Ling 5801: Lecture Notes 20
Linguistic meaning and lambda calculus

Contents

We can build interfaces using gist semantics (transformers), but they hallucinate (confabulate).

To interpret linguistic meanings precisely, interfaces probably need logic (e.g. lambda calculus).

20.1 Basic parts of meanings (ontology)

Linguistic meaning is usually defined over entities, truth values and functions from types to types:

1. Sentences express **propositions** that evaluate to a **truth value** (type `t`):

   E.g. True, False.

2. These propositions may also involve **entities** (type `e`):

   E.g. Kim, History101, etc., like the entities in a database.

3. Propositions may also involve **functions** (type $\alpha \to \beta$ from α to β):

   E.g. OnDuty of type `e` $\to$ `t`, Teach of type `e` $\to$ `e` $\to$ `t`, etc., like relation tables in a database.

We also want a type for hypothetical states of entire databases:

4. Propositions can be used to define **possible world states** (type `s`):

   E.g. the department faculty today, the department faculty if they hire in syntax, etc.

20.2 World models: collections of listeners' associations

Formally, we model language as transmitting associations from speakers' to listeners' minds.

We model listeners'/speakers' minds as **world models** – collections of associations about the world.

A world model is a set of **possible world states** w, which define:

1. a **domain** D^w_α for each type α – a (possibly infinite) set of instances of that type in w;
   for example, in a human resources model, the domain of entities D^w_e may be **Kim** and **Pat**
   (domains for functions are all possible mappings between domains of input and output types);
2. an **interpretation function** $[\![\varphi]\!]^w$ – associating logical expressions φ into these domains;
   for example, the interpretation $[\![\text{OnDuty}]\!]^w$ may be the association/table:

input	output
Kim	: **False**
Pat	: **True**

   We mostly define functions in world models and get sentence meanings via composition rules.
   An interpretation function is itself an association from logical expressions to mental objects.

World states are **complete**. Listeners with incomplete knowledge consider *multiple* world states:

$$[\![\varphi]\!]^M = \forall_{w \in M}\, [\![\varphi]\!]^w$$

20.3 Lambda calculus notation [Church, 1940]

Propositions about entities are described using **lambda calculus** expressions of various types α, β:

1. $\langle\beta\text{-expr}\rangle \to \langle\alpha{\to}\beta\text{-expr}\rangle\ \langle\alpha\text{-expr}\rangle$ – expressions can **apply** functions $\alpha{\to}\beta$ to arguments α.
   E.g.: OnDuty Kim means *Kim is on duty*.
   (Complex α are parenthesized for disambiguation: $\langle\beta\text{-expr}\rangle \to \langle(\alpha){\to}\beta\text{-expr}\rangle\ \langle\alpha\text{-expr}\rangle$.)
   This is interpreted by looking up the $\langle\alpha\text{-expr}\rangle$ in the association table for the $\langle\alpha{\to}\beta\text{-expr}\rangle$:

$$[\![\langle\alpha{\to}\beta\text{-expr}\rangle\ \langle\alpha\text{-expr}\rangle]\!]^w_g = \iota \quad \text{where} \quad [\![\langle\alpha{\to}\beta\text{-expr}\rangle]\!]^w_g =$$

input		output
⋮		⋮
$[\![\langle\alpha\text{-expr}\rangle]\!]^w_g$	:	ι
⋮		⋮

   (The variable binding function g may be empty/undefined if there are no bound variables.)
2. $\langle\alpha{\to}\beta\text{-expr}\rangle \to \lambda_{\langle\alpha\text{-var}\rangle}\ \langle\beta\text{-expr}\rangle$ – expressions can **abstract** functions into input α, output β.
   E.g.: λ_x Prof x means *professors*.
   (Complex α are parenthesized for disambiguation: $\langle(\alpha){\to}\beta\text{-expr}\rangle \to \lambda_{\langle\alpha\text{-var}\rangle}\ \langle\beta\text{-expr}\rangle$.)

This builds a function on α, setting each output $\langle\beta\text{-expr}\rangle$ by binding $\langle\alpha\text{-var}\rangle$ to each α value:

$$[\![\lambda_{\langle\alpha\text{-var}\rangle}\ \langle\beta\text{-expr}\rangle]\!]^w_g \ = \ \begin{array}{|ll|}\hline \text{input} & \text{output} \\ \hline \iota_1 & : \ [\![\langle\beta\text{-expr}\rangle]\!]^w_{\langle\alpha\text{-var}\rangle\ :\ \iota_1,\ \text{other } \chi \text{ in } g\ :\ g\,\chi} \\ \iota_2 & : \ [\![\langle\beta\text{-expr}\rangle]\!]^w_{\langle\alpha\text{-var}\rangle\ :\ \iota_2,\ \text{other } \chi \text{ in } g\ :\ g\,\chi} \\ \vdots & \vdots \\ \hline \end{array} \ \text{for } \iota_1, \iota_2, \ldots \text{ in } D^w_\alpha .$$

(The variable binding function g may be empty/undefined if there are no bound variables.)

3. $\langle\alpha\text{-expr}\rangle \rightarrow (\ \langle\alpha\text{-expr}\rangle\ )$ – expressions can be **parenthesized**.

Note: functions with one or more entities as input and a truth value as output are called **predicates**.

Note: functions with one entity as input and a truth value as output are also **sets**.

In general, expressions can ground out in a variety of constants and variables:

1. $\langle\alpha\text{-expr}\rangle \rightarrow \langle$[A-Za-z0-9]+$\rangle$ – expressions can be defined as **constants**.

   E.g.: $\langle$e-expr$\rangle \rightarrow$ Kim, $\langle$t-expr$\rangle \rightarrow$ True, $\langle$e→t-expr$\rangle \rightarrow$ OnDuty, $\langle$e→e→t-expr$\rangle \rightarrow$ Teach.

2. $\langle\alpha\text{-expr}\rangle \rightarrow \langle\alpha\text{-var}\rangle$ – expressions can be **variables**.

3. $\langle\alpha\text{-var}\rangle \rightarrow \langle$[$a$-$z$]$\rangle$ – variables can be (italicized) letters.

   E.g.: $\langle$e-var$\rangle \rightarrow x$.

