The COCONUT project: Dialogue Annotation Manual

Barbara Di Eugenio
Pamela W. Jordan
Liina Pylkkänen

ISP Technical Report 98-1
December 1998

Contents

1 Introduction **3**

2 Top level **4**

3 Information-Level **5**

3.1 Discussion 7

4 Forward-Communicative Function **8**

4.1 Statement 10

4.2 Influence-on-listener (Influence-addressee-future-action) 11

4.3 Influence-on-speaker (Commit-speaker-future-action) 13

4.4 Other-forward-function 14

4.5 Discussion 15

4.5.1 Potential Actions 15

4.5.2 Trying to get the addressee to do something 18

4.5.3 `Influence-on-Listener` and `Influence-on-Speaker` 19

4.5.4 `Accept`, `Influence-on-Speaker` and `Influence-on-Listener` 19

4.5.5 `Answer`, `Influence-on-Speaker` and `Influence-on-Listener` 19

4.5.6 Elaborations, `Influence-on-Speaker` and `Influence-on-Listener` 21

4.5.7 Conventional acts 22

5 Topic **23**

5.1 ItemFeature 27

6 SurfaceFeatures **28**

7 Backward Communicative Function **28**

7.1 Initiate 30

7.2 Agreement 30

7.2.1 Discussion 34

7.3 Understanding 34

7.3.1 Signal-Non-Understanding 34

7.3.2 Signal-Understanding 36
7.4 Answer 37
7.5 Informational Relations 39
7.6 Coreference / Set Relations 40
7.7 Link 42

8 Segment Tag **43**

9 Fragment **44**

10 Summary **45**

11 Comment **45**

12 Non-Relevant **45**

13 Miscellaneous Conventions **45**

14 File format **46**

1 Introduction

This manual describes the coding scheme we developed to code the dialogues collected within the COCONUT project (http://www.isp.pitt.edu/~intgen/). The COCONUT project has as its long-range goal the creation of a unified architecture for interactive discourse, that includes both interpretation and generation. Using abductive reasoning as a framework, we are developing a discourse simulator for collaborative tasks. The abductive reasoner that manages the dialogue is coupled with a constraint satisfier that reasons about the domain. We are currently analysing the corpus of dialogues we collected. The issue we are concentrating on is how conversation correlates with the problem solving process that the two participants are engaged in. The results of the empirical study inform the computational architecture, by helping selecting the context, i.e., the subset of axioms that the abductive reasoner has at its disposal at a given time, and the heuristics that the constraint satisfier uses.

The COCONUT Corpus. We collected 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 associating points with both individual furniture items and satisfied 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 1, "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. All the examples taken from the Coconut dialogues that are included in this manual appear as they have been recorded. Specifically, typos have not been corrected.

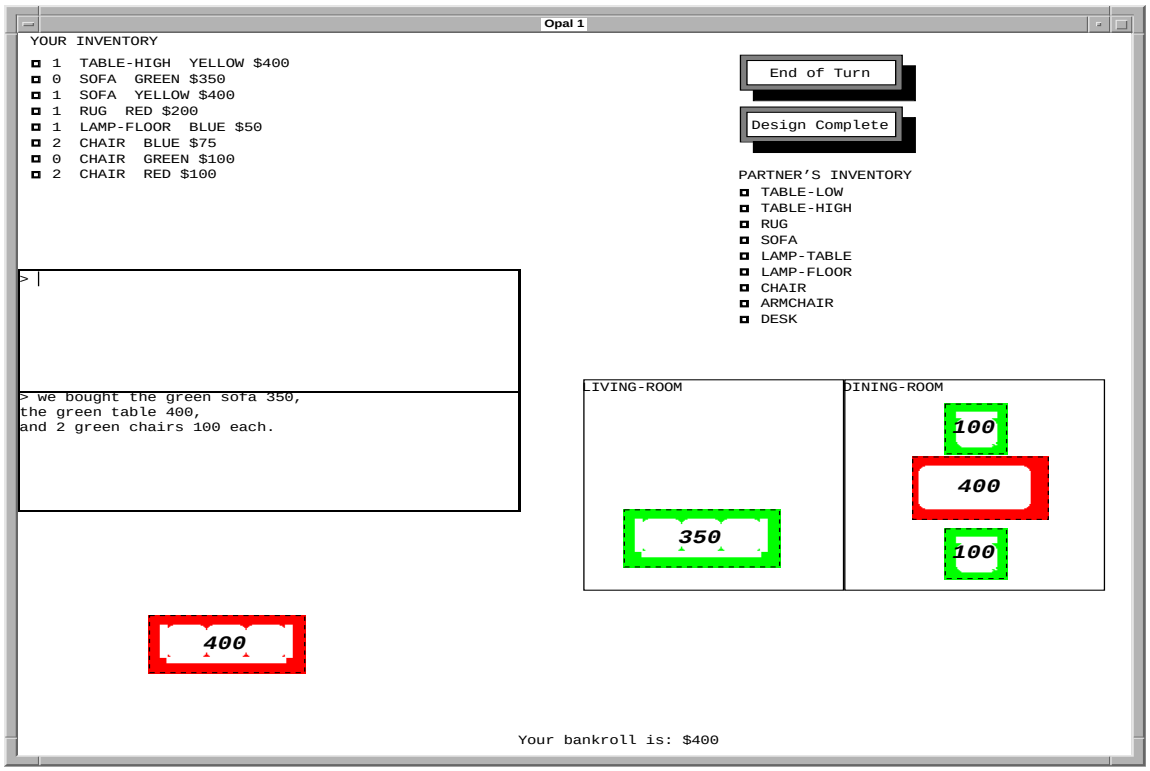

Figure 1: A View of the COCONUT Interface

Coding Schema Approach. We build on top of the DRI coding schema (DAMSL) and extend it in two ways: we add some top level tags, and we further specify other tags. As noted, certain sections of this manual are adapted from the DAMSL manual [Allen and Core, 1997]. The tool we use is *Nota Bene (NB)* [Flammia, 1995]. Pointers to both the DAMSL manual and NB can be found at http://www.georgetown.edu/luperfoy/Discourse-Treebank/dri-home.html, under *Tools and resources*.

2 Top level

The top level menu consists of five major categories that utterances in the Coconut domain are annotated for. These are Information-Level, Forward Communicative Function, Topic, Surface Features and Backward Communicative Function. Every utterance that is a minimal unit (Section 14) must be annotated for these dimensions, except when the utterance unit is Non-Relevant (Section 12) or part of a Summary (Section 10). If the utterance unit is not a tensed clause, it should be tagged as a Fragment; a Fragment will be tagged with any other tag that may apply, see discussion in Section 9.

The tags Non-Relevant, Summary and Fragment all occur at the top level, along with Comment (Sec. 11).

Note. Most of the tag menus that follow don't provide an explicit catch-all *OtherValue* tag, to be used when none of the other tags applies. So for example, `Task` (Section 3) can be specialized as `EvaluatePlan` or as `GameProcedure`. This should not be taken to mean that each utterance which is labeled as `Task` is

tagged as either `EvaluatePlan` or `GameProcedure`; rather, most utterances will be tagged simply as `Task`, without any further specializations. However, `InformationLevel`, the menu to which `Task` belongs, does provide an *OtherValue* subtag, i.e., `OtherLevel`; namely, if none of the other tags applies, `OtherLevel` should be used. The fact that some menus include an `OtherValue` tag, and others don't, is not really an inconsistency. When a menu includes an *OtherValue* tag, it means that the *OtherValue* tag is rare: for example, at the `InformationLevel`, an utterance will be tagged as either `Task`, `TaskManagement`, or `CommunicationManagement` the vast majority of times. Instead, when a menu does not include an *OtherValue* subtag, it means that the included subtags are infrequent, so that utterances will be mostly tagged at the higher level, as in the case of the `Task` tag.

3 Information-Level

(Adapted from DAMSL sec. 2.2.)

This category provides a rough characterization of the content of the utterance. In task-oriented dialogues, we can roughly divide the utterances into those that address the task, and those that address the communication process. In addition, we can subdivide the first category into utterances that advance the task (Task) and those that discuss the problem solving process or experimental scenario (TaskManagement). We follow the current version of DAMSL in subdividing *Information-Level* as follows — boldface indicates the further distinctions we introduce:

* Task ("Doing the task")
  - **EvaluatePlan**
  - **GameProcedure**
* TaskManagement ("Talking about the task")
  - **StrategizeAction**
* CommunicationManagement ("Maintaining the communication")
* OtherLevel

- *Task.* In a task oriented dialogue, most utterances directly move ahead (or attempt to move ahead) the participants' domain goals, i.e., they concern the task. For example, in the two (unrelated) excerpts below, `(a)`, `(b)`, `(c)`, and `(d)`, `(e)`, are all tagged as `Task`:

  ```
  S1: (a) we spent: 300 (sofa), 250 (lamp), 200 (table), and 300 on chairs.
      (b) I had 550.
      (c) We overspent 50.
  ```

  ```
  S1: (d) I say let's start with a blue sofa;
      (e) I have one for 300
  ```

  The vast majority of utterances will be just tagged as `Task`, without any further specification. In other cases, a more specific `Task` tag can be assigned, as follows:

– *EvaluatePlan*, which is used when plans are explicitly talked about as objects. For example, (e) in D's answer would be tagged as < `Task - EvaluatePlan`>:

```
S1: (a) I believe that we should purchase the red sofa (100)
        and the  blue table (50) and blue chair (25).
    (b) Red for the living room and blue for the dining room.
    (c) If we have enough left over we can purchase the red lamp-table.

S2: (d) My funds are limited to 150 with the same credit situation.
    (e) that sounds like a very reasonable plan to me.
    (f) Let's go with it.
```

Note. Whereas `EvaluatePlan` is used when both *complete* and *partial* solutions are under consideration, it should not be used to tag single steps in a plan. For example, (b) below should not be tagged as `EvaluatePlan`:[1]

```
S1: (a) I have a red sofa for 300.
    (b) How does that sound?
```

– *GameProcedure*, used to mark utterances that concern what the players are allowed to do, such as *we can't exceed our budget*, or global domain constraints, such as *colors aren't that many points*. `GameProcedure` does not apply to domain constraints that the participants can give up, such as the color matching goal. Utterances discussing these types of constraints should be treated as `StrategizeActions`. The coder should keep in mind that even the primary goals can be given up and, thus, they don't qualify as a `GameProcedure`.

- *TaskManagement.* Utterances at this level explicitly address the problem solving process or experimental procedure. This includes utterances that involve coordinating the activities of the two speakers (e.g., "Are you keeping track of the time?", "Forget about that problem for a while"), asking for help on the procedures (e.g., "Do I need to state the problem?") or asking about the status of the process (e.g., "Are we done?"). In domains where the task involves building a plan, it is important to distinguish between utterances that concern the plan, which are Task level, and utterances that involve the problem solving process, which are TaskManagement level. For instance, consider a situation where a plan has been proposed; a question "will that work?" will be a Task level utterance because evaluating possible plans is part of the task.

  – We introduce a further distinction, *StrategizeAction*. Utterances marked as `StrategizeAction` concern strategies to follow in the current problem solving scenario, such as *Let's start from the living room*, or *Let's work on getting the train to Avon first.* `(a)` and `(d)` below are tagged as `Strategize-Action`.

```
S1: (a) we can try to color coordinate with chairs.
    (b) i have 3 green (50) 5 yellow (150) 2 red
    (c) but colors aren't that many points...
    (d) maybe we should just try to accumulate furnature.
```

  Note that an utterance simply stating global domain constraints that cannot be changed, such as `(c)` above, *colors aren't that many points*, is tagged as `Game-Procedure`, not as `Strategize-Action`.

[1] Note that the fact that (b) is a question, and thus asks for an evaluation rather than providing one, is not of concern here. The tag `EvaluatePlan` is assigned independently from other dimensions of coding; the fact that e.g. a question may ask for an evaluation, rather than providing one, will be captured by the cooccurrence of the `EvaluatePlan` and `Info-Request` tags (Section 4.2).

It is in principle possible that an utterance states global constraints in order to explicitly direct problem solving, and therefore, that it should be tagged as `Strategize-Action`, not as a `Game-Procedure`; however, we didn't find any such examples in our data. Thus we assume that, apart from rare cases, there is no ambiguity between the two tags `Game-Procedure` and `Strategize-Action`.

- *CommunicationManagement.* This type of utterance typically includes conventional phrases that maintain contact, perception and understanding during the communication process. They include greetings, acknowledging what the speaker said, stalling for time, signaling a speech repair (*oops*), or signaling misunderstanding. Other utterances in this category concern the communication process explicitly: they may establish the communication channel (*Are you there?*), address communication problems (*I didn't understand what you said*), explicitly manage the delays, or maintain the turn (*Wait a minute*).

- *OtherLevel.* Catch all label for utterances that don't fall neatly under any of the other categories, but may be relevant to the dialogue. In Coconut, this category is used for some utterances that relate to actions that are part of the experimental set up, but that neither further problem solving, nor belong to `TaskManagement`. In general, they discuss manipulations of the graphics display that cannot be taken as paraphrases of steps of the solution. For example:

  ```
  S1: (a) and what color are yours [chairs]
      (b) so I can log them in
  ```

  `(b)` is tagged as `OtherLevel`, as updating the graphic display is tangential to the task of furnishing the two rooms. Note that `(a)` is tagged as `Task`, as color of items is considered as an essential property of objects in our domain, even if the speaker here is interested in it only because of the graphics update.

