Instructions for Coding Object Attributes, Intentions and Contextual Constraints (draft)

Pamela W. Jordan

April 15, 2000

Contents

1 Introduction

The coding scheme described in this manual was developed for the dialogues collected as part of the COCONUT project (http://www.isp.pitt.edu/~intgen/). The goal of this coding scheme is to investigate the influences of cognitive constraints, and domain and discourse goals on descriptions of domain entities. In particular we are focusing on how discourse entities get expressed in dialogue and how this interacts with the expression of action and constraint changes. We are not addressing issues regarding when action and constraint changes are accepted or rejected. We are limiting the scope to when actions and constraint changes are introduced or when actions are taken up in the discussion.

1.1 The COCONUT Corpus

The COCONUT corpus is a collection of computer-mediated dialogues in which two subjects collaborate on a simple task, buying furniture for the living and dining rooms of a house. (The task is based on those in [Walker, 1993; Whittaker *et al.*, 1993]). Each subject is given a separate budget and inventory of furniture that lists the quantities, colors, and prices for each available item. By sharing this information during their conversation, the subjects can combine their budgets and can select furniture from each other's inventories. The problem is collaborative in that all decisions have to be consensual; funds are shared and purchasing decisions are joint. The subjects in our collected conversations are equal in status: they are both briefed on the domain knowledge needed for problem solving and neither is an expert at the task.

The subjects' main goal is to negotiate the purchases; the items of highest priority are a sofa for the living room and a table and four chairs for the dining room. The subjects also have specific secondary goals which further complicate the problem solving task. Subjects are instructed to try to meet as many of these goals as possible. The secondary goals are: 1) Match colors within a room, 2) Buy as much furniture as you can, 3) Spend all your money. Subjects are enticed to try to arrive at a solution that achieves the maximum number of goals by a task score that associates points with primary and secondary goals.

The subjects are in separate rooms and can communicate via the computer interface only. They are asked to maintain private graphical representations of their discussions and incremental agreements. We can use this private information as partial evidence of what S's utterance meant and what H understood. Subjects share dialogue windows but the inventories, budgets and updated floor plans are private and show up only on the owner's color display. Figure 1 shows the interface as it looks in the middle of a design session.

The buttons in the upper right corner of Figure **??**, `End of Turn` and `Design Complete`, enforce turn-taking and initiate the incremental recording of the conversation and the graphics updates. During an incremental recording, the most recently transmitted message is recorded as well as the state of the sender's graphics display. The graphics display record is a description of the furniture icons in the two rooms as well as what has been allocated but not assigned to any room. An additional feature of the interface is that each furniture icon is initially displayed with a dashed outline around it. The subjects are given the option of turning off the dashed outline to signal that agreement has been reached on using the item in the solution (i.e. the item has been selected as part of the solution).

Note. The examples that are excerpts from the actual Coconut corpus may have uncorrected typographical errors since we present them unaltered.

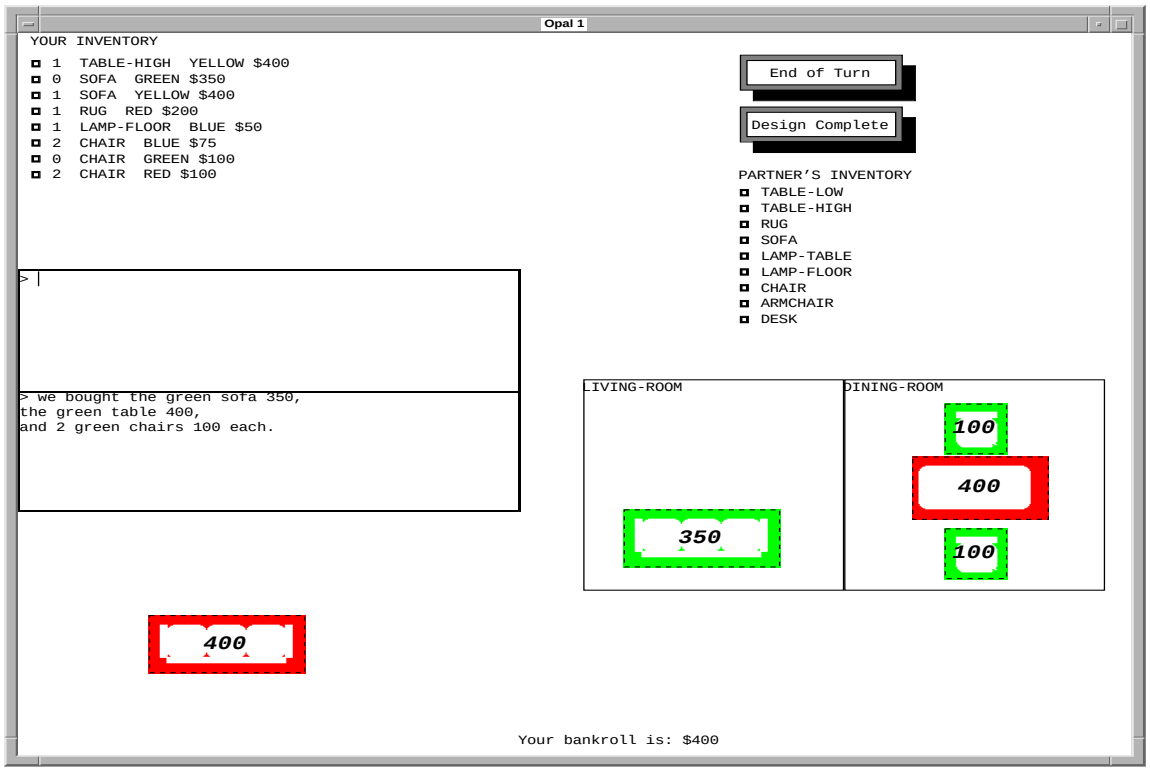

Figure 1: A View of the COCONUT Interface

1.2 Coding Schema Approach

The coding scheme combines elements of the DRI coding schema (DRAMA) for discourse reference, and an extended set of informational relations based on the RDA coding schema. The annotation tool we use is *Nota Bene (NB)* [Flammia, 1995]. Pointers to both discourse reference coding and NB can be found at http://www.georgetown.edu/luperfoy/Discourse-Treebank/dri-home.html, under *Tools and resources*.

2 Top level Menu and Overview

Figure 2 represents the menus and submenus set up for NB and shows all the features and feature value choices for the coding scheme as it applies to the COCONUT corpus. A line out of a box shows the submenu for all the selections in that box. The diamond represents the choice *none of the others applies*. Each submenu and selection will be described below. The ordering of the selections in a menu and the ordering between submenus is not necessarily significant.

The top level menu (which is not shown in Figure 2) consists of two major categories of tags; Utterance-Level and NP-Level tags. In addition there is a Comment tag. Every utterance that is a minimal unit (see Section 5) must be annotated for the Utterance-Level category and every bracketed noun phrase (see Section 5) must be annotated for the NP-Level category, except when the utterance unit is not relevant to the

