evolution_compositionality_under_noise

December 12, 2019

2019-08-06 11:19:34

1 The evolution of compositionality under environmental noise

1.1 Hypothesis:

We hypothesise that other-initiated repair and a compositional language co-evolve under the following assumptions:

1. There is a pressure for communicative success (or more specifically: that the listener makes an effort to find a complete meaning to interpret given the speaker's utterance)
2. Listeners have at least two types of repair initiator at their disposal: (i) open request, and (ii) restricted request
3. There is a pressure to make repair sequences as efficient as possible (in terms of combined utterance length of repair initiator and response)

This hypothesis is based on the idea that if repair sequences are under a pressure to be efficient, then it becomes useful to be able to initiate repair by asking for clarification of just *part* of the utterance, specifically by feeding back the part of the meaning that you *did* get. Under those circumstances, a compositional system is helpful (compared to a holistic system), because parts of the signal (i.e. 'substrings') map to parts of the meaning. ***Note*** that this hypothesis is thus based on the assumption that the 'closed request' type of repair initiator works only if the listener has *understood* part of the *meaning* of the signal, as opposed to just having received part of the *signal* and being able to repeat that back verbatim, in order to prompt the speaker to repeat the part that didn't come through. We feel that this is a reasonable assumption based on what we know from empirical work, but it is a little hard to assess given that natural language *is* compositional.

There are of course parts of language that are holistic though. Mark D. and I thought of the example of proper nouns. For instance, if A says "My favourite writer is Chimamanda Ngozi Adichie", B could say "Chima *who* ?" in order to prompt A to repeat the name. However, our intuition is that it would be quite unnatural for A to then respond to B's repair request by saying only "manda Ngozi Adichie". Most likely A would simply repeat the whole name.

When I was talking to Simon Kirby about it being difficult to think about whether repair could work as efficiently in a holistic system, he thought of colour terms as an example of a holistic part of language. And as an example he gave the repair sequence:

A: "the blue shirt"

B: "you mean the turquoise one?"

In this example, the issue with the holistic signal "blue" is not that part of it was covered by noise, but that it is (apparently) not specific enough to pick out one particular referent. So it's not quite the same as what we were thinking about above: which is the question whether one can do efficient repair (i.e. a restricted request) when given a holistic signal that was partly disturbed by noise, in such a way that the listener does not get even *part* of the meaning. Whereas in the example above, the listener receives the full signal without noise, and therefore is able to interpret part of the meaning (i.e. exclude some potential referents, such as the red shirt), but the problem is that they cannot quite narrow it down to a single unambiguous referent.

2019-08-06 11:19:27

1.2 Predictions:

Here we adapt the model of Kirby et al. (2015)—where they show that a compositional language evolves when there is a pressure for both (i) learnability and (ii) expressivity—to incorporate noise (step 1) and repair (step 2).

In Kirby et al.'s model, the pressure for expressivity (implemented as speakers aiming to avoid ambiguity when they produce a signal) is what causes populations to move towards languages that have one-to-one mappings between meanings and signals, but these can be either compositional or holistic. The pressure for learnability then (implemented as a prior that favours more compressible languages) is what causes populations to move towards compositional languages rather than holistic languages, because compositional languages are more compressible (and therefore, is the assumption, easier to learn).

1.2.1 Strong prediction (excluding pressure for learnability):

Therefore, the strongest prediction given our reasoning above, would be that if we take away the pressure for learnability, but add in environmental noise and the possibility of repair (combined with a pressure for efficient repair sequences), populations would also converge on compositional languages. This may be a bit of a weird prediction in itself, because we *do* believe a pressure for learnability will have played a role. However, by isolating the effects of the two pressures (learnability vs. noise+repair) we could say something about their individual contributions.

1.2.2 'Weaker' prediction (including pressure for learnability):

Not entirely sure what to predict here. Maybe the effects of a noise+repair+efficiency pressure on the one hand, and a pressure for learnability on the other hand, will be additive? So that when combined, they cause populations to converge on a compositional language faster than when only one of these pressures is at play?

1.2.3 Which predictions to test?

We'd like to explore both predictions, starting with the stronger one.

1.2.4 References

Kirby, S., Tamariz, M., Cornish, H., & Smith, K. (2015). Compression and communication in the cultural evolution of linguistic structure. Cognition, 141, 87–102. https://doi.org/10.1016/j.cognition.2015.03.016

1.2.5 Question: How to implement the pressure for communicative success that we have in mind?

What I'm wondering here is whether the pressure for expressivity, as it was implemented by Kirby et al. (2015), in combination with a pressure for efficiency (and the possibility of repair), will have the effect of a pressure for communicative success as we have in mind, or whether we'll have to incorporate an extra or different pressure for communicative success.

Firstly, the way the pressure for expressivity is implemented by Kirby et al. (2015): speakers try to avoid using ambiguous signals. —> this is sufficient for pushing populations towards having either holistic or compositional languages (as opposed to degenerate ones).

—> What could the effect of environmental noise be in this? If data consists of <meaning, form> pairs that were produced by a pair of agents from the previous generation, as in the Kirby et al. (2015) model, noise *without* repair will mean that learners sometimes receive incomplete data (i.e. an incomplete form). Meanings in the <meaning, form> pairs that make up the data should not be incomplete, because (if I understand the Kirby et al., 2015, paper correctly) these represent the speaker's communicative intention, rather than the listener's interpretation.

—> What will that mean for how likely learners are to infer different languages? In general we'd just expect that learners will need more observations in order to reach a given amount of posterior belief in the correct language hypothesis. This means that given a particular bottleneck width, populations who are exposed to noise will be more likely to transition from one language into another than populations who aren't. Will the addition of noise push learners in any particular direction though? I can't think of why that would be the case if learners have a flat prior. However, if learners have a compression-based prior as in the Kirby et al. (2015) model, they would be pushed towards degenerate languages in general, in the same way as it happens in Kirby et al.

—> The pressure for communicative success that Mark D. and I had in mind (I think) is that speaker and hearer try to cooperate to reach mutual understanding; that is, so that the speaker can get their intended meaning across. **Design decision:** I think this pressure is captured if we assume that the hearer will initiate repair whenever (i) they are aware that noise happened, and (ii) they can't derive a single unambiguous meaning based on the form they received. The first condition is a bit artificial I think, because in natural language people do initiate repair even if they have received the full signal perfectly, but can't disambiguate the meaning of (some part of) it. Maybe a better model would be if the listener at least *considers* initiating repair every time they cannot find a single unique meaning that corresponds to the speaker's utterance, but that whether or not the listener *actually* initiates repair depends on (i) just how ambiguous the meaning still is, and (ii) the cost of the required repair sequence (assuming an open request is more costly overall than a restricted request?)?

Assuming we run with the first model (where listeners only initiate repair if they know that noise happened), this would mean that a hearer will not have to initiate repair even when they received an incomplete (i.e. noisy) form. That is, if the hearer is using a language of the 'other' type, in which there is only one form that starts with a 'b', and that form maps to one particular meaning, then they will be able to infer a full meaning, and will therefore not initiate repair. However, such a language will necessarily also contain some degenerate forms, such as 'aa' mapping to both meaning '02' and meaning '03'. *That* is where the pressure for expressivity comes in: speakers will

try to avoid using such ambiguous forms, and therefore learners will be more likely to infer either a holistic or compositional language when receiving data from such a language of the 'other' type.