3.1 Discussion

In coding this dimension, the coder should remember that every utterance has a Communication component in some sense, but that utterances should be marked at the *CommunicationManagement* level only when they make no direct contribution to performing the task. In other words, *CommunicationManagement* level utterances are concerned exclusively with maintaining the conversation: if they were removed, the conversation might be less fluent but would still have the same content relative to the task and how it was solved. For instance, the greeting "hi" could be considered at the *Task* level in the sense that it starts the process of performing the task. Removing the utterance, however, would have no significant effect on the task or the way it was performed, thus we know its function is mainly at the *CommunicationManagement* level.

Distinguishing *TaskManagement* from *CommunicationManagement* or from *Task* is sometimes difficult. The DAMSL manual stipulates that in ambiguous cases the tag *TaskManagement* should be chosen, and we follow this strategy.

The difficulty in distinguishing between *TaskManagement* and *CommunicationManagement* arises because utterances at either level may not add content directly relevant to the task. For example, consider the utterance "Sorry for taking so long" which causes problems because it can be interpreted as an apology both for not answering quickly enough (*CommunicationManagement*) or for slowing down the interaction (*TaskManagement*). If the coder thinks that such an utterance has a component of both types, however, it should be labeled as *TaskManagement*.

Making the distinction between *Task* and *TaskManagement* can be difficult in dialogues where the task involves solving a problem and/or developing a plan. Since the task involves building and evaluating plans, utterances such as "does the plan work?", and "how does that sound" should be marked at the *Task* level. Utterances referring to the current problem to be solved should also be labeled *Task*: "what is the problem", "the problem is we need to get two boxcars of oranges to Avon", "we have to get two boxcars of bananas to Corning", "is there a time limit", etc. The most complicated utterances appear to have interpretations at both levels. For instance, the utterance "does that solve the problem?" could be intended as a request to check if the plan would work (a Task level utterance) or might be intended to ask whether the session is done (i.e., a paraphrase of "Are we done with the task?" which is a *TaskManagement* level utterance). In cases where an utterance seems to do both, it will be marked at the *TaskManagement* level.

Note that when answering a question at one level, the answer is usually, though not always, at the same level. Question-answer pairs can occur naturally at these levels as seen in the following examples:

```
Task                      S1:   (a)  How long does it take to get to Corning?
Task                      S2:   (b)  Three hours.

TaskManagement            S1:   (a)  Do I have to state the problem?
TaskManagement:           S2:   (b)  Yes.

CommunicationManagement   S1:   (a)   Can you hear me.
CommunicationManagement   S2:   (b)   Yes
```

4 Forward-Communicative Function

(Adapted from DAMSL Sec. 2.3)

This dimension characterizes the effect that an utterance has on the subsequent dialogue and interaction.[2] The purposes behind an utterance are very complex. For instance, as the result of an utterance, is the speaker committed to certain beliefs, or to performing certain future actions? Often, there are many different effects simultaneously achieved by an utterance. To allow for this, the coding in this dimension allows four different aspects of every utterance to be coded: *Statement, Influence-on-Listener, Influence-on-Speaker, Other-Forward-Function.*

- Statement
  - Assert
  - Reassert
  - Other-statement
- Influence-on-Listener (Influencing-addressee-future-action)
  - Open-Option
  - Directive
    - Info-Request

[2] In previous versions of the DAMSL coding scheme, it used to be called *Illocutionary Act.*

    * Action-Directive

- Influence-on-Speaker (Committing-speaker-future-action)
  - Offer
  - Commit
- Other-forward-function
  - ConventionalOpening
  - ConventionalClosing
  - ExplicitPerformative
  - Exclamation

It is often difficult to determine what actions the speaker intended to perform with an utterance. Moreover, the effect that an utterance has on the subsequent interaction may differ from what the speaker initially intended by the utterance. For this reason, coders are allowed to look ahead in the dialogue to determine the effect an utterance has on the dialogue.

Naturally, just saying that *coders are allowed to look ahead in the dialogue* is not specific enough: the issue is how far ahead can coders look, the whole dialogue, only the next turn, only the rest of the current turn? Here DAMSL's and our policies differ. DAMSL allows coders to look at the listener's turn in order to determine what effects the speaker's actions have, whereas we limit lookahead to the current turn, i.e., the turn to which the utterance under analysis belongs. There are several reasons why we don't allow coders to look at the listener's turn:

- the decision trees (Figures 2, 3, and 4) are devised from the point of view of an observer trying to infer the speaker's original intentions, rather than trying to infer how the listener reconstructed those intentions;
- we have some insights into the speaker's real intentions by means of the graphics;
- we are interested in studying how the speaker's intentions may be reconstructed or inferred by the listener : to do so, we have to try to identify the speaker's original intentions.

Deciding to limit lookahead to the current turn is not sufficient, as it is still necessary to decide how far in the current turn to look ahead. We chose to use the whole current turn:

- We can't see any principled reason to limit lookahead to 1, 2 or more utterances ahead. The end of the turn seems to provide a natural bound.
- To the extent possible, we use the graphics the subjects update to disambiguate various aspects of the utterance. In our experiment traces, though, it is not possible to determine exactly when the subject updates his/her own graphic display: again, the end of the turn provides a natural bound.

Another important point that the coder should keep in mind is that `ForwardCommunicative Function` does not capture issues of strength: so for example the `Action-Directive` tag will apply both to orders and to proposals or suggestions. The *strength* of the speaker's commitment to beliefs or intentions is *not* captured in this coding scheme.

Finally, the coder should remember that an utterance may have multiple `Forward` tags, depending on how many functions it simultaneously performs. As an example, in the right context, an utterance such as "There is an engine at Avon" could be both an `Assert` and either an `Open-Option` (stating the possibility of using that engine to move some cargo), or an `Action-Directive` (directing the listener to use that engine).

4.1 Statement

The primary purpose of `statements` is to make claims about the world, as in "It's raining outside".Figure 2 shows a decision tree for the statement aspect. The top level provides an applicability condition, *Does S make a claim about the world?*, that tests whether an utterance should be coded along this aspect. The applicability condition per se is not sufficient though, as it needs to be operationalized. The DAMSL manual provides the following test for a statement: *could the utterance be followed by "That's not true"?*. However, we found this test problematic to apply consistently, as there was considerable disagreement in our coder group as to which utterances can be followed by *"That's not true"*. The only utterances we agreed on that would be filtered by such a test are syntactically imperative utterances, such as *Let's take the train from Dansville* or *Bring me that red folder*. However, there are sentences such as *we should start in the dining room*[3] that, in the appropriate context, constitute what is informally called a suggestion: intuitively, we don't want to mark them as statements, but they pass the *"That's not true"* test. Thus, we replaced DAMSL purely semantic test with a test that appeals to syntax (mood and tense), semantics (stativity), and domain knowledge (agreement).

Test: S makes a claim about the world, i.e., utters a statement, if the utterance is a declarative sentence that is

- past; or
- non past, and containing a stative verb; or
- non past, and containing a non-stative verb in which the implied action:
  - does *not* require agreement according to the domain;
  - or is supplying agreement.

Convention. If an utterance explicitly expresses the speaker's opinion by including a matrix such as *I think* or *I believe*, what is tagged along the `Statement` aspect is the argument to the matrix, not the complete utterance.

Some statements:
I bought the blue sofa → statement (past tense)
I just logged in your table. → statement (past tense)
I have a yellow sofa for $100 → statement (stative)
The living room looks good → statement (stative)
I think the yellow ones are better → statement (stative)
I think we're out of money now → statement (stative)

Some non-statements:
Let's buy your sofa → non-statement (imperative)

[3] Of course, such utterances, as well as syntactic imperatives, could be taken as claims about the speaker's intentions; however, in this case the *statement* tag would be applied to syntactic imperatives as well, and therefore it would not be very useful.

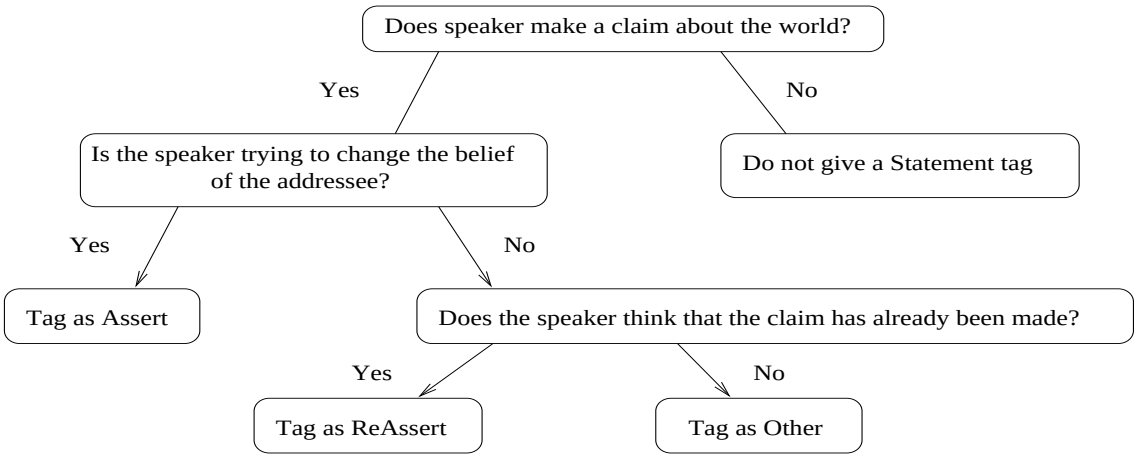

Figure 2: Decision Tree for Statement Aspect

We should start in the living room → non-statement (present, requires agreement)
We will have to forget about the lamp → non-statement (future, requires agreement)

Some utterances that may be statements or non-statements, according to the discourse context:
I'm going to put your table in the living room → a non-statement if unsolicited, a statement if it agrees to a proposal
We could start in the living room → non-statement if meant as a suggestion, i.e. if it requires agreement; a statement if it is meant as *according to the rules of the game, we're allowed to start in the living room.* Note that in the latter case it does not require agreement, as that fact is known to both participants.
Now I will do design complete → a non-statement if unsolicited, a statement if it agrees to a proposal

If the statement aspect is applicable, the decision tree then indicates how to select the appropriate tag. The key distinction for the `Assert` tag is that the speaker is trying to change the beliefs of the hearer. If this is not true, then a further distinction is made depending on whether the claim has been made previously in the dialogue or not.

4.2 Influence-on-listener (Influence-addressee-future-action)

The primary purpose of this aspect is to influence the hearer's future action, as in the case of requests ("Show me the congested cities") or questions ("When will it arrive?").

Test: can the hearer coherently respond with "I can't do that" or "I can't tell you"?

While this test accounts for most cases, there are some examples where an utterance is in this class but the test fails. For instance, in TRAINS, say the participants are discussing how to get some oranges to Bath, and one says "There's an engine at Avon", suggesting that they could use that engine to move the oranges. This utterance falls into this aspect but could not be followed by "I can't do that", although the variant "I can't use that engine" is fine.

There are many verbs in English that describe variations of these acts that differ in strength, including acts like command, request, invite, suggest and plead. These distinctions in strength are not captured in the current annotation scheme. What is captured here is the distinction concerning whether the speaker is merely

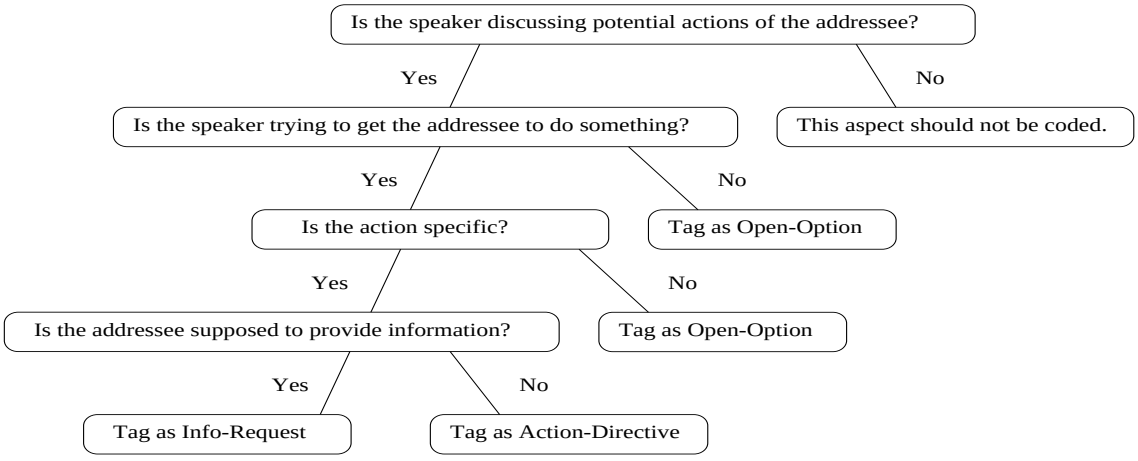

Figure 3: Decision Tree for Influence-on-listener

laying out options for the addressee's future action (the `Open-Option` tag), or actually would personally like the hearer to perform a certain action (the `Directive` tags, `Info-Request` and `Action-Directive`). An example of an `Open-Option` utterance from the TRAINS domain:

```
Assert          S1:   I need to get the train at Avon to Bath.
Open-Option     S2:   You could go through Corning.
```

The `Directive` tags are further subdivided according to whether the action requested involves providing the speaker with information.

`Info-Request` includes all questions, including yes/no questions such as "Is there an engine at Bath?" and "The train arrives at 3 pm right?", and wh-questions such as "When does the next flight to Paris leave?". It also includes actions that are not questions but request information all the same such as "Tell me the time", as well as requests for other actions that can be used to communicate, such as "Show me where that city is on the map".

All other `Directives` are in the `Action-Directive` class, and include examples such as "Take the train to Avon" and "Please speak more slowly". `Action-Directives` are often expressed as questions as in the common utterance "Can you open the door?". In such cases, the coder has to decide whether the utterance counts as an `Action-Directive` in the dialogue. If it does, then it is tagged as an `Action-Directive` and the `Info-Request` aspect (if any) cannot be tagged. This may be modified in the future.

