SMPL Meanings (Paul M. Pietroski, Rutgers University: Oct 31, 2022)

Hypothesis: human linguistic meanings are instructions for how to generate concepts, from a stock of lexical concepts, via simple operations that *don't include* Function Application or Lambda Abstraction.

(**H1**) Lexical concepts can be monadic or dyadic. All the other *generable* concepts are monadic.
(**H2**) The core operations are limited forms of *conjunction, existential closure*, and *polarization*.
(**H3**) There is also a *very limited* operation of *abstraction*, corresponding to relative clause formation.

Pietroski (2018): That's *enough* to handle the usual range of textbook constructions in semantics.
Icard & Moss (2023): That's (provably) *less* than familiar alternatives, but enough to be interesting.

Even if the **Hypothesis** is wrong, it's worth thinking about the possibility of minds that generate concepts in this *computationally spare* way that is *in the ballpark of descriptive adequacy* for human meanings; cp. Chomsky (1957, 1959). It was useful to ask how human grammars—procedures that generate linguistic expressions—differ from context-free rewrite systems (cp. Post 1943) *without* exploiting the "shuffling" power of context-sensitive rules like "CB ➔ CX", much less the full power of a Turing Machine. We can likewise ask how the human procedures that generate linguistically expressible concepts differ from a simple context-free procedure that is powerful enough to be in the ballpark *without* exploiting familiar resources that overgenerate wildly, much less full the power of a Turing Machine (cp. Church 1941).

(**H1**) Lexical concepts can be monadic or dyadic. All the other *generable* concepts are monadic.
(**H2**) The core operations are limited forms of *conjunction, existential closure*, and *polarization*.

M-junction: $M_1(\_)\text{^}M_2(\_)$

conjunction of monadic concepts, thereby forming a concept that applies to a thing, x*, iff each constituent applies to* x

D-junction: $\exists[\Delta(\_, \_)\text{^}M(\_)]$

conjunction-closure of a dyadic concept with a monadic concept, thereby forming a concept that applies to a thing, x*, iff* x *bears the dyadic relation to* <u>*something*</u> *that meets the monadic condition*; cp. '$\exists y[\Delta xy \,\&\, My]$', or in the language of *description logic,* '$\exists\Delta.M$'

—There are only two meaning types, **<M>** and **<D>**. All instances of type **<D>** are lexical.

—Of course, humans (and other animals) have many concepts that are neither monadic nor dyadic. Correlatively, lexical concepts need not be *primitive*. They can be *introduced* via other concepts.

$$\text{SOCRATES}(\_) \equiv_{DF} \text{IDENTICAL}(\_, \text{SOCRATES})$$
$$\text{SEEINGOF}(\_, \_) \equiv_{DF} \text{SEEINGOFBY}(\_, \_, \text{SOME-EXPERIENCER})]$$

—But *sentential* concepts are formally monadic. These "polarized" concepts are Tarskian, not Fregean.

Polarize-UP: $\Uparrow M(\_)$

applies to everything or nothing, depending on whether or not M(_) *applies to something*

Polarize-DOWN: $\Downarrow M(\_)$

applies to nothing or everything, depending on whether or not M(_) *applies to something*

—A polarized concept isn't a special concept that denotes or describes a truth value. The SMPL-concepts don't exhibit Fregean types. There are none of type <t> or type <e>, much less <t, t>, <e, e>, …

—But propositional logic can be reconstructed in terms of polarization:
$\Uparrow M(\_)$ and $\Downarrow\Downarrow M(\_)$ are equivalent, as are $\Uparrow\Downarrow M(\_)$ and $\Downarrow\Uparrow M(\_)$, as are $\Uparrow M(\_)$ and $\Uparrow\Uparrow M(\_)$.
The valid replacements include conjunct *reduction* in positive contexts, *addition* in negative contexts:

$\Uparrow[M(\_)\text{^}P(\_)]$, so $\Uparrow M(\_)$
$\Uparrow[\Uparrow M(\_)\text{^}\Uparrow P(\_)]$, so $\Uparrow M(\_)$

$\Downarrow M(\_)$, so $\Downarrow[M(\_)\text{^}P(\_)]$
$\Downarrow[\Uparrow M(\_)]$, so $\Downarrow[\Uparrow M(\_)\text{^}\Uparrow P(\_)]$

(**H1**) Lexical concepts can be monadic or dyadic. All the other *generable* concepts are monadic.
(**H2**) The core operations are limited forms of *conjunction, existential closure*, and *polarization*.