—> So in our 'strong prediction' scenario, where we *replace* Kirby et al.'s learnability pressure with a pressure for efficient repair (and assuming repair is only initiated when the listener knows that noise happened), we'd expect that populations get pushed in the direction of languages of the 'other' type, as well as languages of the compositional type rather than languages of the degenerate type. I'm not sure which language type would come out as best fit to a pressure for efficient repair however: 'other' or 'compositional'. So let's work through an example:

2019-08-26 17:43:38

1.2.6 Worked example to figure out how each language type would fare under pressure for efficient repair.

Here's a language of the 'other' type which might be specifically useful under a pressure for efficient repair: - 02 –> aa - 03 –> ab - 12 –> aa - 13 –> ba

1. Noisy form a_ –> open request (because a_ does not map reliably to either meaning 0_ or 1_)
2. Noisy form b_ –> no repair (because it maps unambiguously to meaning 13)
3. Noisy form _a –> open request (because _a does not map reliably to either meaning _2 or _3)
4. Noisy form _b –> no repair (because it maps unambiguously to meaning 03)

So all in all, a listener using this language would initiate the more costly form of repair in 50% of the cases where there is noise, and no repair at all in 50% of the cases where there's noise.

Now let's compare this to a case of a listener who uses a compositional language, such as this one: - 02 –> aa - 03 –> ab - 12 –> ba - 13 –> bb

1. Noisy form a_ –> restricted request (because a_ maps unambiguously to meaning 0_)
2. Noisy form b_ –> restricted request (because b_ maps unambiguously to meaning 0_)
3. Noisy form _a –> restricted request (because _a maps unambiguously to meaning _2)
4. Noisy form _b –> restricted request (because _b maps unambiguously to meaning _3)

So which language type allows for the most efficient repair depends on how costly we assume open requests are compared to restricted requests. If, for simplicity's sake, we assume an open request is twice as costly as a restricted request (say c=1.0 for an open request, c=0.5 for a restricted request, and c=0.0 for no repair), a compositional language would be *exactly* as efficient as the other-type language above. Specifically, the average cost of repair for the other-type language would be: $(0.25 * 1.0) + (0.25 * 0.0) + (0.25 * 1.0) + (0.25 * 0.0) = 0.5$. And the average cost of repair for the compositional language would be: $(0.25 * 0.5) + (0.25 * 0.5) + (0.25 * 0.5) + (0.25 * 0.5) = 0.5$.

However, if we assume that there is some baseline cost for doing repair at all (which seems more reasonable to me?), so that the cost of an open versus a restricted request are relatively closer together than the cost of a restricted request versus no repair at all, the other-type language above would win from the compositional language in terms of repair efficiency. To give a simple example, imagine that no repair costs 0.0, a restricted request costs 0.75, and an open request costs 1.0. Then the average cost of repair for the other-type language would still be: $(0.25 * 1.0) + (0.25 * 0.0) + (0.25 * 1.0) + (0.25 * 0.0) = 0.5$, whereas the average cost of repair of the compositional language would now be: $(0.25 * 0.75) + (0.25 * 0.75) + (0.25 * 0.75) + (0.25 * 0.75) = 0.75$.

Perhaps we could base how we parameterise these costs on some empirical findings?

Combining our pressure for efficient repair with Kirby et al.'s pressure for expressivity, should then get rid of the languages of the 'other' type. However, the extent to which this happens should depend on the relative strengths of the pressure for expressivity and the pressure for efficiency.

—> So I guess what I'm saying is that in our model, the pressure for communicative success should be extended somewhat, to not only consist of the (parameterised) assumption that speakers try to avoid ambiguity, but also the assumption that listeners try to arrive at a complete, unambiguous interpretation, rather than just settling for a (wholly or partly) ambiguous meaning.

Importantly however, this analysis does depend on the assumption that the listener is aware of whether noise occured or not, and only initiates repair when that is the case, not for disambiguation in general. If listeners also initiated repair for *general* disambiguation purposes, in the absence of noise (even though their language has a homonym in it, which one could assume the listener could be aware of?), a listener who uses the 'other' type language above would also initiate repair *whenever* they receive the 'aa' form. This would mean they would also initiate repair 50% of the time in the *absence* of noise, making the overall rate of repair higher for languages of the 'other' type than for languages of the 'compositional' type.

Now let's work through a similar example comparing languages of the 'holistic' and 'compositional' type: First, an example of a holistic language: - 02 –> aa - 03 –> bb - 12 –> ba - 13 –> ab

Here, the listener would also have to initiate repair in all cases where there is partial noise, just like with a compositional language. However, repair is less efficient because even if noise is only partial, the listener will have to use an open request in all cases, because they cannot make it more specific, because they don't have any part of the meaning.

1.3 How and where to implement efficiency pressure on repair?

It makes most sense that this pressure would be at play during communication, rather than during learning. We can't assume that learners prefer languages that allow for more efficient repair because they have some kind of a priori awareness of how efficient each of the languages would be in the case that repair is needed.

Therefore, instead of being located in the learner's prior, as the learnability pressure in the Kirby et al. (2015) model, the efficiency pressure should act somewhere during communication, just like Kirby et al.'s expressivity pressure. We could start by creating a simple cost function, where an open request is more costly than a restricted request (and exactly how much more costly it is could be regulated by some parameter?), and we could make the listener's process of whether or not to initiate repair a stochastic one, where they are less likely to initiate repair the more costly the repair sequence is.

2019-09-06 13:20:50

1.4 Decisions about how to implement repair made on 05/09/2019:

- A listener should initiate repair probabilistically, depending on the amount of ambiguity (i.e. uncertainty about the intended referent) that is left after receiving the signal (this could also be expressed as the inverse of the amount of bits of information that the signal provides)

- The reason for initiating repair is therefore always to reduce ambiguity, not just because noise happened. That means that if the listener received a noisy form but is still able to narrow their interpretation down to a single meaning, they will never initiate repair. Similarly, if the listener uses a degenerate language to interpret utterances, they will always initiate repair, even if they've received a complete form. We can however cap the number of repeated repair initiators within a given interaction to 3, because that's where we see people stop trying in real linguistic interactions as well.
- What is the cost of repair? No repair should have no cost (in terms of efficiency), while an open request should have a slightly higher cost than a restricted request. The reasoning behind that being that in a restricted request sequence, parts of the trouble source turn can be 're-used': we assume that the listener can only use a restricted request if they have understood at least part of the meaning (which, in the case of our simple toy languages, is equivalent to being able to narrow down the possible interpretations to two candidates). The listener can then ask for clarification about only the other part of the meaning, which means that the speaker only has to repeat part of the signal.
- Another idea (that might be worth looking into later) is that the costs of repair could be expressed in terms of how many repair sequences are necessary within a given interaction. There you would expect that the costs of repair are higher when using a degenerate language than when using a different type of language.

