Bounded-rational theory of mind for conversational implicature

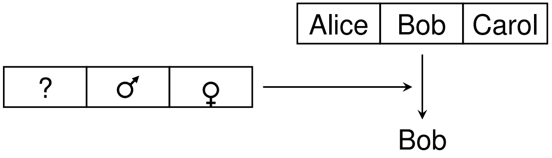
Oleg Kiselyov
FNMOC
oleg@pobox.com

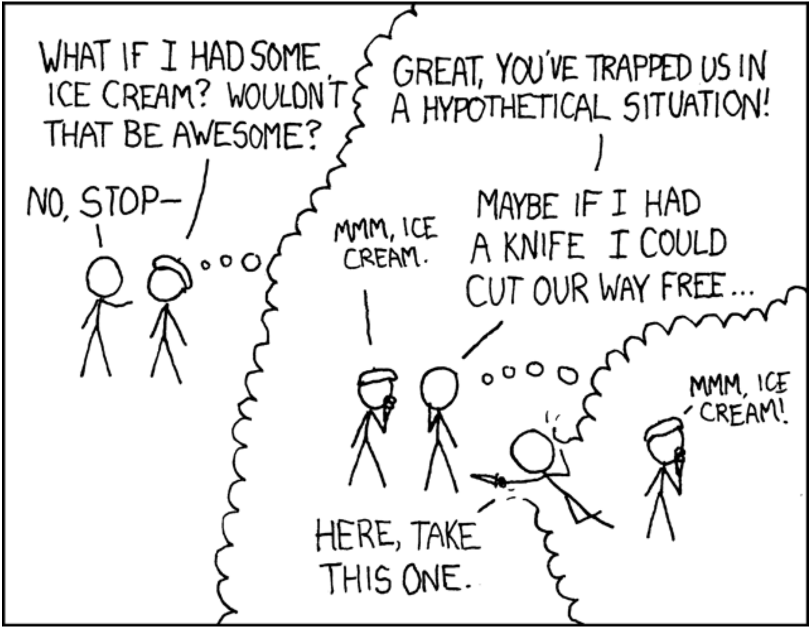
Chung-chieh Shan
Rutgers University
ccshan@rutgers.edu

Logical Methods for Discourse
December 15, 2009

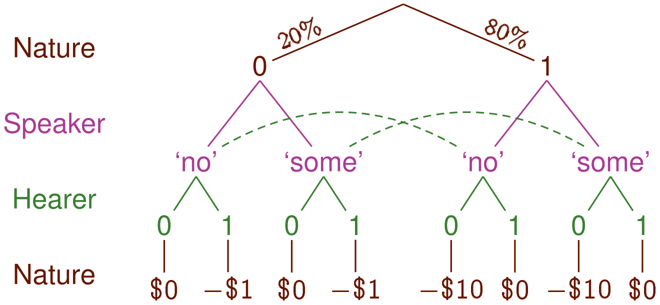
Layers, stages

Continuations when?

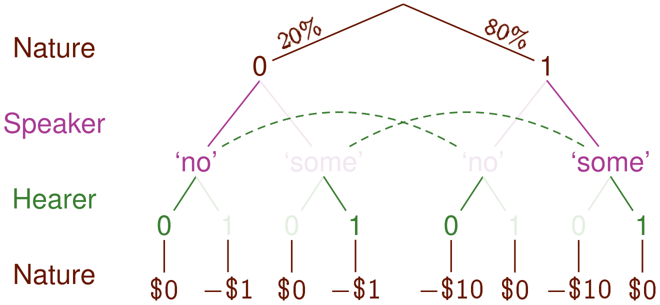
- A: I'll be Wild Bill.
  B: And I'll be Calamity Jane.
  A: Look, Calamity Jane, I've found a gold nugget.
  B: We're rich.
  A: Your dad is here now, so I guess you have to go.
- A: What kind of Scope does your mom use?
  B: What kind of soap?
  A: No, mouthwash; what kind of Scope?
  B: Oh, the regular kind.
- Bush complained about the 'utterly [inaudible] loudspeakers' in the room.
- 


WHAT IF I HAD SOME ICE CREAM? WOULDN'T THAT BE AWESOME?
NO, STOP–
GREAT, YOU'VE TRAPPED US IN A HYPOTHETICAL SITUATION!
MMM, ICE CREAM.
MAYBE IF I HAD A KNIFE I COULD CUT OUR WAY FREE...
MMM, ICE CREAM!
HERE, TAKE THIS ONE.

