An Extension on RSA Model: Lexical Uncertainty Model

Yuxing Chen
Symbolic Systems Program
Stanford University
yxchen28@stanford.edu

1 Introduction

In this project, our focus is on the understanding of utterance and how we can construct computational models to formalize the inferences about meaning in context as well as the reasoning about the two rational agents, the pragmatic speaker and the pragmatic listener, which will be defined later. Our guiding idea is the Rational Speech Acts (RSA) model proposed in [3]. More specifically, in this project, I'm going to explore and implement an extension of our current RSA model, focusing on the study of the lexical uncertainty associated with embedded implicatures.

Hypothesis and Related Concepts

The RSA model and its extensions have shown their ability to reason and predict the understanding/interpretation of utterances that contain conversational implicatures, following Grice's theory. However, problems for the basic RSA model arise when we encounter embedded implicatures (e.g. embedded *some*), where the contexts could be more complicated than those in the examples we previously analyzed. The reason is that this type of implicatures introduce lexical uncertainty to the RSA model. The central hypothesis of this paper is that we can improve the basic RSA model to make it able to work with embedded implicatures as well, by implementing a compositional lexical uncertainty model based on the work of Bergen *et al*. and Potts *et al*. [1, 2, 6], where we are not presuming the participants to share a "single, fixed lexicon mapping word forms to meanings". In this project, I'm going to implement this extension.

RSA model is a quantitative framework for pragmatic reasoning, built on probabilistic models and consistent with the conversational maxims proposed by Grice [4]. With the RSA model, we can make quantitative predictions on the human responses of a variety of utterances. In this context, the **literal listener** uses the literal meaning of the utterance. **Pragmatic speaker** wants to deliver a specific meaning to the listener and reasons about how the listener would interpret the messages. **Pragmatic listener**, on the other hand, wants to figure out the meaning the speaker intended to express and therefore reasons about which meaning would make the speaker to say it in that way. **Embedded implicatures**, as proposed in [7], are implicatures "that arise locally, at a sub-locutionary level, without resulting from an inference in the narrow sense." In [6], Potts *et al*. also informally state that embedded implicatures are cases where "a pragmatically enriched interpretation seems to be incorporated into the compositional semantics". **Lexical uncertainty** occurs when the discourse participants are not restricted to use a single and fixed lexicon mapping from words to meanings. Instead, there are multiple readings for the embedded implicature.

This extension of the RSA model allows us to make fewer assumptions on the discourse participants and hence becomes more applicable in reasoning and solving problems coming from real world scenario.

Proposed Steps and Expectations

The first step is to study the examples where embedded implicatures challenge our basic RSA model. The next step is to understand the mechanism of the model proposed by Bergen *et al*. [1]. Based on the basic RSA model, I'll complete the implementation of the lexical uncertainty RSA model, which

is an extension of the RSA model, allowing lexical uncertainty cases. The design of the model should follow the architecture composed in [1, 6]. Once the new model are able to work, I'm going to test it using examples where lexical uncertainties are introduced. Besides, I'll compare my experiment results with the statistics given in the paper [6]. My expectation is to implement the models composed in [1, 6]. The extensions should outperform the basic RSA model in utterances where there are embedded implicatures. It would be ideal if the experiment results are close to those stated in the paper.

2 Model Description

2.1 Mathematical background

Before diving into the mathematical formulae for the extension, we first recall the fundamental RSA model as proposed in [3]:

(1) Define the reference game as $(R, M, [\![\cdot]\!], P, C)$ where
   a. R is the set of states (i.e. $r_1, r_2, \cdots \in R$, where r_i represent a state).
   b. M is the set of messages (or utterances) (i.e. $m_1, m_2, \cdots \in M$, where m_i represent an utterance).
   c. $[\![\cdot]\!]$ is a semantic interpretation function, mapping from meaning to $\{0, 1\}$.
   d. P is the prior probability distribution over states.
   e. C is the cost function, and here we define costs to be non-positive.

   and denote the pragmatic listener as L, the pragmatic speaker as S and the literal listener as L_0.