   This is interpreted by looking up the variable in the variable binding function:

$$[\![\langle\alpha\text{-var}\rangle]\!]^w_{\begin{array}{|c|}\hline \vdots \quad \vdots \\ \langle\alpha\text{-var}\rangle : \iota \\ \vdots \quad \vdots \\ \hline \end{array}} \ = \ \iota.$$

   (If the variable binding function g does not contain the $\langle\alpha\text{-var}\rangle$, the expression is ill formed!)

Practice:

Write a lambda calculus expression for a function that takes an input and divides it by two.

To simplify logical forms, we'll define substitution in function application using **beta reduction**:

$$[\![(\lambda_\chi\ \ldots\chi\ldots\chi\ldots\chi\ldots)\ \varphi]\!]^w_g \ = \ [\![\ldots\varphi\ldots\varphi\ldots\varphi\ldots]\!]^w_g.$$

20.4 Generalized Quantifiers [Barwise & Cooper, 1981]

Most propositions involve **quantifiers**, to specify number/proportion of entities that hold properties.

1. **Cardinal** quantifiers compare counts of intersected restrictor r and nuclear scope s arguments:

   $$\langle e{\to}(e{\to}t){\to}(e{\to}t){\to}t\text{-expr}\rangle \to \mathsf{Count}_{\geq} \text{ where } [\![\mathsf{Count}_{\geq}]\!]^w_g = [\![\lambda_n\ \lambda_r\ \lambda_s\ |\, r \cap s\,| \geq n]\!]^w_g$$

   For example, we can use this to define:

   $$[\![\mathsf{Some}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \mathsf{Count}_{\geq}\ 1\ r\ s]\!]^w_g$$
   $$[\![\mathsf{AtLeastTwo}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \mathsf{Count}_{\geq}\ 2\ r\ s]\!]^w_g$$

2. **Proportional** quantifiers compare ratios of $r \cap s$ over r arguments:

   $$\langle e{\to}(e{\to}t){\to}(e{\to}t){\to}t\text{-expr}\rangle \to \mathsf{Ratio}_{\geq} \text{ where } [\![\mathsf{Ratio}_{\geq}]\!]^w_g = [\![\lambda_n\ \lambda_r\ \lambda_s\ |\, r \cap s\,| \geq n \cdot |\, r\,|]\!]^w_g$$

   For example, we can use this to define:

   $$[\![\mathsf{All}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \mathsf{Ratio}_{\geq}\ 1.0\ r\ s]\!]^w_g$$
   $$[\![\mathsf{AtLeastHalf}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \mathsf{Ratio}_{\geq}\ 0.5\ r\ s]\!]^w_g$$

   Proportional quantifiers can define conditional probabilities, for probabilistic reasoning:

   $$\mathsf{P}_w(\lambda_\chi\ \omega\,|\,\lambda_\chi\ \alpha) = n \quad \text{iff} \quad [\![\mathsf{Ratio}_{=}\ n\ (\lambda_\chi\ \alpha)\ (\lambda_\chi\ \omega)]\!]^w_g = \textbf{True}$$

   For example:

   $$\mathsf{P}_w(\lambda_x\ \mathsf{OnDuty}\ x\,|\,\lambda_x\ \mathsf{Prof}\ x) = 0.8 \quad \text{iff} \quad [\![\mathsf{Ratio}_{=}\ 0.8\ (\lambda_x\ \mathsf{Prof}\ x)\ (\lambda_x\ \mathsf{OnDuty}\ x)]\!]^w_g = \textbf{True}$$

We can define 'less-than' quantifier in terms of 'greater-than' quantifiers:

$$[\![\mathsf{Count}_{\leq}]\!]^w_g = [\![\lambda_n\ \lambda_r\ \lambda_s\ \mathsf{Count}_{\geq}\ (|r| - n)\ r\ (\lambda_x \neg\ s\ x)]\!]^w_g$$
$$[\![\mathsf{Ratio}_{\leq}]\!]^w_g = [\![\lambda_n\ \lambda_r\ \lambda_s\ \mathsf{Ratio}_{\geq}\ (1.0 - n)\ r\ (\lambda_x \neg\ s\ x)]\!]^w_g$$

For example, we can use this to define:

$$[\![\mathsf{None}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \mathsf{Count}_{\leq}\ 0\ r\ s]\!]^w_g$$
$$[\![\mathsf{AtMostHalf}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \mathsf{Ratio}_{\leq}\ 0.5\ r\ s]\!]^w_g$$

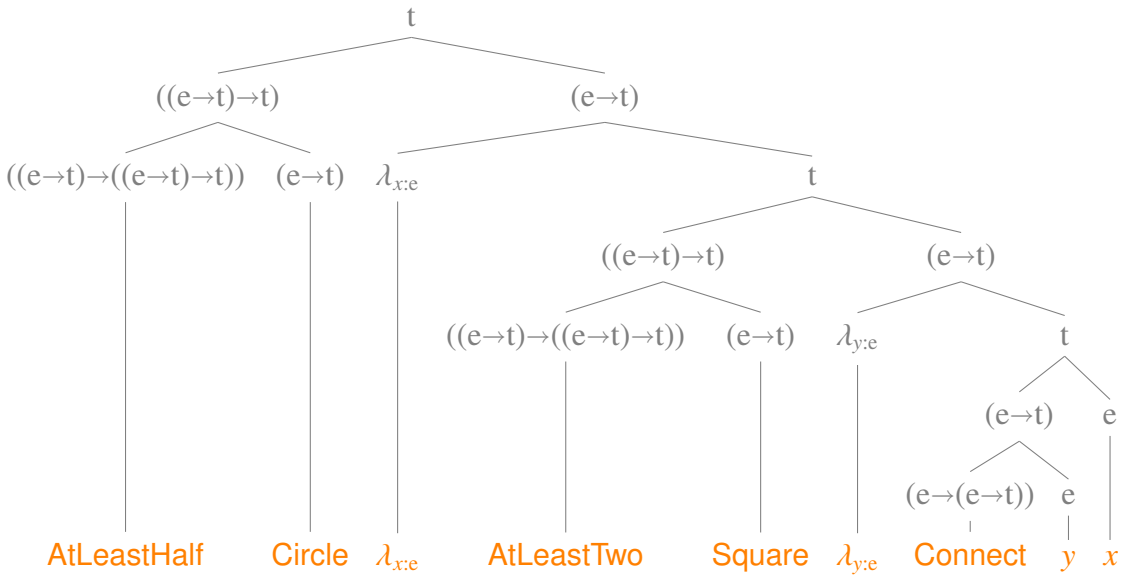
We can then define variants $q_{=}$, $q_{<}$, $q_{>}$ for other comparison operators in terms of these:

$$[\![\mathsf{Count}_{=}]\!]^w_g = [\![\lambda_{n,r,s}\ \mathsf{Count}_{\leq}\ n\ r\ s \wedge \mathsf{Count}_{\geq}\ n\ r\ s]\!]^w_g \quad [\![\mathsf{Ratio}_{=}]\!]^w_g = [\![\lambda_{n,r,s}\ \mathsf{Ratio}_{\leq}\ n\ r\ s \wedge \mathsf{Ratio}_{\geq}\ n\ r\ s]\!]^w_g$$
$$[\![\mathsf{Count}_{<}]\!]^w_g = [\![\lambda_{n,r,s}\ \mathsf{Count}_{\leq}\ n\ r\ s \wedge \neg\mathsf{Count}_{\geq}\ n\ r\ s]\!]^w_g \quad [\![\mathsf{Ratio}_{<}]\!]^w_g = [\![\lambda_{n,r,s}\ \mathsf{Ratio}_{\leq}\ n\ r\ s \wedge \neg\mathsf{Ratio}_{\geq}\ n\ r\ s]\!]^w_g$$
$$[\![\mathsf{Count}_{>}]\!]^w_g = [\![\lambda_{n,r,s}\ \mathsf{Count}_{\geq}\ n\ r\ s \wedge \neg\mathsf{Count}_{\leq}\ n\ r\ s]\!]^w_g \quad [\![\mathsf{Ratio}_{>}]\!]^w_g = [\![\lambda_{n,r,s}\ \mathsf{Ratio}_{\geq}\ n\ r\ s \wedge \neg\mathsf{Ratio}_{\leq}\ n\ r\ s]\!]^w_g$$

For example, we can use this to define:

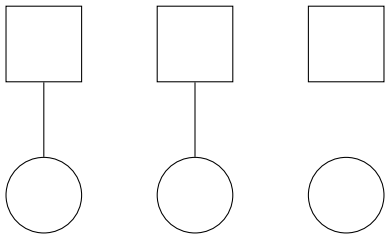
$$[\![\text{ExactlyOne}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \text{Count}_=\ 1\ r\ s]\!]^w_g$$
$$[\![\text{ExactlyHalf}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \text{Ratio}_=\ 0.5\ r\ s]\!]^w_g$$
$$[\![\text{Most}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \text{Ratio}_>\ 0.5\ r\ s]\!]^w_g$$
$$[\![\text{Few}]\!]^w_g = [\![\lambda_r\ \lambda_s\ \text{Ratio}_<\ 0.5\ r\ s]\!]^w_g$$

These quantifiers can be nested inside each other:



Practice:

Interpret the above expression in the below world state:



20.5 Other operators

We also use infix notation for conjunctions:

1. ⟨t-expr⟩ → ⟨t-expr⟩ ∧ ⟨t-expr⟩ where $[\![\varphi \wedge \psi]\!]^w = [\![\text{And}\ \varphi\ \psi]\!]^w$

The conjunction function is:

$[\![\text{And}]\!]^w =$

input	output
False :	input output **False** : **False** **True** : **False**
True :	input output **False** : **False** **True** : **True**

Generalized quantifiers are powerful enough that we don't need other operators.

We can use a 'None' quantifier (and a uniquely satisfied predicate 'Unit') to implement **negation**:

$$(\neg\ \text{ItsRainy}) \Leftrightarrow (\text{Count}_{\leq}\ 0\ (\lambda_x\ \text{Unit}\ x)\ (\lambda_x\ \text{ItsRainy}))$$

We can use negation to implement **disjunction** (via DeMorgan's law):

$$(\text{ItsCloudy} \vee \text{ItsSunny}) \Leftrightarrow (\neg\ ((\neg\ \text{ItsCloudy}) \wedge (\neg\ \text{ItsSunny})))$$

And we can use disjunction to implement **implication** (via double negation law):

$$(\text{ItsRainy} \rightarrow \text{ItsCloudy}) \Leftrightarrow ((\neg\ \text{ItsRainy}) \vee \text{ItsCloudy})$$

or more directly using an 'All' quantifier:

$$(\text{ItsRainy} \rightarrow \text{ItsCloudy}) \Leftrightarrow (\text{All}\ (\lambda_x\ \text{Unit}\ x \wedge \text{ItsRainy})\ (\lambda_x\ \text{ItsCloudy}))$$

(This will simplify our entailments later.)

20.6 Intensions (propositions about propositions) [Carnap, 1947]

We now have a formal system to reason about complex ideas based on sets of entities or tuples.

But what if we have to do something when someone *wants to eat*, where *to eat* is a proposition?

Propositions denote truth values, but the person doesn't want '**False**' (whatever that would mean).

So, define argument propositions as **intensions** – sets of satisfying possible worlds [Carnap, 1947]:

$$\langle \text{s}{\rightarrow}\alpha\text{-expr} \rangle \rightarrow \uparrow \langle \alpha\text{-expr} \rangle$$

The '$\uparrow$' is an **operator**, like the lambda operator, so it has its own interpretation function:

$[\![\uparrow \varphi]\!]^w = [\![\lambda_{w':\text{s}}\ [\![\varphi]\!]^{w'}\ ]\!]^w =$

input		output
w	:	$[\![\varphi]\!]^w$
Star Trek Universe	:	$[\![\varphi]\!]^{\textbf{Star Trek Universe}}$
Marvel Universe	:	$[\![\varphi]\!]^{\textbf{Marvel Universe}}$
⋮	:	⋮
w'	:	$[\![\varphi]\!]^{w'}$
⋮	:	⋮

.

20.7 Entailment

An intension may then **entail** another if its satisfying possible worlds are a subset of the other's:

$$[\![\text{Entail } i\ j]\!]^m \quad \Leftrightarrow \quad i \subseteq j$$

These sets would be hard to calculate! Fortunately we can define entailment in terms of structure!

We reason about these, e.g. test if claim i is in some class j, by *simplifying* rather than enumerating.

This lets us define world models as sets of logic propositions, rather than enormous sets of worlds!