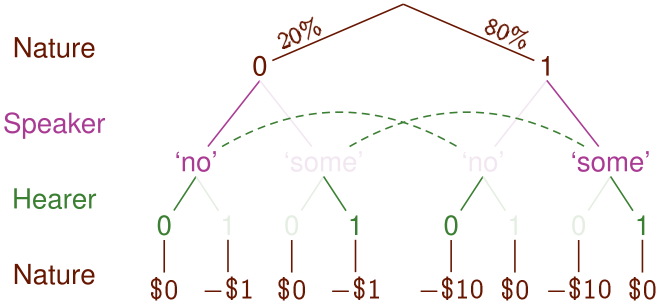
Game-theoretic pragmatics



Game-theoretic pragmatics



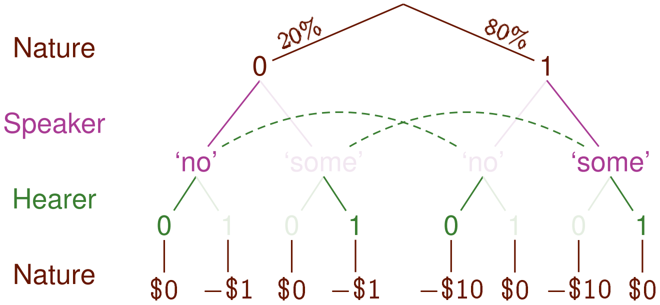
Game-theoretic pragmatics



Game	collaborative task	processing effort
↓		
Solution concept	perfect rationality	bounded rationality
↓		
Strategy	literal meaning	scalar implicature . . .

(Solving online? . . . offline?)

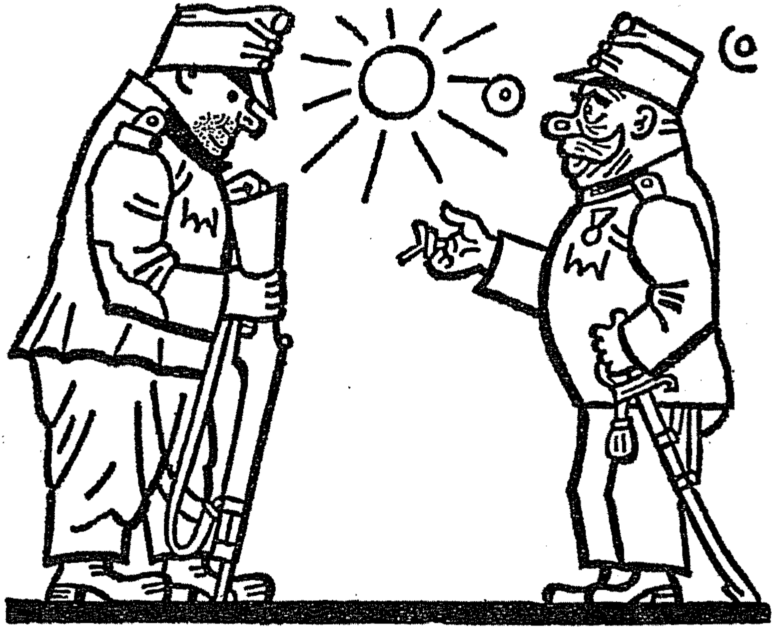
Game-theoretic pragmatics



Game	collaborative task	risk of misinterpretation
↓		
Solution concept	perfect rationality	bounded rationality
↓		
Strategy	literal meaning	scalar implicature . . .

(Solving online? . . . offline?)

