Amortizing Pragmatic Program Synthesis with Rankings

Yewen Pu¹, Saujas Vaduguru², Priyan Vaithilingam³, Elena Glassman³, Daniel Fried²

¹ Autodesk Research, ² Carnegie Mellon University, ³ Harvard University

yewen.pu@autodesk.com, svadugur@cs.cmu.edu, pvaithilingam@g.harvard.edu
glassman@seas.harvard.edu, dfried@andrew.cmu.edu

Abstract

In program synthesis, an intelligent system takes in a set of user-generated examples and returns a program that is logically consistent with these examples. The usage of Rational Speech Acts (RSA) framework has been successful in building pragmatic program synthesizers that return programs which – in addition to being logically consistent – account for the fact that a user chooses their examples informatively. However, the computational burden of running the RSA algorithm has restricted the application of pragmatic program synthesis to domains with a small number of possible programs. This work presents a novel method of amortizing the RSA algorithm by leveraging a global pragmatic ranking – a single, total ordering of all the hypotheses. We prove that for a pragmatic synthesizer that uses a single demonstration, our global ranking method exactly replicates RSA’s ranked responses. We further empirically show that global rankings effectively approximate the full pragmatic synthesizer in an online, multi-demonstration setting. Experiments on two program synthesis domains using our pragmatic ranking method resulted in orders of magnitude of speed ups compared to the RSA synthesizer, while outperforming the standard, non-pragmatic synthesizer.

Introduction

For intelligent systems to be accessible to end users, it is important that they can infer the user’s intent under ambiguity. Imagine a person asking an AI assistant to generate a regular expression that matches the string (123)456-7890. It would be unhelpful if the AI assistant simply returned the regular expression $\Sigma^*$ – the expression that matches all strings – although it is technically correct. The rational speech acts model (RSA) of pragmatics (Frank and Goodman 2012) gives an algorithm for resolving ambiguities by modeling the user as choosing examples that are informative to the system, by using recursive Bayesian reasoning. Given several competing responses, for instance $regex_1 = 33-4$ and $regex_2 = \Sigma^*$, RSA would reason that it is more likely that an informative user would use the utterance “(123)456-7890” to describe $regex_1$ over $regex_2$, allowing it to prefer the intended regex. Recent works (Pu et al. 2020; Vaithilingam, Pu, and Glassman 2023) have leveraged the RSA algorithm to build pragmatic program synthesizers – interactive systems that take in user given examples (e.g. strings) and return programs (e.g. regexes) that are both logically consistent and take into account that the users tend to select informative examples.

A known limitation of the RSA algorithm is its inference speed – exact inference in RSA needs to marginalize across all possible examples (e.g. all strings) and hypotheses (e.g. all regexes) multiple times – making it difficult to scale to domains with a large number of utterances and hypotheses. This drawback is significant in building interactive systems, in which the users expect the system to respond in real-time. Prior works in scaling up RSA computation (Monroe et al. 2017; Andreas and Klein 2016) have largely focused on sampling and re-ranking, curbing RSA’s computation to a small subset of hypotheses and utterances. In this work, we show a simple yet effective way of amortizing RSA via a global ranking – a total ordering of hypotheses that is held constant across every possible set of examples. At training time, the expensive RSA algorithm is used to generate training data in the form of partial rankings, and the global ranking is fitted to be consistent with this partial ranking as much as possible, and cached for use at interaction time (Figure 1). Then, at testing / interaction time, the global ranking is used...
to disambiguate different hypotheses, without the expensive RSA overhead. For instance, the regex $\Sigma^*$ would be ranked very low in the global ranking, making it unlikely to be erroneously chosen compared to other hypotheses.

This work makes the following contributions. First, we prove that in the case of single demonstration RSA with boolean lexicons, studied in works such as (Mukherjee, Hawkins, and Fan 2019; Vogel et al. 2013; Monroe and Potts 2015; Smith, Goodman, and Frank 2013), there always exists a single global ranking that perfectly models the RSA algorithm. Second, we show that in the case of interactive RSA – where the user provides examples one after another – studied in (Cohn-Gordon, Goodman, and Potts 2018b; Pu et al. 2020; Vaithilingam, Pu, and Glassman 2023), a global ranking gives a close approximation for the RSA algorithm in practice. We show that ranking enables scaling (Vaithilingam, Pu, and Glassman 2023) to a larger regex domain, and we conduct a small user study which confirms that end-users can interact effectively with a ranking based program synthesizer. Further, we conduct a simulated user study by replaying the human interactive synthesis data from (Pu et al. 2020) and (Vaithilingam, Pu, and Glassman 2023), finding that our pragmatic ranking method resulted in orders of magnitudes of speed ups compared to the RSA synthesizer, while outperforming the non-pragmatic synthesizer.