While the decision tree in Figure 3 provides some necessary basic distinctions, it contains two tests that need to be operationalized: what counts as a potential action, and what it means to try to get the addressee to do something. We discuss the former issue in Sec. 4.5.1, and the latter issue in Sec. 4.5.2. Sec. 4.5.1 also discusses the test regarding *specificity* of actions in Figure 3, that we added to the original DAMSL decision tree.

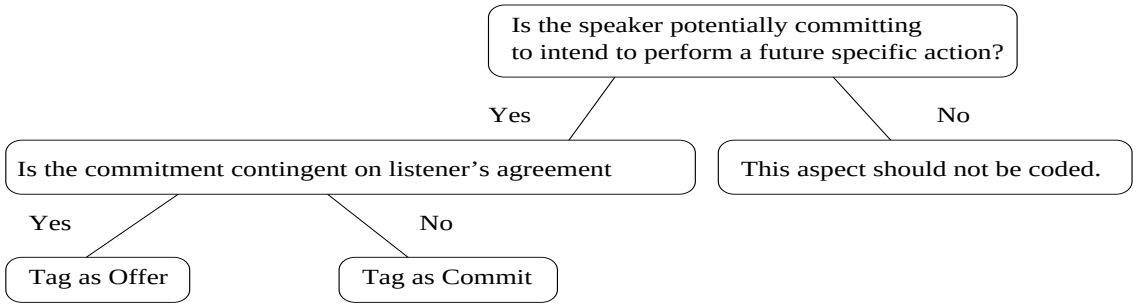

Figure 4: Decision Tree for Commit-speaker-future-action

4.3 Influence-on-speaker (Commit-speaker-future-action)

The defining property of utterances with this aspect is that they potentially commit the speaker (in varying degrees of strength) to some future course of action. Moreover, we only code an utterance along the `Influence-on-speaker` aspect if the action underlying it is specific (Section 4.5.1). The only distinction made within this aspect is whether the utterance's commitment is conditional on the listener's agreement or not. `Commits` that are conditional on the listener's agreement include what are called offers in English: with an offer, the speaker indicates willingness to commit to an action if the hearer accepts it. The prototypical case of a `Commit` not dependent on the listener's agreement is a promise, although the category `commit` may include other weaker forms of commitment such as "I'll probably be at the meeting at 3". Since the speaker does not need the listener's agreement in this case, he or she is not making an offer; the commitment is just not very strong.

Typical `Offers`:

```
Shall I come to your office
I'm free at 3  (in context of setting up a meeting)
```

`Offers` with explicit conditions:

```
I'll be free after four if my meeting ends on time
I can meet at 3 if you're free
```

Typical `Commits`:

```
I'll come to your party
I promise that I'll be there
```

A weak `Commit`:

```
Maybe I'll come to your party
```

Utterances that accept a previous `Directive` or `Open-Option` will typically be a `Commit`, as in the following dialogues:

```
Assert                  S1:  I don't know what to do Saturday night
Open-Option, Offer      S2:  You could go to Bob's party with me
Commit,Accept           S1:  Great, I'll see you there

Action-Directive    S1:  Take the engine from Corning to Bath.
Commit, Accept      S2:  OK
```

In fact, note that in COCONUT a `Commit` will only rarely co-occur with `Initiate` (see Sec. 7.1). An example:

```
Info-Request          S1:  and what colors are yours [chairs]?
Commit, Initiate      S2:  so I can log them in
```

The coder should also remember that in COCONUT we allow lookahead to the end of the current turn. This means that sometimes a potential commitment is not a commitment, because it is cancelled later on in the turn. This happens in the example below, where `(b)` would qualify as an `Offer` if `(d)` were never uttered. Since the commitment is cancelled, `(b)` is not tagged along the `Influence-on-Speaker` dimension. (We explain how `(b)`, a relative clause, can potentially qualify as an `Offer` by itself in Section 4.5.6.)

```
S1: (a) I have 1 sofa for 350
    (b) that is yellow
    (c) which is my cheapest.
    (d) yours sounds good.
```

4.4 Other-forward-function

Acts under this category do not form a natural category, but are grouped together as they are relatively rare. They include (1) conventional conversational acts such as greeting and saying goodbye, (2) explicit performatives, and (3) exclamations.

- Conventional acts:
  - `ConventionalOpening`: Is the utterance a phrase conventionally used to summon the addressee and/or start the interaction (e.g., "Can I help you?", "hi")
  - `ConventionalClosing`: Is the utterance a phrase conventionally used in a dialogueclosing or used to dismiss the addressee (e.g., "Good-Bye")
- `ExplicitPerformative`: Is the speaker performing an action by virtue of uttering the utterance. Typical examples are expressions of psychological states (e.g., "Thank you", "I apologize"); and utterances in which the speaker explicitly declares what is performed, as in firing someone by saying "You're fired", or, in TRAINS, quitting the task by saying "I quit".

  Test: can the word "hereby" be inserted before the verb?
  For example, since "You're hereby fired" has approximately the same meaning as "You're fired", "You're hereby fired" is an explicit performative. The utterance "I want to go to the store" is not, since "I hereby want to go to the store" is difficult to interpret, and in contexts where it might be understandable, "hereby" changes the meaning of the original utterance.

- Exclamation: Is the utterance an exclamation (e.g., "Ouch")

Note that acts that are conventional openings or closings can be coded along other aspects as well. For example, at the start of a dialogue the utterance "Can I help you" could serve both as a `ConventionalOpening` and as an `Offer`.

4.5 Discussion

4.5.1 Potential Actions

The notions of `Influence-on-listener` and of `Influence-on-speaker` rest on what counts as the *potential action* underlying a certain utterance, for the listener in one case and for the speaker in the other. We observed that in COCONUT there are at least two cases that require a precise definition of potential actions, *disjunctions* and *exhaustive listings*. We briefly discuss the problems associated with these two types of utterances, then we provide a convention for recognizing actions in our domain.

1. *Disjunctions.* What are the actions underlying an example such as `I have a red sofa for 150 or a blue one for 200`: the more abstract *buy a sofa* or the two more concrete *buy my red sofa* and *buy my blue sofa*? Adopting the latter solution has the consequence that the utterance should be subdivided into two units of analysis.

2. *Lists of items.* In many cases, our subjects start out by listing their whole inventory, or at least, by exhaustively listing all the items of a certain type they possess (lists of items can still be mentioned later in the dialogue, although the further the negotiation has proceeded, the less likely exhaustive listings are). Intuitively, marking each furniture item in these cases as both `Open-Option` and `Offer` besides `Assert` doesn't seem correct, as no real action is under consideration. This might not be a problem, as the cooccurrence of `Assert, Open-Option, Offer` could indeed characterize initial exchanges in the dialogue; however, our intuition that the "action" labels are not correct arises because it is not clear what the potential actions are in these cases, other than the very abstract *buy furniture items*.

Convention. There are two types of potential actions in Coconut; they correspond to *meta-actions* — all those actions that are labelled `StrategizeAction` or `GameProcedure` at the `Task` or `TaskManagement` level — or to actions in the domain. In both cases, we further distinguish between *general* (potential) actions and *specific* (potential) actions: a specific action is one that has all the necessary parameters set. (Note that our definition of actions does not concern information giving actions, i.e, those underlying an `Info-Request`. Our coding experience shows that they are easy to recognize.)

Specific vs general vs no action. A *specific* action in Coconut is an action in which all necessary parameters are known. It is difficult to pinpoint necessary parameters for *meta-actions* in general: for example, for the *meta-action* of starting furnishing one specific room the necessary parameter is the room itself, as in

`(a) Let's begin from the living room`

In general, we assume that a *meta-action* is specific, unless the disjunction convention below applies.

As regards domain actions, in general, necessary parameters are *type of furniture item*, *price* and *room*; whether *color* is necessary depends on context. Note that the parameter *room* should not be taken as set by default: while *room* is indeed instantiated by default to *living room* for *sofa* and to *dining room* for *table*, no such 1:1 correspondence exists for the other furniture items.

Rule of thumb: *specific* actions are tagged as `Action-Directive` in their *Influence-on-Listener* aspect, as long as the utterance passes the test regarding *the speaker trying to get the addressee to do something*, see Section 4.5.2; and as long as the utterance does not qualify as an `Info-Request`. *Specific* actions are also tagged along the *Influence-on-Speaker* aspect, when applicable.

General actions. They arise either because not all necessary parameters are set, or the action is an abstraction of different choices that the speaker may list in a single utterance.

- *Lack of necessary parameters.*
  Any domain level action missing some of its parameters will qualify as a general action, e.g.

  `(a) I have a blue sofa`

  In a null context, the necessary parameter *price* is missing, so this utterance qualifies as a *general*, not as a *specific*, action.

  It is hard to find examples where necessary parameters are missing at the meta level.

- *Abstraction from explicit choices.* This mainly covers explicit disjunctions.
  At the meta level

  `(a) Shall we begin from the living room or from the dining room`

  can be abstracted as `start from room`, where the parameter `room` can be instantiated as `living room` or `dining room`. The disjunction could also regard the actions itself, rather than parameters of a single action:

  `(a) Should we color coordinate or match furniture?`

  At the domain level, let's consider

  `(a) I have a red sofa for 150 or a blue one for 200`

  The underlying action in `(a)` (in the appropriate context, i.e., excluding the exhaustive listing reading discussed below) is `buy sofa`, where the parameter `sofa` can be instantiated either as `my red sofa for 150` or `my blue sofa for 200`. It will be tagged as `Assert, Open-Option, Offer`.[4]

Rule of thumb: *general* actions will be tagged as an `Open-Option` in their *Influence-on-Listener* aspect; and they will not be tagged along the *Influence-on-Speaker* aspect.

No action. When the coder can individuate neither a specific nor a general action, we assume that there is no potential action underlying the utterance, and that therefore neither `Influence-on-Listener` nor `Influence-on-Speaker` applies.

[4]The same example in the DAMSL manual is tagged in the same way; however, the assumption there is that the underlying action directly includes a disjunction , *we could buy my red sofa or my blue sofa* (DAMSL manual, p. 10).

Convention. Exhaustive listings of items still present a problem, as one could recognize either specific or general actions underlying utterances such as `(a), (e), (f)` and `(g)` below. We tag any occurrence of exhaustive listings, both at the beginning or later in the dialogue, only as `Assert`. The definition of exhaustive listing varies according to whether priority or secondary items are of concern. Any listing of all items of a certain type qualifies as exhaustive if the item is a priority one (e.g., tables); however, a listing of secondary items must mention all of them, not just all of the secondary items of a certain type, to be exhaustive. Namely, if a participant has 6 chairs (primary items), 2 lamps (secondary), and 1 rug (secondary), an utterance like `I have 4 red chairs and 2 green ones` is an exhaustive listing, an utterance like `I have two red lamps` is not, and one like `I have two red lamps and 1 blue rug` is.

Note that sometimes there may be ambiguity on whether the subject is listing all his/her items, or presenting a possible partial solution. This will happen when the number of primary items of a certain type a participant has is exactly the number required for the solution. For example, in the excerpt below, S2 has exactly 1 table and 4 chairs; thus, it is not clear whether `(e) + (f) + (g)` qualifies as an exhaustive listing, or whether S2 is proposing a partial solution, composed of `(e) + (f) + (g)`:

```
S1: (a) I have only 1 table- yellow $400, no yel chairs,
        but 2 blue chairs-$75, 2 green-$100, and 2 red-$100....
    (b) My budget is $400.

S2: (c) my bankroll is $550 this time.
    (d) shall we start the same way, with the table and chairs?
    (e) i have 1 table.
    (f) it's green - $400.
    (g) i also have 2 green chairs for $100 each and 2 yellow chairs for $75 each.
```

In these cases, we look for other sources of evidence that may help disambiguate. Here, a partial solution such as `(e) + (f) + (g)` does not satisfy the secondary goal of choosing furniture items of the same color in each room; as S2 gives no indication that she is giving up the goal of matching colors, it appears that in `(e) + (f) + (g)` she is only listing all her tables and chairs. Thus, we tag all of these utterances simply as `Assert`. Btw, this is a case in which the graphics could help, but it doesn't.

Other notes and examples.

1. The existence presupposition. Underlying our definition of actions, both general and specific, is the assumption that the speaker knows whether a certain items exists. Thus Y/N questions that inquire whether the partner has a certain furniture item don't count as potential actions.

   ```
   S1: (a) Red chairs in the dining room.
       (b) Do you have 2 chair red to go in the dining room
       (c) which will give us 200 pt for the table, 100 pt for chairs +
           10 pts. for same color furniture.
   ```

   There is no meta- or domain action underlying `(b)`, so `(b)` can only be tagged along the `Influence-on-Listener` as an `Info-Request`. The same applies in hypothetical scenarios:

   ```
   S1: (a) If you have two red chairs to go with my red table,
       (b) we'll have a nice set for the dining room.
   ```

There is no domain action underlying (a), as S1 does not know whether S2 has those two red chairs; however, (b) presupposes their existence, so it qualifies as a specific action, and thus, as an Action-Directive.

2. It may happen that there are ill-formed actions, i.e. that an utterance could count as an Action-Directive and/or as an Offer, but that not all necessary parameters are set. The context and the graphics may help the coder disambiguate; the coder should annotate in the *Comment* field which features of the context help make the decision.

3. It may happen that a single utterance gives rise to two potential actions, one generic, the other specific.

   (a) How about your green table and some cheap chairs?