The good soldier Švejk



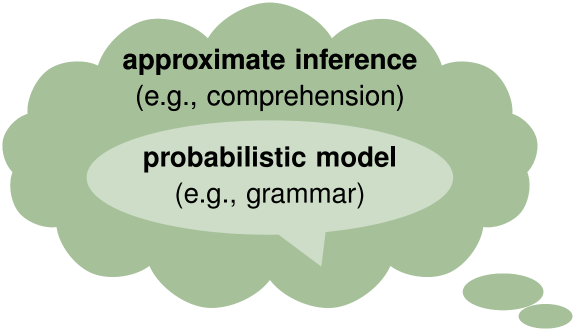
The good soldier Švejk

"The engine that you are to take off to the depot in Lysá nad Labem is no. 4268. Now pay careful attention. The first figure is four, the second is two, which means that you have to remember 42. That's twice two. That means that in the order of the figures 4 comes first. 4 divided by 2 makes 2 and so again you've got next to each other 4 and 2. Now, don't be afraid! What's twice 4? 8, isn't it? Well, then, get it into your head that 8 is the last in the series of figures in 4268. And now, when you've already got in your head that the first figure is 4, the second 2 and the fourth 8, all that's to be done is to be clever and remember the 6 which comes before the 8. And that's frightfully simple. The first figure is 4, the second is 2, and 4 and 2 are 6. So now you've got it: the second from the end is 6 and now we shall never forget the order of figures. You now have indelibly fixed in your mind the number 4268. But of course you can also reach the same result by an even simpler method . . ."

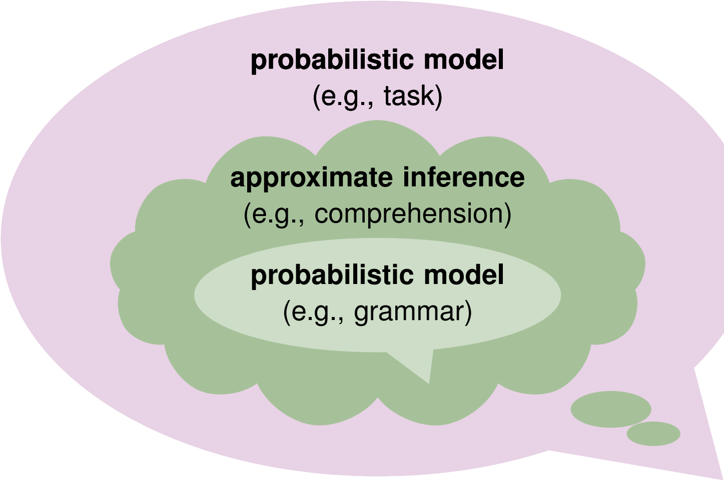
Grice and Marr

probabilistic model
(e.g., grammar)