Then we use the following properties to build the RSA model:

(2) a. $P_{L_0}(r|m) \propto \delta_{[\![m]\!](r)} P(r)$: $P_{L_0}(r|m) = \frac{[\![m]\!](r) \cdot P(r)}{\sum_{r' \in R} [\![m]\!](r') \cdot P(r')}$
   b. $P_S(m|r) \propto e^{\alpha \cdot \log(P_{L_0}(r|m) + C(m))}$
   c. $P_L(r|m) \propto P_S(m|r) \cdot P(r)$

Now back to the design of the RSA extension that allows the occurrence of lexical uncertainties, intuitively, we need a new set of parameters to make our model able to carry more information. The fundamentals of the lexical uncertainty RSA model are then as follows:

(3) The reference game $(R, M, [\![\cdot]\!], P, C, \mathcal{L}, P_{\mathcal{L}})$ is the basically the same as defined in (1), except that there are two more parameters $\mathcal{L}$ and $P_{\mathcal{L}}$ to interpret the reference game:
   a. $\mathcal{L}$ is the set of (equivalence classes of) possible lexica (i.e. $\mathcal{L}^1, \mathcal{L}^2, \cdots \in \mathcal{L}$, where $\mathcal{L}^i$ is a possible lexicon).
   b. $P_{\mathcal{L}}$ is the prior probability distribution over lexica, which is a uniform distribution over all $|\mathcal{L}|$ logically possible lexica.

With this new formation, we need to update the formulae as well, in which we condition not only on the message but also on the lexicon, allowing the agents to reason about distributions over different lexica:

(4) a. For the literal listener L_0, P_{L_0} remains unchanged (see (2)).
   b. $P_S(m|r, \mathcal{L}) \propto e^{\alpha \cdot \log(P_{L_0}(r|m, \mathcal{L}) + C(m))}$
   c. $P_L(r|m) \propto \sum_{\mathcal{L}} P_{\mathcal{L}} \cdot P(r) \cdot P_S(m|r, \mathcal{L})$

In fact, the two rational agents (speaker and listener) are modeling each other and this process can keep going until the listener stop reasoning about the speaker's intention [1]. We may then generalize the model to make it recursive to capture this behavior. The modification is straightforward:

(5) a. $P_{S_n}(m|r, \mathcal{L}) \propto e^{\alpha \cdot \log(P_{L_{n-1}}(r|m, \mathcal{L}) + C(m))}$

b. $P_{L_n}(r|m) \propto \sum_{\mathcal{L}} P_{\mathcal{L}} \cdot P(r) \cdot P_{S_{n-1}}(m|r, \mathcal{L})$

As n goes to infinity, we would expect the model to reach the fixed point, if there is any.

2.2 Code

The skeleton of the code is adapted from [5], which is the Python implementation of the basic RSA model as defined by 1 and 2. With this implementation as the starting point, I implemented the lexical uncertainty RSA model as an extension.

Implementation details

The new model is implemented in the way that is able to run the basic RSA model as well as the lexical uncertainty RSA model. To run the basic model, there is no additional action needed. If we want to apply the lexical uncertainty model, then we need to set `mode='bergen'` and feed data into `lexica` as well as `lexica_num`. The input of `lexica` is a list of dataframes. Each dataframe contains the truth values $\{0, 1\}$ with messages (`msgs`) along the rows and states/worlds (`states`) along the columns. There are `lexica_num` $\in \mathbb{Z}^+$ of such dataframe in each `lexica`. This number is exactly the cardinality of $\mathcal{L}$ defined in (3a), which represents the number of all possible lexica and can be computed ahead[1]. The rest of the model arguments are shared by the two models: we need to provide the (state) prior distribution P, the costs of messages C and pragmatic inference strength parameter α, storing in `prior`, `costs` and `alpha` respectively.

In the original code, `class RSA` takes in `lexicon`, `prior`, `costs` and `alpha`, and can then compute literal listener L_0 predictions P_{L_0} using the member function `literal_listener`. Likewise, we can get the pragmatic speaker predictions P_S and the pragmatic listener predictions P_L from `speaker` and `listener`, respectively.