   When (a) is uttered, the subjects are discussing furnishing the dining room, and the price of the table is mutually known. Therefore, the utterance gives rise to the specific action buy your green table for 200, and to the general one buy (cheap) chairs. Thus it should be tagged as both Action-Directive and Open-Option; however, these two tags are mutually exclusive, so we tag the utterance simply as Open-Option.

4. **Convention.** In some contexts, for instance, when an utterance is associated with both a meta-action and a regular action, some confusion may arise about which action is being committed to. For instance, in the following, in (e) S1 both conditionally commits to a regular action (*using S1's yellow rug for 150*) and unconditionally commits to a meta-action (*not matching colors*, given in (d)). In such cases, the coder should use the Comment field to indicate which Influence-on-Speaker tag refers to which action.

Accept, Commit	S1:	(a) Let's take the blue rug for 250,
		(b) my rug wouldn't match
Open-Option		(c) which is yellow for 150.
Action-Directive	S2:	(d) we don't have to match...
Accept(d), Offer, Commit	S1:	(e) well then let's use mine for 150

4.5.2 Trying to get the addressee to do something

The decision tree in Figure 3 contains a decision point regarding whether the speaker is *trying to get the addressee to do something*. While it is hard to devise a comprehensive test regarding whether the speaker is trying to get the addressee to do something, there is a prototypical case in Coconut in which the speaker is *not* trying to get the addressee to do anything. The speaker mentions one action that the participants could undertake, but, in the same turn makes it clear that that action is not to be performed. For example:

S1: (a) I have a blue sofa for 300.
    (b) It's my cheapest one.

S2: (c) I have one sofa for 350
    (d) that is yellow
    (e) which is my cheapest.
    (f) Yours sounds good.

Given our definitions, a specific action (*get S2's yellow sofa for 350*) underlies (d), which thus qualifies as an Action-Directive (btw, also (a) qualifies as such, as it is not an exhaustive listing). However, because of (f), it is clear that S2 is not offering his own yellow sofa as part of the solution, thus, S2 is not trying to get S1 to use that furniture item. (d) is therefore tagged as an Open-Option.

The same scenario can occur at the meta level:

```
S1: (a) we can try to color coordinate with chairs.
    (b) i have 3 green (50) 5 yellow (150) 2 red
    (c) but colors aren't that many points...
    (d) maybe we should just try to accumulate furnature.
```

(a) could qualify as an Action-Directive, but given that S1 changes her mind in (d), (a) is tagged as Open-Option.

Thus, the coder must remember to use lookahead (in Coconut, the whole turn) to decide whether an utterance qualifies as an Open-Option or as a Action-Directive.

Another problematic case regarding the notion of *trying to get the addressee to do something* arises when the specific action is provided as an answer to a question, as we discuss in Section 4.5.5.

4.5.3 Influence-on-Listener and Influence-on-Speaker

Given the nature of our task, we expect that these two aspects will almost always co-occur whenever there is a specific action underlying an utterance. They will not co-occur in those few cases in which subjects think each of them can only buy from his/her own furniture list, or when they discuss updating their own private graphics displays.

4.5.4 Accept, Influence-on-Speaker and Influence-on-Listener

There is a class of utterances such as `the lamp floor sounds good`, `your red set is a good idea`, etc, that Accept an Open-Option or an Action-Directive proposed by the other speaker — see Sec. 7.2. The question is how they should be tagged at the Forward-Function level. The Statement dimension clearly applies (they are present declarative sentences that include a stative verb). What about Influence-on-Speaker and Influence-on-Listener? Our convention is that if we can individuate a specific action underlying such utterances, either because it is contained in the antecedent, or because the speaker identifies such an option by means of the utterance itself the utterance should be tagged along those two dimensions.
Rule of thumb. Such utterances will typically be tagged as Action-Directive and Commit.

4.5.5 Answer, Influence-on-Speaker and Influence-on-Listener

The fact that answers are always solicited causes them to be treated slightly differently from other types of utterances. To see this, compare the exchanges in (a-b-c) and (a'-b'-c'):

```
S1: (a) OK, we're done in the dining room.
    (b) Let's move on to the living room.
```

```
S2: (c) I have a blue sofa for $350.

S1: (a') OK, we're almost out of money.
    (b') Do you have anything for $50 or under?
S2: (c') I have a blue rug for $50. (Context: S2 has no other items for under $50.)
```

There are specific actions underlying both (c) and (c') and, therefore, according to the decision tree in Figure 3, both of these utterances could potentially be Action-Directives. However, the action *get S2's blue rug for 50* underlying (c') comes into focus only because S1 asks for it, whereas in (c) S2 chooses to mention his blue sofa from among several alternatives. Based on this, it is reasonable to assume that only in (c), and not in (c'), is S2 actually trying to get S1 to do something. Thus, S2's response should be tagged as an Action-Directive in (c) and as an Open-option in (c').

Similar considerations apply to tagging answers for the Influence-on-Speaker function. In most cases answers do not involve commitment. This is especially the case with answers that simply provide some additional information, such as (d) below; in some other contexts, (d) might be tagged as an Offer, but not when it simply answers a question as in this case:

```
S1: (a) how about 2 of yours [chairs] and 2 of mine?
    (b) mine are green.
    (c) what colors are yours?

S2: (d) The 2 chairs that i have are red.
```

In (d), S2 simply answers S1's questions in (c), and does not provide any further information as regards the solution proposed by S1 in (a). While the price of S2's chairs is known, so that there is a specific action underlying (d) (get S2's red chairs), it does not appear that S2 is trying to get S1 to use S2's chairs; therefore (d) does not pass the test regarding trying to get the addressee to do something.

Answers that express opinions are, however, somewhat less straightforward. For instance, an answer such as the one in (c) below does seem committal:

```
S1: (a) I have a lamp floor (blue), a green table and 4 chairs for 100.
    (b) What do you think?
S2: (c) The lamp floor sounds good.
```

But note how making the answer a comparison, rather than a simple evaluation, removes some of that commitment:

```
S1: (a) I have a lamp floor (blue), a green table and 4 chairs for 100.
    (b) What do you think?
S2: (c) The lamp floor sounds better than the table.
```

The reason why a comparison seems less committal might be that saying x is better than y does not commit the speaker to the view that x is a good solution; it is just better than y. However, a response such as The lamp floor sounds good does potentially commit a speaker to that solution. Thus, there seems to be a general tendency for answers that are opinions to be less committal if they relate two solutions than if thay simply state a positive or a negative opinion about one. However, always consider the context when making a decision about this dimension.

4.5.6 Elaborations, `Influence-on-Speaker` and `Influence-on-Listener`

It often happens that an item is introduced and then elaborated in subsequent utterances: the question is, given that often introducing and elaborating items gives rise to potential actions, and therefore to tagging the utterance with `Influence-on-Speaker` and/or `Influence-on-Listener` tags, should these tags be used on every such utterance? Here we provide some guidelines on when to include and when to omit these tags.

When to omit the `Influence-on-Speaker/Listener` tag Proposals which consist of more than one consecutive utterances raise the question of how to indicate that these utterances share their `Influence-on-Speaker` and `Influence-on-Listener` functions. These cases could be treated in three different ways: (1) all the utterances could be tagged for the appropriate communicative functions individually, (2) the utterances could be grouped together as a segment and the functions could be tagged at the segment level or (3) one of the utterances could be treated as the main proposal and the others as elaborations which, in a sense, "inherit" the communicative functions of the main proposal. Since the first option seems quite redundant, and the second would introduce more segments (which we try to use conservatively), we opt for the third option. Thus, in the following, even though at an intuitive level `(a)` and `(b)` together form an `Offer` and an `Action-Directive`, only `(a)` is given these tags:

```
Action-Directive, Offer       S1: (a) I have a blue sofa 300
ElaborateItem, SameItem(a)        (b) it's my cheapest one.
```

`(b)` is linked to `(a)` via elaboration and via coreference (`SameItem`), thus it "inherits" the `Action-Directive, Offer` tags.

When not to omit the `Influence-on-Speaker/Listener` tag

Changes in function. Note that the elaboration in `(b)` above does not give rise to an action description different from the one underlying `(a)`, at least in the terms we have defined actions in this manual. However, other elaborations change the action description from general to specific, and therefore change the `Influence-on-Speaker/Listener` functions from one utterance to the next: obviously in this case those functions should be annotated for.

This most often happens when the initial introduction of an item does not qualify as a specific action (see Section 4.5.1 on potential actions). In these cases it is often the elaboration that fixes the missing parameters and makes the action specific. For instance, in the example below, `(a)` is not specific enough to be an `Action-Directive` or an `Offer`. Therefore, it is just an `Open-Option`.[5] The missing color parameter is then filled in in `(b)`, which is, hence, tagged as an `Action-Directive` and an `Offer`.

```
Open-Option                   S1: (a)  I have one table for 350.
Action-Directive, Offer           (b)  it is green.
```

Given that non restrictive relative clauses are considered as independent units of analysis (Section 14), this convention explains how a non restrictive relative clause that elaborates on an item description may be tagged as `Action-Directive, Offer`. If we change `(b)` from the preceding example to a non restrictive relative clause, as follows:

[5] However, in a context where the color matching goal had explicitly been given up, (a) would qualify for an Action-Directive and an Offer.

```
Open-option                  S1: (a)   I have one table for 350,
Action-Directive, Offer          (b')  that is green.
```

exactly the same analysis applies.

Lookahead. Another situation in which the communicative function of an elaboration should be coded for is when the function of the "main proposal" and the elaboration is the same only because the speaker later on in the turn backtracks out of a commitment or a request. Such a case is illustrated below. In this example (d) would be an Action-Directive and an Offer, just like (b) and (b') in the examples above, if it were not followed by (f). However, since (f) makes it clear that S2 does not want her sofa to be bought, (d) is only an Open-option. By not omitting this tag we indicate that (d) is an Open-option not because of inheritance, but because its actual communicative function was cancelled.

```
Action-Directive, Offer             S1: (a)   I have a blue sofa for 300.
(``inherits'' AD, Offer from (a))       (b)   It's my cheapest one.
Open-option (non-specific)          S2: (c)   I have 1 sofa for 350.
Open-option                             (d)   that is yellow.
(``inherits'' OO from (d))              (e)   which is my cheapest
Accept (a), Commit                      (f)   yours sounds good.
```

4.5.7 Conventional acts

The previous version of DAMSL had a separate category for conventional acts like thanking and apologizing. These acts now fall into different categories based on the tests presented. For instance, an utterance like "I'm sorry" is an Assert as it makes a claim about the world, in fact it is a present, declarative utterance that contains a stative verb. The utterance "Thank you", on the other hand, cannot be analyzed in such a way and only falls into the explicit performative category (observe that "I hereby thank you" is fine).

Short utterances such as "okay" and "yes" can have many different interpretations depending on how they are being used. In fact, they can be Asserts, Commits and even Other-forward-functions. If the utterance conveys information in answering a question, then the utterance is only an Assert, as in:

```
Info-Request   S1:  (a) is there a train at Chicago?
Assert         S2:  (b) yes.
```

On the other hand, if the utterance commits the speaker to a certain action, say in response to a request or invitation, then it should only be labeled as a Commit

```
Action-Directive    S1: (a) tell me if the route gets too long
Commit              S2: (b) okay
```

Cases where the speaker utters an "okay" to accept a request and then performs the act requested are also marked as a Commit, even though the action to which the speaker is committing is performed immediately after:

```
Action-Directive     S1: (a) can you tell me the time?
Commit               S2: (b) okay.
Assert                   (c) three o'clock.
```

Utterances such as "okay" can be used not only to convey information but also to manage the dialog, possibly to hold the turn, or to signal the introduction of a new topic. In these cases, the Forward Function of these utterances is not directly captured in the annotation scheme, and they should be marked as (an unspecified) `Other-forward-function`.

5 Topic

This menu tries to capture what the utterance is about. There are two semi-independent aspects we code for in this dimension: what could be called *Topic* proper — whether the utterance concerns furniture items, money or points; and *Attitude*, which indicates the speaker's attitude towards the furniture items, budget or point accumulation under discussion, or towards a (possibly partial) solution or plan. Note that most of the time only one of these two aspects will be tagged — see below for some notes on when to tag with both.

Tags related to *Topic* proper are: concerning furniture items, *needItem, haveItem, getItem, elaborateItem, otherItem*; concerning money, *budgetAmount, budgetRemains, costAccum*; concerning points, *pointAmount, pointAccum.*

Tags expressing an *Attitude* of the speaker are *Eval* and *Relate.*

Tags related to Furniture Items.

- *needItem, haveItem, getItem.* They closely reflect the surface form of the utterance: e.g. *We need a cheaper sofa* is tagged as <needItem - pos>. However, surface form is not all: for example *buy the chairs (2)* is tagged as <getItem - pos>, even if no explicit *get* is included in the utterance.
  Rule of thumb. The criterion for inferring one of these tags if they don't explicitly appear in surface form is that the corresponding verb (*need, have, get*) can be either substituted or inserted in the utterance (if e.g. the utterance is elliptical).

  For example, `Your table deal is good` in the exchange below is not tagged as *getItem*, because no direct lexical substitution or insertion of *get* is possible.

  ```
  S1: (a)  This time I have all of 550.
      (b)  I have a red high table for 300 and a blue sofa for 300.
      (c)  Also 2 red chairs for 50,
      (d)  which would match the table.

  S2: (e)  Your table deal is good
  ```

  On the other hand, utterances such as `(a)` and `(b)` below are tagged as <getItem pos>, as they're paraphrasable as in `(a')` and `(b')` respectively:

  ```
  (a) I was thinking about your yellow table and 2 blue chairs and my 2 yellow chairs
  (a') I was thinking about [getting] your yellow table and 2 blue chairs and my 2 yellow
  chairs


  (b) What do you plan on doing, with this 100 we saved?
  (b') What do you plan on getting, with this 100 we saved?
  ```

  A further requirement for inferring one of these tags is that the utterance must concern a specific item

or a type of item. Therefore, (c) below is would not be labeled as `getItem`, even though the verb *accumulate* could be replaced by *get*.