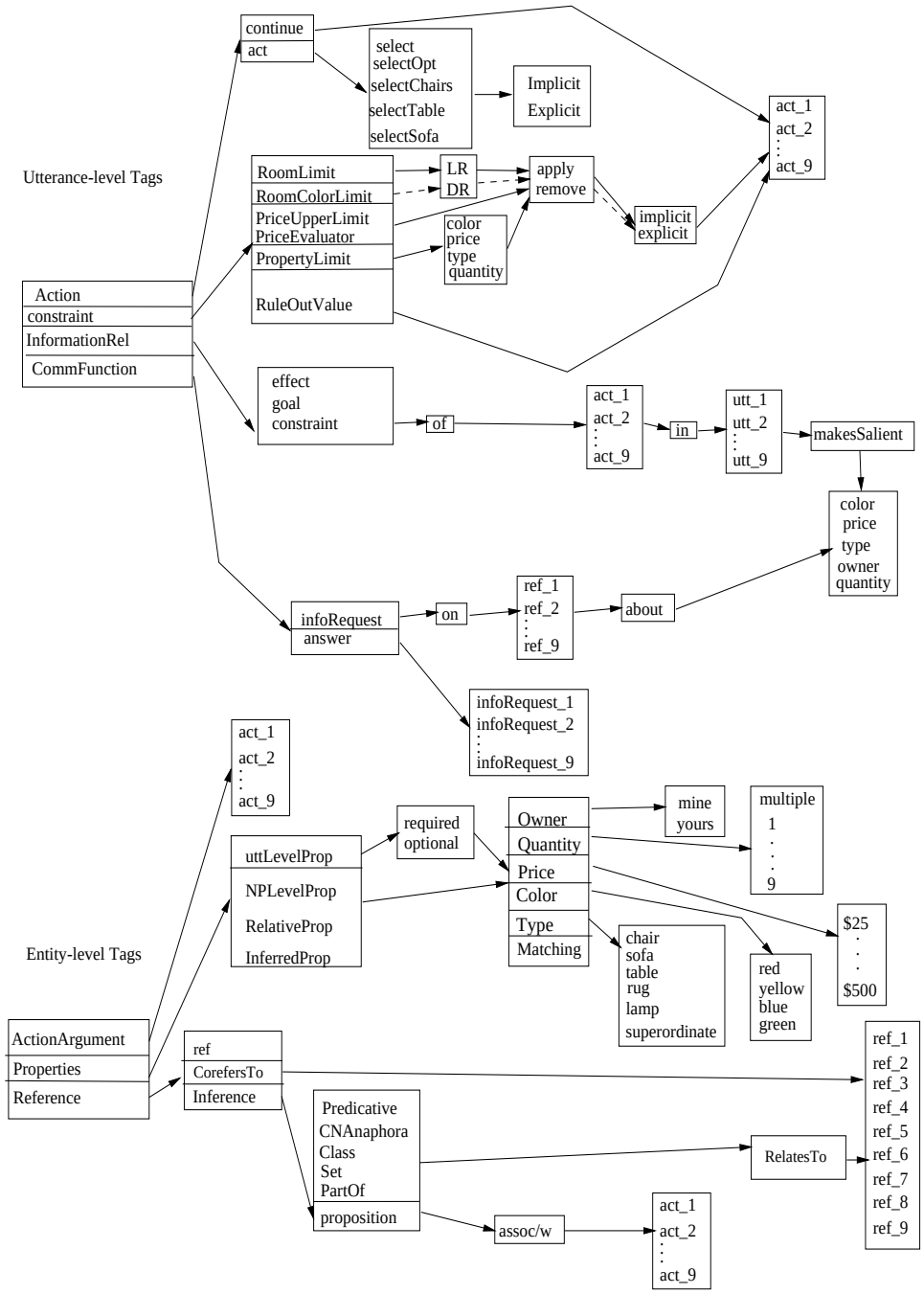

Figure 2: Overview of Coding Scheme

task or the utterance is part of a summary or repair[1]. In cases where more than one tag in a category applies or if the same category applies more than once for a particular annotated unit, then separate annotations should be made for each application. In some cases, the tags within a category are mutually exclusive. Unless noted otherwise, the annotator should assume that tags are not mutually exclusive.

3 Utterance-Level tags

The utterance-level tags capture information about the domain actions and constraints that an utterance communicates and the communicative and domain relationships between the utterances.

3.1 Information about domain Actions - Action

There is one general action type of interest for us in COCONUT: *select furniture for a room.* But there is a hierarchy of *select* action types for annotation purposes that ranges from this most general action type to the most specific action types that are one level up from fully defined executable actions, such as *select a table for the dining room.* **For each utterance we will code the most specific action types that are unambiguously recognizable**. For example, if an utterance unit discusses tables and chairs and it is inferrable that the primary goal of selecting a table and chairs is being addressed, then the utterance will be coded as having two action types: *select chairs for the dining room* and *select a table for the dining room* instead of the more general *select furniture for the dining room.* On the other hand if it is not inferrable that a primary goal is being addressed, the utterance would be coded as having one action type; *select optional items for a room.*

Coding for the most general action type of *select furniture for a room* would be appropriate when a participant lists all of the furniture that he has available and it is not clear how these would define an unambiguous set of more specific action types. For example, a listing of all of a participant's furniture that includes multiple sets of chairs could be either *select chairs for dining room* or *select optional items for a room.* Because it is ambiguous in the initial context, it should be coded as a more general action type to represent the ambiguity. However, if initially a speaker lists all the items they have of only the types that are most strongly associated with particular actions then it is reasonable to infer that the speaker is addressing those actions. For example, if the speaker lists all the chairs and tables he has then it is reasonable to infer he is addressing *select chairs for the dining room* and *select table for the dining room.* However if he also listed some rugs or lamps in the same utterance then the situation is ambiguous and should be annotated as a more general action type.

Note that if the items of highest priority according to the problem description (see Section 1.1) have already been decided by the participants, then one can typically infer that any other tables, chairs and sofas that are not replacements are additional optional items. When no decision has been made yet about the priority items, one can infer that the location for sofas is the living room and the location for tables and chairs is the dining room.

If the most specific action type description for an utterance is the first occurrence of that description then the annotator should begin the coding for the utterance by first selecting `act` from the `Action` menu. This selection causes the tool to assign a unique identifier (e.g. `act_40`) to the new action type.

The action type hierarchy is represented as a combination of choices under the `Action` and `constraint`

[1] We do this to limit the scope of the current research project.

menus. Under the `Action act` submenu the relevant choices are `select`, `selectOpt`, `selectChairs`, `selectTable` and `selectSofa`. Each will be discussed in more detail below. Under the `constraint` menu the relevant choice is `RoomLimit` and this will be discussed in more detail below in Section 3.2.

The menu choice `select` is used to indicate the most general action type of *select furniture for a room.* If the location is more specific, then this will be indicated by the `RoomLimit` choice under the `constraint` menu to be described below in Section 3.2. The choice `selectOpt` should be used when the furniture under discussion is clearly meant to be an optional item. If it is clear which room is meant then this can be indicated with the `RoomLimit` under the `constraint` menu as described in Section 3.2. The choices `selectChairs` should be used when there is a specific action type of *select chairs for dining room*, `selectTable` for *select table for dining room*, and `selectSofa` for *select sofa for living room.*

Finally, when describing a **new** action type, the annotator should indicate whether the action type description required inferencing on the part of the annotator. Selecting `implicit` indicates that the `select` action and/or the furniture argument type were inferred. Otherwise, `explicit` should be selected. In general, the action type is not explicit. **To determine whether the action type was inferred, consider the utterance in isolation from the rest of the dialogue, but not from the initial problem description. If it is clear what the action type is without this additional context, then the action type is explicit.** For example, in (1), the action type could be either `selectSofa` or `selectOpt`. It is not clear without considering the preceding dialogue as well. However, in (2), it is clear that the action types are `selectTable` and `selectChairs` by considering just the utterance and the initial problem description. If something happened in the dialogue to alter the action type descriptions in this case then it should be annotated for the action type that is consistent with the dialogue. However, it would then have to be annotated as `implicit` since the dialogue has to be taken into account.

(1) Let's get my blue sofa.

The annotator should not consider whether the room is inferred when annotating inference for an action type. This will be taken care of as part of the constraint annotation. For example, (2) would be annotated as `selectTable Explicit` and `selectChairs Explicit`. A continuation of this discussion such as (3) would be annotated as a `continue` action for `selectTable`. On the other hand, if a dialogue started with (3), it would be annotated as `selectTable Implicit`.

(2) Let's decide on the table and the chairs.

(3) I have a $100 blue table for the dining room.

If the action type is a continued discussion of a previously described action type then the annotator simply annotates which action type is being continued by selecting `continue` under the `Action` menu and then indicating which automatically assigned unique identifier was given to the previously described action. Action identifiers appear as `act_[1-9]` in the menu. If the number portion of the automatically assigned identifier is larger than 9, then the annotator can select a close identifier from the menu and edit it to get the correct identifier. The correct action identifier can be found by looking for a previous utterance with the same action type description and placing the mouse cursor on any part of the utterance other than an NP. This placement of the cursor causes the tool to show the annotation for the utterance-level.