If $[\![\uparrow \varphi]\!] \subseteq [\![\uparrow \psi]\!]$, then if we have φ in our world model, we can safely add ψ preserving uncertainty.

An example entailment — **conjunction elimination** (where φ and ψ are of type t):

$$\begin{aligned} [\![\uparrow \varphi \wedge \psi]\!]^w_g \quad &\subseteq \quad [\![\uparrow \varphi]\!]^w_g \\ [\![\uparrow \varphi \wedge \psi]\!]^w_g \quad &\subseteq \quad [\![\uparrow \psi]\!]^w_g \end{aligned}$$

For example, if we have these conjoined propositions in our world model:

$$\text{Prof Kim} \wedge \text{OnDuty Pat}$$

then we can safely add this without loss of generality (without restricting our possible worlds):

$$\text{Prof Kim}$$

Another entailment — **universal modus ponens** for generalized quantifiers (φ, ψ:(e→t), χ:e):

$$[\![\uparrow \text{Ratio}_{=}\ 1.0\ \varphi\ \psi\ \wedge\ \varphi\ \chi]\!]^w_g \quad \subseteq \quad [\![\uparrow \psi\ \chi]\!]^w_g$$

For example, if we have these conjoined propositions in our world model:

$$\begin{aligned} &\text{Ratio}_{=}\ 1.0\ (\lambda_t\ \text{Time}\ t) \\ &\qquad (\lambda_t\ \text{Ratio}_{=}\ 1.0\ (\lambda_x\ \text{AdmittedStudent}\ t\ x \wedge \neg\text{EnrolledStudent}\ t\ x) \\ &\qquad\qquad (\lambda_x\ \text{Ratio}_{=}\ .3\ (\lambda_u\ \text{PossibleSuccessor}\ t\ u) \\ &\qquad\qquad\qquad (\lambda_u\ \text{EnrolledStudent}\ u\ x)) \wedge \\ &\text{Time StartTime} \wedge \\ &\text{AdmittedStudent StartTime Kim} \wedge \neg\text{EnrolledStudent StartTime Kim} \end{aligned}$$

then we can safely add this without loss of generality:

$$\begin{aligned} &\text{Ratio}_{=}\ .3\ (\lambda_u\ \text{PossibleSuccessor StartTime}\ u) \\ &\qquad (\lambda_u\ \text{EnrolledStudent}\ u\ \text{Kim}) \end{aligned}$$

We can also reason about quantifiers (where $Q \in \{\text{Count}, \text{Ratio}\}$ and $\varphi, \psi, \psi' : (e \to t)$ and $\chi : e$):

$$\begin{array}{rcl} n \geq n' & \Rightarrow & [\![\uparrow Q_{\geq}\ n\ \varphi\ \psi]\!]^w_g \subseteq [\![\uparrow Q_{\geq}\ n'\ \varphi\ \psi]\!]^w_g \\ n \leq n' & \Rightarrow & [\![\uparrow Q_{\leq}\ n\ \varphi\ \psi]\!]^w_g \subseteq [\![\uparrow Q_{\leq}\ n'\ \varphi\ \psi]\!]^w_g \\ \forall_{\chi \text{ well-formed}}\ [\![\uparrow \psi\ \chi]\!]^w_g \subseteq [\![\uparrow \psi'\ \chi]\!]^w_g & \Rightarrow & [\![\uparrow Q_{\geq}\ n\ \varphi\ \psi]\!]^w_g \subseteq [\![\uparrow Q_{\geq}\ n\ \varphi\ \psi']\!]^w_g \\ \forall_{\chi \text{ well-formed}}\ [\![\uparrow \psi\ \chi]\!]^w_g \supseteq [\![\uparrow \psi'\ \chi]\!]^w_g & \Rightarrow & [\![\uparrow Q_{\leq}\ n\ \varphi\ \psi]\!]^w_g \subseteq [\![\uparrow Q_{\leq}\ n\ \varphi\ \psi']\!]^w_g \end{array}$$

For example, if we have the left proposition in our world model, then we can safely add the right:

$$[\![\uparrow \text{Count}_{\geq}\ 2\ (\lambda_x\ \text{Hut}\ x)\ (\lambda_x\ \text{Straw}\ x)]\!]^w \subseteq [\![\uparrow \text{Count}_{\geq}\ 1\ (\lambda_x\ \text{Hut}\ x)\ (\lambda_x\ \text{Straw}\ x)]\!]^w$$
$$[\![\uparrow \text{Count}_{\geq}\ 1\ (\lambda_x\ \text{Hut}\ x)\ (\lambda_x\ \text{Straw}\ x \wedge \text{Round}\ x)]\!]^w \subseteq [\![\uparrow \text{Count}_{\geq}\ 1\ (\lambda_x\ \text{Hut}\ x)\ (\lambda_x\ \text{Straw}\ x)]\!]^w$$

This kind of reasoning by simplifying is sometimes called **natural logic** [van Benthem, 1986].

20.8 Decision theory [von Neumann & Morgenstern, 1944]

This linguistic meaning representation can now be used in **probabilistic planning**.

We first define 'conditional entailment' requiring a contribution of intension i toward entailing j:

$$i \subseteq_k j \quad \Leftrightarrow \quad k \not\subseteq j \ \wedge\ i \cap k \subseteq j$$

We also define a special probability over 'trials' υ, which are **possible successor** times to τ:

$$\text{P}_{M,\tau}(\lambda_\upsilon\ \omega \,|\, \alpha) \quad = \quad \text{P}_M(\lambda_\upsilon\ \omega \,|\, \lambda_\upsilon\ \alpha \wedge \text{PossibleSuccessor}\ \tau\ \upsilon)$$

(We'll use $\text{Suc}\ \tau$ to define a unique **actual successor** to τ, and $\text{Cur}\ \tau$ to flag τ as **current**.)