`(c) Maybe we should just try to accumulate furnature.`

However, if the utterance is a question, such as `(b) - (b')` above, `getItem` applies as long as the expected answer concerns specific items.

The value *pos(itive)* that appears in the tags just given as examples corresponds to the semantic polarity of the utterance (a tag for syntactic polarity is available, see Sec. 6). The value *neg(ative)* is also available. It it is often used when an explicit negation is present, as in *I don't have a cheaper sofa*, tagged as <`haveItem - neg`>: note that here the tag doesn't mean that the speaker doesn't have a sofa, but that he doesn't have a *cheaper* sofa. However, an explicit negation is not necessary: *We will have to forget about the lamp* is tagged as <`getItem - neg`>.

For questions, *neg / pos* corresponds to the polarity of the question: *Do you have a blue rug* will be tagged as <`haveItem - pos`>, *Don't you have any green chair* as <`haveItem - neg`>.

It may happen that when speakers give a list of items they own or need, they also mention things they don't have or don't need. Such lists consist at times of a main verb plus a string of coordinated NPs: as we don't consider NPs to constitute independent utterances in such contexts (Section 14), the utterance will be tagged as `haveItem` or `needItem`, without specifying `pos` or `neg`.

- *elaborateItem*: used to code elaborations on a description of a *specific* item already introduced. It is used both when the speaker asks for values of certain features (*How much is the red sofa*) or asks for clarification on the values of certain features (*so are the chairs 25 each?*); and when further information is provided (*No, the chairs are 100 each*). It is the only value on the Topic menu that points to the *ItemFeature* submenu — we think that when an item is elaborated in some way it may be important to encode which particular feature is specified; however, we decided not to code for the features mentioned in utterances tagged e.g. as *haveItem*. Also note that *elaborateItem* is independent from the *coreference* tag *sameItem*, see discussion in Sec. 7.4.

  Rule of Thumb. *elaborateItem* should be preceded by *haveItem* or *getItem*, unless the "antecedent" appears in a question. For example, (a) below is tagged as `otherItem`; then (b) should not be tagged as *elaborateItem*, but as *haveItem*.

  ```
  S1: (a) I say let's start with a blue sofa;
      (b) I have one for 300
  ```

- *otherItem*. The utterance concerns (an) item(s) of furniture, but none of the other *x-Item* tags applies. (a) above — `so I say let's start with a blue sofa` — is an example of *otherItem*.

  Note. To be tagged as *otherItem*, the utterance should contain an explicit reference to a furniture item: namely, either the furniture item is present in surface form, or it is recoverable if it's not verbalized (as in ellipsis). A case such as

  `(a) I think we should change our ugly red dining room around`

  is not tagged as `otherItem` (it's tagged as <`Eval - Neg - Color`>).

Tags related to budget.

- *budgetAmount*: to tag utterances where the available initial budget is mentioned, such as *I have 550*.

- *budgetRemains*: to tag utterances where the amount of money left is discussed, as (b):

  S1: (a) let's do the yellow/blue mix,
  (b) leaving us with 250.

- *costAccum*: to tag utterances where the costs of various choices are summed up:

  (a) we spent: 300 (sofa), 250 (lamp), 200 (table), and 300 on chairs

Tags related to points.

- *pointAmount*: to tag utterances where the points associated with a certain choice or choices are mentioned:

  (a) Gained 200 for sofa, 200 for high table, 10 points for
  matching in livingroom and 10 points for the yellow chair
  in diningroom.

- *pointAccum*: to tag utterances where the points associated with the solution are summed up. (c) in the excerpt below is tagged *pointAccum*:

  S1: (a) How about I buy two of these Beautiful 50 green chairs over here.
  (b) that would expire our money
  (c) and give us 630 pts.

Evaluate and Relate.

- *Eval(uate)*, with the two possible values: *pos, neg*.
- *Relate*, used for comparisons, with possible values *better, worse, same, dfft (different)*.

These tags concern the attitude of the speaker towards: either furniture items, budget amounts and costs, and point amounts; or a (possibly partial) solution or plan. They don't apply to utterances that express an attitude towards other things talked about, such as `Btw, your screen is right`, or to utterances tagged as *CommunicationManagement*.

Eval and *Relate* point — through their values *pos, neg* and *better, worse, same, dfft* — to the *Item Features* submenu, that details the particular features that are evaluated or compared. For example *The chairs sound too expensive* is tagged as <Assert - Eval - Neg - Price>; *I do not have a sofa for a better price* as <Assert - Relate - Worse - Price>.

Note. Sometimes the evaluation or comparison is not directly explicit in the utterance, but requires some inference. We use this tag only when inferencing is minimal, and derives only from the utterance being tagged and the previous context — in particular, it shouldn't be inferred by looking ahead to the rest of the turn.[6] Thus, *Eval / Relate* will be used only

- when the utterance explicitly conveys an evaluation or comparison, as in the examples just mentioned. Other examples are *That sounds good*; *I like red*; etc.

[6] We don't regard this as inconsistent with using the full turn as lookahead for determining the forward function of the utterance.

- or when the utterance conveys an evaluation or comparison via explicit signals such as contrastive cues. (b) is tagged as <Assert - Relate - Better - Price>:

  S1: (a) The blue sofa costs 400.
      (b) My yellow sofa is only 350 however.

  (c) below will be tagged as <Assert - Eval - Neg - Price> — in the immediately preceding exchange, the two speakers have eliminated a lamp they had previously agreed upon because it costs $250, and exceeds their budget.

  S1: (a) do you have a rug or something blue?
  S2: (b) I have a blue rug,
      (c) but it is 250 also.

An example that should not be tagged as Eval is (b) below. While the speaker must think this is a good solution, there isn't anything clearly evaluative about his utterance:

S1: (a) do you have 2 chair red to go in the dining room
    (b) which will give us 200 pt for the table, 100 pt for chairs
        + 10 pts. for same color furniture.

Conventions.

1. *Better / Worse* refers to the conceptual level of comparison between the two entities X and Y, where X is the item mentioned in the utterance, or the first item mentioned if both X and Y appear. In I do not have a sofa for a better price, X is *my sofa*, Y is *your sofa*.

2. Often an utterance tagged with Eval or with Relate will be linked via a *sameItem* or *subset* coreference tag to a previous utterance that mentions the same item(s) (see Sec. 7.4). In the case of a Relate, as two items are compared, it is possible that the current utterance is linked to two antecedents. As will be discussed below, we will edit the *link* value to include both references, but only when computing these 2 links is immediate. For example:

   S1: (a) I have a red sofa for 300.
   S2: (b) I also have a red sofa,
       (c) but it's only 200.

   In (c), X is S_2's red sofa, and Y is S_1's red sofa. Therefore, (c) will be tagged as <Relate - better - Price> and <sameItem - (b),(a)>, where (b) indicates coreference to X and (a) to Y.

When to use two Topic tags. As mentioned, we don't intend pairs of topics to appear on every utterance. Obviously, the *Attitude* tags won't appear unless the utterance expresses an attitude of the speaker; but even when an *Attitude* tag is warranted, doubly tagging the utterance with one of the other topic tags should happen only when the coder thinks that the utterance really mentions two topics. This is more likely to happen when the utterance comprises more than one clause, each of which actually conveys one of the two topics. For example, in

```
S1: (a) Just curious,
    (b) how much do your green chairs cost
    (c) because i have a green table over here worth only 200!!!
```

(c) is tagged with a double topic, <haveItem - pos> and <Relate - better - price>.
This can of course happen in other cases, such as

```
(a) I do not have a sofa for a better price
```

which is tagged as <haveItem - neg>[7] and as < Relate - worse - price>.
Although two topics most commonly occur when a speaker introduces a referent in the domain and expresses an attitude towards it in the same utterance, there are also other cases where two topics clearly exist. For example,

```
(a) Alright,  we will go with the two red chairs (310 pts.!)
```

is tagged with both pointAccum and getItem. What shouldn't happen is that an utterance is tagged as both otherItem and with an *Attitude* tag, given that otherItem doesn't really add any information in this case.

5.1 ItemFeature

This menu specifies which properties of items are discussed, compared or evaluated:

- obviously, *price, color, type, points*
- the possible combination of the furniture item properties: *typeColor, typePrice, colorPrice, all.*
- *genl*, for *general*. It should be used for general assessment about a solution or a plan, such as *That sounds good* (in the appropriate context).

Note. The coder should be as specific as possible in individuating the relevant features, and use genl only as a last resort. When specific items are mentioned, such as in

```
the lamp and table sound good
```

the specific properties relevant from context should be tagged for, such as type or typeColor. If no specific properties are relevant, then all should be used. genl should be used only when no specific furniture items are explicitly discussed, as in (b), or when a global solution is discussed:

```
S1: (a) We should start from the living room.
S2: (b) That sounds good.
```

Note. The composite tags in the *ItemFeature* menu, especially those involving *type* — *typeColor* and *typePrice* — rarely appear with *Eval* or *Relate*. An evaluation or comparison on *Type* by itself seems rare as well; it could occur with either non-priority items — *The lamp is better than the rug, we need some light in the living room* — or with priority items when the money available is so scarce that one can't afford all of them — *I prefer the sofa to the table.*

[7] A reminder that, as discussed above, <haveItem - neg> does not mean that the speaker doesn't have any sofas.

6 SurfaceFeatures

There are two menus that capture the surface features of the utterance, `Word-Surface-Features`, and `Syn-Surface-Features`. The former include Matrix, Modal and Subject, and the latter Tense, Mood and NegPolarity. They are not conceptually different, they are separated in the Nb configuration file simply for efficiency reasons. A tensed utterance will be coded for all surface features (apart from NegPolarity, used only if the utterance includes an explicit negation). Fragments will be coded only for those features that have non-nil or non catch-all values, see Section 9 for details.

`Word-Surface-Features` includes three cascaded menus, *Matrix*, *Modal* and *Subject*. Note that *Modal* and *Subject* are used to tag for features of the embedded proposition, if there is a matrix, or of the main clause, if there is no matrix.

- *Matrix*. Its possible values are surface *matrix* indicators, *iSay, iGuess, iThink, iKnow, letMeKnow, otherMatrix, noMatrix*. Notice that we consider *matrix* indicators only those expressions that have a *full tensed* clause as argument — thus, *I want to buy those two chairs* is tagged as *noMatrix*.
- *Modal* has possible values: *howAbout, lets, can, could, must, should, would, otherModal, noModal*.
  Convention. `otherModal` is used for all the other modals, such as *have to* or *may*, that are not explicitly listed in the menu.
- *Subject* has possible values: *i, you, we, otherSubject*.
  Convention. *otherSubject* is used whenever the subject is not *i* or *you* or *we*, or when there is no subject.

`Syn-Surface-Features` comprises three cascaded menus, *Tense, Mood, Neg-Polarity*. They apply to the embedded proposition, if there is a matrix, or to the main clause, if there is no matrix.

- *Tense* has possible values: *pres, past, future, presProg, pastProg, presPerfect, pastPerfect, otherTense, noTense*.
- *Mood* has possible values: *questionY/N, questionWH, indir(ect)-question (e.g. I forgot how much the sofa was), imperative, declarative, otherMood*
- *Neg-Polarity* is used only when an explicit negation is present.

Conventions.

- *let's*. It is tagged as *imperative* (Mood) and as *we* (Subject).
- *would, should, could*. The value of the *tense* feature is *present* when they are followed by a bare infinitive, as in *it would expire our money.*

7 Backward Communicative Function

(Adapted from DAMSL, sec. 2.4.)

Backward Looking communicative functions indicate how the current utterance relates to the previous discourse. For example, an utterance might answer, accept, reject, or try to correct some previous utterance or

utterances. To capture these *Backward* functions, we need to both identify the type of *Backward* function and indicate the previous stretch of discourse that is being affected.

The previous utterance or segment (set of utterances) being responded to by the current utterance will be called the antecedent. The relations we consider here are local: their antecedent typically is either the previous utterance or segment, and usually appears in the previous turn. Constraints on what can be an antecedent and how to annotate complicated cases will be discussed after the basic set of tags is introduced.

There are four semi-independent aspects coded in this dimension. The first concerns relations affecting agreement about the task or whatever the topic of discussion is. The second concerns relations that signal understanding (or non-understanding). The third marks utterances that answer previous information requests. The fourth concerns the relationships between the informational content in the utterance and its antecedent and is for now left unspecified in DAMSL: we provide a minimal set of such relations below, in Sec. 7.4.

We include here the DAMSL hierarchy for *Backward* communicative function. In the following list, boldface identifies tags we add to DAMSL; tags in boldface and in parentheses are those that we don't include, as they do not occur in our corpus.

- **Initiate**
- Agreement
  - Accept
  - Accept-Part
  - Maybe
  - Reject-Part
  - Reject
  - Hold
  - **ClarificationRequest**
- Understanding
  - Signal-non-understanding
  - Signal-understanding
    - Acknowledge
    - RepeatRephrase
    - **(Completion)**
  - CorrectMisspeaking
  - **CorrectAssumption**
- Answer
- Informational Relations
- **Coreference / Set Relations**

7.1 Initiate

The utterance is unsolicited. It may appear odd to include `Initiate` as a value for `Backward-Looking` function; after all, `Initiate` means that the utterance does *not* perform a `Backward-Looking` function. However, we keep marking for `Initiate`, whereas DAMSL leaves initiatives unmarked, mainly to make sure that the coder has not forgotten to tag for `Backward Function`.