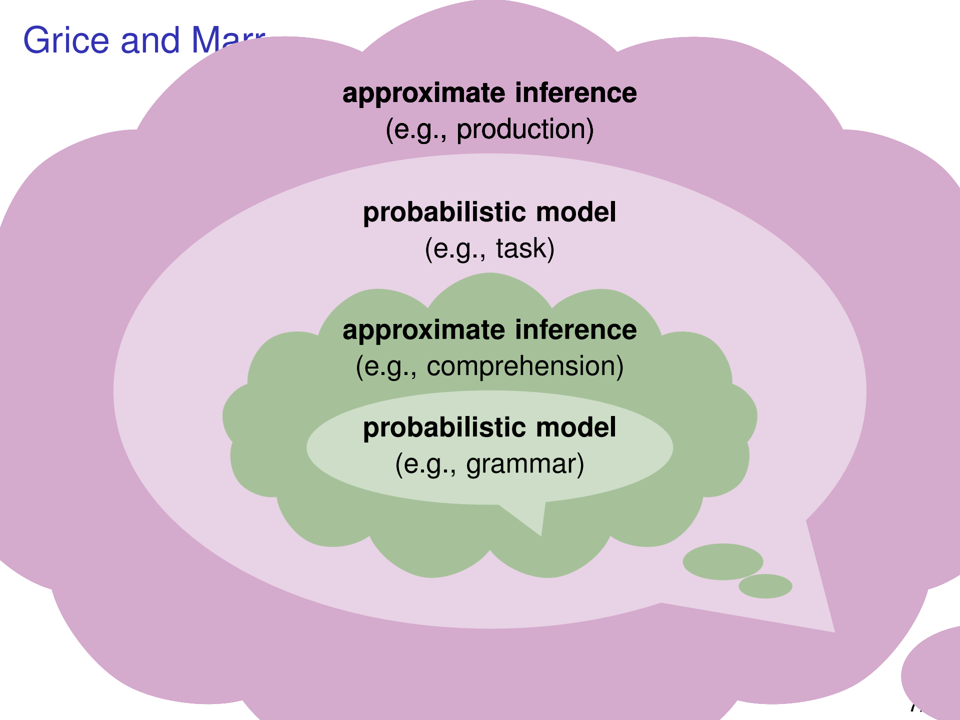
Grice and Marr



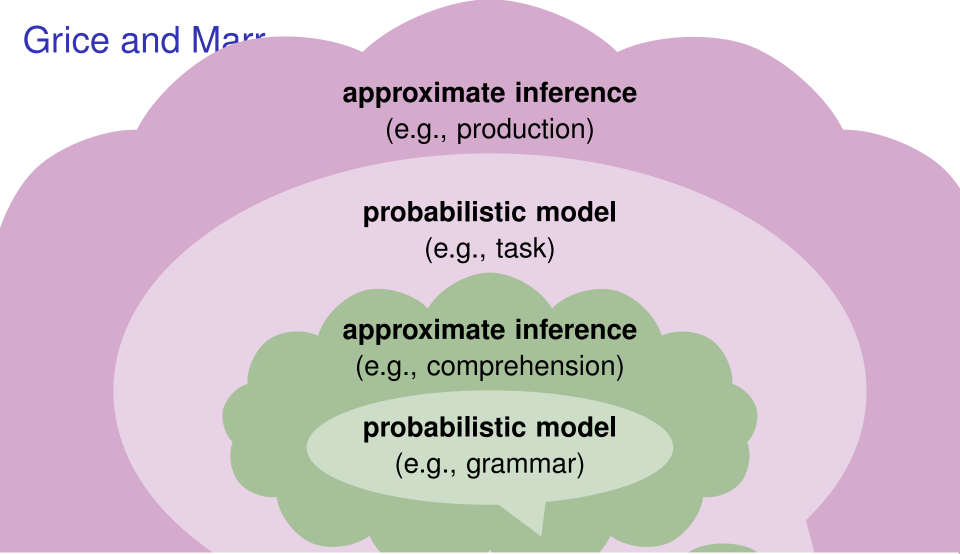
Grice and Marr



Grice and Marr



Grice and Marr



Probabilistic models invoke inference.
Random choices manipulate continuations.
Multiple layers track who thinks what.

Roadmap

Probabilistic models invoke inference.
Random choices manipulate continuations.
Multiple layers track who thinks what.

- Probabilistic models
- Inference algorithms
- The hearer's program
- The speaker's program

We have a hammer. (Nails: anaphora? vagueness? . . .)

Probabilistic models

Program	Type	Denotation	Operation
`flip`	$\overline{n}$	$\lambda c.\,\lambda g.\ \underset{c(0)(g)}{\overset{50\%}{\swarrow}}\ \underset{c(1)(g)}{\overset{50\%}{\searrow}}$	fork server

$$\overline{A} \overset{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Probabilistic models

Program	Type	Denotation	Operation
`flip`	$\overline{n}$	$\lambda c.\, \lambda g.\ \overset{50\% \qquad 50\%}{c(0)(g) \quad c(1)(g)}$	fork server
`+`	$n \to n \to n$	$\lambda x.\, \lambda y.\, x + y$	primitive

$$\overline{A} \stackrel{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Probabilistic models

Program	Type	Denotation	Operation
`flip`	$\overline{n}$	$\lambda c.\, \lambda g.$ 50% $c(0)(g)$ 50% $c(1)(g)$	fork server
`+`	$n \to n \to n$	$\lambda x.\, \lambda y.\, x + y$	primitive
`flip + flip`	$\overline{n}$	$\lambda c.\, \lambda g.$ 50% (50% $c(0)(g)$ 50% $c(1)(g)$) 50% (50% $c(1)(g)$ 50% $c(2)(g)$)	

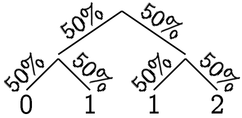
$$\overline{A} \stackrel{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Probabilistic models