We then use these probabilities in a decision process to calculate time-averaged expected utility:

$$\text{AEU}(\tau,\ M,\ \pi) = \overbrace{\text{R}_M(\tau)}^{\text{reward}} \cdot \begin{cases} \text{if } \exists_\alpha \overbrace{[\![\uparrow \pi]\!] \subseteq_{[\![\uparrow \text{Cur}\ \tau]\!] \cap M} [\![\uparrow \alpha]\!]}^{\text{next action } \alpha \text{ of } \pi}\text{:} & \overbrace{\textstyle\sum_\omega \text{P}_{M,\tau}(\omega \,|\, \alpha)}^{\text{sum prob. of outcomes } \omega \text{ of } \alpha} \cdot \overbrace{\text{AEU}(\text{Suc}\ \tau,\ \underbrace{[\![\uparrow \alpha \wedge \omega\ (\text{Suc}\ \tau)]\!] \cap M}_{\text{new } M \text{ at next } \tau},\ \pi)}^{\text{repeat with } M,\ \alpha \text{ and } \omega \text{ as new } M} \\ \text{otherwise:} & \frac{1}{\tau} \end{cases}$$

(It assumes the plan π is perfectly specific: at most one next action α for each world model M.)

This tells us how good a plan is. We can use it to choose among plans to maximize average reward!

For example, if the goal hill is one step away, we get a reward in one step, so $\text{AEU}(\tau,\ M,\ \pi) = 1$.

But if it's muddy and we slip half the time and don't go anywhere, then:

$$\begin{aligned} \text{AEU}(\tau,\ M,\ \pi) = & \begin{cases} .5 \text{ (slip)} & \times \begin{cases} .5 \text{ (slip)} & \times \begin{cases} .5 \text{ (slip)} & \times \ldots \\ +.5 \text{ (no slip)} & \times \frac{1}{3} \text{ (arrive in 3 steps)} \end{cases} \\ +.5 \text{ (no slip)} & \times \frac{1}{2} \text{ (arrive in 2 steps)} \end{cases} \\ +.5 \text{ (no slip)} & \times \frac{1}{1} \text{ (arrive in 1 step)} \end{cases} \\ \approx & .7 \end{aligned}$$

So if we have two plans (clear path and muddy path) and we know mud slows us, we can avoid it.

20.9 Example: grid-world navigation

Here's a plan and world model that solves the muddy hill problem in a grid world:

1. a **plan** π to move Me to MyHill (assuming the following predicates:
   - Cur t is true for the most recent time point t in an AEU evaluation;
   - PrecedeOrEqual s t is true if time s precedes or is equal to t;
   - TryMoveToward t a x is true if a tries to move toward x at time t;

   where 'All's iterate over entities, 'None's give conditions):

   $$\begin{aligned}&\text{All } (\lambda_t \text{ Cur } t \wedge \\ &\quad \text{None } (\lambda_s \text{ PrecedeOrEqual } 0\ s\ \wedge \text{PrecedeOrEqual } s\ t) \\ &\quad\quad (\lambda_s \text{ At } s \text{ Me MyHill})) \\ &\quad (\lambda_t \text{ TryMoveToward } t \text{ Me MyHill})\end{aligned}$$

   (Here time '0' is when the plan is created – I have to reach the goal *after* that to succeed.)

   When used in an AEU, this gives us our actions a – in this case: TryMoveToward predicates.

2. **world knowledge** M that moving through mud may fail (assuming the following predicates:
   - Adjacent x y is true if grid squares x and y are adjacent;
   - Aligned x y z is true if a grid square y lies on a line from x to z;
   - Muddy y and Clear y are true if grid square y is muddy or clear, respectively;
   - PossibleSuccessor t u is true if time t immediately precedes time u;

   where 'Half's give probability cost):

   $$\begin{aligned}&\text{All } (\lambda_{t,a,z} \text{ TryMoveToward } t\ a\ z) \\ &\quad (\lambda_{t,a,z} \text{ All } (\lambda_x \text{ At } t\ a\ x) \\ &\quad\quad (\lambda_x \text{ All } (\lambda_y \text{ Adjacent } x\ y\ \wedge \text{Aligned } x\ y\ z\ \wedge \text{Muddy } y) \\ &\quad\quad\quad (\lambda_y \text{ Half } (\lambda_u \text{ PossibleSuccessor } t\ u)\ (\lambda_u \text{ At } u\ a\ y) \wedge \\ &\quad\quad\quad\quad \text{Half } (\lambda_u \text{ PossibleSuccessor } t\ u)\ (\lambda_u \text{ At } u\ a\ x))))\end{aligned}$$

   and that moving through clear terrain always succeeds:

   $$\begin{aligned}&\text{All } (\lambda_{t,a,z} \text{ TryMoveToward } t\ a\ z) \\ &\quad (\lambda_{t,a,z} \text{ All } (\lambda_x \text{ At } t\ a\ x) \\ &\quad\quad (\lambda_x \text{ All } (\lambda_y \text{ Adjacent } x\ y\ \wedge \text{Aligned } x\ y\ z\ \wedge \text{Clear } y) \\ &\quad\quad\quad (\lambda_y \text{ All } (\lambda_u \text{ PossibleSuccessor } t\ u)\ (\lambda_u \text{ At } u\ a\ y))))\end{aligned}$$

   When used in an AEU, this gives us our outcome events ω – in this case: At predicates.

Dropping these plans and world models into the AEU function defines a rational decision process. Since they are *simple* and *work*, they are in some sense a 'natural' representation of complex ideas.

We'll therefore use these expressions as the complex ideas that get communicated using language.

But note these look very different from how we might represent these ideas in natural language.

20.10 Discourse

Entailment predicates can be used to evaluate if a desired intension i is in some class j:

$$\begin{array}{l}\text{All } (\lambda_t \text{ Cur } t) \\ \quad (\lambda_t \text{ All } (\lambda_c \text{ Clerk } c) \\ \qquad (\lambda_c \text{ All } (\lambda_a \text{ Person } a) \\ \qquad\quad (\lambda_a \text{ All } (\lambda_x \text{ Have } t\ c\ x \wedge \\ \qquad\qquad \text{Some } (\lambda_j \text{ Equal } j\ (\uparrow \text{Count}_{\geq}\ 1\ (\lambda_u \text{ PossibleSuccessor } t\ u) \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad (\lambda_u \text{ Eat } u\ a\ x))) \\ \qquad\qquad\qquad\quad (\lambda_j \text{ Some } (\lambda_i \text{ Want } t\ a\ i) \\ \qquad\qquad\qquad\qquad\quad\ (\lambda_i \text{ Entail } i\ j))) \\ \qquad\qquad\quad (\lambda_x \text{ Give } t\ c\ a\ x))))\end{array}$$

(If a clerk has something, and someone wants it [perhaps among other things], give it to them.)