7.2 Agreement

The agreement aspect codes how the current utterance unit affects what the participants believe they have agreed to, typically at the task level. These relations occur in contexts where one agent has made some kind of proposal such as a request that the hearer do something, an offer that the speaker do something, or a claim about the world. The current utterance then indicates the other participant's view of the proposal. In general, the agent may explicitly accept or reject all or part of the proposal, or may simply be noncommittal on the proposal, or may leave the proposal open by requesting additional information or exploring its consequences. Note that agreement does not necessarily require that the agent proposing and the agent (dis)agreeing be distinct: occasionally an agent may accept or, more likely, reject his/her own proposal:

```
S1: (a) we can try to color coordinate with chairs.
    (b) i have 3 green (50) 5 yellow (150) 2 red
    (c) but colors aren't that many points...
    (d) maybe we should just try to accumulate furnature.
```

In `(d)` S1 rejects `(a)`, her own proposal about color coordination.

Assuming the current speaker directly addresses a previous proposal, they could respond in one of the five ways shown in the example below. Note that the label of `Maybe` applies to cases like the one below where the speaker explicitly states that they cannot give a definite answer at that moment.

```
                      S1: (a)    Would you like the book and its review?
Accept(a)             S2:        Yes please.
Accept-Part(a)        S2:        I'd like the book.
Maybe(a)              S2:        I'll have to think about it
                                    (intended literally rather than a polite reject)
Reject-Part(a)        S2:        I don't need the review.
Reject(a)             S2:        No thanks
```

Agreement can apply to cases other than proposals, as shown in the two examples below. In the first example, `a` is an `Open-Option` as it simply presents a possible option for solving a problem. `(b)` is still considered an `Accept` though. Note, `Accept`'s are often words such as "alright", "yes", and "okay" as well as repetitions.

```
Open-Option      S1: (a)  we can unload them and then reuse the boxcars
                            on the way to Corning
Accept(a)        S2: (b)  alright
```

An `Accept` can also be used as a response to an `Info-Request`. In this example, the possible action accepted is that of supplying information (see Figure 5).

```
Info-Request     S1: (a)  can you tell me the time?
Accept(a)        S2: (b)  yes
Answer(a)            (c)  it's 5 o'clock
```

(c) is marked as an Answer, as discussed later.

Although an utterance tagged with an Agreement tag will most often be linked to an utterance or segment tagged as a Influence-on-Listener, it is also possible for it to Accept an Influence-on-Speaker or the content of an Assert. In this example, the Accept indicates that the information conveyed in the Assert is accepted.

```
Assert           S1: (a)  boxcars don't travel by themselves
Accept(a)        S2: (b)  okay
```

Accept-Part / Reject-Part. An issue arises with respect to Accept-Part/Reject-Part: namely, should one code for the implicit "Reject-Part" in the case of an "Accept-Part", and vice versa? The assumption in DAMSL is that we code only for what is explicitly addressed in the utterance; the implicated part is not coded for, as it could be cancelled right away, as in (c) below:

```
                    S1: (a) Would you like the book and its review?
Accept-Part(a)      S2: (b) I'd like the book,
Accept-Part(a)          (c) and I also want the review.
```

A response like I'll take the book but not the review will be segmented into two units, ''I'll take the book'' and ''[..] but not the review'', respectively tagged as Accept-Part and Reject-Part.

Hold. The Hold tag applies to cases in which the participant does not address the proposal but performs an act that leaves the decision open pending further discussion. This includes cases such as counter-proposals and questions that request additional information in order to help the participant make a decision (i.e., one sense of clarification request). This tag does not apply to cases where the speaker explicitly expresses uncertainty such as uttering "maybe".

In the dialogue below, *S1*'s request is vague since there are two possible paths; *S2* makes a clarification request by asking a question.

```
Action-Directive        S1: (a)  take the train to Corning
Info-Request, Hold(a)   S2: (b)  should we go through Dansville or Bath
Assert, Answer(b)       S1: (c)  Dansville
```

Instead of answering a question, a responder may make a clarification statement as shown below. This is also marked as a Hold.

```
Info-Request    S1: (a)  how long will that take
Hold(a)         S2: (b)  you want to go from Avon to Dansville
Answer(a)           (c)  that's 5 hours
```

In the examples below, *S2*'s suggestion of an alternative route should be marked as a `Hold` if the coder thinks that *S2* is leaving the original option open, but as a `Reject-Part` if not.

```
Action-Directive          S1: (a)  take the train to Avon via Bath
Open-Option, Hold(a)      S2: (b)  How about we go via Corning instead
```

```
Action-Directive                        S1: (a)  take the train to Avon via Bath
Action-Directive, Reject-Part(a)        S2: (b')  Go via Corning instead
```

The following exemplifies an instance of `Hold` in the Coconut domain. Instead of directly addressing S1's proposal, S2 provides a list of of his chairs, thus putting the proposal on hold. Since this Backward Function is shared by three utterances, they are marked as a Segment. (See Section 8 for more specific instructions on how to use segments.)

```
S1: (a) How about your green table and some cheap chairs?
S2: <Segment><Hold>
    (b) I have 6 chairs
    (c) that are green for 100
    (d) or 2 chair for 50?</Hold></Segment>
    (d) so what do you think?
```

`Hold`, however, should not be used if the utterance provides information that was requested for. For instance, in the following S1 makes a proposal in `(a)` to which S2 does not respond. Instead, S2 provides further information about the items in question, which makes his response look like a `Hold`. However, since the information in `(d)` was requested for, S2 is responding to something in S1's turn. Thus, we have no evidence for claiming that by saying `(d)`, S2 holds the proposal in `(a)`.

```
Offer, Action-Directive    S1:    (a) How about 2 of your chairs
                                  (b) and I have two also.
Info-Request                      (c) What colors are yours?
Answer (c)                 S2:    (d) The 2 chairs that I have are red.
```

ClarificationRequest. We add this tag to DAMSL. As mentioned above, `Hold` captures one sense of a clarification request, i.e., the request of additional information in order to help the participant make a decision. However, there are at least two other senses of *clarification request* captured by our explicit `ClarificationRequest` tag.

- A `ClarificationRequest` may occur when the participants review their decisions, because they have forgotten what was previously decided, the decision was after all not so clear, or they're wrapping up the game:
  `Did we decide on the red sofa?`
- A `ClarificationRequest` may also cooccur with a SNU (Signal-non-Understanding), or may occur while trying to repair a SNU.[8]

[8] One possibility is therefore that `ClarificationRequest`'s belong to the *Understanding*, rather than to the *Agreement*, dimension. We put them under *Agreement* because one sense of `ClarificationRequest` is captured by *Hold*. We will reconsider this issue in the future.

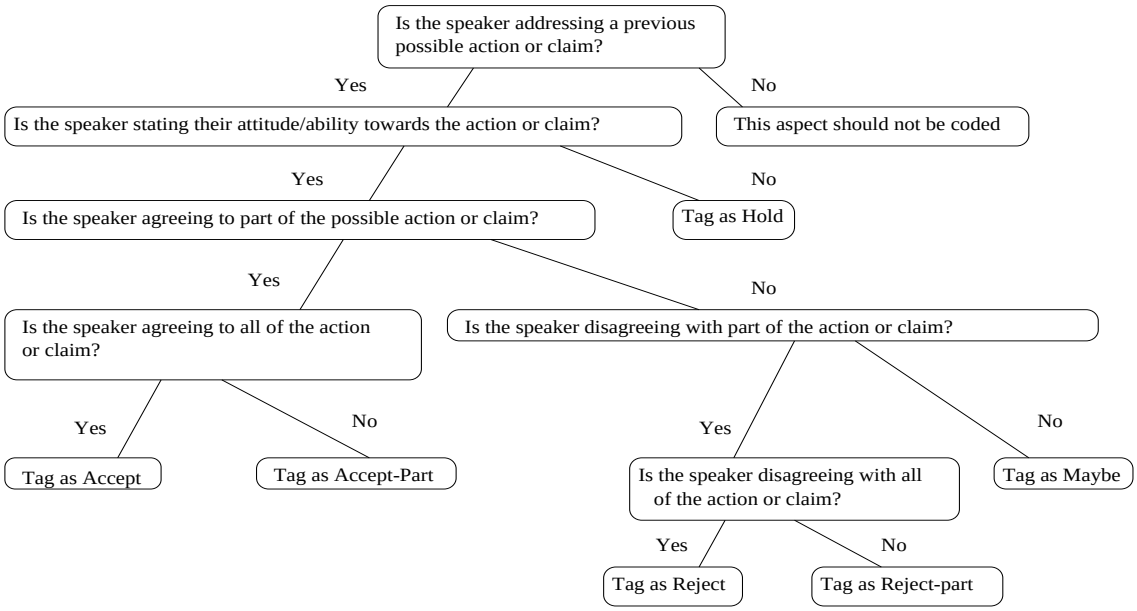

Figure 5: Decision Tree for Agreement Aspect

```
S1: (a) I have a lamp floor (blue), a green table and 4 chairs for 100.
S2: (b) The lamp and table sound good,
    (c) but the chairs seem expensive.
    (d) I have 4 blue for 100
    (e) they should look OK together
S1: (f) again that sounds good.
    (g) So the chairs are 25 a piece?
```

(g) is tagged as a SNU (see below) and as a `ClarificationRequest`. Note that (g) is not a `Hold`, because in (f) S1 agrees to S2's proposal. The fact that (f) is marked as `Accept` deserves further discussion. It is true that (f) represents an agreement conditional on the chairs costing $25 each, which in fact turns out to be false in this dialogue; however, it does not qualify as a `Hold`, as S1 is expressing his attitude towards (d). Often the conditional aspect of a (dis)agreement will be captured at the `Informational relation` level, via the `act:condition` tag.

Convention. We have chosen not to distinguish utterances that ask for confirmation from utterances that ask for clarification. Therefore, the tag `ClarificationRequest` should also be used for confirmation requests, such as (b) below:

```
S1: (a) I'm going to do 'design complete'.
    (b) OK?
```

The link value of a `ClarificationRequest` should point back to the action that is hindered by the lack of clarity.

7.2.1 Discussion

There may be cases in which it is unclear whether an utterance is an `Initiate` or an `Accept` (or another *Agreement* tag). Sometimes the graphics information may help the coder decide. The coder should annotate these ambiguities in the *Comment* field and if a choice is possible, an explanation of the choice.

Convention. Even though in conversation proposals can be accepted more than once, in COCONUT we make the assumptions that acceptances are not generally re-accepted. Thus, we do not allow the same proposal or claim to be accepted or rejected more than once.

It is also worth mentioning that acceptances are not always explicit. Sometimes they are inferred from other conversational behavior, such as silence. While in COCONUT we do not explicitly analyze implicit acceptances, the coder should still be aware of them, as sometimes they may affect the functions of some other utterances. For instance, in the example below, S1 infers that S2 has accepted the table from S2's not mentioning it during his turn. To make this inference explicit, she utters `(g)`. If `(g)` was uttered by S2, it would be coded as an `Accept`. However, since `(g)` is uttered by S1, who originally made the proposal, there is reason to believe that she thinks the proposal has already been accepted, and just wants to make the hearer aware of this, too. By tagging `(g)` only as a `Commit` and not also as an `Accept`, we indicate `(g)` functions more as a reminder of the acceptance than as the acceptance itself (see also the convention above about multiple acceptances).

```
S1:     (a) How about your green table and some cheap chairs?
S2:     (b) I have 6 green chairs for 100
        (c) or 2 chairs for 50.
        (d) What do you think?
S1:     (e) Let's take the 50 ones
        (f) and I have two also.
        (g) I just put in your table.
```

7.3 Understanding

This aspect concerns the actions that speakers take in order to make sure that they are understanding each other as the conversation proceeds. There are many levels of "understanding", ranging from merely hearing the words to fully identifying the speaker's intention. Here we group most of these levels together so that if the hearer is said to have understood the speaker, then the hearer knows what the speaker meant by the utterance.

7.3.1 Signal-Non-Understanding

Utterances that explicitly indicate a problem in understanding the antecedent are labeled as *Signal-non-understanding (SNU)*.

Test: the coder should be able to roughly paraphrase a Signal-non-understanding utterance as "What did you say/mean?".

```
        A:    Take the train to Dansville
SNU     B1:  Huh?  (i.e., What did you say?)
SNU     B2:  What did you say?  (i.e., What did you say?)
```

```
SNU     B3:  to Dansville?  (i.e., What did you say?)
SNU     B4:  did you say Dansville?  (i.e., What did you say?)
SNU     B5:  Dansville, New York?  (i.e., What did you mean?)
SNU     B6:  Which train?  (i.e., What did you mean?)
```

On the other hand, responses that query how to comply with the speaker's request/proposal, or that question its desirability ("why are we doing this") are marked as `Hold` at the Agreement level and are not marked as SNU. Clarification questions labeled as `Hold` involve acquiring additional information about how or why something was requested or proposed, and do not signal misunderstanding. B_7 through B_9 provide examples.

```
             A: (a) Take the train to Dansville
Hold(a)     B7:       through Avon?  (i.e, How shall we take the train)
Hold(a)     B8:       to get the oranges?  (i.e., Why are we taking the train)
Hold(a)     B9:       Should it leave immediately?  (i.e., When should we take the train)
```

Note. We expect that SNU's of the *What did you mean?* type will often be marked as `ClarificationRequest` as well.

