes disparate, entries of single words. For example, here is definition for direction words *right, left, front, back*:

```
def direction-word(loc) {
  prp+(of-np, loc, location, prep-all)
  prp+(of-np, loc, whereto, prep-2)
  prp+(no-arg, loc, direction, prep-2)
  noun+of-np(loc, e-region, cc-none)
  noun(loc, e-region, cc-none)
  adj-none(loc, q-location, )
}
```

Thus, the macro call: **direction-word(right)** 'calls' each of these 'atomic' macros creating the entire gamut of entries needed for handling the various behaviours of this word:

I am right of the table
go right of the table
go right
I am to the right of the table
I am to the right
I want the right one (i.e. not the left one)

Similar macros exist for numbers and determiners/context-nps/group-heads. In fact, the macro **pronoun** described in §8.2 above also operates in this way.

A useful future task would be to re-organize other areas of the dictionary in this way. Thinking in this way would also help in removing the redundancy in the ontology (etc. e-location, q-location, m-location).

9 Using MOLOKO

This section contains

1. help on getting MOLOKO running on your machine

2. information concerning MOLOKO's file structure
3. a general overview of the layout and organization of the grammar.
4. instructions for adding words, or more substantial grammatical alterations
5. instructions for compiling the grammar and testing it.

9.1 Getting Started

MOLOKO is written in DotCCG, the 'higher level' grammar writing language which extends OpenCCG. Consequently, you must first install both of these, along with all of their dependencies (Java, Python, etc.) See http://openccg.sourceforge.net for the relevant downloads, installation notes, and introductory documentation. Particularly, if you want to understand the OpenCCG native XML format better, please refer to the *OpenCCG Rough Guide*. The Rough Guide is provided with the OpenCCG distribution, under *doc/guide* (use `pdflatex` to compile the file `guide.tex`).

9.2 Folder Contents

To begin, MOLOKO (like any OpenCCG grammar) consists of two types of files: **generated** and **non-generated**.

> DO NOT MAKE CHANGES TO GENERATED FILES DIRECTLY.
> THESE CHANGES WILL NOT BE SAVED.
> FIND THE APPROPRIATE NON-GENERATED FILE AND CHANGE THAT.

Generated Files:

- the complete Dot-CCG file: `MOLOKO.ccg`
- the compiled Open-CCG files: `grammar.xml, rules.xml,` etc

Non-Generated Files:

- a batch file for compiling the grammar called `build-MOLOKO` (see below) (this should be placed in your OpenCCG `bin` directory)
- the folder `ccg-files` containing:
  - all of the Dot-CCG component files forming the grammar (see below)
  - a perl script `merge.pl` used to combine these `.ccg` files to create `MOLOKO.ccg`

9.3 Grammar Layout

The MOLOKO Grammar, contained in `./ccg-files` is divided into 3 parts:

1. the grammar signature
   all of the syntactic categories, semantics, lexical families, hierarchies and rules
2. the dictionary
   all of the words and their assignment to lexical families, i.e. the entries
3. the testbed
   a list of examples used in testing the grammar, and illustrating its features

9.3.1 The Grammar Signature

The grammar proper is divided into a number of different files of the form:

`#_description-of-contents.ccg`.

The reason the files are numbered like this is because the Dot-CCG compiler is ordered, and consequently any macro-definitions that are used must be defined prior to that.

The first two files `types-ontology.ccg` and `types-features.ccg` contain hierarchies used in the grammar, and in the case of the ontology, for outside inferencing as well. The ontology is a hierarchy of semantic sorts (see §2.1). The feature hierarchy contains both syntactic and semantic feature categories and a listing of their possible (hierarchically organized) values. These definitions also function as 'macros' (in the old OpenCCG sense of the word), i.e., values that can be 'used' within the grammar and dictionary (i.e. lexically specified).

For a syntactic example, it contains a feature category (vform) for verbal forms, including values like verb-ing, base, past-particple, infinitive, etc. In the grammar, this is used, for example, to create the family **v+verb-inf** i.e., the family of verbs which take a verbal compliment in infinitive form (ex. *I want to go home*). It is also used in dictionary entries to specify, e.g., that the past-partiple form of *sing* is *sung*.

For a semantic example, it contains the feature $\langle Aspect \rangle$ with values perfect and imperfect. These values are attached to events via auxillary verb entries.

The remainder of the files in the Grammar Signature contain the syntactic and semantic categories, families, rules and dictionary entry creating macros. Syntactic and semantic categories are combinined to make lexical families, like ditranstive verbs. They are also used in rules which are used for a variety of purposes in the grammar. Dictionary-entry creating macros are a new component of the grammar. They utilize the power of the Dot-CCG language to massively simplify the task of adding new words to the grammar. For example, here are some examples of noun entries from dictionary-open.ccg (see below for more details):

```
noun-irr(man, men, person,)
noun(box, thing,)
```

name(GJ, person,)

These grammar components have been divided among the files according to two general organizational principles. First, encapsulation, e.g. `#_adj.ccg` contains everything particular to adjectives. you want to find the families, rules, and dictionary macros for adjectives, see this file. That is, everything **but** the adjective words themselves, which are in the dictionary files (see the next section). Second, generality and efficiency: minimize overlap and put generalized components together, e.g., adjectives, prepositions and adverbs overlap a lot in their grammatical information (like semantics and their syntactic categories). Thus, this common info is contained in `#_modifiers.ccg`.