Not all utterance units will directly address the domain action hierarchy of *select furniture for a room.* For example, a simple answer to a yes/no question to clarify a value or property of an item would not

directly address any domain actions. **To determine whether an utterance directly addresses any domain actions of interest and should be annotated with an action type, check to see if there is a bracketed NP or a [0] in the utterance that has a discourse entity relation to another furniture discourse entity .** (See Section 4.3 for annotating discourse entity relations and Section 5 for an explanation of bracketed NPs and insertions of [0] in utterances.) For example, (4) does not directly address any of the action types in the *select furniture for a room* hierarchy since it has no discourse entity relations to a furniture discourse entity. It is only indirectly related since it is a side-effect of some collection of action types. (See Section 3.3 for annotating side-effects of action types.)

(4) I have $50 left.

Consider another example. In (5), the first utterance contains the elliptical NP, *two*, that refers to some previously discussed chairs (see CNAnaphora in Section 4.4) and thus would be annotated as a continued action for *select chairs for the dining room* (assuming the proper contextual information). However, the *yes* response does not directly relate to any furniture discourse entities so it would not be annotated as having an action type.

(5) A: So we can only get [two]?

B: yes

If on the other hand the response had been as in (6), then it would have an action type annotation. This may seem inconsistent since the two responses don't seem substantially different in meaning, but we adopted this convention for the sake of achieving consistency between multiple coders and to eliminate coding information that can be recovered by other parts of the annotation scheme.

(6) B: We can only get [two].

A general rule of thumb is that if no furniture item or furniture template is mentioned in the utterance, then it should be coded for actions. For these utterances, the annotator should only consider informtation relations for the utterance level coding (see Section 3.3).

Note that there should be no more than 6 specific actions annotated for a dialogue. These include `SelectSofa`, `SelectTable`, `SelectChairs`, `SelectOpt` constrained to the living room, and `SelectOpt` constrained to the dining room. If an optional item is decided on for a particular room, and then later it is decided to consider an additional optional item for that same room, it should be annotated as a continuation of that same action. If it is unclear which room the additional optional item is intended for then it should be coded as a different SelectOpt with no room constraint and then once the room constraint is determined then it should be annotated as a continuation of the general and a continuation of the more specific with the first occurrence of the addition of the room constraint. If the constraint is later changed to the other room and there is no optional item in that room yet then the general action will become that more specific action for that room.

3.2 Information about Constraints that Limit Actions - constraint

The `constraint` menu, allows the annotator to indicate the constraints that are placed on an action type. **Constraints should be annotated relative to the changes that have been made since the most**

recent description or constraint change for the relevant action type (e.g. the last utterance with the same action type identifier), or when a constraint setting is made explicit. Also, since an utterance can specify multiple action types, each constraint change needs to be annotated as to which action type description it applies to. This is done by including the action type identifier as part of the constraint annotation.

There is one exception for including an action type identifier in a constraint annotation. This is when there is a constraint to match colors in a room and is explained below.

An action constraint should only be annotated if there is an action introduced or continued by the utterance. If the utterance constraints an action but does not introduce or continue an action, then it should be annotated as an informational relation (see Section 3.3. Recall that for an utterance to introduce or continue an action, it must have a furniture discourse entity in it (see Section 3.1).

There are six constraints to be annotated. In the general case, besides the details of each constraint (which will be explained below) and which action type description it applies to, whether the constraint is inferred or not should be annotated using the selections `implicit` and `explicit`. The only case in which it is not necessary to note whether a constraint is inferred or not happens when the constraint excludes a particular furniture item (see `RuleOutValue` below). In general, it can be easy to overlook an inference that changes a constraint. For example, if you annotate a property of `matching` (see Section 4.2) and didn't already have the room location set for the action the "matching" item is an argument for, then you should be able to infer a room location for that action.

All the other details for annotating constraints vary according to the constraint.

Constraining the room of an action type. Selecting `RoomLimit` indicates that the room where the furniture item is to be placed has been constrained. This constraint always applies to action types that have been annotated with `select` and `selectOpt` since there is no default room. For all the other action types (e.g. `selectSofa`), the `RoomLimit` constraint should only be annotated if it is made explicit as in (7). When describing the room limit constraint, the annotator should indicate which room the action is constrained to (`LR` for living room and `DR` for dining room) and whether the constraint is being applied or removed.

(7) Let's put my blue sofa in the living room

To understand what has been described so far for annotating constraints, consider a context where the two previous utterances had the action type descriptions, `selectOpt` and `selectSofa`, and where no constraints changes were annotated. Assume that the next utterance, (8), is a continuation of the two action type descriptions, `selectOpt` and `selectSofa`. In this case the only constraint change relative to the previous `selectOpt` and `selectSofa` action descriptions is an inference that the optional rug is to be constrained to the living room. We would just annotate that a constraint has been implicitly applied to the `selectOpt` action type to limit its location to the living room. Assuming that `selectOpt` has the action type identifier `act_2`, the annotation would be `RoomLimit LR apply implicit act_2`.

(8) Let's put the rug with the sofa.

Constraining items in a room to have matching colors. Constraints involving color matches are indicated by selecting `RoomColorLimit`. In the initial case it is understood that there is a color match constraint set for both the living room and the dining room. It is more general to place a color match

constraint on a room rather than on particular action types since the room might not be specified until later in the dialogue. For this reason, this is the only constraint description that excludes an action type identifier.

Since the initial case is to have a RoomColorLimit set for the living room and a separate one for the dining room, this constraint is typically removed rather than applied. Note that in Figure 2, the dashed arrows indicate the path through the submenus for describing the `RoomColorLimit` constraint.

To see how the room color constraint interacts with other constraints, consider a case where there are yellow items in the living room and blue items in the dining room, and a blue sofa has been indicated as an optional item. One can reasonably infer that the blue sofa is for the dining room. Otherwise the color match constraint in the living room would be violated. If the blue sofa were explicitly limited to the living room, then one should annotate that the `RoomColorLimit` for the living room has been removed.

Setting an upper limit on the cost of an action type. Selecting `PriceUpperLimit` indicates that an upper limit has been placed on the price of the item selected for the furniture argument of the action type. For example, in (9) there is a constraint that the sofa price be no more than $200 for the `selectSofa` action type. Assuming that `selectSofa` has the identifier `act_11`, the annotation would be `PriceUpperLimit apply explicit act_11`.

(9) We only have $200 left for the living room.
Do you have a sofa for that price?

Since this constraint is not indicated in the initial problem description, it is typically applied rather than removed when it first arises in a dialogue. (Select `apply` to indicate this.) However, the constraint can also be removed later should the participants find it necessary to do some backtracking. (Select `remove` to indicate this.)

Note that conceptually, (10) could be considered a `PriceUpperLimit` but since it would not have a specific action annotated for the utterance, it should not have a action constraint annotated either. However, it should have informational relation annotations (see Section 3.3).

(10) I have $500 left.

Setting a filtering function for the cost of an action type. Selecting `PriceEvaluator` indicates that a filtering function has been set that depends on the price of the item selected for the furniture argument of the action type. For example, in (11) the filter is that the chairs should be as inexpensive as possible. A filtering function differs from setting an upper limit on the price in that it is vaguer. **The annotator should choose the constraint definition that is the most specific one possible**. Again this constraint is not indicated in the initial problem description so it will first be applied (select `apply`) and may later be removed (select `remove`) should the participants decide to backtrack.

(11) Do you have any cheap yellow or green chairs?

Constraining the properties associated with an action type argument. The participants might choose to constrain the values for one property of a furniture type argument. This is annotated by selecting `PropertyLimit`. There are three properties that can be limited in this way: color, price and type. In general,

this constraint acts to specify a query template for items that are being sought. The `PropertyLimit type` will not often occur and should only be set for a `SelectOpt` action.

