

A description of the algorithms for 'Integrated pragmatic values'

Christopher Potts

May 2, 2006

1 Basic description

Theoretical background The scripts implement the theory presented in Potts 2006b, Potts To appear, and Potts 2006a. The manuscript Potts 2005 is a predecessor, though it differs in substantive theoretical ways and was much less developed.

'Integrated pragmatic values' The user inputs a world-count, a proposition modeling the speaker's belief state, a proposition modeling the hearer's belief state, a contextual (quality) threshold C_τ , and a question under discussion. The output is a ranking of utterances according to felicity in that context. Even when the space of potential utterances is very large, the output set is typically small, since the theory performs a series of reductions to arrive at the felicitous region.

'Infer speaker beliefs' The user inputs a world-count, a contextual threshold C_τ , a question under discussion, and an utterance. The output is the set of belief states that, according to the theory, are consistent with what the speaker said and the assumption that he is playing by the pragmatic rules.

2 Integrated pragmatic values

2.1 The initial probability distribution

Both discourse participants begin (so to speak) in the state of complete ignorance, which is just an even distribution of probabilities over the entire world space.

```
(1) 1: initialize %initial_pd
    2: for  $n$  (1..world_count) do
    3:   %initial_pd{ $n$ } =  $\frac{1}{\text{world\_count}}$ 
    4: end for
```

Central data structure %initial_pd, a hash mapping numbers in the world count to numbers in $[0, 1]$

2.2 The space of propositions

Utterances are assumed to denote propositions, so the script operates with propositions. The user supplies a world-count n . The script generates the power-set of the set of numbers $\{1 \dots n\}$. That is the space of potential utterances.

Central data structure @ps, an array of arrays

2.3 Conditionalizing probability distributions

The speaker's and hearer's belief states are defined by the user as propositions. The script converts these into hashes, i.e., probability distributions:

```
(2) conditionalization( $p$ , %initial_pd)
    1: initialize %new_pd ▷ the output
    2: @isect = the intersection of  $p$  and the keys of %initial_pd
    3: for  $k$  in the keys of %initial_pd do
    4:   if  $k$  is a member of @isect then
    5:     %new_pd{ $k$ } =  $\frac{1}{|\text{@isect}|}$  ▷ even distribution over  $p$ 's members
    6:   else
    7:     %new_pd{ $k$ } = 0 ▷ non-members of  $p$  go to 0
    8:   end if
    9: end for
   10: return %new_pd
```

Central data structure %new_pd, a hash mapping numbers in the world count to numbers in $[0, 1]$. The input p is the speaker's belief state or the hearer's belief state:

- %speaker_pd = conditionalization(p , %initial_pd), where p is the user-supplied belief state (proposition) for the speaker
- %hearer_pd = conditionalization(p , %initial_pd), where p is the user-supplied belief state (proposition) for the hearer

2.4 Quality and quantity ratings

This portion of the code cycles through the set of propositions, @ps, determining the quality and quantity ratings for each:

```
(3) 1: initialize %quality_ratings
    2: initialize %quantity_ratings
    3: for  $p$  in @ps do
    4:    $P_S(p) = 0$  ▷ initialize speaker's probability for  $p$ 
    5:    $P_H(p) = 0$  ▷ initialize hearer's probability for  $p$ 
    6:   ▷ get the speaker and hearer probabilities for  $p$ 
    7:   for  $w$  in  $p$  do
    8:      $P_S(p) = P_S(p) + \text{\%speaker\_pd}\{w\}$ 
    9:      $P_H(p) = P_H(p) + \text{\%hearer\_pd}\{w\}$ 
    10:  end for
    11:  ▷ QUALITY
    12:  ▷ throw  $p$  out if its quality rating is below the standard
    13:  if  $P_S(p) > C_\tau$  then
    14:    %quality_ratings{ $p$ } =  $P_S(p)$  ▷ quality is speaker's probability
    15:  end if
    16:  ▷ QUANTITY
    17:  if  $P_H(p) > 0$  then
    18:    %quantity_ratings{ $p$ } =  $-\log_2(P_H(p))$  ▷ quantity is the  $-\log_2$ 
    19:  end if
    20: end for
```

Central data structures

- %quality_ratings, which maps propositions to their quality-ratings (defined only for propositions that are above the threshold C_τ)
- %quantity_ratings, which maps propositions to their quantity-ratings

2.5 Ans values

The Ans-value of a proposition p relative to a question Q is the number of propositions in Q with which p has a nonempty intersection. In the script, p is the utterance we are testing, and Q is the user-supplied question under discussion @qud.

```
(4)  relevance( $p$ ,  $Q$ )
      1: intersect_count = 0
      2: for  $q$  in  $Q$  do
      3:   if  $q \cap p \neq \emptyset$  then
      4:     increase intersect_count by 1
      5:   end if
      6: end for
      7: return %intersect_count
```

Central data structure This subroutine is used to build a hash %answer_values that maps each proposition to its answer value (i.e., its %intersect_count as determined in (4)).