Within any of these files, there is an order to the components.

1. syntactic and semantic categories.
2. def-macros used for simplifying the building of lexical families
3. the various families themselves
4. associated type-changing rules
5. dictionary-entry macros

The syntatic and semantic categories created via def-macros are typically those used frequently within its file, and those used externally. For example, the noun file contains a syntactic category n() which corresponds to 'any old generic noun', i.e. a non-bound n . It can be further specified by adding feature values to its parameters, e.g. n(3rd s-sg). Similarly, ENTITY() is the corresponding semantic representation (both share index) . Again, features and arguments can be further specified by adding them to the parameters.

9.3.2 The Dictionary

The dictionary, i.e. the words in the grammar, are divided into **closed** and **open** class entries, located in `X_dictionary-open.ccg` and `X_dictionary-closed.ccg` respectively. Each of these files contains a sorted listing of all the entries currently contained in the grammar. The vast majority are simply 'calls' to (instances of) the various dictionary entry macros specified in the grammar signiture files. For example:

noun(box, thing,)
verb(give, giving, gave, given, m-class-1, action-non-motion, tv dtv)
pronoun(I, 1st, sg, I , me , my , mine , person,)

Some of the more irregular or singular words (specifically closed class function words) are simply build using the default Dot-CCG syntax. Here, e.g., is the entry for *most* as in *the most ugly*:

word form: Family: {other forms: args;}

word most : Most-adj: superlative;

Furthermore, some 'higher level' dictionary macros are located within the dictionary files themselves. They combine a series of other 'base level' macros to create the appropriate lexical entries. For example, the macro **number** takes the ordinal and cardinal forms of numbers (*two, second*) and then 'calls' a large number of other macros, like **adj** (*those two balls*), **context-n** (*those two*), **sgroup-np** (*two of the balls*) etc. to account for all of the 'uses' of numbers. This eliminates redundany by collecting all of these entries into a meaningful conceptual category, instead of having them spread throughout the dictionary.

9.3.3 The Testbed

The testbed, contained in `X_testbed.ccg` is a listing of sample sentences used to test and to 'showcase' the grammar. Each line in the testbed consists of a sentence and the expected number of parses. This listing can be tested for both parsing and generation using the command **ccg-test** (see below)

We have organized these test-sentences in an attempt to illustrate the capability of the grammar. See the file itself for more comments.

9.4 Modifying the Grammar

Obviously, we can only give a brief overview of some of the most common ways that the grammar is modified.

9.4.1 Adding New Words (i.e. dictionary entries) to the grammar

The simplest and most common way of modifying the grammar is to add new words using the current dictionary-entry building macros. In most cases, simply parroting or mimicing one of the currently existing entries will suffice. However, as discussed above, these macros are defined throughout the grammar and for a full description of the parameters involved, you'll have to look into these files (or better yet §8.)

If none of the existing dictionary macros fit your exact needs, this does not necessarily mean that you underlying lexical families, rules and features cannot support what you want. In some cases, you can overload one of the more general dictionary macros by adding extra feature values in the argument list. If this still won't work you may be able to combine these appropriately by resorting to the Dot-CCG word entry syntax:

unique-id: families (class, pred): {
form1: associated feature-values;
form2: associated feature-values;
...
}

9.4.2 Changing the Ontological Hierarchy

The semantic sorts/classes/types attached to the semantic objects produced in the semantic representations used in the grammar are specified in `X_types-ontology.ccg`

If you wish to modify this hierarchy by either collapsing, expanding, or altering its components and their relations, in some cases, this can be done by simply modifying this file. In other cases, however, this will require further changes within other parts of the grammar.

See `#_types-ontology.ccg` for specifics

9.4.3 Classes

It is relatively easy to extend the various classes defined in the grammar. See `#_types-features.ccg` for details.

9.4.4 Other changes (features, new lexical families, rules, etc)

We have tried to organize the grammar in such a way that it can easily be navigated and extended. Moreover, some of the grammar files include instructions on extending the more 'open' elements of the grammar. Good Luck!

9.5 Compiling and Testing the Grammar

Compiling the grammar actually consists of 2 steps:

1. merging all of the `.ccg` files to create `MOLOKO.ccg` (using the `merge.pl` script)
2. converting `MOLOKO.ccg` into the OpenCCG `.xml` files (using the Dot-CCG Parser `ccg2xml`)

These have been combined into a single batch file called `build-MOLOKO`.
This file should be moved to your openccg bin director.

There are two ways of testing the grammar. These are identical to the old OpenCCG methods:

1. Command line parsing and generation using the `tccg` tool
2. Running through the testbed using the `ccg-test` tool