**Background and Motivation**

In this section, we provide background on a reference game framework of program synthesis, which affords building a pragmatic synthesizer that can infer a user’s intended program from few examples (Pu et al. 2020). We illustrate this framework using a toy example from a small version of the regular expression domain of this work. We then give an informal overview of how one can use a global pragmatic order, the contribution of this paper, to make pragmatic synthesis scalable to more complex domains, such as unrestricted regular expressions.

### Synthesis as a Reference Game

Consider the problem where a user gives a few example strings to a synthesis system, and asks the synthesizer to find a regular expression that can match them. This process can be modeled as a reference game, where a speaker (the user) chooses a few utterances (strings) to give to the listener (the synthesizer), with the intention that the listener can infer the correct hypothesis (regular expression). This reference game is characterized by the lexicon $M$, a boolean matrix of 1s and 0s (Figure 2). In $M$, each row corresponds to an utterance and each column corresponds to a hypothesis, and 1s indicating consistency of its corresponding utterance and a hypothesis: whether the regular expression matches the string. As we can see, a given utterance (such as $001$) may be consistent with multiple hypotheses ($0^+1$, $0(2)^+$, and $0^+1$).

![Figure 2: A boolean lexicon for a small reference game of regular expressions. The rows are the utterances (strings) and the columns are hypotheses (regexes), and each entry denotes if a string is consistent with a regex.](image)

### A Literal Program Synthesizer

![Figure 3: Given the utterance $01$, a naive synthesizer $L_0$ will predict $0^+1$ and $0^+1$ as equally probable.](image)

How might we build a system that takes an utterance (say $01$) and produces the intended hypothesis $0^+1$? As $01$ is consistent with multiple hypotheses ($0^+1$ and $0^+1$), a naive strategy is to treat these two as equally likely. A synthesizer built this way is a literal listener $L_0$ (Bergen, Levy, and Goodman 2016), which we can construct by normalizing the rows of the matrix $M$, resulting in a probability distribution over hypotheses $W$ given utterances $u$.

$$L_0(u | w) = \frac{M[u, w]}{\sum_{w'} M[u, w']}$$

The result of this normalization is shown in Figure 3. As we can see, given the utterance $01$, this listener predicts an equal probability of $0^+1$ and $0^+1$ being the intended program. However, as Pu et al. (2020) find, end users do not communicate effectively with $L_0$, requiring longer interactions compared to a model that reasons about how users choose utterances informatively.

### A Pragmatic Program Synthesizer

A key insight to improving on the literal synthesizer is to consider that a user is cooperatively choosing an utterance to be informative about the intended program to the

---

1Note that in synthesizers for more complex domains, we cannot explicitly enumerate the entire lexicon $M$, which may be extremely large or infinite, and have to generate and score programs dynamically.
Reference Games with Multiple Utterances

So far, we have considered the problem of inferring a hypothesis based on the speaker producing a single utterance. However, in complex domains such as regular expressions, the users will have to clarify their intent interactively, by giving a sequence of utterances in multiple turns \( u = u_1, u_2, \ldots, u_n \). The synthesizer must infer the intended program after every turn. This is an instance of incremental RSA (Cohn-Gordon, Goodman, and Potts 2018b), which synthesizer. The Rational Speech Acts (RSA) framework models this informative choice of utterances using recursive Bayesian reasoning (Frank and Goodman 2012). By reasoning about why a speaker might have chosen a particular utterance (examples), rather than possible alternatives, the listener (synthesizer) can disambiguate the hypothesis (program) to which the speaker was referring. Formally, the RSA framework produces a chain of alternating listeners and speakers beginning with the \( L_0 \) model above, each of which reasons about the previous agent in the chain.