2.6 Relevance ranking

There are two main steps. First, we sort the space of quality-rated utterances into equivalence classes based on their Ans-values, which are calculated by relevance (described in (4)). Second, we get rid of the members in this class that do not have the lowest quantity rating for that class.

```
(5)  Sort into equivalence classes based on Ans-values
      1: initialize %answer_values
      2: initialize %equivalence_classes
      3: for  $n$  in 1...world_count do
      4:   for  $p$  in the set of quality-rated utterances do
      5:     if %answer_values{ $p$ } =  $n$  then
      6:       put  $p$  into the array @equivalence_classes[ $n$ ]
      7:     end if
      8:   end for
      9: end for
```

Central data structure @equivalence_classes, a set of sets of equivalence classes of propositions

(6) **Get rid of all but the least informative members of each class**

```

1: for  $n$  in @equivalence_classes do
2:   initialize @quantity_class
3:   for  $p$  in @equivalence_classes[ $n$ ] do
4:     place quantity_ratings{ $p$ } in @quantity_class
5:   end for
6:   set min to the smallest element in @quantity_class
   ▷  $p$  is relevance-ranked iff  $p$ 's quantity rating is minimal in its class
7:   for  $p$  in @equivalence_classes[ $n$ ] do
8:     if quantity_ratings{ $p$ } is identical to min then
9:       %relevance_ranking{ $p$ } = %answer_values{ $p$ }
       ▷ relevance rankings are just answer values
10:    end if
11:  end for
12: end for

```

Central data structure %relevance_ranking, which contains just the minimally informative members of each Ans equivalence class

2.7 Final ranking (felicitous utterances)

In the end, we determine a ranking of the propositions that survived the initial quality cut (section 2.4, line 10) and then also made it through the relevance-ranking phase (section 2.6).

(7)

```

1: initialize %utterable
2: define  $f(p) = 1$  iff  $p$  has a quality rating
3: for  $p$  in the class of relevance-ranked things do
4:   if  $f(p) = 1$  then
5:     %utterable{ $p$ } = 1
6:   end if
7: end for
8: sort the keys of %utterable according to their quantity ratings

```

Central data structure %utterable. The sorted keys of %utterable are what the user is presented with (along with information about each of those keys).

3 Infer speaker beliefs

At first, we proceed exactly as above: generate the set of propositions, as in section 2.2, and create the initial probability distribution, as in section 2.1.

Important differences:

- The utterance is supplied this time. We don't care about quantity, so we get a single quality rating via a simplified version of the procedure in section 2.4.
- We need just one Ans-value (the one for our utterance) so we run the `relevance` subroutine (section 2.5) just once, and there is no need for the complicated sorting of section 2.6.
- However, we do cycle through the set of all propositions. Here, though, they are used to model the speaker's belief state. So, we cycle through, running the algorithm of section 2.3 to obtain the full set of potential belief states for the speaker.
- We then print out all the probability distributions P such that
 1. P assigns the speaker's utterance a value that is above the user-supplied threshold C_τ ; and
 2. the Ans-value of the speaker's utterance is less than or equal to the Ans-value of the proposition used to conditionalize P (this eliminates belief states in which the speaker did not answer as completely as he could have).

References

- Potts, Christopher. 2005. Integrated pragmatic values. URL <http://semanticsarchive.net/Archive/WFmYjRmM/>, Ms., UMass Amherst.
- Potts, Christopher. 2006a. Clausal implicatures via general pragmatic pressures. In Eric McCready, ed., *Proceedings of Logic and Engineering of Natural Language Semantics 2006*. Tokyo: Japan Society for the Promotion of Science.
- Potts, Christopher. 2006b. Integrated pragmatic values. URL <http://people.umass.edu/potts/talks/potts-pragmatic-values-brown.pdf>, Handout for a talk given at Brown University, March 6.
- Potts, Christopher. To appear. How far can pragmatic mechanisms take us? Commentary on truckenbrodt. *Theoretical Linguistics* URL <http://semanticsarchive.net/Archive/jViODayZ/>.