Discussion.[9] The reader may be puzzled by the fact that SNU is used also when an utterance has been heard, but the hearer has not recovered its meaning. The problem is "where to draw the line" in order to say that the hearer has "recovered" the explicit semantic content of the antecedent. Possible criteria are:

1. Has the hearer fully identified the speaker's intention in uttering the antecedent?
2. Has the hearer correctly identified the senses of the lexical items and the roles that constituents play in the semantic structure? Under this proposal, recovering the semantic content implies that no lexical or syntactic ambiguities remain, but does *not* imply that the hearer was able to resolve referents.

   Using this criterion, B_1–B_4 would be marked as **signal non-understanding**, whereas B_7-B_9 implicitly qualify as **signal understanding**. However, B_5 and B_6 do not provide any evidence of non-understanding: the hearer has heard the correct words, but can't perform reference resolution. Thus, B_5 and B_6 are not SNU's under this criterion.
3. The acceptance test: Could an indication of acceptance have been included in the response? For example:

   ```
   S1:  (a) Take some oranges to Dansville.
   S2:  (b) OK I will.  Should it leave immediately?
   ```

   sounds fine, whereas

   ```
   S1:  (a) Take some oranges to Dansville.
   S2:  (b) OK I will.  Dansville New York?
   ```

   sounds odd.

   Using this criterion, B_1–B_6 above are all marked as **signal non-understanding**, whereas B_7-B_9 still qualify as **signal understanding**.

Criterion 3 is adopted because Criterion 1 requires too much subjective reasoning about intentions and the recognition of intentions and Criterion 2 is difficult to explain without resorting to terms like "reference resolution" and "cospecification". Criterion 3 seems to have a simple test, and thus, it makes it easy to write clear instructions that enable coders, who are not necessarily trained in linguistics, to reliably code the data.

[9] The following discussion is taken from Johanna Moore's report from the Dagstuhl meeting.

7.3.2 Signal-Understanding

Utterances that explicitly signal understanding are marked with a Signal-understanding tag. Note that any utterance that doesn't explicitly indicate non-understanding implicitly indicates understanding. It is not necessary to mark such cases. Rather, there are some specific mechanisms used to explicitly signal understanding that we are interested in. In many cases, such utterances may also count as `Accept` acts at the Agreement level. However, the examples below are Signal-understanding utterances (acknowledgments) that are not acceptances.

Acknowledgments are utterances consisting of short phrases such as "okay", "yes", and "uh-huh", that signal that the previous utterance was understood without necessarily signaling acceptance, as in b:

```
              S1: (a)  Take the Avon train to Dansville
Ack(a)        S2: (b)  Okay
Hold(a)           (c)  But wouldn't using the Bath train be faster?
```

Acknowledgments may also interrupt or overlap with the other speaker's sentence. Given that in COCONUT the interaction is written and turn taking is strictly enforced, these kinds of acknowledgements do not occur.

RepeatRephrase. The `RepeatRephrase` tag is used for utterances that repeat or paraphrase what was just said in order to signal that the speaker has been understood. Like acknowledgments, a `RepeatRephrase` does not necessarily make any further commitment as to whether the responder agrees with or believes the antecedent. We do not expect `RepeatRephrase` to often occur in COCONUT.

```
                      S1: (a) do you need the bananas [in boxcars](1) at Bath
RepeatRephrase(a)     S2:         [the bananas](1)
```

CorrectMisspeaking. The `CorrectMisspeaking` tag is used for utterances that offer a correction and indicate that the hearer believes that the speaker has not said what he or she actually intended. In the example below, *S1* misspeaks by saying "engine E" instead of "engine E one" and *S2* offers a correction.

```
                    S1: (a)   so we should move the engine at Avon engine E to
CorrMisspeak(a)     S2: (b)   engine E one
Accept(b)           S1: (c)   E one to Bath
```

This category only applies to cases where another speaker makes a correction. If *S1* had corrected himself then no `CorrectMisspeaking` label would be applied to this section of the dialogue. There is currently no dimension in this scheme for annotating such speech repairs. We don't expect this tag to occur in the COCONUT corpus.

CorrectAssumption. In the previous version of DAMSL, `Correction` was used to tag corrections of either misspeaking or wrong assumptions. While DAMSL now only codes for corrections at the speaking level, we provide a tag to code for corrections at the semantic level as well. `CorrectAssumption` should, however, only be used when there is direct evidence for the wrong assumption. (i) below is marked as

CorrectAssumption. Note that (i) is a self-correction: we consider this acceptable for CorrectAssumption (as opposed to CorrectMisspeaking, that does not allow for self-corrections).

```
S1: (a) I have a lamp floor (blue), a green table and 4 chairs for 100.
S2: (b) The lamp and table sound good,
    (c) but the chairs seem expensive.
    (d) I have 4 blue for 100
    (e) they should look OK together
S1: (f) again that sounds good.
    (g) So the chairs are 25 a piece?
S2: (h) I goofed
    (i) the chairs are 100 each.
```

7.4 Answer

The Answer aspect is simply a binary dimension where utterances can be marked as complying with an Info-Request action in the antecedent. A standard question answer pair is shown here:

```
Info-Request          S1: (a)   can I take oranges um on tankers from Corning
Assert, Answer(a)     S2: (b)   no you may not they must be in boxcars
```

Test. An Answer can occur only if its antecedent is an Info-Request.
Rule of thumb. An Info-Request and its Answer will generally share the same *InformationLevel*.

Most questions are answered with one or more declarative sentences although it is possible to answer a question with an imperative as shown in the direction giving example below. The answer is an Open-Option because it describes one option for *S1*'s future action, but there is no indication that *S2* is trying to get *S1* to do it.[10]

```
Info-Request               S1: (a)   How do I get to Corning?
Open-Option, Answer(a)     S2: (b)   Go via Bath.
```

Sometimes a speaker will answer an implicit or indirect question. The general rule of thumb is to consider whether the implicit or indirect question was obvious enough to lead you to make the antecedent an Info-Request. The example below is similar to the previous one except that *S1* has not asked a question and we would not mark (a) as an Info-Request. Thus *S2*'s utterance is an Action-Directive, and is not considered an Answer that gives information and makes a suggestion.

```
Assert               S1: (a)   I need to get the train to Corning.
Action-Directive     S2: (b)   Go via Bath.
```

Of course there are borderline cases where these distinctions are a matter of degree. For instance, if *S1* says "I don't know how to get oranges to Corning", in many contexts this strongly suggests an implicit question and implies that a response like "You could get them from Bath" is an Answer. These difficult cases are left

[10] Our analysis of this example is different from DAMSL's. In DAMSL, this imperative answer is also marked as an Assert, as it also provides information. However, syntactic imperatives do not pass out Statement test.

to the coder's intuition, but the two utterances should be annotated consistently, so that if the second is an `Answer`, the first is an `Info-Request`.

Consider an unambiguous case where `(b)` is not marked as `Info-Request`, and hence, `c` is not marked as `Answer`. Here S1 is trying to avoid going to a meeting, but S2 does not cooperate.

```
S1: (a)   I should be at the meeting.
    (b)   Luckily, I don't know what time it is.
S2: (c)   It's 3 o'clock.
```

Note that clarification requests and other utterances that refuse to answer an `Info-Request` are not considered answers. For instance, in the following dialogue fragment *S2* states that he doesn't know the answer (and hence can't answer the question). Thus, `(b)` is treated as a `Reject`, just as in cases for non-information requests as in the next dialogue fragment:

```
Info-Request          S1:  (a)  How can I get oranges to Corning?
Assert, Reject(a)     S2:  (b)  I don't know.

Action-Directive      S1:  (a)  Please open the door
Assert, Reject(a)     S2:  (b)  I can't, my arm is broken.
```

It is also often the case that a question is answered by more than one utterance. In these cases the coder should try to determine whether one of the utterances functions as the direct response and the others as elaborations. If this is the case, only the direct response should be tagged as an Answer. For example, in the following, `(b)` is the Answer and `(c)` an elaboration.

```
Info-Request      S1: (a) Do you have any high tables?
Answer(a)         S2: (b) Yes I do,
ElaborateItem         (c) A high table blue, green, red, at 400, 200, 400 respectively.
```

The same principle should also be applied for some other backward function, such as when `Accept` is expressed my multiple utterances.

Rule-of-thumb. To test whether an utterance can function by itself as `Answer` (or as `Accept`), substitute the elaboration with something that e.g. changes the item in focus, like

```
(c') but let's think about sofas for now
```

in the example just discussed. `(b)` still functions as an answer in this revised context, so only `(b)` in the original example should be tagged as an answer.

Finally, it is possible that an `Answer` also functions as an `Accept` (or `Reject`). For example, in the situation in which only $50 is left, (b) can be taken as an `Accept`:

```
Info-Request      S1: (a) Do you have a rug or something blue?
Answer, Accept    S2: (b) yes, I do have a blue rug for  $50
```

Some of these ambiguities may be solved by looking at the graphics, see Sec 13. For example, if S2 updates his own display after uttering (b) and the graphics indicates that the blue rug has become part of the solution, we can safely conclude that (b) is both an `Answer` and an `Accept`. The fact that we use the tag `Accept` even if the antecedent does not qualify as a potential action may be puzzling but it is possible that the acceptance is for an inferred action. Note that `(a)` does not qualify as a potential action because the existence presupposition is not satisfied, as discussed in Section 4.5.1.

7.5 Informational Relations

Informational relations are used mostly to capture information about budget and about specific furniture items. We provide only the following three:

- *Act:Condition*. *Act* depends on a *Condition*, namely, *Condition* is a necessary condition (precondition) for *Act*. The *Condition* may be either a state or another act. *Condition* may already hold, or (be hypothesized to) hold in the future.
- *Act:Consequence*. *Act* can be expressed as an action or a state, i.e., it refers either to a previous action that had an effect or to the effect directly. *Consequence* refers to necessary consequences. *Act:Consequence* is used especially for budgetary consequences or for including / excluding certain furniture items from the solution.
  Note that to use this relation there must be an *Act*, even if expressed through its effects as a state. (d) below should not be tagged as `Act:Consequence`, but as *otherInfo*, as there is no *Act* here, only initial conditions:

  ```
  S1: (a)  I have got 550.
      (b)  How much have you got
  S2: (c)  I have 450
  S1: (d)  That gives us in total 1000
  ```

- *otherInfo*: catch all label for all the rest. It is used when the coder has a strong intuition that utterances are related at the informational level, as in the preceding example, and neither *Act:Condition* nor *Act:Consequence* apply.

Both *Act:Condition* and *Act:Consequence* point to another menu, that includes the two values *LabelOrder* and *ReverseOrder*. These are used to mark whether the two related utterances appear in the order mentioned in the label or not; i.e. *LabelOrder* is used when *Act* precedes *Condition/Consequence*, *ReverseOrder* is used for the reverse order.

Convention. As mentioned, informational relations are used mostly to capture information about budget and about specific furniture items; (d) below is tagged as < `act:consequence LabelOrder (c)` >

```
S1: (a)  we spent: 300 (sofa), 250 (lamp), 200 (table), and 300 on chairs.
    (b)  I had 550.
    (c)  We overspent $50.
    (d)  We will have to forget about the lamp.
```

They should not be taken to have intentional content, i.e. they should not be used to capture relations such as 'support' or 'evidence', which are not annotated for in Coconut. We are not trying to capture how

arguments are constructed. In the example above, the fact that (a) and (b) are support for (c) is not going to be annotated. The constraint that *Condition* must be a precondition to qualify as *Act:Condition* is meant to help the coder exclude these other informational relations.

7.6 Coreference / Set Relations

We provide three tags that allow us to capture a limited notion of coreferentiality: *sameItem*, *subset*, and *MutExclusive*.

- *sameItem*. It is used when an utterance discusses the same item as its antecedent. We also use it when the utterance discusses the same type of items as its antecedent, but no *MutExclusive* relation obtains. For example, `(c)` below is tagged `<sameItem (b)>` (here `(b)` is interpreted as referring to a type rather than to the specific red chairs in `(a)`, otherwise `(c)` couldn't refer to `(b)`):

  ```
  S1: (a) Ok, I have a red table and two chairs in DR.
  S2: (b) Red chairs in dining room.
      (c) Do you have two chairs red to go in dining room?
  ```

  Thus, *sameItem* is not strictly a coreference relation. However, if an utterance has two possible antecedents, one discussing the same item and the other the same type, it should be linked according to strict coreferentiality. If the antecedent discusses more than one item, to be tagged as *sameItem* the current utterance must discuss exactly the same items, otherwise the tag *subset* should be used — see below.
  Note. We allow a specific item to refer back to a type, but we don't allow a type to refer back to a specific item. We do allow *sameItem* to link two utterances that discuss the same type of item but only when the type is exactly the same. For instance, an utterance discussing red chairs cannot be linked to an utterance discussing cheap chairs.
  Note. *sameItem* is different from *elaborateItem*, discussed above under the *Topic* menu. *sameItem* does not mean that the utterance provides more information about a certain item, only that it is about the same item as its antecedent. *elaborateItem* and *sameItem* will certainly co-occur, but this is not necessary.

- *subset*. We use this label when the current utterance refers to a subset of the items discussed in its antecedent. For example:

  ```
  S1: (a)  I have a red table for 200, a blue one for 300, and a yellow one for 400.
  S2: (b)  Let's go with your red table.
  ```

  `(b)` will be tagged as `< subset (a)>`.

  Note: Accept-Part/Reject-Part and subset. Since we explicitly code for `subset` as part of our *coreferential* relations, `Accept-Part/Reject-Part` usually co-occur with `<subset>`. For example, `(b)` will be tagged as `<Accept-Part (a)>` and `<subset (a)>`, `(c)` as `<Reject-Part (a)>` and `<subset (a)>`.

  ```
  S1: (a) I have a lamp floor (blue), a green table and 4 chairs for 100.
  S2: (b) The lamp and table sound good,
      (c) but the chairs seem expensive.
  ```

There are few cases in which `Accept-Part/Reject-Part` occur, but `subset` doesn't. This happens when the accepted or rejected part of the proposal is not an item, but an item feature or a room, as in the example below:

```
                    S1: (a) How about my red sofa for the living room for $400?
<Accept-Part (a)>   S2: (b) A red sofa for the living room sounds good,
<Reject-Part (a)>       (c) but I have a cheaper one for $300.
```

In `(b)`, S2 accepts the color, type and room, but in `(c)` s/he rejects the specific item.