For example, in (12), the color property for `selectChairs` is restricted to green or yellow. This differs from a constraint to match colors in a room in that it specifies a subset of colors that are ruled out and this constraint is independent of the colors of anything else. An example for limiting the price property is (13). This differs from a constraint to set an upper limit on a price in that only sofas that cost exactly $100 are requested. This circumstance might arise when the participants are trying to spend all of their remaining money. An example for limiting the type is (14) in the context of `selectOpt`. Note that any furniture type can suffice for the `selectOpt` action type.

(12) Do you have any green or yellow chairs?

(13) Do you have a sofa for $100?

(14) Do you have any rugs or lamps?

Since these constraints aren't in effect at the initiation of the dialogue, they will first need to be applied (select `apply`) and can later be removed (select `remove`) should the participants decide to backtrack in their problem solving effort.

Distinguishing between `PropertyLimit price`, `PriceEvaluator`, and `PriceUpperLimit`. If you can infer a price range for an item but not a single price value, then the annotator should choose the `PriceUpperLimit` constraint. For example, (15) should be annotated as `PriceUpperlimit`, assuming it is mutually known how much money the speaker has left.

(15) I can get the red sofa with the money I have left.

With a `PropertyLimit` on price, there should be an item or items with a specific price value or total price value that are being sought. An example of this is (16). Note that with this example, an appropriate response could be that there are n items that total $50. The key difference between the `PropertyLimit` on price and `PriceUpperLimit` is that the speaker's intention is to spend out the budget and not to buy a certain quantity of items.

(16) Do you have anything for $50 to spend out the budget?

To distinguish between `PriceUpperLimit` and `PriceEvaluator` is that `PriceEvaluator` is for cases when the speaker may not know any of the prices or it is not clear to the annotator which chairs are being referred to. For example, in (17), assuming that no chairs have been introduced prior to this utterance, then it should be annotated as a `PriceEvaluator`. On the other hand, (18) should be annotated as `PriceUpperlimit` since it relates to the mutually known price of an entity. In this example, if the chairs are too expensive, then it seems reasonable to infer that speaker wants to look for something cheaper. *Your chairs* is not completely ruled out unless a set with a lower price can be identified. One could use the rule of thumb that price = some value equates to `PropertyLimit price`, price > some value equates to `PriceUpperLimit` and otherwise it is `PriceEvaluator`.

(17) Let's get cheap chairs to go with your blue table.

(18) Your chairs are too expensive.

Ruling out a particular furniture item. Selecting `RuleOutValue` indicates that a particular furniture item has been ruled out for the furniture argument of the action type. For example, in (19), there is a unique lamp that has been explicitly ruled out. In (20), the items of the same type that have been mentioned recently in the discourse, that don't cost $50, are implicitly ruled out.

(19) We will have to forget about the lamp.

(20) I think the 50 ones are better.

This constraint can be either implicit or explicit but the difference is not noted in the annotation .

3.3 Informational Relations - InformationRel

An informational relation between two utterances describes how the content of the two utterance are related in the domain. **Note that informational relations should only be annotated between two different utterances or two different clauses in the same utterance and must be associated with a particular domain action**. In deciding what informational relations occur between the two utterances or clauses, it is necessary to think about whether the information in an utterance or clause specifies actions or states and how the two actions, or an action and a state, are related. If one cannot associate the informational relation to an action then that informational relation should not be annotated.

In the following annotation choices for informational relations, an action type as described in Section 3.1 is not equivalent to an action for purposes of annotating informational relations, nor is a constraint as described in Section 3.2 equivalent to a state. However, the annotations for action type and constraints can offer some guidance in making decisions about informational relations. As we will see below, some informational relations provide more information about the further specification of one action type. All the informationally related units should have the same action type identifier when both have action type annotations. For example, utterance units annotated with `goal` should be limited to utterances where the goal and the associated action have matching action type identifiers.

We define an action, in the case of informational relations, as an utterance that has an action type and that specifies values or potential values for the action type. Included in the informational relation annotation is the action type identifier associated with the action types involved in the informational relation and which other utterance is part of the informational relation. In addition, if a furniture entity property is made more salient or brought to mind by the informational relation, it will also be included in the annotation. An informational relation will not always make some property more salient than another. If this is the case, do not annotate any of the properties as being more salient than the others. Examples of annotations will be included below with the descriptions for each of the informational relations. **

For standardization, the annotation should always go on the clause or utterance that plays the role indicated by the relation name. Note that an informational relation can point forward or backwards in the discourse. **

Effect. This unit is an effect or consequence of the action expressed in another unit. An effect is something that happens because of an action but may not be a goal of the action. For example, in (21), the first clause is the action and the second is the effect. Assuming the action type identifier is `act_10` and the first clause is `utt_10` and the second `utt_11`, the annotation on `utt_11` would be `effect of act_10 in utt_10 makesSalient price`. Again, select the action utterance that is in closest proximity to the effect utterance. This will usually be in the same turn as the effect utterance.

(21) If we buy my rug, we will be out of money.

Constraint. This unit is the constraint on an action . **This is a state that exists or needs to exist that cannot be planned for**.

As an example of a constraint informational relation, consider (22). The first utterance is the constraint and the second is the action that is being constrained. **For purposes of standardizing the annotation, the constraint utterance should be related to the action that is in closest proximity to which the constraint is meant to first be applied**.

(22) We only have $100 left.
We could get your rug instead.

Goal. The situation presented in this unit is a goal for the action presented in another unit. A goal is a desired state or a higher-level action and the action in the other unit is part of or a way of achieving the goal. For example, in (23) the first utterance is a goal and the second an action that helps in achieving the goal. Note that both utterances have an action type of `selectTable`. The first utterance cannot also be the action for the goal since it does not specify any values or potential values for the table argument. Assuming ** the action type identifier is `act_10` and the first utterance has the identifier `utt_10`, the annotation on `utt_10` would be; `goal of [act_10 in utt_11] makesSalient price`. The question to use in testing for ** what property (if any) is made salient, is to ask whether the goal of this action brings a particular furniture property more to mind than another. For example, in (23), the price of the table in the action is made more ** salient than it might otherwise be without the statement of the goal.

(23) We need to decide on a cheap table.
I have a blue one for $100.

Typically if an action type is explicit or becomes explicit, as in (24), then a `goal` information relation should be annotated. In this example, both utterances have the action type `selectTable`. The first introduces it and the second continues it. While the second utterance is the `goal` it would not be annotated as the action as well although it has an appropriate action type. The action annotated should be the most specific one that is in closest proximity to the goal. The first utterance in (24) is a more specific action type since values for the action type arguments are supplied. Assuming the action type identifier ** is `act_10` and the first utterance has the identifier `utt_10`, the annotation on `utt_11` would be; `goal of [act_10 in utt_10] makesSalient type`.

(24) I have a blue table and a yellow table.
So let's decide on the table first.

Although a goal utterance can be related to all actions with the same action type, the annotator should just mark one such relationship. Look for the action in closest proximity to the goal that the goal is clearly meant to apply to. In ambiguous cases, generally an action utterance in the same turn as the goal utterance would be considered to be in closer proximity than an action utterance by a different speaker.

Distinguishing goal, constraint and effect informational relations can be difficult. A goal is something that the participants are working towards trying to achieve and in this domain usually indicates which action they are going to address. A constraint, on the other hand, is something that an action must comply with to qualify as an executable action. To tell a goal from an effect, if the utterance is a motivation for an action, it is a goal. If the utterance is something that results from executing the action then it is an effect. One exception is that points for items should not be annotated with informational relations. This particular type of effect is not of interest to us. **Note that an utterance can have multiple informational relations associated with it.**

An utterance can be involved in more than one informational relation. Oftentimes, if there is a `constraint` informational relation there is also an `effect` informational relation. For example, in (25), the remaining $200 is an effect of some action but is also a constraint on the action of buying for the LR.

(25) This leaves us with $200 for the LR.