M-junction: $M_1(\_)\text{^}M_2(\_)$ **D-junction**: $\exists[\Delta(\_, \_)\text{^}M(\_)]$ *abbreviation*: $\exists\Delta.M$

Polarize-UP: $\Uparrow M(\_)$ **Polarize-DOWN**: $\Downarrow M(\_)$

Icard & Moss (2023) precisify this system, SMPL, introduced in Pietroski (2018).

Some Examples

Assume: BROWN(_) applies to x iff x is BROWN
COW(_) applies to x iff x is a COW
SOCRATES(_) applies to x iff x is identical to SOCRATES
ABOVE(_, _)—or abbreviating, ABOVE—applies to <x, y> iff x is ABOVE y
AGENT(_, _)—or abbreviating, AGENT—applies to <x, y> iff x is an EVENT DONE BY y

BROWN(_)^COW(_) applies to x iff x is both BROWN and a COW
∃ABOVE.COW(_) applies to x iff x is ABOVE something that is a COW
∃ABOVE.[BROWN(_)^COW(_)] applies to x iff x is ABOVE something that is both BROWN and a COW
∃AGENT.COW(_) applies to x iff x is an EVENT DONE BY something that is a COW
⇑COW(_) applies to x iff something is a COW
⇓[BROWN(_)^COW(_)] applies to x iff nothing is both BROWN and a COW
⇓BROWN(_)^COW(_) applies to x iff nothing is BROWN and x is a COW
⇑∃ABOVE.COW(_) applies to x iff something is ABOVE something that is a COW
⇓∃AGENT.COW(_) applies to x iff nothing is an EVENT DONE BY something that is a COW
⇓[SOCRATES(_)^COW(_)] applies to x iff nothing is both identical to SOCRATES and a COW

Some facts (via Icard & Moss) about SMPL*, which doesn't yet allow for any abstraction*

—While propositional logic can be reconstructed, no concept of concept-negation can be defined.
Polarizing provides an analog of Aristotelian predicate-*denial*, but no analog of predicate-*complement*.
'Socrates is not a cow' corresponds to ⇓[SOCRATES(_)^COW(_)], presupposing ⇑SOCRATES(_).
But COW(_) can't be used, given SMPL operations, to build a concept logically equivalent to ~COW(_).
(Likewise, there is no way to define a concept of concept-disjunction.)

— While sentential concepts do not have truth values, logically trivial concepts can be defined.
Let '⊥' stand for some instance of Φ(_)^⇓Φ(_). Then as a matter of logic, ⊥ applies to nothing.
Let '⊤' stand for ⇓⊥. Then as a matter of logic, ⊤ applies to each thing.

—Given a notion of concept *equivalence* (presumably available in natural deductive reasoning), one can characterize a binary universal quantifier, and then *reconstruct syllogistic logic* in a familiar way.

Assume a mind such that if it can generate concepts Φ and Ψ, it can treat them as *interchangeable* (i.e., materially equivalent) for purposes of simple reasoning. If it can form thoughts of the form $\Phi \approx \Psi$, it can form thoughts of form $\Phi\text{^}\Psi \approx \Phi$, according to which the Φs that are Ψs *are* the Φs (i.e., every Φ is Ψ). In set-theoretic terms: every Φ is Ψ iff $\{x{:}\Phi(x) \,\&\, \Psi(x)\} = \{x{:}\Phi(x)\}$. Or put another way, if 'Φ' specifies the domain of the (restrictable) quantifier, then 'Ψ' eliminates nothing from the domain iff every Φ is Ψ. This is the basis for the classical *dictum de omni*, illustrated with examples like the one below.

Whether or not every cow is brown, whatever holds of cows holds of brown cows. But if every cow is brown, the brown cows *are* the cows. In which case, whatever holds of brown cows holds of cows. So if every cow is brown, 'cow' and 'brown cow' are equivalent: substitution is licensed in *both* directions.