- *Mut(ually)Exclusive*. It is not really a *coreferential* relation, rather, it is a *set* relation. *MutExclusive* is the relation linking two propositions p_1 and p_2, where p_1 mentions a set of items $\mathcal{S}_1$, and p_2 provides an alternative $\mathcal{S}_2$ to that same set of items. Note that *MutExclusive* holds when $\mathcal{S}_1$ and $\mathcal{S}_2$ are mutually exclusive, not when there are several open options that are not mutually exclusive, and one is selected. Examples:

  1. (b) — spoken by a different speaker than (a) — is tagged <`Reject (a)`> and <`MutExclusive (a)`>.

     ```
     S1: (a) Let's get a blue sofa
     S2: (b) No, let's get a yellow table.
     ```

     Note that `(b)` in this (constructed) example constitutes an alternative only when the blue sofa and the yellow table are mutually exclusive: this will happen when the budget is too tight to allow the participants to satisfy all the priority item constraints (get a sofa for living room and a table and four chairs for dining room).

  2. (a) and (b) are spoken by the same speaker S_1, after S_2 has mentioned 4 other chairs. (b) is tagged as <`Initiate` > and <`MutExclusive (a)`>:

     ```
     S1. (a) ... but the chairs seem expensive.
         (b) I have 4 blue for 100.
     ```

  3. (a) and (c) are spoken by different speakers. (c) is tagged as <`Accept (a)`> and <`MutExclusive (a)`>:

     ```
     S1: (a) I have a blue sofa for $300.
         (b) What do you think?
     S2: (c) Sure, I don't have a sofa for a better price.
     ```

  4. (a) and (b) are spoken by the same speaker. (b) is tagged as <`Initiate` > and <`MutExclusive (a)`> — this example shows that the label `MutExclusive` can be used also when a certain item has been already explicitly rejected, as the lamp is in (a):

     ```
     S1: (a) We will have to forget about the lamp.
         (b) Do you have a rug or something that is blue?
     ```

  Note that *MutExclusive* is purely a *set* relation, and does not constitute per se a counterproposal. In fact, we want to explore whether *MutExclusive* is one of the characterizing features of (certain) counterproposals.

Note: Antecedents. Coreferential/set relations can apply to both explicit and elliptical referents. The two excerpts below illustrate coreference where the linked referent is elliptical. The material in square brackets indicates ellipsis.

```
              S1: (a) Do you have any high tables?
sameItem(a)   S2: (b) Yes, I do [have high tables].

                  S1: (c) I have 6 chairs for 150
                      (d) or 2 for 50.
sameItem(c),(d)       (e) What do you think [of these]?
```

However, the elliptical argument must be specific in a prior utterance; the antecedent cannot be something derived from the context. Also, be careful in distinguishing elliptical arguments from implicit ones, to which coreference/set relations do not apply. In the following two excerpts, `(b)` exemplifies ellipsis with a specific argument, and `(b')` non ellipsis with an implicit argument:

```
S1: (a)  Who ate the cake?
S2: (b)  John did.

S1: (a') Why is the table set?
S2: (b') John ate.
```

Comparisons are elliptical.
Note: Multiple Labels. It is possible that a single utterance refers to several items, which are contained in different utterances. If the potential multiple labels are all the same, then a single label with appropriate, multiple `Links` will suffice. Instead, multiple coreference labels must be assigned if different labels, such as *sameItem* and *subset*, apply. In the dialogue excerpt below, (e) is tagged <`sameItem (d)` > — linking to (d), the most recent explicit reference to S2's lamp – and <`subset (a)`> — linking to (a), the most recent explicit reference to S1's green chairs:

```
S1: (a) I have 1 blue floor lamp 50, 2 green and 2 red chairs both for 100 each,
        also a red rug for 200.
S2: (b) i dont think i have anything that would match any of that in our budget.
    (c) the only thing I have that could work is a floor lamp for 150.
    (d) it is green.
    (e) we could use it and one of your green chairs.
```

7.7 Link

The `Answer` tag and all the tags under the `Agreement, Understanding, Informational` and `Coreference` menus point to the *Link* menu. The link between `Informational` and `Link` is via *LabelOrder / ReverseOrder*.

The *Link* menu is just a list of numbers (1 through 9), and it refers to the label for p_1, where p_1 contains the closest antecedent for p_2. As mentioned above, only explicit and elliptical, not implicit, antecedents count. The value for *Link* will often need to be edited.

Convention. p_2 may actually be linked to more than one p_1; in this case, if it is clear what the multiple links are, we edit the link value by adding all the utterances the one being tagged is linked to, starting with the closest one.
Example:

```
S1: (a) i do have a lamp-floor, blue (250).
```

```
    (b) i also have a green table (200).
    (c) (d) (e)

S2: (f) (g) (h)
    (i) the lamp and table sound good
```

(i) is tagged as < `sameItem (b), (a)` >: `(b)` is the closest utterance where the table is mentioned, and `(a)` is the closest utterance where the lamp is mentioned.

Convention. Even if the coder has recognized the existence of an *informational* or of a *coreference* relation between the current utterance and the previous context, it is not always possible to mark *Link* explicitly: this happens when the relation between the current utterance and the previous ones is too complex, e.g. if the current utterance is a summary of many things previously discussed. No value for *Link* will be provided then.

8 Segment Tag

The *link* menu just discussed captures links between a single utterance and one or more individual previous utterance. Sometimes, though, a group of utterances functions as a whole, i.e., as a segment. While other coding schemes, in particular RDA [Moser *et al.*, 1996] and HATT [Nakatani *et al.*, 1995] are based on complex notions of segmentation, we use the *segment tag* in the same way as in DAMSL, to mark portions of the dialogue motivated by one single `Backward` function. Thus, we only allow the `Agreement` tags and the `Answer` tag to motivate a segment. As a consequence, either *response* or *antecedent* segments may arise.

N consecutive utterances within a single turn will form a *Response* segment only in the following two cases:

1. if the same type of `Agreement` tag or the `Answer` tag applies to each of the n utterances; or
2. if no single utterance performs the backward function of `Agreement` or `Answer` by itself. E.g., if the answer to the question *What tables do you have?* indicates 1 item per utterance when there are multiple tables, then no single utterance can count as the answer.

N consecutive utterances within a single turn will form an *Antecedent* segment, if the `Backward` function associated with a subsequent utterance or segment requires those n contiguous utterances as antecedent.

Note. In our current scheme, no other functions or dependencies are grounds for segmentation. In particular, neither two members of an informational pair, nor two utterances linked by syntactic dependence or by coreference constitute a segment. For example, in the following excerpt `(a)` constitutes the *Act* and `(b)` constitutes the *Consequence* of an *Act:Consequence* relation, and moreover, `(b)` is syntactically dependent on `(a)`: neither reason warrants tagging `(a)` and `(b)` together as a segment.

```
S1:  (a) let's do the yellow/blue mix,
     (b) leaving us with 250.
```

Convention. When coding a *response* segment, only the backward functions should be coded at the segment level, while all other tags will be at the utterance level. When coding an *antecedent* segment, no tag other than `Segment` should apply to the whole segment. Also, when a segment is both an *antecedent* and a *response*

segment, it should be doubly coded with the `Segment` tag, and a comment should be added to make it clear which tag refers to the segment as an *antecedent*, and which to the segment as a *response*.

Even if the backward function of a *response* segment is coded at the segment level, utterances internal to the segment can still have further backward functions. For example, in the following `(c)`, `(d)` and `(e)` constitute a response to `(b)` and thus share a single `Answer` tag. However, `(d)` is also an `Answer` to `(a)`, and therefore it is individually linked to `(a)`.

```
S1:   (a) I forget what you had.
      (b) what do we have left if anything?

S2:  <Segment_1><Backward_1="Answer (b)">
      (c) we spent: 300 (sofa), 250 (lamp), 200 (table), and 300 on chairs.
      (d) <Backward_2="Answer (a)">
          I had 550.  </Backward_2>
      (e) We over spent $50.
    </Segment_1></Backward_1>
```

Convention. It is not necessary to tag utterances internal to a *response* segment as `Initiate`, when they only carry the segment `Backward` function.

Convention. When we link an utterance (or segment) back to an *Antecedent* segment, we need to indicate that *Link* points to a segment. As there are no absolute segment labels, we'll use the segment position, relative to its response. To do so, we edit the Link values as follows: we add *seg* to the appropriate number, as in <Backward_2="Answer 1seg">. The back pointer skips over single utterances and counts segments only: namely, `1seg` refers to the first preceding segment, independently of any intervening utterances between the current utterance and the segment.

9 Fragment

Fragment is a top level tag which is used for untensed clauses. If the utterance is a cue word such *ok* or *well*, it is marked only as a Fragment and no other Surface-Features are specified. In the other cases, a Fragment is coded for all the surface features that apply. For example, in the following the fact that (c) is a question should be captured by tagging it as *questionWH* even though it is a Fragment:

```
S1: (a) I have two [chairs] also
    (b) mine are green and red
    (c) which?
```

Similarly, an utterance such as *How about my red chairs?* should be annotated as a *howAbout* at the modal dimension and as a *questionY/N* at the mood dimension even though it has no other Surface-Features.

Given that some of our Surface-Features menus are embedded, if the coder needs to code for an embedded feature, s/he should go through the appropriate `no-X` or `other-X` values at the higher level, but should stop as soon as no other embedded features are applicable.

As regards tags other than `Surface-Features`, there are no clear guidelines, but the coder should code only for those tags that don't result in a *nil* or *catch-all* label. So for example a particle such as `OK` in the

appropriate context is tagged as `Accept` at the `Backward` level, and as `Task` at the `Information-level`; however, it should not be tagged at the other levels, as the corresponding tags are *nil* or *catch-all*.

10 Summary

A summary of the items purchased often occurs at the end of a dialogue. These are annotated with the label *Summary* and marked as a segment if they are longer than one utterance. No further coding is performed on summaries.

11 Comment

This tag is used to record notes, doubts etc. In fact, whenever the coder is not sure about a particular coding decision, s/he should make a comment about it, i.e. a note about the ambiguity or the alternatives that one cannot decide between.

12 Non-Relevant

If the utterance is not relevant to the coding purposes, it is tagged as such, and no further tagging is performed on it. For example, in the Coconut domain, all comments of a social nature (e.g. *let's have some fun*, *My Mom hit the lottery*, *Which year are you?*) are marked as *NonRelevant*, and no further tagging is performed on them.
Note. In the current version of DAMSL, the tag `NonRelevant` has been discarded. It used to be included under *Communicative-Status* (DAMSL sec. 2.4), together with tags such as `Uninterpretable, Self-Talk`, etc. We don't include the aspect *Communicative-Status* in this manual, as tags such as `Uninterpretable` or `Self-Talk` don't occur in our corpus.

13 Miscellaneous Conventions

- **Missing values.** At any level, if the coder wants to code for a certain feature X and has a specific value Y in mind, but there is neither a tag that applies, nor a "catch-all" tag of the type `otherValue` is provided, the coder must leave the feature empty, and possibly add a comment. This will mainly happen with the `Topic` tags. For example, none of our listed topics applies to the following utterance, so it is coded as follows:

  ```
  <Topic_17><Comment_1 Topic=Room>
  (a) so let's begin in the dining room.
  </Topic_17></Comment_1>
  ```

  Note that the previous coding does not mean that `(a)` has no *topic*, but that it has a *Topic* whose value is left undefined. Utterances with no *Topic*, such as fragments like *OK*, will not have *Topic* among their tags at all.

- **Using the graphics.** The coder can use the graphics if it helps disambiguate; eg. for coreference, for understanding speaker's intentions and effects on conversation, etc. In general, not too much should be expected from the graphics, as subjects were not too consistent in using it.

14 File format

We preprocess the files to be tagged by defining the minimal units for tagging; each will correspond to a single line. We follow Passonneau's approach [Passonneau, 1994] in segmenting the dialogue into *functionally independent clauses*. Passonneau's criteria to identify 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, non restrictive 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 coder 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 cut, 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.

Acknowledgements

This work is supported by the National Science Foundation under Grant No. IRI–9314961, "Integrated techniques for natural language generation and interpretation."
We also acknowledge all the researchers who actively participate in DRI, and whose efforts have shaped the DAMSL manual. Among them, a special thanks to Johanna D. Moore for feedback on this manual, to David Traum for many discussions, electronic and otherwise, and to Mark Core and James Allen for assembling the DAMSL manual, and making it available to us.

References

[Allen and Core, 1997] James Allen and Mark Core. Draft of DAMSL: Dialog act markup in several layers. Available from http://www.georgetown.edu/luperfoy/Discourse-Treebank/dri-home.html, under *Tools and resources*, 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.

[Moser *et al.*, 1996] Megan Moser, Johanna D. Moore, and Erin Glendening. Instructions for Coding Explanations: Identifying Segments, Relations and Minimal Units. Technical Report 96-17, University of Pittsburgh, Department of Computer Science, 1996.

[Nakatani *et al.*, 1995] Christine H. Nakatani, Barbara J. Grosz, David D. Hahn, and Julia Hirschberg. Instructions for annotating discourses. Technical Report TR-25-95, Harvard University Center for Research in Computing Technology, 1995.

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

[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.