\[
S_1(u|w) = \frac{L_0(w|u)}{\sum_{u'} L_0(w|u')}
\]

\[
L_1(w|u) = \frac{S_1(u|w)}{\sum_{w'} S_1(u|w')}
\]

Applying this framework amounts to normalizing the columns of the \( L_0 \) matrix to obtain \( S_1 \), the normalize the rows of \( S_1 \) to obtain a pragmatic listener (synthesizer), \( L_1 \). The result is shown in Figure 4. As we can see, given the utterance \( O1 \), this listener prefers \( \theta^+1\{1\} \) over \( \theta^+1* \), reflecting the reasoning that if the user wanted to refer to \( \theta^+1* \), they might have provided an example that highlights the possibility of no 1s in the string.

The Issue: RSA is Slow

In computing \( L_1 \) using RSA, it needs at worst case \( O(|W|) \) calls to \( S_1 \). Each call to compute \( S_1 \) requires \( O(|U|) \) calls to \( L_0 \), which in turn requires \( O(|W|) \) operations to determine a set of consistent programs. In small domains, the results of each computation – \( L_0, S_1, L_1 \) – can be stored explicitly in matrix form, as we’ve shown in Figure 3 and Figure 4. However, in incremental domains with a large number of hypotheses and utterances, it becomes infeasible to cache all the computation results explicitly, forcing the \( L_1 \) listener to recompute from scratch. In practice, the pragmatic synthesizer \( L_1 \) runs in \( O(|W|^2|U|) \) time. In the incremental RSA setting with multiple (say \( l \)) utterances, the run-time of \( L_1 \) is \( O(|W|^2|U|l) \). As the number of hypotheses and utterances becomes large in a program synthesis domain, it becomes infeasible to compute \( L_1 \) at a speed required for end-user interactions. For instance, Vaithilingam, Pu, and Glassman (2023) were only able to build a pragmatic synthesizer for the regex domain consisting of 350 regular expressions.

This Work: Pragmatic Synthesis using Rankings

We are now in a position to explain our work – using a global pragmatic ranking to amortizing the RSA inference. Imagine we had a pre-computed, global ranking of programs, we would simply need to find consistent programs and rank them, which requires only \( O(|W|) \) steps. Consider the example from earlier where a speaker provides the utterance models the informative speaker \( S_1 \) as follows:

\[
S_1(u|w) = \frac{1}{\sum_{u'} S_1(u'|w)}
\]

\[
S_1(u|w) = \frac{1}{\sum_{u'} S_1(u'|w)}
\]
01. If we had determined that the global ordering of programs was $0\{1\} > 0+1\{1\} > 0\{2\}1+ > 0+1*$, we could identify $0+1\{1\}$ as the most preferred program that was consistent by progressing through the list, and checking consistency. Even after providing a second utterance 001, the outcome of this remains unchanged and still correct. In the rest of this paper, we describe how, for the case of single utterance (the user can only provide 1 string, e.g. 001), such a ranking provably exists. And for the case of multiple utterances (the user provides multiple strings incrementally, e.g. 001.01), even though we may not have guarantees, we empirically verify that we can compute and use such a ranking to make pragmatic program synthesis much more efficient.

**Global Pragmatic Ranking**

We formally define a global pragmatic ranking, and show how it can be used to construct a rank-based listener.

**Global pragmatic ranking** A global pragmatic ranking $\sigma_L$ for a listener $L$ is an ordering of hypotheses such that:

$$\forall w, w', u. \text{if } L(w|u) > 0 \land L(w'|u) > 0,$$

then $L(w|u) > L(w'|u) \iff \sigma_L[w] > \sigma_L[w']$ \hspace{1cm} (1)

Which is to say, for any two competing hypotheses $w, w'$ given utterance $u$, if both have non-zero probabilities under $L(\cdot|u)$, then both $\sigma_L$ and $L(\cdot|u)$ will sort these two hypotheses in the same order.

**A global ranking is utterance agnostic** The most salient feature of a ranking $\sigma$ is that it is *utterance agnostic*: the same ranking is used to order all the hypotheses irrespective of the utterance given. This is counter-intuitive, because one might expect that given different utterances $u, u'$, it is possible for a listener $L$ to give different rankings for $w$ and $w'$, even if both hypotheses are consistent with both utterances:

$$L(w|u) > L(w'|u) > 0 \land L(w'|u') > L(w|u') > 0$$ \hspace{1cm} (2)

However, we will show that – in the case of boolean lexicons with a single utterance and an RSA listener as described in the previous section – a global pragmatic ranking $\sigma_L$ consistent with $L$ must exist, and in the case of multiple utterances, a global ranking $\sigma_L$ can well approximate the true $L$ ranking empirically.