2019-08-26 17:29:10 2019-07-31 15:27:33

1.5 Production in the Kirby et al. (2015) model:

$$P(f \mid l,t) \propto \begin{cases} (\frac{1}{a})^{\gamma}(1-\epsilon) & \text{if } t \text{ is mapped to } f \text{ in } l \\ \frac{\epsilon}{|F|-1} & \text{if } t \text{ is not mapped to } f \text{ in } l \end{cases}$$

where f stands for form (in the sense of a complete signal, such as aa), l stands for language, t stands for topic, a stands for ambiguity (i.e. the number of meanings in M that map to form f in l), γ specifies the extent to which ambiguous utterances are penalised, and ϵ stands for production error.

From Kirby et al. (2015, pp. 92-93): > "If $a = 1$ (f is unambiguous) and/or $\gamma = 0$ then this yields a model of production where the 'correct' form is produced with probability $1 - \epsilon$. However, when $\gamma > 0$ and f is ambiguous (i.e., $a > 1$), then the 'correct' mapping from t to f is less likely to be produced (the probability $P(f \mid l,t)$ is reduced by the factor $(\frac{1}{a})^{\gamma}$) and the remaining probability mass is spread equally over the other possible forms, leading to increased probability of producing $f' \neq f$. Therefore, $\gamma > 0$ introduces a penalty for languages whose utterances are ambiguous."

2019-07-31 15:27:52 ## How do we add *environmental* noise to this production model?

If we leave Kirby et al.'s production error ϵ aside for a moment, we could add environmental noise by allowing production to have two possible outcomes. Given a particular topic t, a speaker could produce either: - form f if t is mapped to f in l, with probability $1 - n$ (where n stands for the probability of noise happening) - form f_{noisy}, where f_{noisy} is a 'noisy variant' of f *and* t is mapped to f in l. This should happen with probability $\frac{n}{|F_{noisy}|}$ (where F_{noisy} is the full set of possible noisy variants of the form f that maps to t in l)

To give an example, using the example compositional grammar from Kirby et al. (2015, p. 92):

S –> A B
A:0 –> a
A:1 –> b
B:2 –> a
B:3 –> b

Given this grammar, the only form that maps to topic 02 is *aa*.
Design decision: If we allow for both 'partial' and 'full' noise, this form has three possible noisy variants: - a_ - *a* - _

where _ stands for a noisy part of the signal that the listener could not perceive.

Decision Mark D. + Marieke (01/08/2019): Only partial noise is enough for now. We do want to allow for both open and restricted repair requests being a possibility, because the intuition that our hypothesis is based on, is that whereas a compositional system allows for the usage of both open and restricted requests, a holistic system only allows for open requests. However, to allow for open requests we don't necessarily need a form that is entirely noise (i.e. ___); we only need to assume that if the listener is able to disambiguate *part* of the meaning, they use a restricted request, and if they cannot disambiguate any of the meaning they us an open request.

So if 02 is the speaker's intended topic (and if we exclude the possibility of the speaker making a production error for now), the speaker will produce the following signals with the following probabilities: - aa with probability $1-n$ - [a_, _a] with probability $\frac{n}{2}$ (because there are 2 possible noisy variants)

Now let's add the probability of making a production error ϵ back in, which gives us:

$$P(f \mid l,t) \propto \begin{cases} (\frac{1}{a})^{\gamma}\,(1-\epsilon)\,(1-n) & \text{if } t \text{ is mapped to } f \text{ in } l \text{ and } f \text{ is intact} \\ (\frac{1}{a})^{\gamma}\,(1-\epsilon)\,\frac{n}{|F_{noisy}|} & \text{if part of } t \text{ is mapped to } f \text{ in } l \text{ and } f \text{ is not intact} \\ \frac{\epsilon}{|F|-1}\,(1-n) & \text{if } t \text{ is not mapped to } f \text{ in } l \text{ and } f \text{ is intact} \\ \frac{\epsilon}{|F|-1}\,\frac{n}{|F_{noisy}|} & \text{if no part of } t \text{ is mapped to } f \text{ in } l \text{ and } f \text{ is not intact} \end{cases}$$

where n stands for the probability of noise.

Actually, the probability for when f is *not* mapped to t, *and* f is a noisy variant, is not quite as simple as stated above. Depending on how many forms f there are that *don't* map to t, some of those error forms will generate the same noisy variant. For example, if the form that *is* mapped to the intended topic is 'aa', and the forms ['ab', 'ba', 'bb'] form the set of forms that *aren't* mapped to t, some noisy variants will only occur given one of these error forms being what the speaker produced ('a_' in this case), while others will occur given multiple possible error forms that the speaker might produce (e.g. '_b' can be a noisy variant of either 'ab' or 'bb'). The probabilities for the different noisy variants should reflect this. (I.e. in the example above, noisy variant '_b' should be twice as probable as noisy variant 'a_'.)

2019-07-31 15:28:14

1.6 First, let's reproduce Kirby et al.'s (2015) model of production:

In order to do that we first need to set some general parameters:

[1]:
```
# First some parameters:
meanings = ['02', '03', '12', '13']  # all possible meanings
forms_without_noise = ['aa', 'ab', 'ba', 'bb']  # all possible forms, excluding␣
↪their possible 'noisy variants'
noisy_forms = ['a_', 'b_', '_a', '_b']  # all possible noisy variants of the␣
↪forms above
all_forms_including_noisy_variants = forms_without_noise+noisy_forms  # all␣
↪possible forms, including both complete
                                                                    # forms and␣
↪noisy variants
error = 0.05  # the probability of making a production error
gamma = 2  # parameter that determines strength of ambiguity penalty (Kirby et␣
↪al. used gamma = 2 for communication
           # condition and gamma = 0 for condition without communication)
turnover = True  # determines whether new individuals enter the population or␣
↪not
b = 20  # the bottleneck (i.e. number of meaning-form pairs the each pair gets␣
↪to see during training (Kirby et al.
       # used a bottleneck of 20 in the body of the paper.
rounds = 1*b  # Kirby et al. (2015) used rounds = 2*b, but SimLang lab 21 uses␣
↪1*b
popsize = 2  # If I understand it correctly, Kirby et al. (2015) used a␣
↪population size of 2: each generation is simply
            #  a pair of agents.
runs = 50  # the number of independent simulation runs (Kirby et al., 2015 used␣
↪100)
gens = 150  # the number of generations (Kirby et al., 2015 used 100)
noise = False  # parameter that determines whether environmental noise is on or␣
↪off
noise_prob = 0.1  # the probability of environmental noise masking part of an␣
↪utterance
```

Then we need some functions to generate all possible languages and classify languages according to the categories specified by Kirby et al. (2015):