Here, even if the intension i that the agent wants contains other conjuncts, it still entails j:

$$\begin{array}{l}\text{Some } (\lambda_i \text{ Equal } i\ (\uparrow \text{Count}_{\geq}\ 1\ (\lambda_u \text{ PossibleSuccessor 10:00:00 } u) \\ \qquad\qquad\qquad\qquad\qquad\quad (\lambda_u \text{ Eat } u \text{ Me Apple1} \wedge \text{Drink } u \text{ Me Juice1}))) \\ \qquad (\lambda_i \text{ Want 10:00:00 Me } i)\end{array}$$

So if the above is true, the clerk will recognize that I want to eat an apple and give it to me.

20.11 Mapping from syntax

We compose lambda calculus expressions the same way we build trees, using semiring substitution.

First intialize unit spans with triples of probability, lambda term and syntactic category:

$$v_\top(w) = \begin{cases} \langle \text{P}(\text{most} \mid \textbf{N-b(N-aD)}, \ldots),\ \lambda_R\ \lambda_S\ \text{Most}\ R\ S,\ \textbf{N-b(N-aD)} \rangle & \text{if } w = \text{most} \\ \langle \text{P}(\text{shapes} \mid \textbf{N-aD}, \ldots),\ \lambda_x\ \text{BeingAShape}\ x,\ \textbf{N-aD} \rangle & \text{if } w = \text{shapes} \\ \langle \text{P}(\text{are} \mid \textbf{V-aN-b(A-aN)}, \ldots),\ \lambda_P\ \lambda_Q\ P\ Q,\ \textbf{V-aN-b(A-aN)} \rangle & \text{if } w = \text{are} \\ \langle \text{P}(\text{red} \mid \textbf{A-aN}, \ldots),\ \lambda_Q\ Q\ (\lambda_x\ \text{BeingRed}\ x),\ \textbf{A-aN} \rangle & \text{if } w = \text{red} \\ \langle \text{P}(\text{square} \mid \textbf{A-aN}, \ldots),\ \lambda_Q\ Q\ (\lambda_x\ \text{BeingSquare}\ x),\ \textbf{A-aN} \rangle & \text{if } w = \text{square} \\ \vdots \end{cases}$$

(Recall $v_\top$ is just the semiring identity for $\otimes$, now extended to depend on word w.)

We then extend 'prod_pair' to compose terms $a = \langle p_a,\ \varphi_a,\ c_a \rangle$ and $b = \langle p_b,\ \varphi_b,\ c_b \rangle$:

$$a \otimes b = \begin{cases} \langle \mathsf{P(Aa}, \upsilon \,|\, \tau, \ldots) \cdot p_a \cdot p_b,\ \varphi_b\ \varphi_a,\ \tau \rangle & \text{if } c_a = \upsilon,\ c_b = \tau\text{-}\mathbf{a}\upsilon \\ \langle \mathsf{P(Ab}, \upsilon \,|\, \tau, \ldots) \cdot p_a \cdot p_b,\ \varphi_a\ \varphi_b,\ \tau \rangle & \text{if } c_a = \tau\text{-}\mathbf{b}\upsilon,\ c_b = \upsilon \\ \langle \mathsf{P(Ma}, \upsilon \,|\, \tau, \ldots) \cdot p_a \cdot p_b,\ \lambda_x\ \varphi_a\ (\lambda_P\ P\ x) \wedge \varphi_b\ x,\ \tau \rangle & \text{if } c_a = \upsilon\text{-}\mathbf{a}\mathbf{N},\ c_b = \tau \\ \langle \mathsf{P(Mb}, \upsilon \,|\, \tau, \ldots) \cdot p_a \cdot p_b,\ \lambda_x\ \varphi_b\ (\lambda_P\ P\ x) \wedge \varphi_a\ x,\ \tau \rangle & \text{if } c_a = \tau,\ c_b = \upsilon\text{-}\mathbf{a}\mathbf{N} \\ \vdots \end{cases}$$

where Aa, Ab, Ma, Mb, ... are just names for the corresponding operations on lambda terms.

To simplify logical forms, we'll define substitution in function application using **beta reduction**:

$$(\lambda_A\ \ldots A \ldots A \ldots A \ldots)\ B \ = \ \ldots B \ldots B \ldots B \ldots$$

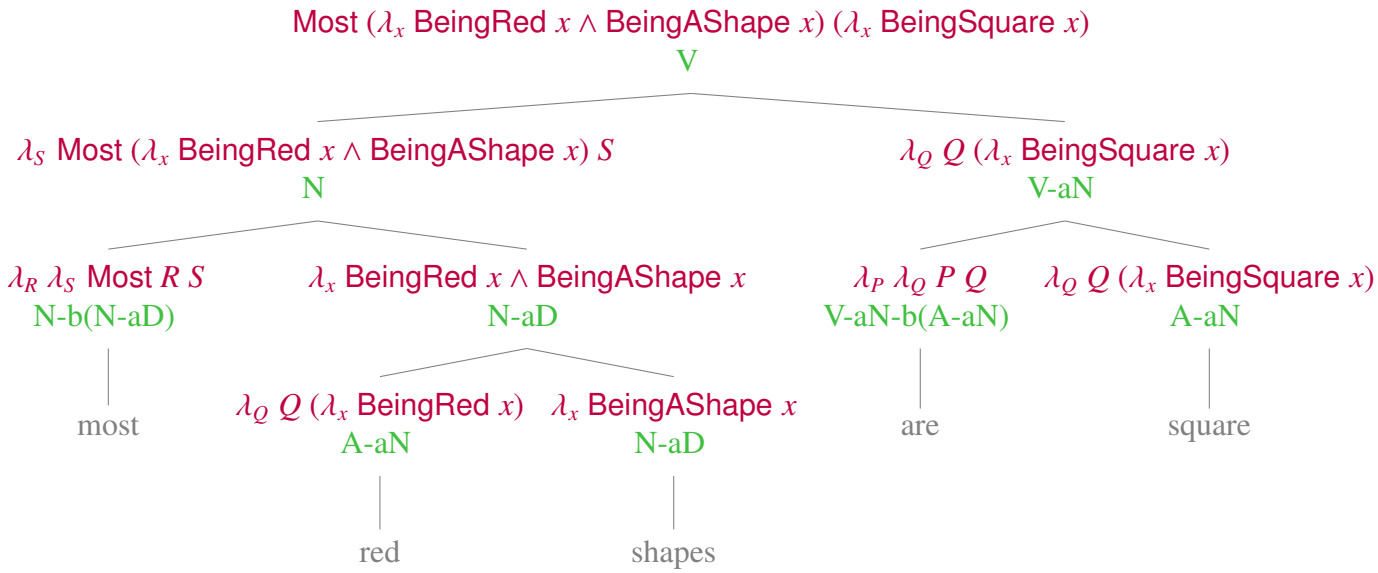
For example (as happens in '*red shapes*' below):