Program	Type	Denotation	Operation
`flip`	$\overline{n}$	$\lambda c.\,\lambda g.\ c(0)(g) \quad c(1)(g)$	fork server
`+`	$n \to n \to n$	$\lambda x.\,\lambda y.\,x + y$	primitive
`flip + flip`	$\overline{n}$	$\lambda c.\,\lambda g.\ c(0)(g)\ c(1)(g)\ c(1)(g)\ c(2)(g)$	
`Lower`	$\overline{A} \to \mathsf{tree}\ A$	$\lambda m.\,m(\lambda v.\,\lambda g.\,v)(\emptyset)$	new thread

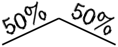
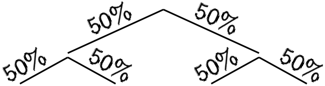
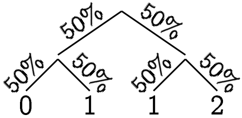
$$\overline{A} \stackrel{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Probabilistic models

Program	Type	Denotation	Operation
`flip`	$\overline{n}$	$\lambda c.\,\lambda g.\ c(0)(g)\quad c(1)(g)$	fork server
`+`	$n \to n \to n$	$\lambda x.\,\lambda y.\,x + y$	primitive
`flip + flip`	$\overline{n}$	$\lambda c.\,\lambda g.\ c(0)(g)\ c(1)(g)\ c(1)(g)\ c(2)(g)$	
`Lower`	$\overline{A} \to \mathtt{tree}\,A$	$\lambda m.\,m(\lambda v.\,\lambda g.\,v)(\emptyset)$	new thread
`Lower(flip + flip)`	$\mathtt{tree}\,n$		

$$\overline{A} \stackrel{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Probabilistic models

Program	Type	Denotation	Operation
`flip`	$\overline{n}$	 $\lambda c.\, \lambda g.\ c(0)(g) \quad c(1)(g)$	fork server
`+`	$n \to n \to n$	$\lambda x.\, \lambda y.\, x + y$	primitive
`flip + flip`	$\overline{n}$	 $\lambda c.\, \lambda g.\ c(0)(g)\ c(1)(g)\ c(1)(g)\ c(2)(g)$	
`Lower`	$\overline{A} \to \mathtt{tree}\, A$	$\lambda m.\, m(\lambda v.\, \lambda g.\, v)(\emptyset)$	new thread
`Lower(flip + flip)`	$\mathtt{tree}\, n$		
`ExactExpect`	$\mathtt{tree}\, n \to n$		enumerate tree leaves

$$\overline{A} \stackrel{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Perceptual observations

Program	Type	Denotation	Operation
`fail`	$\overline{A}$	$\lambda c.\, \lambda g.\, \mathtt{empty\ tree}$	exit server

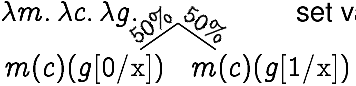
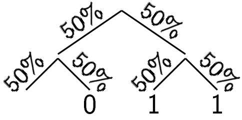
$$\overline{A} \overset{\text{def}}{=} (A \to \mathtt{assignment} \to \mathtt{tree}) \to \mathtt{assignment} \to \mathtt{tree}$$

Perceptual observations

Program	Type	Denotation	Operation
`fail`	$\overline{A}$	$\lambda c.\,\lambda g.\,\texttt{empty tree}$	exit server
`x`	$\overline{n}$	$\lambda c.\,\lambda g.\,c(g(\mathtt{x}))(g)$	get var
`x := flip;`	$\overline{A} \to \overline{A}$	$\lambda m.\,\lambda c.\,\lambda g.\, \underset{m(c)(g[0/\mathtt{x}]) \qquad m(c)(g[1/\mathtt{x}])}{\overset{50\% \quad 50\%}{\wedge}}$	set var

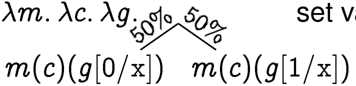
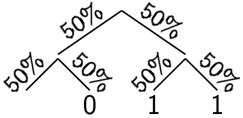
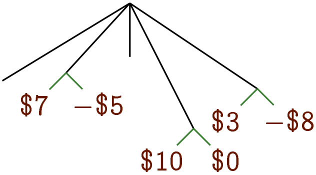
$$\overline{A} \stackrel{\text{def}}{=} (A \to \texttt{assignment} \to \texttt{tree}) \to \texttt{assignment} \to \texttt{tree}$$