[2]:
```
import itertools
import numpy as np

# Some functions to create and classify all possible languages:
def create_all_possible_languages(meaning_list, forms):
    """Creates all possible languages

    :param meaning_list: list of strings corresponding to all possible meanings
    :param forms: list of strings corresponding to all possible␣
↪forms_without_noisy_variants
```

```
    :returns: list of tuples which represent languages, where each tuple␣
↪consists of forms_without_noisy_variants and
    has length len(meanings)
    """
    all_possible_languages = list(itertools.product(forms,␣
↪repeat=len(meaning_list)))
    return all_possible_languages


def classify_language(lang, forms, meaning_list):
    """
    Classify one particular language as either 'degenerate' (0), 'holistic'␣
↪(1), 'other' (2)
    or 'compositional' (3) (Kirby et al., 2015)

    :param lang: a language; represented as a tuple of␣
↪forms_without_noisy_variants, where each form index maps to same
    index in meanings
    :param forms: list of strings corresponding to all possible␣
↪forms_without_noisy_variants
    :param meaning_list: list of strings corresponding to all possible meanings
    :returns: integer corresponding to category that language belongs to:
    0 = degenerate, 1 = holistic, 2 = hybrid, 3 = compositional, 4 = other␣
↪(here I'm following the
    ordering used in the Kirby et al., 2015 paper; NOT the ordering from␣
↪SimLang lab 21)
    """
    # TODO: See if I can modify this function so that it can deal with any␣
↪number of forms_without_noisy_variants and
    #  meanings.
    class_degenerate = 0
    class_holistic = 1
    class_hybrid = 2  # this is a hybrid between a holistic and a compositional␣
↪language; where *half* of the partial
    # forms is mapped consistently to partial meanings (instead of that being␣
↪the case for *all* partial forms)
    class_compositional = 3
    class_other = 4

    # First check whether some conditions are met, bc this function hasn't been␣
↪coded up in the most general way yet:
    if len(forms) != 4:
        raise ValueError(
            "This function only works for a world in which there are 4 possible␣
↪forms_without_noisy_variants"
        )
```

```
    if len(forms[0]) != 2:
        raise ValueError(
            "This function only works when each form consists of 2 elements")
    if len(lang) != len(meaning_list):
        raise ValueError("Lang should have same length as meanings")

    # lang is degenerate if it uses the same form for every meaning:
    if lang[0] == lang[1] and lang[1] == lang[2] and lang[2] == lang[3]:
        return class_degenerate

    # lang is compositional if it makes use of all possible␣
↪forms_without_noisy_variants, *and* each form element maps
    # to the same meaning element for each form:
    elif forms[0] in lang and forms[1] in lang and forms[2] in lang and forms[
        3] in lang and lang[0][0] == lang[1][0] and lang[2][0] == lang[3][0]␣
↪and lang[0][
        1] == lang[2][1] and lang[1][1] == lang[3][1]:
        return class_compositional

    # lang is holistic if it is *not* compositional, but *does* make use of all␣
↪possible forms_without_noisy_variants:
    elif forms[0] in lang and forms[1] in lang and forms[2] in lang and␣
↪forms[3] in lang:
        # within holistic languages, we can distinguish between those in which␣
↪at least one part form is mapped
        # consistently onto one part meaning. This class we will call 'hybrid'␣
↪(because for the purposes of repair, it
        # is a hybrid between a holistic and a compositional language, because␣
↪for half of the possible noisy forms that
        # a listener could receive it allows the listener to figure out *part*␣
↪of the meaning, and therefore use a
        # restricted request for repair instead of an open request.
        if lang[0][0] == lang[1][0] and lang[2][0] == lang[3][0]:
            return class_hybrid
        elif lang[0][1] == lang[2][1] and lang[1][1] == lang[3][1]:
            return class_hybrid
        else:
            return class_holistic

    # In all other cases, a language belongs to the 'other' category:
    else:
        return class_other


def classify_all_languages(language_list, complete_forms, meaning_list):
    """
```

```
    Classify all languages as either 'degenerate' (0), 'holistic' (1), 'other' (2) or 'compositional' (3)
    (Kirby et al., 2015)

    :param language_list: list of all languages
    :param complete_forms: list containing all possible complete forms; corresponds to global variable
    'forms_without_noise'
    :param meanings: list of all possible meanings; corresponds to global variable 'meanings'
    :returns: 1D numpy array containing integer corresponding to category of corresponding
    language index as hardcoded in classify_language function: 0 = degenerate, 1 = holistic, 2 = hybrid,
    3 = compositional, 4 = other (here I'm following the ordering used in the Kirby et al., 2015 paper; NOT the ordering
    from SimLang lab 21)
    """
    class_per_lang = np.zeros(len(language_list))
    for l in range(len(language_list)):
        class_per_lang[l] = classify_language(language_list[l], complete_forms, meaning_list)
    return class_per_lang
```

[3]:
```
# Let's try out our create_all_possible_languages() function:

all_possible_languages = create_all_possible_languages(meanings, forms_without_noise)
# print("all_possible_languages are:")
# print(all_possible_languages)
print("number of possible languages is:")
print(len(all_possible_languages))
```

```
number of possible languages is:
256
```

[4]:
```
# Let's test our classify_language() function using some example languages
# from the Kirby et al. (2015) paper (and the numbering of classes used
# in SimLang lab 21):

degenerate_lang = ('aa', 'aa', 'aa', 'aa')
print('')
print("degenerate_lang is:")
print(degenerate_lang)
class_degenerate_lang = classify_language(degenerate_lang, forms_without_noise, meanings)
```

```
print("class_degenerate_lang is:")
print(class_degenerate_lang)

holistic_lang = ('aa', 'ab', 'bb', 'ba')
print('')
print("holistic_lang is:")
print(holistic_lang)
class_holistic_lang = classify_language(holistic_lang, forms_without_noise,␣
 ↪meanings)
print("class_holistic_lang is:")
print(class_holistic_lang)

other_lang = ('aa', 'aa', 'aa', 'ab')
print('')
print("other_lang is:")
print(other_lang)
class_other_lang = classify_language(other_lang, forms_without_noise, meanings)
print("class_other_lang is:")
print(class_other_lang)

compositional_lang = ('aa', 'ab', 'ba', 'bb')
print('')
print("compositional_lang is:")
print(compositional_lang)
class_compositional_lang = classify_language(compositional_lang,␣
 ↪forms_without_noise,
                                              meanings)
print("class_compositional_lang is:")
print(class_compositional_lang)
```

```
degenerate_lang is:
('aa', 'aa', 'aa', 'aa')
class_degenerate_lang is:
0

holistic_lang is:
('aa', 'ab', 'bb', 'ba')
class_holistic_lang is:
2

other_lang is:
('aa', 'aa', 'aa', 'ab')
class_other_lang is:
4

compositional_lang is:
```

```
('aa', 'ab', 'ba', 'bb')
class_compositional_lang is:
3
```

Let's start with implementing the production function. Here is the equation from Kirby et al. (2015) again:

$$P(f \mid l, t) \propto \begin{cases} (\frac{1}{a})^{\gamma}(1-\epsilon) & \text{if } t \text{ is mapped to } f \text{ in } l \\ \frac{\epsilon}{|F|-1} & \text{if } t \text{ is not mapped to } f \text{ in } l \end{cases}$$

where f stands for form (in the sense of a complete signal, such as aa), l stands for language, t stands for topic, and ϵ stands for production error.