$$\begin{aligned} (\lambda_Q\ Q\ (\lambda_x\ \mathsf{BeingRed}\ x))\ (\lambda_P\ P\ x) &= (\lambda_P\ P\ x)\ (\lambda_x\ \mathsf{BeingRed}\ x) \\ &= (\lambda_x\ \mathsf{BeingRed}\ x)\ x \\ &= \mathsf{BeingRed}\ x \end{aligned}$$

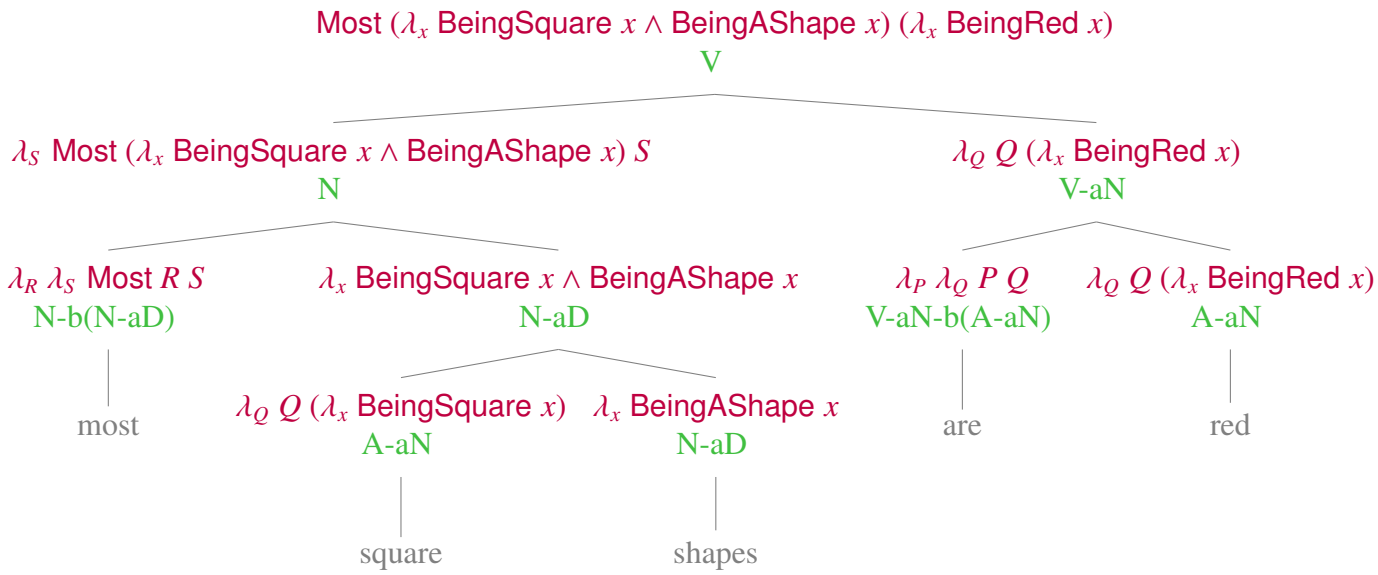
Another example (as happens in '*are square*' below):

$$\begin{aligned} (\lambda_P\ \lambda_Q\ P\ Q)\ (\lambda_S\ S\ (\lambda_x\ \mathsf{BeingSquare}\ x)) &= \lambda_Q\ (\lambda_S\ S\ (\lambda_x\ \mathsf{BeingSquare}\ x))\ Q \\ &= \lambda_Q\ Q\ (\lambda_x\ \mathsf{BeingSquare}\ x) \end{aligned}$$

Example translation of '*Most red shapes are square.*':



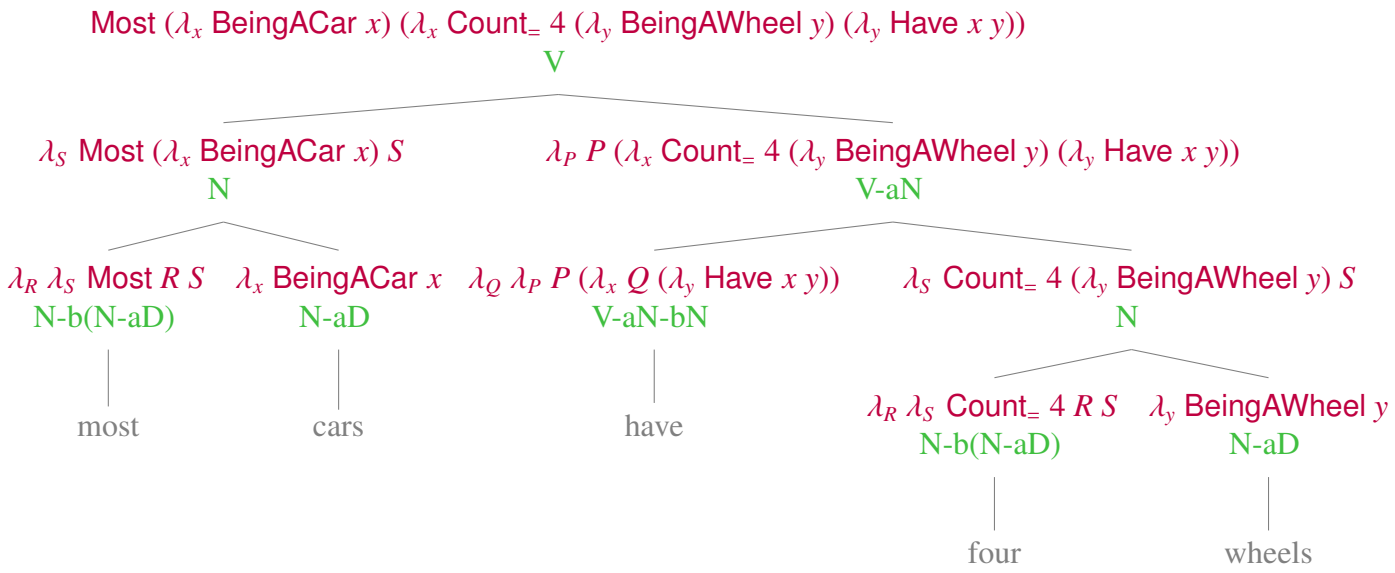
as distinct from '*Most square shapes are red.*':