**Rank-based listeners** Given any ranking $\sigma$, one can use it to construct a rank based listener $L_\sigma : U \Rightarrow W^*$, that takes in an utterance $u$ and returns a sorted list of all satisfying hypotheses by progressing through the list of ranked hypotheses in $\sigma$ and filtering for consistency with $u$. As we can see, both the time and space complexity of $L_\sigma$ are $O(|W|)$.

**Existence of Ranking for Single Utterance**

We prove that in the case of boolean lexicons, a global pragmatic ranking must exist for any listeners $L_0, L_1, \ldots$.

\[\text{Theorem:}\] For a sequence of listeners in the RSA algorithm $L_0, L_1, \ldots$ over a boolean-valued lexicon $M$, there exists a sequence of global pragmatic rankings $\sigma_{L_0}, \sigma_{L_1}, \ldots$ such that:

$$\forall w, w', u. \text{if } L_i(w|u) > 0 \land L_i(w'|u) > 0,$$

then $L_i(w|u) > L_i(w'|u) \iff \sigma_{L_i}[w] > \sigma_{L_i}[w']$ \hspace{1cm} (3)

Let $M$ be a boolean lexicon of size $m$ rows and $n$ columns. Let $r_0 = r_1 \ldots r_n^m$ be the row-normalizing vector such that $r_0 = (\sum M[j, ;])^{-1}$, which is to say, each element $r_0[i]$ is the normalization term for row $i$ of $L_0$. Let $\ast \ast$ denote row-wise multiplication:

$$L_0 = M \ast \ast r_0$$

Which is to say, starting from $M$, $L_0$ can be obtained by scaling each row $j$ by their respective normalization constant $r_0[j]$. Let $c_0 = c_1 \ldots c_n$ be the col-normalizing vector such that $c_0 = (\sum L_0[; j])^{-1}$, which is to say, each element $c_0[i]$ is the normalization term for column $i$ of $S_1$. Similarly, let $\ast \ast$ denotes column-wise multiplication

$$S_1 = L_0 \ast \ast c_1 = M \ast \ast r_0 \ast \ast c_1$$

Computing $L_i$ under RSA amounts to applying row and column normalization alternatively multiple times:

$$L_i = M \ast \ast r_0 \ast \ast c_1 \ast \ast r_i$$

Let $\ast$ be element-wise multiplication, let $\otimes$ be outer-product, we can rearrange the terms:

$$L_i = M \ast ((r_0 \ast \ast r_i) \otimes (c_1 \ast \ast c_{i-1}))$$

$$= M \ast (r_0 \ast \ast c_i \ast \ast c_{i-1})$$ \hspace{1cm} (4)

Here, $r_0 \ast \ast r_i = r_0 \ast \ast \ast r_i$ is a vector of size $m$, and $c_1 \ast \ast c_{i-1}$ is a vector of size $n$. As we can see, following the RSA algorithm, $L_i$ can be decomposed to multiplication of 2 parts: the lexicon $M$, and a matrix that is formed by the outer product $r_0 \ast \ast c_i \ast \ast c_{i-1}$.

**Claim** The ordered indexes of $c_{i-1}$ is the global pragmatic ranking $\sigma_M$:

$$\sigma_M[w] \iff c_{i-1}[w] > c_{i-1}[w']$$

**Proof:** We show both sides of the $\iff$ in 3. Suppose that for some $w, w', u$, both $L_i(w|u) > 0$ and $L_i(w'|u) > 0$ (i.e. $M[u, w] = M[u, w'] = 1$).