[5]:
```
# A reproduction of the production function of Kirby et al. (2015):

# Now let's define a function that calculates the probabilities of producing␣
↪each of the possible forms_without_noisy_
# variants, given a particular language and topic:
def production_likelihoods_kirby_et_al(language, topic, ambiguity_penalty,␣
↪error_prob):
    """
    Calculates the production probabilities for each of the possible␣
↪forms_without_noisy_variants given a language and
    topic, as defined by Kirby et al. (2015)

    :param language: list of forms_without_noisy_variants that has same length␣
↪as list of meanings (global variable),
    where each form is mapped to the meaning at the corresponding index
    :param topic: the index of the topic (corresponding to an index in the␣
↪globally defined meaning list) that the
    speaker intends to communicate
    :param ambiguity_penalty: parameter that determines the strength of the␣
↪penalty on ambiguity (gamma)
    :param error_prob: the probability of making an error in production
    :return: 1D numpy array containing a production probability for each␣
↪possible form (where the index of the
    probability corresponds to the index of the form in the global variable␣
↪"forms_without_noisy_variants")
    """
    for m in range(len(meanings)):
        if meanings[m] == topic:
            topic_index = m
    correct_form = language[topic_index]
    ambiguity = 0
    for f in language:
        if f == correct_form:
            ambiguity += 1
```

```
    prop_to_prob_correct_form = ((1./ambiguity) ** ambiguity_penalty) * (1. -␣
↪error_prob)
    prop_to_prob_error_form = error_prob / (len(forms_without_noise) - 1)
    prop_to_prob_per_form_array = np.zeros(len(forms_without_noise))
    for i in range(len(forms_without_noise)):
        if forms_without_noise[i] == correct_form:
            prop_to_prob_per_form_array[i] = prop_to_prob_correct_form
        else:
            prop_to_prob_per_form_array[i] = prop_to_prob_error_form
    return prop_to_prob_per_form_array
```

1.7 Now let's implement noisy production:

Let's do this by creating a variant of the production_likelihoods_kirby_et_al() function above, where we add the possibility of noisy variants ('a_' etc.), and assigns them a likelihood as well (depending on the settings of the parameters 'noise' and 'noise_prob'):

[6]:
```
def create_noisy_variants(form):
    """
    Takes a form and generates all its possible noisy variants. NOTE however␣
↪that in its current form, this function
    only creates noisy variants in which only a single element of the original␣
↪form is replaced with a blank! (So it
    creates for instance 'a_' and '_b', but not '__'.)

    :param form: a form (string)
    :return: a list of possible noisy variants of that form
    """
    noisy_variant_list = []
    for i in range(len(form)):
        noisy_variant = form[:i] + '_' + form[i+1:]
        # Instead of string slicing, another way of doing this would be to␣
↪convert the string into a list, replace the
        # element at the ith index, and then convert it back into a string␣
↪using the 'join' method,
        # see: https://www.quora.com/
↪How-do-you-change-one-character-in-a-string-in-Python
        noisy_variant_list.append(noisy_variant)
    return noisy_variant_list



# we also need a function that removes every instance of a given element from a␣
↪list (to use for
# removing the 'correct' forms_without_noisy_variants from a list of possible␣
↪forms_without_noisy_variants for a given
```

```
# topic:
def remove_all_instances(my_list, element_to_be_removed):
    """
    Takes a list, and removes all instances of a given element from it

    :param my_list: a list
    :param element_to_be_removed: the element to be removed; can be of any type
    :return: the list with all instances of the target element removed
    """
    i = 0  # loop counter
    length = len(my_list)  # list length
    while i < len(my_list):
        if my_list[i] == element_to_be_removed:
            my_list.remove(my_list[i])
            # as an element is removed
            # so decrease the length by 1
            length = length - 1
            # run loop again to check element
            # at same index, when item removed
            # next item will shift to the left
            continue
        i = i + 1
    return my_list



def production_likelihoods_with_noise(language, topic, ambiguity_penalty,␣
↪error_prob, prob_of_noise):
    """
    Calculates the production probabilities for each of the possible forms␣
↪(including both forms without noise and all
    possible noisy variants) given a language and topic, and the probability of␣
↪environmental noise

    :param language: list of forms that has same length as list of meanings␣
↪(global variable), where each form is
    mapped to the meaning at the corresponding index
    :param topic: the index of the topic (corresponding to an index in the␣
↪globally defined meaning list) that the
    speaker intends to communicate
    :param ambiguity_penalty: parameter that determines the strength of the␣
↪penalty on ambiguity (gamma)
    :param error_prob: the probability of making an error in production
    :param prob_of_noise: the probability of environmental noise masking part␣
↪of the utterance
```

```
    :return: 1D numpy array containing a production probability for each␣
↪possible form (where the index of the
    probability corresponds to the index of the form in the global variable␣
↪"all_forms_including_noisy_variants")
    """
    for m in range(len(meanings)):
        if meanings[m] == topic:
            topic_index = m
    correct_form = language[topic_index]
    error_forms = list(forms_without_noise)  # This may seem a bit weird, but a␣
↪speaker should be able to produce *any*
    # form as an error form right? Not limited to only the other forms that␣
↪exist within their language? (Otherwise a
    # speaker with a degenerate language could never make a production error).
    error_forms = remove_all_instances(error_forms, correct_form)
    if len(error_forms) == 0:  # if the list of error_forms is empty because␣
↪the language is degenerate
        error_forms = language  # simply choose an error_form from the whole␣
↪language
    noisy_variants_correct_form = create_noisy_variants(correct_form)
    noisy_variants_error_forms = []
    for error_form in error_forms:
        noisy_variants = create_noisy_variants(error_form)
        noisy_variants_error_forms = noisy_variants_error_forms+noisy_variants
    ambiguity = 0
    for f in language:
        if f == correct_form:
            ambiguity += 1
    prop_to_prob_correct_form_complete = ((1./ambiguity) ** ambiguity_penalty)␣
↪* (1. - error_prob) * (1 - prob_of_noise)
    prop_to_prob_error_form_complete = error_prob / (len(forms_without_noise) -␣
↪1) * (1 - prob_of_noise)
    prop_to_prob_correct_form_noisy = ((1. / ambiguity) ** ambiguity_penalty) *␣
↪(1. - error_prob) * (prob_of_noise / len(noisy_forms))
    prop_to_prob_error_form_noisy = error_prob / (len(forms_without_noise) - 1)␣
↪* (1 - prob_of_noise) * (prob_of_noise / len(noisy_forms))
    prop_to_prob_per_form_array = np.
↪zeros(len(all_forms_including_noisy_variants))
    for i in range(len(all_forms_including_noisy_variants)):
        if all_forms_including_noisy_variants[i] == correct_form:
            prop_to_prob_per_form_array[i] = prop_to_prob_correct_form_complete
        elif all_forms_including_noisy_variants[i] in␣
↪noisy_variants_correct_form:
            prop_to_prob_per_form_array[i] = prop_to_prob_correct_form_noisy
        elif all_forms_including_noisy_variants[i] in␣
↪noisy_variants_error_forms:
```

```
            prop_to_prob_per_form_array[i] = prop_to_prob_error_form_noisy
        else:
            prop_to_prob_per_form_array[i] = prop_to_prob_error_form_complete
    return prop_to_prob_per_form_array
```