3.4 Communicative Functions - CommFunction

We are primarily interested in one pairing of communicative functions; questions and their answers. However, we are focusing only on question / answer pairings that relate specifically to properties and property values of domain object units. **Although we recommend annotating all of the utterance level tags first, the annotator should postpone the `CommFunction` annotation until all the referential indices have been assigned (see Section 4.3), since the referential identifier is a necessary part of the `CommFunction` annotation.**

Any utterance that directly or indirectly asks for information about a property or property value for an ActionArgument should be annotated as an `infoRequest`. An utterance should not be annotated as an `infoRequest` if it is a general request for information as in(26). The `infoRequest` feature is meant to record information requests for values of particular properties for mutually known furniture entities. A question about the existence of an item should never be annotated as an `infoRequest`. The reason for the restriction to asking about property values of mutually known items is that we are focusing of the question of why a particular property value is mentioned in a subsequent reference and not why a particular item is subsequently referred to. So in the case of a request for a property value of a mutually known item, that is an obvious reason why the proprty value would be included in a subsequent reference (i.e. the answer).

Note that questions about the existence of particular items should most likely be coded as an action constraint. For example, in (26), there should be `PropertyLimit` action constraints on the type and color.

(26) Do you have a blue rug?

As part of the `infoRequest` feature, the annotator should also indicate which furniture discourse entity the information is requested. This is accomplished by indicating the furniture discourse entity's referential

identifier. Finally, we want to know which properties information is being requested on. For example, in (28), assuming that *your chair* has the referential identifier `ref_10`, the first utterance should be annotated as `infoRequest on ref_10 about color`. When the annotation is made, the tool automatically assigns a unique identifier `infoRequest_[1-n]` to each request. If information on more than one property is requested, then each property request should receive a separate annotation. Note that if the annotator cannot indicate which property information is requested on, then the `infoRequest` tag should not be assigned.

(27) What did we just buy?

(28) A: What color is your chair?

B: It is red.

To annotate an answer to a property information request, the annotator should select the utterance in closest proximity that supplies the requested information and should indicated the unique identifier, `infoRequest_[1-9]`, for the request it responds to. For example, in (28), above, assuming that the first utterance was annotated with `infoRequest_23`, the annotator would tag the second utterance with `answer infoRequest_23`.

4 Entity-Level Tags

The Entity-Level tags capture information about a discourse entity described in an utterance. **Each bracketed object unit within an utterance unit should be annotated with an Entity-Level tag**. We define an object unit as an NP that denotes a discourse entity of type furniture or action. Note that an NP is not the only part of an utterance that contributes to a discourse entity's description. An utterance can convey additional information about the discourse entity (e.g. in (29) the ownership of the chair is conveyed by the utterance and not by the NP *a red chair*). **So although we are annotating just the NP forms in the dialogue we will actually be recording all the relevant information about the underlying discourse entities in the utterance**. For example, in (29), the annotation for *a red chair* will include the ownership information.

(29) I have [a red chair].

See the section on file format (Section 5) for more on how object units and utterance units are identified.

4.1 ActionArgument

Any object unit that further specifies the furniture argument of any action type in the *select furniture for a room* hierarchy (as described in Section 3.1), or refers to a specific furniture entity or corefers to an entity that is an `ActionArgument`, should be annotated as an ActionArgument. This means that any utterance annotated with an `ActionArgument` should also be coded with the action it is an argument of. The reverse does not necessarily hold since the conditions for annotating an action are less stringent with regards to the type of furniture discourse entities that must be present.

We stipulate that coreferential entities (see Section 4.3, must also be annotated with a separate `ActionArgument` because it is possible for an entity to become an argument to a different action. For example, a participant might discuss a rug as an optional item for the living room and then if it isn't selected for the living room, he might later discuss it as an option for the dining room solution.

If multiple object units in an utterance have a subsumption relation between them or are in a predicative relationship, only the most specific ones should be coded as ActionArguments. For example, in 30, *a blue one* and *a red one*, should both be annotated as ActionArguments for `select table for dining room`. The object unit *my cheapest tables* subsumes these other two entities. Note that *a blue one* and *a red one* do not subsume one another, they are instead representational siblings. **Also, although wh-pronouns are bracketed as object units, they do not typically count as action arguments since they do not further specify the furniture argument**.

(30) [My cheapest tables] are [a blue one] and [a red one] both for $200

The object unit further specifies a furniture argument if it denotes a discourse entity of type furniture and is not a generic NP. This includes for example, pronouns and full noun phrases that refer to specific furniture items as with *a blue chair* and *it* in (31), NPs that rule out values for the furniture argument as with *mine* in (32), and non-referential NPs that put constraints on the values that can be assigned to the furniture argument as with *any blue chairs* in (33) and (34). Generic NPs, such as *tables* and *rugs* in (35), would not be annotated as ActionArguments although they would indicate a particular action type. Instead generics, like those in (35), indicate that a particular goal is being addressed.

(31) I don't have [any blue sofas],
but I have [a blue chair for $150].
Let's get it.

(32) [Your chair] is cheaper than [mine].

(33) Do you have [any blue chairs]?

(34) I don't have [any blue chairs].

(35) [Tables] are more important than [rugs].

Every object unit that is identified as an ActionArgument should be annotated as to which action type description it further specifies. The applicable action type description is the most recent, specific action type description that has a furniture argument type that subsumes that of the object unit. The unique identifier for this action type description is associated with the ActionArgument by selecting the closest identifier in annotation submenu (i.e. `act_1--act_9`) and editing it to get the correct identifier.

We add the stipulation of subsumption of the furniture type because sometimes the entity denoted by the object unit will be related to a general domain action such as *get furniture for the living room* or *get furniture for the rooms*. For example, in the constructed utterance in (36), there is an implied action type of *get furniture for the rooms*. In this case, each of the object units will be individually related to this implied action type since it is the most recent action type and the furniture type *furniture* subsumes the furniture types *sofa*, *table*, and *chair* for the three ActionArguments.

(36) I have [a $200 red sofa], [a $300 yellow table] and [4 green chairs for $25 each].

In general, there should only be one `ActionArgument` assignment per object entity. If the annotator finds a case where the domain furniture entity is initially introduced where they think the entity is a argument to more than one action then the annotator should review whether they have coded the action itself at the right level. If the domain furniture entity is mutually known to the participants (this does not equate to cases of initial reference), then it is allowable for an object entity to be the argument of more than one action. This circumstance could arise if a dialogue participant refers to an entity that has a set discourse relation to a variety of other furniture items. The entity is an initial reference but it is made up of mutually known domain entities. For example, consider (37) in a context where there are 4 chairs selected for the dining room and 2 for the living room. In this case *the chairs* are an `ActionArgument` for the actions `SelectChairs` and `SelectOpt`.

(37) The chairs cost $200 total, the sofa $250 and the table $300.

Since the annotator needs to associate an action type identifier with each ActionArgument, utterance-level action information should be coded first in order to code more efficiently. (See Section 3.1.) Also, recall that an utterance can have multiple action types associated with it. Because of this, care should be taken to examine all the action types defined for an utterance before selecting one.

4.2 Property Value Codings - Properties

The Properties category codes specific property values (e.g. the color is blue) of the object units annotated as ActionArguments, that are expressed by either the containing utterance or in the noun phrase itself. At the utterance level there are three possible choices: `RelativeProps` (relative properties), `InferredProp` (inferred properties), and `uttLevelProp` (utterance level properties). At the NP level there is one choice: `NPLevelProp` (NP level properties). All of these choices are discussed in detail below. For property value codings, the goal is to record any evidence that is local to the utterance that allows us to recognize a property value for a furniture discourse entity. The property values should be noted for every bracketed entity and not just cases of initial reference. We are particularly interested in seeing which properties are repeated either implicitly (`InferredProp` and `RelativeProps`) or explicitly (`uttLevelProp` and `NPLevelProp`).

The annotation scheme allows any property value to be expressed at either the utterance or NP level, although it is more likely that some property values will be preferentially expressed at one level or the other. For example, ownership of an object is frequently expressed at the utterance level as in (38), but can be expressed at the NP level as well as in (39). At the utterance level, property values can be expressed in two special ways that are of interest to us: relative to the property values of another object (`RelativeProp`) and by inferring the property value (`InferredProp`). Other ways of expressing property values at the utterance level are not distinguished and are grouped in the general category of property values expressed at the utterance level (`uttLevelProp`). If there is a discourse inference relation (see Section 4.4) supplied by another bracketed NP in the utterance, then no property tag should be assigned for information that is gained via the discourse inference relation. **

For the general category of `uttLevelProp`, the annotator should also indicate whether the property value information is obligatory according to the syntax or semantics of the utterance by selecting `required` in the menu and `optional` otherwise.