1. Show $\Rightarrow$: Suppose $L_i(w|u) > L_i(w'|u)$. We have

$$L_i(w|u) = L_i[u, w] = r_0 \ast \ast [u] \ast c_{i-1}[w]$$

$$L_i(w'|u) = L_i[u, w'] = r_0 \ast \ast [u] \ast c_{i-1}[w']$$

As $r_0 \ast \ast [u]$ is a constant, we have

$$L_i(w|u) > L_i(w'|u) \Rightarrow c_{i-1}[w] > c_{i-1}[w']$$

2. Show $\Leftarrow$: Suppose $c_{i-1}[w] > c_{i-1}[w']$.

$$c_{i-1}[w] > c_{i-1}[w']$$

$$M[u, w] \ast r_0 \ast \ast [u] \ast c_{i-1}[w]$$

$$L_i[u, w] > L_i[u, w']$$

$$L_i(w|u) > L_i(w'|u)$$

Thus, $c_{i-1}$ is the global ranking $\sigma_M$ as claimed. \[\blacksquare\]
Approximate Ranking for Multiple Utterances

In the case of multiple utterances given incrementally, the orderings of hypotheses are utterance dependent; for some hypotheses, their orderings can swap depending on utterances (Equation 2). In this section, we give a simple approximation algorithm that attempts to find a global ordering that is maximally consistent with the ordering given by an RSA listener \( L \), consisting of first obtaining a communication dataset \( D \) consisting of orderings, then annealing a global ordering to be as close to the orderings in \( D \) as possible. This ordering is obtained offline during training, and cached for interaction with users. We sample a dataset of interactions between a speaker and a listener by unrolling the interaction and having the speaker select each utterance greedily based on the utterances given up to that point. We then obtain a ranking over programs using the listener probabilities given the utterances. This is detailed in Algorithm 1. We then use the dataset of interactions we collect to anneal a global ranking, as shown in Algorithm 2.

**Algorithm 1: Algorithm to obtain a dataset of simulated interactions between a speaker \( S \) and listener \( L \). For each turn of each interaction, a ranking of programs is obtained.**

Require: Set of programs \( P \)

Require: Length of specification to generate \( N \)

Require: Speaker model \( S(u|w, u) \)

Require: Listener model \( L(w|u) \)

Require: Function \( \text{MAKERANKING} \) that ranks samples from a distribution based on the probability

\[ D \leftarrow \{ \} \]

for \( p \) in \( P \) do

\( u \leftarrow [ ] \)

for \( i = 1 \) to \( N \) do

\( u_{\text{next}} \leftarrow \arg \max_u S(u|p, u) \)

\( u \leftarrow u + [u_{\text{next}}] \)

\( \bar{\sigma} \leftarrow \text{MAKERANKING}(L(\cdot|u)) \)

\( D \leftarrow D \cup \{(p, \bar{\sigma}, u)\} \)

end for

end for

return \( D \)

**Experiments**

To validate the usefulness and efficiency of an approximate ranking listener \( L_\sigma \), we perform the following two sets of experiments. First, we conduct a small \((n = 8)\) human experiment by building a ranking based synthesizer in a regular expression synthesis domain, a domain where it is infeasible to run the vanilla RSA algorithm \( L_0 \) at interaction time. Second, we conduct two replay studies taking the human interaction data from Vaithilingam, Pu, and Glassman (2023) and Pu et al. (2020). We seek to answer the following two research questions. **Q1:** Is \( L_0 \) performant, in terms of communication accuracy when communicating with humans, compared to the naive synthesizer \( L_0 \) and the vanilla RSA synthesizer \( L_1 \)? **Q2:** Is \( L_\sigma \) efficient, in terms of inference time, compared to \( L_0 \) and \( L_1 \)?

**Human Experiment**

To validate the efficacy of the ranking-based listener in interaction with users, we conducted a user study where people interacted with the ranking-based \( L_\sigma \) and literal synthesizers \( L_0 \) on the domain of regular expression synthesis.

**The Regex Domain** The regex domain is a scaled up version of Vaithilingam, Pu, and Glassman (2023), where they considered a total of 350 regular expressions from their grammar (Figure 6; this subset forms the hypothesis space \( W \)). They built \( L_1 \) and \( L_0 \) models based on RSA, and found in their experiments that \( L_1 \) is better. For this study, we expanded the space of programs to 3500 regular expressions from the same grammar – a setting that would make live interaction infeasible with the \( L_1 \) synthesizer they propose based on RSA.