Perceptual observations

Program	Type	Denotation	Operation
`fail`	$\overline{A}$	$\lambda c.\, \lambda g.\, \texttt{empty tree}$	exit server
`x`	$\overline{n}$	$\lambda c.\, \lambda g.\, c(g(\mathtt{x}))(g)$	get var
`x := flip;`	$\overline{A} \to \overline{A}$	$\lambda m.\, \lambda c.\, \lambda g.$ 	set var
`Lower (x := flip; y := flip; if x ∨ y then x else fail)`	$\texttt{tree}\, n$		

$$\overline{A} \overset{\text{def}}{=} (A \to \texttt{assignment} \to \texttt{tree}) \to \texttt{assignment} \to \texttt{tree}$$

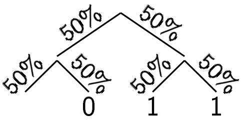
Perceptual observations

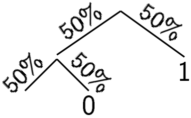
Program	Type	Denotation	Operation
`fail`	$\overline{A}$	$\lambda c.\, \lambda g.\, \text{empty tree}$	exit server
`x`	$\overline{n}$	$\lambda c.\, \lambda g.\, c(g(\mathrm{x}))(g)$	get var
`x := flip;`	$\overline{A} \to \overline{A}$	$\lambda m.\, \lambda c.\, \lambda g.$ 	set var
`Lower (x := flip; y := flip; if x ∨ y then x else fail)`	tree n		
`Lower (w := ...; if w ⊨ u then a := act;` $U(\mathrm{a} \mid \mathrm{w})$ `else fail)`	tree u		

$$\overline{A} \overset{\text{def}}{=} (A \to \text{assignment} \to \text{tree}) \to \text{assignment} \to \text{tree}$$

More tractable inference

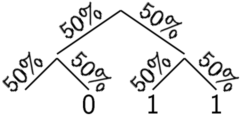
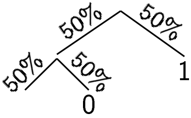
Program	Type	Denotation	Operation

`Lower`
`(x := flip; y := flip;`
` if x ∨ y then x`
` else fail)`
	$\mathsf{tree}\, n$		

			lazy evaluation (branching heuristic)

$$\overline{A} \overset{\text{def}}{=} (A \to \mathsf{assignment} \to \mathsf{tree}) \to \mathsf{assignment} \to \mathsf{tree}$$

More tractable inference

Program	Type	Denotation	Operation
`Lower (x := flip; y := flip; if x ∨ y then x else fail)`	tree n		
			lazy evaluation (branching heuristic)
`ExactExpect`	tree $n \to n$		enumerate tree leaves
`ApproxExpect`	tree $n \to \overline{n}$		sample tree leaves

$$\overline{A} \overset{\text{def}}{=} (A \to \mathrm{assignment} \to \mathrm{tree}) \to \mathrm{assignment} \to \mathrm{tree}$$

The bounded-rational hearer's program

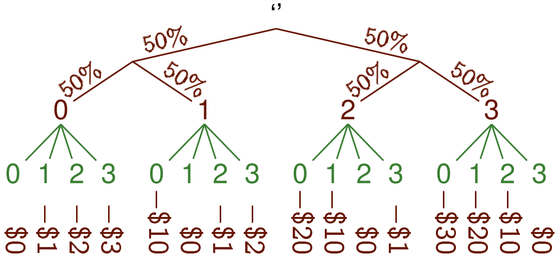
```
ApproxExpect
(Lower(count := 2 * flip + flip;
       conjunction := flip;
       if count,conjunction ⊨ some,not_all
       then a := act; U(a | count)
       else fail))
```

The bounded-rational hearer's program

```
ApproxExpect
(Lower(count := 2 * flip + flip;
       conjunction := flip;
       if ((some ∧ not_all) → conjunction)
          ∧ (some → count > 0) ∧ (not_all → count < 3)
       then a := act; U(a | count)
       else fail))
```