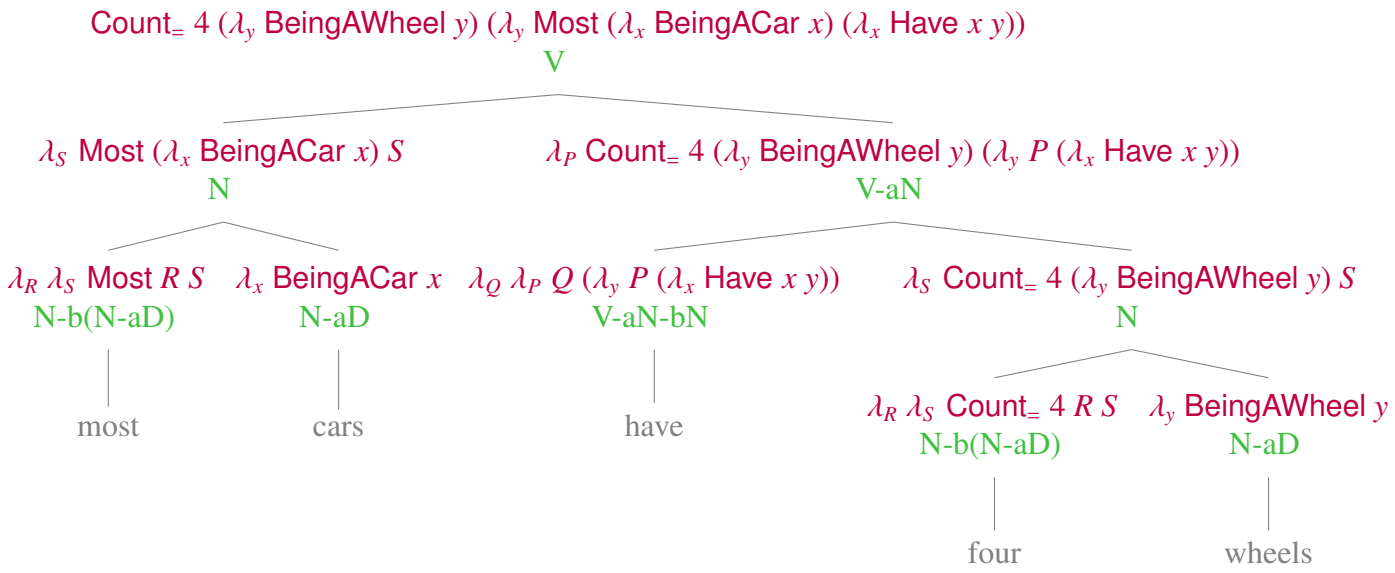
We can also model (local) scope inversion:

$$v_{\mathsf{T}}(w) = \begin{cases} \vdots & \\ \langle \mathsf{P}(\text{have} \,|\, \textbf{V-aN-bN}, \ldots),\ \lambda_Q\ \lambda_P\ P\ (\lambda_x\ Q\ (\lambda_y\ \mathsf{Having}\ x\ y)),\ \textbf{V-aN-bN} \rangle & \text{if } w = \text{have} \\ \langle \mathsf{P}(\text{have} \,|\, \textbf{V-aN-bN}, \ldots),\ \lambda_Q\ \lambda_P\ Q\ (\lambda_y\ P\ (\lambda_x\ \mathsf{Having}\ x\ y)),\ \textbf{V-aN-bN} \rangle & \text{if } w = \text{have} \\ \langle \mathsf{P}(\text{four} \,|\, \textbf{N-b(N-aD)}, \ldots),\ \lambda_R\ \lambda_S\ \mathsf{Count}_{=}\ 4\ R\ S,\ \textbf{N-b(N-aD)} \rangle & \text{if } w = \text{four} \\ \langle \mathsf{P}(\text{cars} \,|\, \textbf{N-aD}, \ldots),\ \lambda_x\ \mathsf{BeingACar}\ x,\ \textbf{N-aD} \rangle & \text{if } w = \text{cars} \\ \langle \mathsf{P}(\text{wheels} \,|\, \textbf{N-aD}, \ldots),\ \lambda_y\ \mathsf{BeingAWheel}\ y,\ \textbf{N-aD} \rangle & \text{if } w = \text{wheels} \\ \vdots & \end{cases}$$

Example translation of '*Most cars have four wheels*' with usual scope:



Example translation of '*Most cars have four wheels*' with inverted scope:



We can also model conjunction:

$$a \otimes b = \begin{cases} \vdots & \\ \langle \mathrm{P(Ca} \,|\, \tau, \ldots) \cdot p_a \cdot p_b,\ \lambda_Q\, \varphi_a\, Q \wedge \varphi_b\, Q,\ \tau \rangle & \text{if } c_a = \tau,\ c_b = \tau\text{-}\mathbf{c}\tau \\ \langle \mathrm{P(Cb} \,|\, \tau\text{-}\mathbf{c}\tau, \ldots) \cdot p_a \cdot p_b,\ \lambda_Q\, \varphi_a\, Q \wedge \varphi_b\, Q,\ \tau\text{-}\mathbf{c}\tau \rangle & \text{if } c_a = \tau\text{-}\mathbf{c}\tau\text{-}\mathbf{d}\tau,\ c_b = \tau \\ \vdots & \end{cases}$$

Not covered yet:

- Coreference (may have to be separately inferred).
- Scope (may have to be separately inferred).

References

Barwise, J. & Cooper, R. (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4.

Carnap, R. (1947). *Meaning and Necessity: A Study in Semantics and Modal Logic*. Chicago: University of Chicago Press.

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2), 56–68.

van Benthem, J. (1986). Natural logic. In *Essays in Logical Semantics*. Dordrecht, the Netherlands: Kluwer.

von Neumann, J. & Morgenstern, O. (1944). Theory of games and economic behavior. *Science and Society*, 9(4), 366–369.