**Participants** We recruited 8 participants from our institution, with 7 males and 1 female. 6 participants reported they had 2–5 years of experience with regular expressions, and
Figure 7: Success rate of the literal and ranking-based synthesizers in the human experiments. The ranking-based synthesizer achieves a success rate of 93.75%, while the literal synthesizer achieves only 65.63%. The ranking-based synthesizer also achieves higher success with fewer utterances.

2 participants reported less than a year of experience using regular expressions. Each received a $20 gift certificate.

Procedure Each participant was given a short tutorial on how to use the interface, then completes a total of 4 communication tasks. For each task, the participant was asked to communicate a target regex using examples to both the literal $L_0$ and the ranking $L_{\sigma}$ synthesizer, anonymized as simply a “green robot” and a “blue robot”, one after another in a randomized order. For each regex, the participants can take any numbers of turns (each turn consists of providing an additional example string) until the regex is recovered by the synthesizer, or give up early. The communication is interactive: When the participant adds a new example, they are immediately shown the current top-1 guess of the synthesizer, which allows them to adjust the next example accordingly.

Results Figure 7 shows the communication success rate over turns for both the literal and ranking-based synthesizers. As we can see, not only does the ranking-based synthesizer eventually succeed more often than the literal synthesizer (with the user successfully communicating about X% of targets to the ranking-based synthesizer as opposed to Y% to the literal synthesizer), but it also achieves a higher success rate for smaller numbers of turns, allowing the user to successfully communicate at a lower cost. We conclude that $L_{\sigma}$ performs better than $L_0$ for the regex domain (Q1).

Replay Studies

We also compare $L_{\sigma}$ vs $L_1$ and $L_0$ by replaying the interaction data collected from (Vaithilingam, Pu, and Glassman 2023) and (Pu et al. 2020) – small pragmatic program synthesis domains where it is feasible to use the vanilla RSA synthesizer $L_1$.

Replay Data During the human studies of (Vaithilingam, Pu, and Glassman 2023) and (Pu et al. 2020), a human $H$ is given a target hypothesis $w$, and attempt to get the synthesizer ($L_0$ or $L_1$) to infer the target using a sequence of examples $u = u_1, u_2, \ldots$. Thus, two sets of data are generated, one where the human is interacting with the literal synthesizer $L_0$, which we term $H_0$, and one where the human is interacting with the pragmatic synthesizer $L_1$, which we term $H_1$. Specifically, from each domain we extract the following dataset \{$(w, u_j)$|$w \in W_s, j \in P, i \in \{0, 1\}$\}. Here, $W_s$ are the set of hypotheses used for the human study (the stimuli), $P$ is the set of participants, and $i$ indicates if the participant is communicating with $L_0$ or $L_1$.

Experiment Setup We can simulate an user interaction by using the replay data. Given a datapoint $w, u$, we create a simulated user that iteratively gives the utterances $u_1, u_2, \ldots$ in multiple turns to communicate a given target hypothesis $w$. At every turn, the synthesizer returns the top-1 responses, $L_{\text{top-1}}(u_1), L_{\text{top-1}}(u_1, u_2), \ldots$, and we can check if any of them matches the target hypothesis $w$. If they do, we mark the communication as successful and stop early. Otherwise, we keep adding utterances until the $u$ runs out, and we mark the communication as unsuccessful. Note that our evaluation cannot account for a user adapting their choice of examples to $L$, as the simulated user can only give scripted utterances according to the replay data.

Domain 1: Animals Pu et al. (2020) uses a domain of grid patterns generated by an underlying domain-specific language (see Appendix for the grammar of the DSL and semantics). The space contains 17,976 semantically distinct programs and 343 possible examples, where a user uses a sequence of multiple examples to communicate a target program. They conducted a study with 48 human subjects,
In Conclusion In the human study, end users are more successful at interacting with the ranking based synthesizer compared to the non pragmatic synthesizer (Q1). In the replay experiment, for both domains, the ranking based synthesizer $L_\sigma$ significantly outperforms the non-pragmatic synthesizer $L_0$, while being orders of magnitudes faster than the exact inference $L_1$ synthesizer (Q1, Q2).

Related Work

Rational Speech Acts (RSA) The RSA framework (Frank and Goodman 2012) models human cognition as cooperative Bayesian inference, and has been successfully applied to model a variety of human communicative and linguistic behaviour (Goodman and Frank 2016; Kao, Bergen, and Goodman 2014; Kao et al. 2014; Wang et al. 2020).