(38) I have [a blue chair for $200].

(39) [My blue chair] is too expensive.

`RelativeProp` is used to encode when a property value is defined relative to some other already known property value, e.g. in (40), the price property value of *your chair* is compared with the price property value of another.[2] Although it is mainly the prices and points (note, however, that we will not annotate anything regarding points) associated with the COCONUT furniture items that will be presented in this way, other property selections have been made available for coding.

For (40) to qualify as a `RelativeProp`, the property value for the object must be defined relative to a mutually known property value. **Typically, it will not be possible to annotate a single value as part of the `RelativeProp` annotation so no actual value selection should be made for this category**. Note that (39) should not be coded as a `RelativeProp price` since the price value for the chair is not defined relative to a known price value. In this particular example, the price related information should be annotated as an implicit `PriceEvaluator` constraint (see Section 3.2).

(40) [Your chair] is cheaper than [mine].

The utterance level manner, `InferredProp`, is provided for cases when the property value is not explicitly mentioned but a specific value is inferrable locally from the content of the utterance. For example, in a context where it is mutually known that the agents have a certain amount of money remaining, then with (41) and (42), the price of the blue chair can be inferred[3].

(41) I have [a blue chair] which leaves us with $100.

(42) I can get [a blue chair] with the money I have left.

Another example, is where the number of items is locally inferrable from the NP article or the number of the noun (e.g. singular or plural), or both. In 42, we can infer that there is 1 chair in this discourse entity. Although this may seem to be explicit information about the number of items it is not. Compare (42) with an utterance that substitutes *1* for *a*. Note that with an inferred quantity, one will generally only be able to infer quantity property values of 1 or multiple.

Property values that are inferred on the basis of an discourse inference relation (see Section 4.4), or a coreference relation (see Section 4.3) should not be annotated. If two referents in the same utterance are related by a coreference relation or an inference relation, then the property value codings that are shared between the two by virtue of the relation or are provided at the utterance level, need only be annotated on the more specific of the two NPs. For example, in 43, the two bracketed NPs have two separate reference identifiers and are related by the `Set` inference relation (see Section 4.4). The first NP is more specific so it would carry the utterance level price property annotation and the second would not need to have this specified.

(43) [The yellow chairs] are $75 for [each one].

[2]Note that with the right context, (39) could be annotated with a `RelativeProp` for price. It depends on whether there is another chair in the current context.

[3]Note that with (42), if you can't infer a value for the price of the chair, it should be annotated as a `PriceUpperLimit` constraint. See Section 3.3.

Property values for furniture items include: type of furniture, color, ownership (i.e. whose inventory it is listed in), quantity (the number of identical entities grouped in the set represented by the entity, price, and whether the item explicitly has "matching" associated with it. There is such an association if the NP contains the word "matching" or the utterance has the word "match" in it. For example, (44), the table entity would be annotated with "matching" and would also have the `inferredProp` for color with a value of red.

(44) I have 2 red chairs and a matching table.

The values for the type of furniture also includes a choice of `superordinate` for NPs that have a category that is superordinate for the base-level types. For example, *furniture*, *items*, and *stuff* are superordinates for specific furniture entities in the domain.

If no property values are expressed at a particular level or in a particular manner, then it is not necessary to code that level (where the levels are; `uttLevelProp` and `NPLevelProp`, and the manners are; `RelativeProp` and `InferredProp`). When encoding the property values for a particular level or manner, the property value is either present or absent and a binary selection should be made for each possible object property value. Note that there for an object entity, there should be no more than one value annotated for each property. Finally, there is an option to escape out of the sequence of property value selections. Escapes will be interpreted as an indication that the remaining property values are not expressed at this level.

4.3 Reference Relations - Reference

The reference relations were adapted from the DRAMA coding scheme and manual.

If a noun phrase introduces a discourse entity that is new to the discourse, it should be assigned a new referential index.[4] If the entity refers to a discourse entity that was referred to previously, it should be assigned the same index that was used for all previous references to the relevant entity.

The menu selection, `ref`, assigns a new referential index to a new discourse entity while `CorefersTo` links a discourse entity to an existing referential index. The selection, `CorefersTo` has a submenu with the referential indices `ref_[1-9]`. These values should be edited to get the correct unique referential index.

In some cases, it may be unclear whether an object unit introduces a new discourse entity. **When the annotator is unsure whether an object unit reintroduces an entity, say X, the annotator should test whether the information in the new utterance is consistent with everything that has already been said about X**.

If a single entity cannot be identified for an object unit because there are multiple possibilities that are equally likely to be what the speaker or writer intended (i.e. if there is genuine ambiguity), then the annotator should indicate the list of possibilities in the Comment tag. Note that this should be done only in cases where the annotator finds the alternatives to be truly equally likely. Keep in mind that the more natural interpretation is that the speaker intends to refer to the most recently mentioned entity.

If there is vagueness or uncertainty that prevents the annotator from determining whether an object unit refers to a particular discourse entity then no annotation should be made for coreference.

There are many cases where the entity for an object unit is very closely linked with an entity already in the discourse context, sometimes so closely that the annotator may be tempted to use `CorefersTo` to capture

[4]Note that we are not limiting the assignment of referential indices to NPs that are referring expressions.

partial identity. However, most of these relations are captured by the annotation of inference relations, discussed in the next section.

Each bracketed entity should have either an initial reference identifier annotated or a coreference annotation (this applies to generics as well). An object entity should only have one reference relation tag annotated. Should an annotator believe that more than one coreference relation exists they should instead be annotating an initial reference and some discourse inference relations (see Section 4.4).

(45) I have [yellow chairs] that I can buy [0].

If there is a co-reference relation to an unbracketed NP, then the annotator should go back and annotate that particular unbracketed NP. **Note that not all levels of object units are bracketed. Only the smallest non-identical subcomponents are bracketed.**[5]

To annotate an unbracketed NP, simply drag the mouse across the part of the NP that defines the entity that was referred to. For example, in 46, note that the first sentence has 3 NPs but only the two lowest level ones are bracketed but not the higher level NP that is the two conjoined lower level NPs. In the next sentence, however, *those*, refers to a discourse entity that is defined by this higher level NP. In this case the annotator should assign a referential index to this higher level NP and then note the co-reference relation between those and this higher level NP.

(46) I have [2 $50 green chairs] and [2 $75 red chairs].
Let's get [those].

4.4 Inference Relations between Discourse Entities - Inference

The inference relations for discourse entities are taken from the DRAMA coding scheme and manual and were adapted by adding some examples from the COCONUT corpus. Note that this annotation category is under the Reference relation in the annotation menus although it is not a reference relation. It is placed there because reference relations and inference relations function to relate discourse entities.

The referential identifiers discussed in the preceding section make it possible to identify all object units that are related by strict coreference. Another relation among discourse entities that constrains how a referent will be referred to is whether it can be inferred from previously mentioned referents. The types of inferential relations proposed to play a role in discourse reference include, for example, part/whole, instance/type, set/subset and so on.

There are two types of inferences that relate distinct referents: linguistic and conceptual. We will code for both but we will not make a distinction between the two types in the annotation scheme. An inference is linguistic if it is derived directly from the linguistic semantic representation of an expression, linguistic knowledge about the elements of this representation, and general inference rules. In one type of linguistic inference, the progressive use of a verb implies that the event evoked by the clause extends indefinitely into the past and present. In contrast, an inference is conceptual if it depends on world knowledge, and commonsense reasoning. Excerpt (47) illustrates both types of inference. A linguistic inference rule associated with the verb *pick* implies a change of location for the argument corresponding to the picked referent. That is, the pears that were picked have one location prior to the picking event, and a new location afterwards. World knowledge supports the conceptual inference that the default location for pears prior to the picking event is

[5] See Section 5 for information on how object units are identified.

in pear trees. Thus linguistic knowledge about the argument structure of pick and commonsense knowledge that pears grow on pear trees supports the derived logical form illustrated in (47a), namely that the man picking a set of pears is picking them from a corresponding set of pear trees.