To get the lexical uncertainty listener predictions, I introduce two new member functions to `class RSA`:

- `def bergen_lexical_uncertainty_listener(self):`
  This function reads in `self.lexica`, `self.lexica_prior` and `self.prior`, iteratively computes the literal listener predictions $P_{L_0}(r|m, \mathcal{L})$ using `literal_listener` and the (first level) pragmatic speaker predictions $P_{S_1}(m|r, \mathcal{L})$ using `speaker`, given a lexicon $\mathcal{L}$. Then it sums up the results from each iteration and calculates the (first level) lexical uncertainty listener predictions $P_{L_1}(r|m)$.
- `def run_lu_model(self):`
  This is the helper function to get the predictions from the lexical uncertainty RSA model. It calls `bergen_lexical_uncertainty_listener`, `speaker` and `listener` in a sequence. It returns a list of predictions: the first level lexical uncertainty listener predictions $P_{L_1}(r|m)$, the second level pragmatic speaker predictions $P_{S_2}(m|r, \mathcal{L})$ reasoning on $P_{L_1}(r|m)$, and the second level lexical uncertainty listener $P_{L_2}(r|m)$. In fact, we can keep this back-and-forth procedure to get higher level predictions.

3 Results

Before going into the examples, we first assume the following notations for the quantificational determiners:

(6) a. $[\![\text{no}]\!] = \lambda f(\lambda g(\text{T if } \{x : f(x) = \text{T}\} \cap \{x : g(x) = \text{T}\} = \varnothing, \text{else F}))$
b. $[\![\text{every}]\!] = \lambda f(\lambda g(\text{T if } \{x : f(x) = \text{T}\} \subseteq \{x : g(x) = \text{T}\}, \text{else F}))$
c. $[\![\text{(at least) some}]\!] = \lambda f(\lambda g(\text{T if } \{x : f(x) = \text{T}\} \cap \{x : g(x) = \text{T}\} \neq \varnothing, \text{else F}))$
d. $[\![\text{some but not all}]\!] = \lambda f(\lambda g(\text{T if } \{x : f(x) = \text{T}\} \cap \{x : g(x) = \text{T}\} \neq \varnothing \text{ and } \{x : f(x) = \text{T}\} \nsubseteq \{x : g(x) = \text{T}\}, \text{else F}))$

We then define the refinement sets for the lexical items that appear later in this section, following the notion in [2, 6]:

[1] We will discuss this in the next section in detail.

(7) a. Let φ be a set-denoting expression. R is a refinement of φ if and only if $R \neq \varnothing$ and $R \in [\![\varphi]\!]$.

b. $\mathcal{R}_c(\varphi)$, the set of refinements for φ in context c, is constrained so that $[\![\varphi]\!] \in \mathcal{R}_c(\varphi)$ and $\mathcal{R}_c(\varphi) \subseteq \wp([\![\varphi]\!]) - \varnothing$.

(8) Assuming there is only one person, Alice, in our context:

a. $\mathcal{R}_c(\text{Alice}) = \{[\![\text{Alice}]\!]\}$

b. $\mathcal{R}_c(\text{scored}) = \{[\![\text{scored}]\!], [\![\text{scored and didn't ace}]\!]\}$

c. $\mathcal{R}_c(\text{aced}) = \{[\![\text{aced}]\!]\}$

(9) Assuming there are only two people, Alice and Bob, in our context:

a. $\mathcal{R}_c(\text{Alice}) = \{[\![\text{Alice}]\!], [\![\text{only Alice}]\!]\}$

b. $\mathcal{R}_c(\text{Bob}) = \{[\![\text{Bob}]\!], [\![\text{only Bob}]\!]\}$

c. $\mathcal{R}_c(\text{some}) = \{[\![\text{(at least) some}]\!], [\![\text{some but not all}]\!]\}$

d. $\mathcal{R}_c(\text{every}) = \{[\![\text{every}]\!]\}$

e. $\mathcal{R}_c(\text{no}) = \{[\![\text{no}]\!]\}$

f. $\mathcal{R}_c(\text{scored}) = \{[\![\text{scored}]\!], [\![\text{scored and didn't ace}]\!]\}$

g. $\mathcal{R}_c(\text{aced}) = \{[\![\text{aced}]\!]\}$

Based on these assumptions, we can generate labels for possible states and find possible lexica. In (8), the possible states are `N` (no score at all), `S` (scored but not aced) and `A` (aced). Since the refinement set for 'scored' has two elements, there are two possible lexica: (i) When we say someone 'scored', we allows the case that 'he/she aced' to be true; (ii) When we say someone 'scored', we strictly rule out the case that 'he/she aced' (i.e. It is false). As mentioned before, we assume uniform prior distribution (P) and uniform lexica prior distribution ($P_{\mathcal{L}}$). Setting $C(0) = 5$ and $C(m) = 0$ for the other messages, we can replicate the results in [6] (rounded to the nearest hundredth):