The closest work to ours is a pragmatic approach to program synthesis developed by Pu et al. (2020). Their work, however, is limited to a domain that affords simple programs and demonstrations which are efficiently enumerable. We show that our ranking-based approach also produces pragmatic behavior on this domain (Animals), but also allows us to scale to the more realistic regular expressions domain.

RSA has also been applied to improve the performance of language interfaces in a variety of other domains, such as image description (Andreas and Klein 2016; Cohn-Gordon, Goodman, and Potts 2018a,b), instruction generation and interpretation (Fried, Andreas, and Klein 2018; Fried et al. 2018), and grounded interaction (Fried, Chiu, and Klein 2021; Lin et al. 2022). As exact RSA inference in these domains, like in our synthesis domain, is intractable due to a large space of utterances and hypotheses, these approaches all use approximations which normalize over a smaller set of utterances or hypotheses, which are sampled from a proposal model trained on data from people (Monroe et al. 2017). Our approach, in contrast, requires no human-produced data. On the other hand, sampling a subset of utterances and hypotheses can be easily adopted into our framework during the generation of interaction data, in case the full RSA algorithm is too slow even for generating training data.

Ranking Functions in Synthesis Prior works on resolving ambiguity in program synthesis typically fall into two categories: using a human crafted ranking function and learning from human generated data. Works such as Singh and Gulwani (2015); Polozov and Gulwani (2015) use scoring functions to penalize certain properties of programs (e.g. discouraging the use of constants), effectively inducing a global ranking over all programs; Ellis and Gulwani (2017) uses a set of hand-crafted features to learn a naturalistic ranking from data. Synthesis algorithms that use a large neural code model to sample a large number of programs (Chen et al. 2021) implicitly rank the programs based on their naturalistic distributions in its training data. Our approach is unique in that (1) the learned ranking is rooted in efficient communication rather than hand-crafted and (2) our approach does not require human annotated training data.
References


Ellis, K.; and Gulwani, S. 2017. Learning to Learn Programs from Examples: Going Beyond Program Structure. IJCAI.


Appendix

Code

Code for this work can be found at https://github.com/evanthebouncy/pragmatic_synthesis_ranking

Simulated Studies

Ranking Always Exists

We empirically validate that in the case of single utterances, a ranking can always be found. See simulation/single_utter/exp_exists_orders.py

Stability of Ranks Across RSA Iterations

We’ve shown that for every $L_0, L_1, \ldots$, there exists a corresponding global, utterance agnostic ranking $\sigma_{L_0}, \sigma_{L_1}, \ldots$. We now explore the relationship between these rankings as a function of the RSA iteration $i$. Specifically, how stable is the relative ranks of $w$ and $w'$ once it is formed?

Stable Order

A pair-wise order between $w$ and $w'$ is stable from iteration $i$ onward if:

$$\text{stable}(i, w \succ w') \iff \bigwedge_{j \in \{i+1, \ldots, \infty\}} \sigma_{L_j}[w] \succ \sigma_{L_j}[w']$$

Which means the relative ranking of $\sigma_{L_0}[w] \succ \sigma_{L_0}[w']$ holds true for every subsequent iterations until $\sigma_{L_\infty}$. Let the minimal-index of a stable pair-wise ordering be the first iteration $i$ such that $w \succ w'$ becomes stable:

$$i_{\text{min}}(w \succ w') = \arg\min_j \text{stable}(j, w \succ w)$$

As $\sigma_{L_i}$ is the first time any ranking can exist ($L_0$ is a uniform distribution over valid hypotheses, i.e. no rankings), we explore the following: For a lexicon $M$, what fraction of stable orderings have a minimal-index of 1?

$$\frac{\text{frac-stable}_{L_i}(M)}{\text{lexicon size}} = \frac{|\{w \succ w' \mid i_{\text{min}}(w \succ w') = 1\}|}{|\{w \succ w' \mid \exists i. \text{stable}(i, w \succ w')\}|}$$