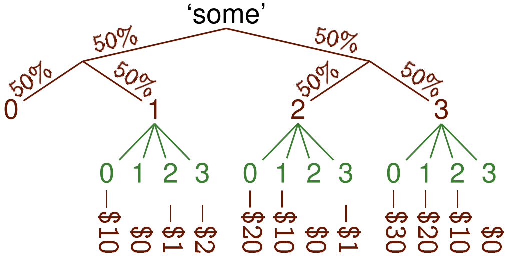
The bounded-rational hearer's program

```
ApproxExpect
(Lower(count := 2 * flip + flip;
       conjunction := flip;
       if ((some ∧ not_all) → conjunction)
           ∧ (some → count > 0) ∧ (not_all → count < 3)
       then a := act; U(a | count)
       else fail))
```



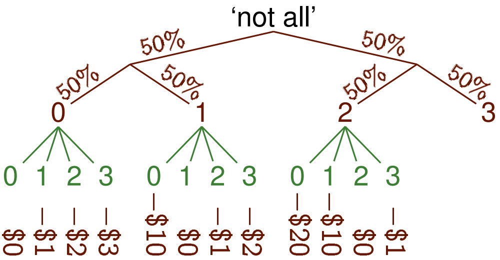
The bounded-rational hearer's program

```
ApproxExpect
(Lower(count := 2 * flip + flip;
       conjunction := flip;
       if ((some ∧ not_all) → conjunction)
          ∧ (some → count > 0) ∧ (not_all → count < 3)
       then a := act; U(a | count)
       else fail))
```



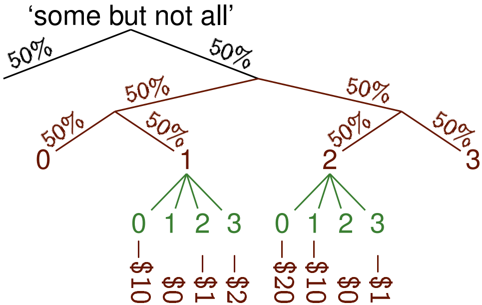
The bounded-rational hearer's program

```
ApproxExpect
(Lower(count := 2 * flip + flip;
       conjunction := flip;
       if ((some ∧ not_all) → conjunction)
           ∧ (some → count > 0) ∧ (not_all → count < 3)
       then a := act; U(a | count)
       else fail))
```



The bounded-rational hearer's program

```
ApproxExpect
(Lower(count := 2 * flip + flip;
       conjunction := flip;
       if ((some ∧ not_all) → conjunction)
           ∧ (some → count > 0) ∧ (not_all → count < 3)
       then a := act; U(a | count)
       else fail))
```



Going meta

The hearer

- believes utterance is grammatical and true
  (constrains unobserved random variables)
- desires to maximize expected utility
- **processes complex utterances less accurately because they trigger more constraints** (e.g., ‘but’ deepens tree)

Going meta

The hearer

- believes utterance is grammatical and true
  (constrains unobserved random variables)
- desires to maximize expected utility
- **processes complex utterances less accurately because they trigger more constraints** (e.g., ‘but’ deepens tree)

The speaker

- believes private world knowledge
- desires to maximize expected utility
- **trades off informativity against complexity**
  (e.g., omission, white lies)

Going meta

The hearer

- believes utterance is grammatical and true
  (constrains unobserved random variables)
- desires to maximize expected utility
- **processes complex utterances less accurately because they trigger more constraints** (e.g., ‘but’ deepens tree)

The speaker

- believes private world knowledge
- desires to maximize expected utility
- **trades off informativity against complexity**
  (e.g., omission, white lies)

The linguist

- **invokes inference algorithms in probabilistic models**
  (but can abstract; e.g., layperson model of meteorologist)
- **programs in an intuitive and expressive language**

Roadmap

Probabilistic models invoke inference.
Random choices manipulate continuations.
Multiple layers track who thinks what.

- Probabilistic models
- Inference algorithms
- The hearer's program
- The speaker's program

We have a hammer. (Nails: anaphora? vagueness? ...)

`http://okmij.org/ftp/kakuritu/`
`http://okmij.org/ftp/kakuritu/incite.ml`