M

	N	S	A
scored	0	1.00	1.00
aced	0	0	1.00
0	1.00	1.00	1.00

	N	S	A
scored	0	1.00	0
aced	0	0	1.00
0	1.00	1.00	1.00

	N	S	A
scored	0	0	1.00
aced	0	0	1.00
0	1.00	1.00	1.00

L_0

	N	S	A
scored	0	0.5	0.5
aced	0	0	1
0	0.33	0.33	0.33

	N	S	A
scored	0	1	0
aced	0	0	1
0	0.33	0.33	0.33

	N	S	A
scored	0	0	1
aced	0	0	1
0	0.33	0.33	0.33

S_1

	scored	aced	0
N	0	0	1.00
S	1.00	0	0.00
A	0.33	0.67	0.00

	scored	aced	0
N	0	0	1.00
S	1.00	0	0
A	0	1.00	0

	scored	aced	0
N	0	0	1.00
S	0	0	1.00
A	0.50	0.50	0.0

L_1 (lexical uncertainty listener)

	N	S	A
scored	0	0.71	0.29
aced	0	0	1.00
0	0.75	0.25	0.00

If we add one more pair of layers of reasoning, we can observe that the predictions start to converge to some fixed point (rounded to the nearest hundredth):

L_2

	N	S	A
scored	0	0.81	0.19
aced	0	0	1.00
0	1.00	0.00	0.00

L_3

	N	S	A
scored	0	0.86	0.14
aced	0	0	1.00
0	1.00	0.00	0.00

L_4

	N	S	A
scored	0	0.89	0.11
aced	0	0	1.00
0	1.00	0.00	0.00

In this way, the lexical uncertainty RSA model is able to predict that when the listener hears 'Alice scored', he or she assumes that the most possible state is `S` and that when the message is null, he or she assumes that the most possible state is `N`. We also observe that there are probabilities that are not equal to 1, which means the lexical uncertainty remains. However, if the rational agents keep reasoning on each other, eventually the probability of the most possible state will be close to 1, as

shown in the tables for L_2, L_3, L_4. Meanwhile, if we further assume the listener makes a categorical decision, then the model has no problem choosing the state that is of the highest predicted probability.

The next example (9) is more complicated. The semantic meanings of the quantificational determiners follow the assumptions stated in (6). Similar to what we did for (8), we first generate labels for possible states and possible lexica. The meaning of `N`, `S`, `A` are as defined previously. The label `NN` means that both Alice and Bob had no score at all, the label `SA` means that Alice scored but didn't ace and Bob aced, and etc. All 9 possible states are `NN NS NA SN SS SA AN AS AA`. Since each of the refinement sets for the person, 'some' and 'scored' has two elements, the total number of possible lexica is $2 \times 2 \times 2 = 8$ and they are formed by taking combinations of different elements from each refinement set (see Appendix for details). Again, we assume uniform prior distribution (P) and uniform prior distribution ($P_{\mathcal{L}}$). Setting $C(0) = 5$ and $C(m) = 0$ for the other messages, we get the following predictions for the first level lexical uncertainty listener (rounded up to the hundredth):

	NN	NS	NA	SN	SS	SA	AN	AS	AA
Alice scored.	0	0	0	0.42	0.19	0.22	0.12	0.03	0.02
Alice aced.	0	0	0	0	0	0	0.50	0.40	0.10
Bob scored.	0	0.43	0.12	0	0.10	0.04	0	0.29	0.02
Bob aced.	0	0	0.49	0	0	0.42	0	0	0.09
Some person scored.	0	0.28	0.09	0.28	0.06	0.09	0.09	0.09	0.02
Some person aced	0	0	0.25	0	0	0.22	0.25	0.21	0.06
Every person scored.	0	0	0	0	0.62	0.15	0	0.13	0.10
Every person aced.	0	0	0	0	0	0	0	0	1.00
No person scored.	0.65	0	0.15	0	0	0	0.15	0	0.04
No person aced.	0.28	0.27	0	0.27	0.18	0	0	0	0
0	0.15	0.14	0.11	0.14	0.10	0.10	0.11	0.10	0.07

We may also compute the second or higher level lexical uncertainty listener predictions. The pattern is similar to what we've observed in example (8): the predicted probabilities are converging to some fixed points.