And finally, let's write a function that actually produces an utterance, given a language and a topic:

[7]:
```
def produce(language, topic, ambiguity_penalty, error_prob, noise_switch,␣
↪prob_of_noise):
    """
    Produces an actual utterance, given a language and a topic

    :param language: list of forms_without_noisy_variants that has same length␣
↪as list of meanings (global variable),
    where each form is mapped to the meaning at the corresponding index
    :param topic: the index of the topic (corresponding to an index in the␣
↪globally defined meaning list) that the
    speaker intends to communicate
    :param ambiguity_penalty: parameter that determines the strength of the␣
↪penalty on ambiguity (gamma)
    :param error_prob: the probability of making an error in production
    :param noise_switch: turns noise on when set to True, and off when set to␣
↪False
    :param prob_of_noise: the probability of noise happening (only relevant␣
↪when noise_switch is set to True);
    corresponds to global variable 'noise_prob'
    :return: an utterance. That is, a single form chosen from either the global␣
↪variable "forms_without_noise" (if
    noise is False) or the global variable "all_forms_including_noisy_variants"␣
↪(if noise is True).
        """
    if noise_switch:
        prop_to_prob_per_form_array =␣
↪production_likelihoods_with_noise(language, topic, ambiguity_penalty,␣
↪error_prob, prob_of_noise)
        prob_per_form_array = np.divide(prop_to_prob_per_form_array, np.
↪sum(prop_to_prob_per_form_array))
        utterance = np.random.choice(all_forms_including_noisy_variants,␣
↪p=prob_per_form_array)
    else:
        prop_to_prob_per_form_array =␣
↪production_likelihoods_kirby_et_al(language, topic, ambiguity_penalty,␣
↪error_prob)
        prob_per_form_array = np.divide(prop_to_prob_per_form_array, np.
↪sum(prop_to_prob_per_form_array))
        utterance = np.random.choice(forms_without_noise, p=prob_per_form_array)
```

```
    return utterance
```

Let's test our production function:

[8]:
```
produce(compositional_lang, "02", gamma, error, True, noise_prob)
```

[8]:
```
'aa'
```

We also need a function that interprets an utterance (given a language):

[9]:
```
import random

def receive_without_repair(language, utterance):
    """
    Takes a language and an utterance, and returns an interpretation of that␣
↪utterance, following the language

    :param language: list of forms_without_noisy_variants that has same length␣
↪as list of meanings (global variable),
    where each form is mapped to the meaning at the corresponding index
    :param utterance: a form (string)
    :return: an interpretation (string)
    """
    possible_interpretations = []
    for i in range(len(language)):
        if language[i] == utterance:
            possible_interpretations.append(meanings[i])
    if len(possible_interpretations) == 0:
        possible_interpretations = meanings
    interpretation = random.choice(possible_interpretations)
    return interpretation
```

Let's test our reception function:

[10]:
```
receive_without_repair(compositional_lang, 'aa')
```

[10]:
```
'02'
```

1.8 Let's implement our repair initiation function

Above we decided that repair should be initiated probabilistically, depending on the level of ambiguity that is left after having received the speaker's utterance. We should also implement the possibility of turning on or off an efficiency pressure which, when on, makes the listener decide to initiate repair not just based on ambiguity, but also on the costs of the type of repair initiator that is presumed to be necessary. To start us off, we'll start with an open request costing 2/3, a restricted request costing 1/3, and no repair costing 0.

Now we can write the function that does the actual receiving of a signal and deciding whether or not to do repair. There are two variables that play a role in the process of deciding whether

or not to initiate repair: the amount of ambiguity left after receiving the utterance, and the cost of the require repair initiator. The three possible responses a listener can give when receiving an utterance are:

1. no repair (i.e. just interpret the utterance using the receive_without_repair() function above)
2. restricted request (ask for clarification about part of the meaning)
3. open request (ask for clarification about the whole meaning

To start simple, let's say that these are the probabilities of initiating the three options, depending on the ambiguity u (I'm calling this u for uncertainty, because a for ambiguity already has a different meaning above, in Kirby et al.'s, 2015 production function), and the cost of the required repair initiator c:

$$P(r \mid I_u, c) \propto \begin{cases} \frac{1}{|I_u|} - c_r & \text{if } r \text{ is no repair} \\ (1 - \frac{1}{|I_u|}) - c_r & \text{if } r \text{ is a repair initiator} \end{cases}$$

where r stands for response, I_u for the set of possible interpretations of utterance u (i.e. the set of meanings that map to the set of complete forms that are compatible with the form that was received), and c_r for the cost of response r.

Which repair initiator is available depends on whether the listener has a partial grasp of the meaning or not. If so, the listener will always use a restricted request when they decide to go for repair, and if not, the listener will always use an open request.

To start off with, let's defined the cost vector c as: - c = 0.0 for no repair - c = 0.2 for restricted request - c = 0.4 for open request

(So the cost ratio of restricted:open is 1:2) Let's work through some examples to see how this definition of repair initiation works out.

Say the listener receives the signal 'a_'. Remember that the order of possible meanings that each language assumes is: ['02', '03', '12', '13'].

If the listener uses the following compositional language: ['aa', 'ab', 'ba', 'bb'], the set of possible interpretations I_u consists of ['02', '03']. Thus, $|I_u| = 2$. Furthermore, the signal 'a_' maps unambiguously to meaning '0_' in the listener's language, which means that the listener has partial grasp of the meaning. Therefore, the repair initiator that is available to the listener is the restricted request '0?'. Given our definition of the probabilities of initiating repair or not above (as well as the cost vector c), this means that initiating repair or not will have the following probabilities for this listener given this utterance:

$$P(repair \mid I_u, c) \propto (1 - (1/2)) - 0.2 P(norepair \mid I_u, c) \propto (1/2) - 0.0$$

Which, when normalized, yields the following probabilities:

[11]:
```
prop_to_prob_repair_compositional = (1.-(1./2.))-0.2
prop_to_prob_no_repair_compositional = (1./2.)-0.0

prop_to_probs_compositional = np.array([prop_to_prob_repair_compositional,␣
↪prop_to_prob_no_repair_compositional])
```

```
normalized_probs_compositional = np.divide(prop_to_probs_compositional, np.
 ↪sum(prop_to_probs_compositional))

print('P(repair | compositional language) =␣
 ↪'+str(normalized_probs_compositional[0]))
print('P(no repair | compositional language) =␣
 ↪'+str(normalized_probs_compositional[1]))
```

```
P(repair | compositional language) = 0.37499999999999994
P(no repair | compositional language) = 0.625
```