Simulation

We measure $\text{frac-stable}_{L_i}(M)$ on a population of sampled random boolean lexicons. We sample square lexicons of size $\text{lexicon.size} \in 2 \times 2 \ldots 100 \times 100$. Each lexicon is sampled with $P_{\text{true}} \in \{0.1, 0.2, 0.5\}$, where larger value of $P_{\text{true}}$ makes the lexicon have more 1s. We make sure each sampled lexicon is valid in the following sense: (1) all rows are unique – every utterance must communicate a unique subset of valid hypotheses (2) all columns are unique – every hypothesis has a unique set of utterances that can refer to it. For every combination of $(P_{\text{true}}, \text{lexicon.size})$ we randomly sample 100 lexicons. As it is infeasible to run RSA until iteration $\infty$, we run RSA for 100 iterations for each lexicon (i.e. $L_{100} \approx L_{\infty}$). We measure $\text{stable}_{L_i}$ for each sampled lexicon. The result is shown in 11. As we can see, of all the stable pair-wise orderings, a large fraction ($> 0.8$) are formed during $\sigma_{L_i}$, this is increasingly true as we (1) increase $P_{\text{true}}$, making the boolean lexicons having more number of 1s – i.e. the lexicon is more ambiguous for a literal speaker and listener and (2) increase $\text{lexicon.size}$.

We suspect this is due to faster “mixing time” of the RSA algorithm under these conditions, but this is just a guess.

Takeaway

This study may provide an alternative explanation as to why humans do not perform RSA for more than few iterations (Franke and Degen 2016). In addition to it being computationally expensive, it is also not necessary as the majority of top-k orderings becomes available at $\sigma_{L_i}$, and remains stable for all subsequent iterations of the RSA algorithm. In another word, $L_{i>1}^{\text{top-k}} \approx L_{i>1}^{\text{top-k}}$. Code in simulation/single_utter

Animals domain

In the Animals domain, a program is a pattern on a grid formed from a set of objects. These objects may be a colourless pebble, or a chicken or pig that may be red, green or blue. An utterance reveals one square on the grid, and the speaker has to communicate the pattern by choosing which square to reveal. The pattern is formed according to rules specified in the domain-specific language in Figure 12. Examples of programs shown in Figure 13. The description of the domain-specific language and the examples are due to Vaduguru, Ellis, and Pu (2022).

Human study interface

The interface for the human study on regular expression programs is shown in Figure 14.

Figure 11: Fraction of stable orders that were formed in $\sigma_{L_i}$ as a function of increasing lexicon size. Points are raw samples (n=100 per lexicon size and $P_{\text{true}}$), bars are 95% bootstrapped CI (nboot = 1000). Overall, increasing $P_{\text{true}}$ and lexicon size increases the fraction of stable orders that were formed in $\sigma_{L_i}$.
Program $\rightarrow$ \langle Shape, Colour \rangle

Shape $\rightarrow$ Box(Left, Right, Top, Bottom, Thickness, Outside, Inside)

Left $\rightarrow$ 0 | 1 | 2 | 3 | ... | 6
Right $\rightarrow$ 0 | 1 | 2 | 3 | ... | 6
Top $\rightarrow$ 0 | 1 | 2 | 3 | ... | 6
Bottom $\rightarrow$ 0 | 1 | 2 | 3 | ... | 6
Thickness $\rightarrow$ 1 | 2 | 3
O $\rightarrow$ chicken | pig
I $\rightarrow$ chicken | pig | pebble
Colour $\rightarrow$ [red, green, blue][A1(A2(A3))]

A1 $\rightarrow$ x | y | x + y
A2 $\rightarrow$ $\lambda z:0|\lambda z:1|\lambda z:2|\lambda z:z\%2|\lambda z:z\%2+1|\lambda z:2*(z\%2)$

Figure 12: Grammar of the DSL

Figure 13: Two patterns in our layout domain and their corresponding programs, represented as a sequence of production rules: [Program, Shape, Left, Right, Top, Bottom, Thickness, O, I, Colour, A1, A2]. The symbol $\cdot$ indicates rules which only have 1 choice of expansion (Program, Shape, and Colour). The rules where these two programs differ are marked with a box.
Figure 14: User interface for the regex domain

Regex: 1*0{1}

<table>
<thead>
<tr>
<th>REGEX</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>'1' (0 or more times (matching the most amount possible))</td>
</tr>
<tr>
<td>0{1}</td>
<td>'0' (1 time)</td>
</tr>
</tbody>
</table>

Enter examples here:

0
10

Guess