The overall pattern of the lexical uncertainty listener predictions shown in the table above is consistent with Table 4 in [6], except for some numerical differences that may come from different parameter settings (i.e. costs C and number of layers). According to our predictions, when the listener hears 'Alice scored', he or she assumes that `SN`-'Alice scored but not aced and Bob didn't score at all' is the most possible intended meaning, which matches our intuition. When the listener hears 'Some person scored', our model suggests that he or she assumes that `NS`-'Alice didn't score at all and Bob scored but not aced' and `SN`-'Alice scored but not aced and Bob didn't socre at all' are the two most possible states and there should be no preference between these two states. This is also reasonable, because in real life, 'Alice scored' and 'Bob scored' contribute equivalently to the statement that 'Some person scored' and we don't actually care who scored.

We expected the predictions on 'Alice scored' and 'Bob scored' (and on 'Alice aced' and 'Bob aced') should be symmetric. However, the results I got (presented above) are not symmetric, even though they're very close to each other. My assumption is that I might make some mistakes when assigning the truth values to the entries.

4 Conclusion and Future Work

The lexicon uncertainty RSA model is able to play with cases where there are lexical uncertainties introduced by the embedded scalar implicatures, as we've shown in examples (8) and (9). In addition, if we keep this back-and-forth reasoning going, the predictions will converge to fixed points. We may apply this model to other examples as well. For example, we can make predictions on sentences that contain numbers: given three entities 1, 2, and 3, in different lexica, $[\![one]\!]$, $[\![two]\!]$, $[\![three]\!]$ might be interpreted differently [2]. We may also use this model to simulate the division of pragmatic labor generalization that is problematic for the basic RSA model.

Currently, it is time-consuming to come up with possible lexica and assign corresponding truth values to each `(lexicon, message, states)`-tuple. Meanwhile, it is easy to make mistakes when doing

this manually. Therefore, one potential improvement for this project would be writing the script to automatically generate possible lexica as well as possible states and then assign truth values to them. It is also very interesting to see if there is a way to construct a learned/neural RSA with lexical uncertainty and how good the results would be.

References

[1] Leon Bergen, Noah Goodman, and Roger Levy. That's what she (could have) said: How alternative utterances affect language use. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 34, 2012.

[2] Leon Bergen, Roger Levy, and Noah Goodman. Pragmatic reasoning through semantic inference. *Semantics and Pragmatics*, 9, 2016.

[3] Noah D Goodman and Michael C Frank. Pragmatic language interpretation as probabilistic inference. *Trends in cognitive sciences*, 20(11):818–829, 2016.

[4] H Paul Grice. Logic and conversation. *1975*, pages 41–58, 1975.

[5] Christopher Potts. Rsa implemented in python. https://web.stanford.edu/class/linguist130a/materials/rsa130a.py.

[6] Christopher Potts, Daniel Lassiter, Roger Levy, and Michael C Frank. Embedded implicatures as pragmatic inferences under compositional lexical uncertainty. *Journal of Semantics*, 33(4):755–802, 2016.

[7] François Recanati. Embedded implicatures. *Philosophical perspectives*, 2003.

A Example Lexicon for (9)

(10) Lexicon:

a. ⟦some⟧ is defined as 'at least some' (see (6c)).

b. ⟦scored⟧ means 'scored but didn't ace'.

c. ⟦Alice⟧ doesn't mean 'only Alice' (and same for ⟦Bob⟧).

Then the truth values should be:

	NN	NS	NA	SN	SS	SA	AN	AS	AA
Alice scored.	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
Alice aced.	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
Bob scored.	0.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0
Bob aced.	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
Some person scored.	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
Some person aced	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0
Every person scored.	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0
Every person aced.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
No person scored.	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
No person aced.	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0