(47) a. A man is picking some pears(j).
man(i) & pear set(j) & event(k) & is picking(k i j) & pear trees(m) & is picking from(k i j m)
b. He puts some pears(n) he has picked into a basket.
c. A boy steals the pears(n).
d. The pears(j) make a sound when the man picks them.

We will not distinguish between the two types of inference relations in the annotation. Both the conceptual and the linguistic inference relations will be grouped at the same level in the menu and relations that are only distinguished by whether they are conceptual or linguistic will use the same tag.

If there is an inference relation to an unbracketed NP, then the annotator should go back and annotate that particular unbracketed NP. Note that not all levels of object units are bracketed. Only the smallest non-identical subcomponents are bracketed.[6] Note that NPs that denote events, lists, inventories and rooms are not bracketed but may participate in some of the inference relations. **A relation to an unbracketed NP should be annotated only when it contributes to the inference of some property value for the related furniture entity**. Remember that we will not include this inferred property value as part of the related furniture entity's annotation, the property value annotation will be on the unbracketed NP.

For each inference relation, the inference relation annotation should go on the second NP in the dialogue so that the relation always points backward. This convention standardizes which of the related NPs receives the annotation. **

Set: If the referent X of a discourse entity can be inferred to have one of the following relationships with referent Y of a previous discourse entity, and this relationship between X and Y has not yet been recorded, the corresponding relation between X and Y should be assigned the Set feature.

$X \in Y$, $Y \in X$
$X \subset Y$, $Y \subset X$

An example of a set/subset relation is all the pears that have been picked by a pear picker, versus one basketful of these pears.

We expect this relation to occur quite frequently in the COCONUT domain. For example, in 47, *the green set* is a new discourse entity and has a set conceptual inference relation to the three distinct discourse entities for *2 $25 green chairs*, *2 $100 green chairs* and *$200 green table.*

(48) A: I have [2 $25 green chairs] and [a $200 green table].

a. B: I have [2 $100 green chairs].
Let's get [the green set].

[6] See Section 5 for information on how object units are identified.

Plural linguistic inference relations will also be included as part of the **Set** annotation. If a plural NP is used to refer to a set of referents, and any members of the set are already known, then the relation of the set to the members should be coded. A variety of possibilities occur, which could be encoded using set notation. The referent of a plural NP may be inferentially related to the referent of a previous singular NP, as in (49) below. Or, vice versa, the referent of a singular NP may be inferentially related to the referent of a previous plural NP, as in (50).

For example, in (49a) and (49b), two referents have been discussed, a boy (j) and a girl (k). The plural pronoun subject (m) in (49e) refers to set with members (j) and (k).

(49) a. and the little boy's-j going along,
b. and um then you see this little girl-k.
c. Coming on a bicycle in the opposite direction,
d. and uh you wonder how she-k's going to figure in on this.
e. And uh they-m come,

In (50), a set of three boys (m) is introduced first, and then a member p of m, and a subset n of m (the three boys) is later referred to. In (50), the little boy who returned the bicyclist's hat (p) is described as taking the reward of three pears back to share with his two friends (n). The singular NP subject in (50d) is related to the referent of the earlier plural NP through set membership. The plural NP in (50i) is related to the original set of three boys by set difference.

(50) a. And then he picks up uh the li
b. the three little boys-m sort of go in the opposite direction,
c. down the road, : : :
d. And the little boy-p ,
f. who fetched his hat,
g. took the pears,
h. and goes back,
i. to his friends-n,
j. and /they/ each ;p gives them-n each a pear.

In (51), from the COCONUT domain, assuming that A's red chairs have the reference identifier `ref_1` and B's `ref_2`, *the chairs* in B's second utterance would be annotated as `Set RelatesTo ref_1` and `Set RelatesTo ref_2`.

(51) A: I have [2 red chairs for $25 each].

B: I have [a red table for $125] and [2 red chairs for $50 each].
Let's definitely get [the chairs].

As noted earlier, inference relation annotations should always point back to the related entities and never forward.

PartOf: If the referent X of a furniture discourse entity can be inferred to be part of some entity Y that has already been mentioned in the discourse then this relationship should be annotated. In the COCONUT domain this happens mainly between furniture entities and unbracketed entities such as *my list*, *my inventory*,

the living room. The PartOf annotation should be used only when the entity type of the unbracketed entity is something the related furniture entity could be part of. If there is a subsumption relation between the two discourse entities in question then the PartOf annotation should not be assigned. For example, one could consider the entity type of *inventory* to be **list**. The list entity does not subsume a furniture entity but a furniture entity could be part of what defines a particular list entity. In general, this relation would not exist between two bracketed NPs since bracketed NPs should all be of type furniture. If the annotator is considering this tag for two bracketed NPs then it is most likely a Set of Class inference relation.

The PartOf annotation will generally go on the bracketed entity since it is usually later in the discourse. This annotation should only be done if the PartOf relationship enables the inference of some property about the furniture entity. For example, in 52, the ownership of the items is inferred via the PartOf relationship to *your inventory.*

(52) From your inventory, I have 3 green chairs for $50 each and a red sofa for $200.

Class: If the referent X of a furniture discourse entity can be inferred to have a subsumption relationship with referent Y of a previous discourse entity, and this relationship between X and Y has not yet been recorded, the corresponding relation between X and Y should be assigned the Class feature. For example, in 53, *the table* and *your green one* have a subsumption relationship.

(53) Let's decide on the table for the dining room.
How about your green one?

The `Class` annotation should always point back to an entity and should only be noted for those in close proximity to one another.

CNAnaphora: If a common noun anaphora, such as one anaphora or null anaphora, or a pronoun, is used to refer to a discourse entity or element of a discourse entity, then the relation to that entity should be coded. A simple rule of thumb is that any bracketed entity that would not receive a property value annotation for its type is generally a case of CNAnaphora. For example, in (54), each of the NPs in the second utterance has a null anaphora relation to the NP in the preceding utterance. Assuming that *a variety of high tables* has the reference identifier `ref_10`, the annotations for *green*, *red*, and *yellow* would each be `CNAnaphora RelatesTo ref_10`. Note that this example also has a `Class` relation as well.

(54) I have [a variety of high tables]
,[green], [red] and [yellow] for 400, 300, and 200.

Note that not all `CNAnaphora` relations are also coreference relations (see (55)).

(55) I have a lot of [green chairs], I have [2 $100 ones], [3 $50 ones] and [2 $125 ones].

Proposition: The referent of a discourse referential NP may depend on the interpretation of a proposition. In the COCONUT corpus, this pertains primarily to references to actions in the `select furniture for room` hierarchy. For example, in (56), *That* refers to the action type `select sofa for living room`. The annotator should only annotate the NPs that refer to actions in the action type hierarchy. Assuming that `selectSofa` has the action type identifier, `act_12`, the annotation for *that* would be `proposition assoc/w act_12`.

(56) A: Let's get my blue sofa.

B: [That] sounds good.

The annotator would not annotate *it* as a proposition in (58) since it refers to *a decision* and not an action in the action type hierarchy. Also pleonastic *it* should not be annotated as a proposition, as in (58).

(57) It is up to you.

(58) It is better to buy [a cheaper table].

This annotation should be placed on the pronoun for the propositional entity and indicate the action identifier for the action it refers to.

Predicative: If the referent of a discourse entity is related by a predicative relationship such *is*, then the two entities should be coded as having this relationship. For example, in 58, the entities defined by *my cheapest table* and *a blue one for $200* are not the same discourse entities in the definitional sense but the information about one provides more information about the other. The annotation should be placed on the second entity, *a blue one for $200*. Note that this example also includes `CNAnaphora` and `Class` inference relations.

(59) [My cheapest table] is [a blue one for $200].