Now let's work through the same example, but with a listener using the following holistic language: ['aa', 'ba', 'ab', 'bb]. Remember that the order of meanings is: ['02', '03', '12', '13']. Given the same utterance, 'a_', the set of possible interpretations I_u now consists of ['02', '12']. Thus, just like for the compositional language, $|I_u| = 2$. However, signal 'a_' does not map unambiguously to any partial meaning in this holistic language. Therefore, the repair initiator that is available to the listener is the open request '??'. Given our definition of the probabilities of initiating repair or not above (as well as the cost vector c), this means that initiating repair or not will have the following probabilities for this listener given this utterance:

$$P(repair \mid I_u, c) \propto (1 - (1/2)) - 0.4 P(norepair \mid I_u, c) \propto (1/2) - 0.0$$

Which, when normalized, yields the following probabilities:

[12]:
```
prop_to_prob_repair_holistic = (1.-(1./2.))-0.4
prop_to_prob_no_repair_holistic = (1./2.)-0.0

prop_to_probs_holistic = np.array([prop_to_prob_repair_holistic,␣
 ↪prop_to_prob_no_repair_holistic])
normalized_probs_holistic = np.divide(prop_to_probs_holistic, np.
 ↪sum(prop_to_probs_holistic))

print('P(repair | holistic language) = '+str(normalized_probs_holistic[0]))
print('P(no repair | holistic language) = '+str(normalized_probs_holistic[1]))
```

```
P(repair | holistic language) = 0.16666666666666663
P(no repair | holistic language) = 0.8333333333333334
```

So given the holistic language, initiating repair is a bit less likely (see below exactly how much less likely), because the required repair initiator is more costly.

[13]:
```
print("P(repair | compositional language) / P(repair | holistic language) = ")
normalized_probs_compositional[0] / normalized_probs_holistic[0]
```

```
P(repair | compositional language) / P(repair | holistic language) =
```

[13]: 2.25

Now let's consider an example language of the 'other' category: ['aa', 'aa', 'ab', 'bb]. Remember that the order of meanings is: ['02', '03', '12', '13']. Given the same utterance, 'a_', the set of possible interpretations I_u thus consists of ['02', '03', '12']. So $|I_u| = 3$. Signal 'a_' does not map unambiguously to any partial meaning in this holistic language, which means that again the listener can only use an open request repair initiator ('??'). This means that initiating repair or not will have the following probabilities for this listener given this utterance:

$$P(repair \mid I_u, c) \propto (1 - (1/3)) - 0.4 P(norepair \mid I_u, c) \propto (1/3) - 0.0$$

Which, when normalized, yields the following probabilities:

[14]:
```
prop_to_prob_repair_other = (1.-(1./3.))-0.4
prop_to_prob_no_repair_other = (1./3.)-0.0

prop_to_probs_other = np.array([prop_to_prob_repair_other,␣
↪prop_to_prob_no_repair_other])
normalized_probs_other = np.divide(prop_to_probs_other, np.
↪sum(prop_to_probs_other))

print('P(repair | other language) = '+str(normalized_probs_other[0]))
print('P(no repair | other language) = '+str(normalized_probs_other[1]))
```

```
P(repair | other language) = 0.4444444444444445
P(no repair | other language) = 0.5555555555555555
```

Finally, let's consider an example degenerate language: ['aa', 'aa', 'aa', 'aa']. Remember that the order of meanings is: ['02', '03', '12', '13']. Given the same utterance, 'a_', the set of possible interpretations I_u thus consists of ['02', '03', '12', '13']. So $|I_u| = 4$. Signal 'a_' does not map unambiguously to any partial meaning in this holistic language, which means that again the listener can only use an open request repair initiator ('??'). This means that initiating repair or not will have the following probabilities for this listener given this utterance:

$$P(repair \mid I_u, c) \propto (1 - (1/4)) - 0.4 P(norepair \mid I_u, c) \propto (1/4) - 0.0$$

Which, when normalized, yields the following probabilities:

[15]:
```
prop_to_prob_repair_degenerate = (1.-(1./4.))-0.4
prop_to_prob_no_repair_degenerate = (1./4.)-0.0

prop_to_probs_degenerate = np.array([prop_to_prob_repair_degenerate,␣
↪prop_to_prob_no_repair_degenerate])
normalized_probs_degenerate = np.divide(prop_to_probs_degenerate, np.
↪sum(prop_to_probs_degenerate))

print('P(repair | degenerate language) = '+str(normalized_probs_degenerate[0]))
print('P(no repair | degenerate language) =␣
↪'+str(normalized_probs_degenerate[1]))
```

```
P(repair | degenerate language) = 0.5833333333333334
P(no repair | degenerate language) = 0.4166666666666667
```

Thus, maximal uncertainty is the only situation in which initiating repair becomes more likely than just making a guess.

Now, let's capture what we've defined above in an actual function:

First we need a function that takes a noisy form and works back to determine which possible complete forms it could have stemmed from:

[16]:
```
def noisy_to_complete_forms(noisy_form, complete_forms):
    """
    Takes a noisy form and returns all possible complete forms that it's␣
↪compatible with.

    :param noisy_form: a noisy form (i.e. a string containing '_' as at least␣
↪one of the characters)
    :param complete_forms: The full set of possible complete forms (corresponds␣
↪to global parameter
    'forms_without_noise')
    :return: A list of complete forms that the noisy form is compatible with
    """
    possible_complete_forms = []
    amount_of_noise = noisy_form.count('_')
    for complete_form in complete_forms:
        similarity_score = 0
        for i in range(len(noisy_form)):
            if noisy_form[i] == complete_form[i]:
                similarity_score += 1
        if similarity_score == len(complete_form)-amount_of_noise:
            possible_complete_forms.append(complete_form)
    return possible_complete_forms
```

Let's test our noisy_to_complete_forms() function:

[17]:
```
compatible_forms = noisy_to_complete_forms('a_', forms_without_noise)
print("compatible_forms are:")
print(compatible_forms)
```

```
compatible_forms are:
['aa', 'ab']
```

Now we need a function that, given a set of complete forms that are compatible with the noisy form that was received, determines which possible interpretations of the noisy signal are available

[18]:
```
def find_possible_interpretations(language, forms):
    """
    Finds all meanings that the forms given as input are mapped to in the␣
↪language given as input
```

```
    :param language: list of forms_without_noisy_variants that has same length␣
↪as list of meanings (global variable),
    where each form is mapped to the meaning at the corresponding index
    :param forms: list of forms
    :return: list of meanings (type: string) that the forms given as input are␣
↪mapped to in the language given as input
    """
    possible_interpretations = []
    for i in range(len(language)):
        if language[i] in forms:
            possible_interpretations.append(meanings[i])
    return possible_interpretations
```

Let's test our possible_interpretations() function:

[19]:
```
example_language = ['aa', 'ab', 'ba', 'bb']

possible_interpretations = find_possible_interpretations(example_language,␣
↪compatible_forms)
print("possible_interpretations are:")
print(possible_interpretations)
```

```
possible_interpretations are:
['02', '03']
```

Now we need a function that takes a language and a noisy form, and determines whether or not the noisy form maps unambiguously onto a partial meaning (i.e. whether the listener has at least a partial grasp of the meaning):