When coding inference relations, there is a submenu of numbers `ref_[1-9]`. The number selected indicates the unique reference id number of the related discourse entity. If identifiers with numbers greater than 9 are needed, they can be obtained using the editor. The inference relation should only be annotated when a new discourse entity is involved.

5 File format

We preprocess the files to be tagged by defining the minimal units for tagging; object units and utterance units. First we define the object units. The object units are all the NPs that denote entities of type furniture or pronouns that denote entities of type event that are in the `select furniture for room` hierarchy. The aim is to note the distinct entities defined by an NP. A set counts as a distinct entity if all the members are identical. A complex NP can often be broken into simpler NPs that define distinct entities but sometimes the complex NP defines a more distinct entity. For example in the NP *2 of the green chairs*, the entire NP should be marked as the object unit since it is more specific or limiting than the subcomponent *the green chairs* (i.e. this entity would be defined as having more than 2 green chairs). On the other hand, the NP *the green table and chairs* should be marked as two object units *table* and *chairs*. The full NP should not be delimited as an object unit in this particular example. The portion of the NP that is an object unit will be indicated with the delimiters [ ].

An implicit argument of a verb should also be inserted as [0] whenever it is syntactically a required argument of the verb and it denotes an entity of type furniture or an entity of type event that is in the `select furniture for room` hierarchy. Note that in (59), although there is a semantically implied argument it is

not syntactically a required argument. Note that for the implied argument to be syntactically realized, "than" must be added as well.

(60) I do not have a sofa for a better price.

Wh-pronouns that have intended fillers of type furniture should also be marked as object units. Missing arguments for object units of type event will not be inserted. For example, *My rug matches*, should be formatted as *[My rug] matches [0]* since the verb *matches* requires two objects as arguments. However, the deverbal *the match* should not be bracketed as an object unit or have an implicit argument added since this NP refers to a matching event.

Also, NPs such as *my inventory*, should not be delimited as object units. In this case, the NP refers to an entity of type list and not an entity of type furniture. Although this entity has furniture items as part of it, it is not representationally a furniture item.

Next we segment the dialogue into functionally independent clauses following Passonneau's approach [Passonneau, 1994]. This section is taken unaltered from the COCONUT manual[Di Eugenio *et al.*, 1997]. Each clause corresponds to a single line.

Passonneau's criteria for identifying a functionally independent clause are:

- It must contribute a proposition to the discourse that is semantically complete and fully specified, and if it is a full syntactic clause it must be syntactically independent and maximal.
- It must not be a formulaic clause serving the function of an interjection.

As a consequence, functionally independent clauses include, besides main tensed clauses, coordinate clauses, subordinate adjuncts, nonrestrictive relative clauses, clauses containing verb phrase ellipsis and response fragments.

We differ from Passonneau in considering backchannels, some gerundives and some interjections as separate units, as well.

Conventions

1. *Cue words*, such as 'yes', 'no' and 'alright'. When 'yes' or 'no' are part of a response, as in 'Yes, I have a blue rug', they are are not treated as independent utterances. Cues such as 'OK' and 'alright' can similarly share a backward tag with the following utterance. But this is a matter of analysis. Therefore, in preprocessing the files, we treat 'OK' and'alright' as separate units. If the annotator considers them part of a larger response, s/he should make a segment of the cue and the rest of the response. An example in which 'OK' is part of a response:

```
S1:  (a)  I guess you can buy the yellow rug for $150.

S2: <Segment_1><Backward_1=Accept>
     (b)  OK,
     (c)  I'll buy the rug for $150. <Coref=sameItem (a)>
    </Segment_1></Backward_1>
```

An example in which 'OK' is independent:

```
S1:  (a) What do we have left if anything?
S2:  (b) OK.
     (c) We spent: 300 (sofa), 250 (lamp), 200 (table), and 300 on chairs...
```

2. *Gerunds.* There appears to be two different kinds of gerunds: we call one type *resultative*, as it describes the effect of a possibly complex action, and the other *depictive*, as it denotes a state. The excerpt below provides examples of both, *resultative* in (f), and *depictive* in (b) and (d):

```
S1: (a) That means we just bought two chairs
    (b) that are yellow costing 75 a piece
    (c) and two chairs
    (d) that are green costing 100 a piece
    (e) for a total of 675,
    (f) leaving us with  275
```

   The excerpt also shows the consequences of this distinction for segmentation: only *resultative* gerunds are considered as independent utterances.

   While the distinction is not totally clear, we think it may be related to the distinction proposed in [Stump, 1985] between *weak* and *strong* free adjuncts.

```
(a)  Having unusually long arms,  John can touch the ceiling.
(b)  Standing on the chair,  John can touch the ceiling.

(a)  Being a businessman, Bill smokes cigars.
(b)  Lying on the beach,  Bill smokes cigars.
```

   Stump calls the adjuncts in both (a) sentences *strong*, because their actual truth is uniformly entailed. and those in the (b) sentences *weak*, because their actual truth can fail to be entailed. Our examples may correlate with Stump's *weak / strong* distinction: *costing 100 a piece* would be 'strong', as it is an unchangeable property in the context of the game, whereas *leaving us with 100* would be 'weak', as its truth depends on the chosen solution.

3. *List of NPs.* When a list of items is given via coordinated NPs — e.g. *I have a green table, two sofas, one blue and one yellow, and lots of chairs* — individual NPs are not considered as separate utterances.

References

[Blok, 1993] Peter Blok. *The Interpretation of Focus: An Epistemic Approach to Pragmatics.* PhD thesis, Rijksuniversiteit Groningen, Groningen, Holland, 1993.

[Carlson and Tanenhaus, 1988] G. Carlson and M. Tanenhaus. Thematic roles and language comprehension. In W. Wilkins, editor, *Syntax and Semantics, vol.21: Thematic Relations*, pages 263–288. Academic Press, San Diego, 1988.

[Di Eugenio *et al.*, 1997] Barbara Di Eugenio, Pamela W. Jordan, and Liina Pylkkänen. The COCONUT project: Dialogue annotation manual. Available at http://www.isp.pitt.edu/~intgen/research-papers, 1997.

[Flammia, 1995] Giovanni Flammia. N.b.: A graphical user interface for annotating spoken dialogue. In *AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, pages 40–46, Stanford, CA, 1995.

[Grosz and Sidner, 1986] Barbara J. Grosz and Candace L. Sidner. Attentions, intentions and the structure of discourse. *Computational Linguistics*, 12:175–204, 1986.

[Partee, 1991] Barbara Partee. Topic, focus, and quantification. In Steven Moore and Adam Zachary Wyner, editors, *Proceedings from Semantics and Linguistic Theory I*, pages 159–187, Ithaca, New York, 1991. Cornell University. Available from CLC Publications, Department of Linguistics, Morrill Hall, Cornell University, Ithaca, NY 14853-4701.

[Passonneau, 1994] Rebecca J. Passonneau. Protocol for coding discourse referential noun phrases and their antecedents. Technical report, Columbia University, 1994.

[Roberts, 1997] Craige Roberts. Focus, the flow of information, and universal grammar. In Peter Culicover and Louise McNally, editors, *The Limits of Syntax*. Academic Press, New York, 1997.

[Rooth, 1992] Mats Rooth. A theory of focus interpretation. *Natural Language Semantics*, 1(1):75–116, 1992.

[Rooth, 1996] Mats Rooth. Focus. In Shalom Lappin, editor, *The Handbook of Contemporary Semantic Theory*, pages 271–297. Blackwell Publishers, Oxford, 1996.

[Stump, 1985] Gregory Stump. *The semantic variability of absolute constructions*. D. Reidel Publishing Company, 1985.

[Walker, 1993] Marilyn A. Walker. *Informational Redundancy and Resource Bounds in Dialogue*. PhD thesis, University of Pennsylvania, December 1993.

[Whittaker *et al.*, 1993] Steve Whittaker, Erik Geelhoed, and Elizabeth Robinson. Shared workspaces: How do they work and when are they useful? *IJMMS*, 39:813–842, 1993.

Index