[20]:
```
def find_partial_meaning(language, noisy_form):
    """
    Checks whether the noisy_form given as input maps unambiguously to a␣
↪partial meaning in the language given as
    input, and if so, returns that partial meaning.

    :param language: list of forms_without_noisy_variants that has same length␣
↪as list of meanings (global variable),
    where each form is mapped to the meaning at the corresponding index
    :param noisy_form: a noisy form (i.e. a string containing '_' as at least␣
↪one of the characters)
    :return: a list containing the partial meaning that the noisy_form maps␣
↪unambiguously to, if there is one
    """
    part_meanings_as_ints = []
    for i in range(len(meanings)):
        for j in range(len(meanings[0])):
```

```
            part_meanings_as_ints.append(int(meanings[i][j]))
    max_part_meaning = max(part_meanings_as_ints)
    count_per_partial_meaning = np.zeros(max_part_meaning+1)
    for i in range(len(noisy_form)):
        if noisy_form[i] != '_':
            for j in range(len(language)):
                if language[j][i] == noisy_form[i]:
                    count_per_partial_meaning[int(meanings[j][i])] += 1
    n_features = 0
    for i in range(len(meanings)):
        if meanings[i][0] == meanings[0][0]:
            n_features += 1
    if np.sum(count_per_partial_meaning) == n_features:
        part_meaning_index = np.where(count_per_partial_meaning==n_features)[0]
    else:
        part_meaning_index = []
    if len(part_meaning_index) == 1:
        return part_meaning_index
    else:
        return []
```

Let's test our find_partial_meanings() function:

[21]:
```
example_lang = ['aa', 'ab', 'aa', 'bb']

partial_meaning = find_partial_meaning(example_lang, 'b_')
print('')
print("partial_meaning is:")
print(partial_meaning)
```

```
partial_meaning is:
[]
```

Now that we have all these functions in place, we can finally build our full receive_with_repair() function:

[22]:
```
def receive_with_repair(language, utterance, mutual_understanding_pressure,␣
↪minimal_effort_pressure):
    """
    Receives and utterance and gives a response, which can either be an␣
↪interpretation or a repair initiator. How likely
    these two response types are to happen depends on the settings of the␣
↪paremeters 'mutual_understanding' and
    'minimal_effort' (and, if minimal_effort is set to True, the parameter␣
↪'cost_vector'). These three parameters are
    all assumed to be global variables.
```

```
    :param language: list of forms_without_noisy_variants that has same length␣
↪as list of meanings (global variable),
    where each form is mapped to the meaning at the corresponding index
    :param utterance: an utterance (string)
    :param mutual_understanding_pressure: determines whether the pressure for␣
↪mutual understanding is switched on or off
    (i.e. set to True or False); corresponds to global variable␣
↪'mutual_understanding'
    :param minimal_effort_pressure: determines whether the pressure for minimal␣
↪effort is switched on or off
    (i.e. set to True or False); corresponds to global variable 'minimal_effort'
    :return: a response, which can either be an interpretation (i.e. meaning)␣
↪or a repair initiator. A repair initiator
    can be of two types: if the listener has grasped part of the meaning, it␣
↪will be a restricted request, which is a
    string containing the partial meaning that the listener did grasp, followed␣
↪by a question mark. If the listener did
    not grasp any of the meaning, it will be an open request, which is simply '?
↪?'
    """
    if not mutual_understanding_pressure and not minimal_effort_pressure:
        raise ValueError(
            "Sorry, this function has only been implemented for at least one of␣
↪either mutual_understanding or minimal_effort being True"
        )
    if '_' in utterance:
        compatible_forms = noisy_to_complete_forms(utterance,␣
↪forms_without_noise)
        possible_interpretations = find_possible_interpretations(language,␣
↪compatible_forms)
        if len(possible_interpretations) == 0:
            possible_interpretations = meanings
        partial_meaning = find_partial_meaning(language, utterance)
        if mutual_understanding_pressure and minimal_effort_pressure:
            prop_to_prob_no_repair = (1./
↪len(possible_interpretations))-cost_vector[0]
            if len(partial_meaning) == 1:
                prop_to_prob_repair = (1.-(1./
↪len(possible_interpretations)))-cost_vector[1]
                repair_initiator = str(partial_meaning[0])+'?'
            elif len(partial_meaning) == 0:
                prop_to_prob_repair = (1.-(1./
↪len(possible_interpretations)))-cost_vector[2]
                repair_initiator = '??'
        elif mutual_understanding_pressure and not minimal_effort_pressure:
```

```
            if len(possible_interpretations) > 1:
                prop_to_prob_no_repair = 0.
                prop_to_prob_repair = 1.
                if len(partial_meaning) == 1:
                    repair_initiator = str(partial_meaning[0])+'?'
                elif len(partial_meaning) == 0:
                    repair_initiator = '??'
            elif len(possible_interpretations) == 1:
                prop_to_prob_no_repair = 1.
                prop_to_prob_repair = 0.
        elif not mutual_understanding_pressure and minimal_effort_pressure:
            prop_to_prob_no_repair = 1.
            prop_to_prob_repair = 0.
            if len(partial_meaning) == 1:
                repair_initiator = str(partial_meaning[0])+'?'
            elif len(partial_meaning) == 0:
                repair_initiator = '??'
        prop_to_prob_per_response = np.array([prop_to_prob_no_repair,␣
↪prop_to_prob_repair])
        for i in range(len(prop_to_prob_per_response)):
            if prop_to_prob_per_response[i] < 0.0:
                prop_to_prob_per_response[i] = 0.0
        normalized_response_probs = np.divide(prop_to_prob_per_response, np.
↪sum(prop_to_prob_per_response))
        selected_response = np.random.choice(np.arange(2),␣
↪p=normalized_response_probs)
        if selected_response == 0:
            response = random.choice(possible_interpretations)
        elif selected_response == 1:
            response = repair_initiator
    else:
        response = receive_without_repair(language, utterance)
    return response
```

Now, let's test our receive_with_repair() function:

[23]:
```
# First, we'll need to set some parameters:

minimal_effort = True
mutual_understanding = True
cost_vector = [0.0, 0.2, 0.4]  # costs of no repair, restricted request, and␣
↪open request, respectively

example_lang = ['aa', 'ba', 'aa', 'bb']

for i in range(5):
    print('')
```

```
    print('i is:')
    print(i)
    response = receive_with_repair(example_lang, 'a_', mutual_understanding,␣
↪minimal_effort)
    print("response is:")
    print(response)
```

```
i is:
0
response is:
02

i is:
1
response is:
02

i is:
2
response is:
??

i is:
3
response is:
??

i is:
4
response is:
12
```

1.9 For the rest of the code, see evolution_compositionality_under_noise.py (i.e. the python file)

Which can be found in the following GitHub repository: https://github.com/marieke-woensdregt/repair_compositionality

1.9.1 References

Kirby, S., Tamariz, M., Cornish, H., & Smith, K. (2015). Compression and communication in the cultural evolution of linguistic structure. Cognition, 141, 87–102. https://doi.org/10.1016/j.cognition.2015.03.016