More facts (via Icard & Moss) about SMPL, *which doesn't yet allow for any abstraction*

—Polarization provides obvious analogs for 'Some Φ is Ψ' and 'No Φ is Ψ'. Given the trivial concepts ⊤ and ⊥, binary analogs of 'Some…is…' and 'No…is…' can also be defined: ⇑(Φ^Ψ) ≈ ⊤; Φ^Ψ ≈ ⊥. But the "4th corner" of the traditional square of opposition ('Some…is not…') eludes characterization.

One can imagine a quantifier 'Sumaint' such that 'Sumaint Φ is Ψ' would be equivalent to
'Some Φ *is not* Ψ', in which '*not*' indicates predicate-denial (rather than predicate-complement).
Such a quantifier would be restrictable, and thus "conservative" in Barwise and Cooper's (1981) sense:
'Sumaint Φ is Ψ' would be equivalent to 'Sumaint Φ is a Φ that is Ψ',
and also equivalent to 'Some Φ *is not* a Φ that is Ψ', which is equivalent to 'Some Φ *is not* Ψ'.

But just as SMPL *doesn't* provide the resources needed to define concept-negation, adding a capacity to characterize the three classical quantifiers ('every', 'some', and 'no') in terms of concept-equivalence *doesn't* yield a capacity to characterize the 4th-corner quantifier 'Sumaint'.

—Reasoning in the system is, computationally, no harder (or easier) than reasoning in propositional logic.
—There are soundness and completeness proofs. (Unsurprisingly, *proving* soundness is easier.)

Potentially Relevant Sidebars regarding negation and complementation:

—There's no reason to think that having a concept C(_) guarantees a capacity to form a corresponding concept ~C(_), or any equivalent concept that applies to an entity, x, iff C(_) doesn't apply to x. Having a concept with which one can think about things *as cows* does not ensure having a capacity to think about things as *things that are not cows*, or to think about things as *not-cows*; where the *not-cows* include prime numbers, colors, and black holes. Complementation can seem simple, from a set-theoretic perspective, given a stipulated domain. But cognitively, complementation is *weird*.

—At least in nonhuman animals, natural cognition seems to abhor negation, or at least complementation.

—There are various ways in which negation (and contrast/contraries) get expressed in natural language; see Horn (1989). But strikingly, none of them seem to be a simple sentential prefix of type <t, t>.

—Pietroski et. al. (2009) and Lidz et. al. (2011) argue, based on experimental data, that the quantifier 'most' is not understood in terms of concept-negation. One can imagine a mind that forms a thought that speakers of English can report with '(that) most of the dots are blue', and forms such a thought by using a concept of negation along with (i) concepts that allow for comparison of cardinalities, or (ii) concepts of one-to-one correspondence and leftovers. For example:

#ιX:[DOT(x)^BLUE(x)] > #ιX:[DOT(_)^~BLUE(_)]; or
ONETOONEPLUS{ιX:[DOT(x)^BLUE(x)], ιX:[DOT(x)^~BLUE(x)]}.

But human speakers seem to *understand* 'Most of the dots are blue' as an instruction to build a thought that involves a concept of cardinality-*subtraction*, rather than concept-*negation*:

#ιX:[DOT(x)^BLUE(x)] > (#ιX:DOT(x) − #ιX:[DOT(_)^BLUE(_)]).

Hypothesis: human linguistic meanings are instructions for how to generate concepts, from a stock of lexical concepts, via simple operations that *don't include* Function Application or Lambda Abstraction.

(**H1**) Lexical concepts can be monadic or dyadic. All the other *generable* concepts are monadic.
(**H2**) The core operations are limited forms of *conjunction, existential closure*, and *polarization*.
(**H3**) There is also a *very limited* operation of *abstraction*, corresponding to relative clause formation.

Given the facts about SMPL, which *doesn't* allow for the abstraction posited by (**H3**), it's worth getting clear about the posited abstraction operation and what it *adds* to SMPL.

(**H3**) There is also a *very limited* operation of *abstraction*, corresponding to relative clause formation:

[who$_1$ [[a poet] [saw t$_1$]]]	[who$_2$ [t$_2$ [saw [a poet]]]]
IS-SUCH-THAT-A-POET-SAW-HER$_1$(_)	IS-SUCH-THAT-SHE$_2$-SAW-A-POET(_)

—This operation exploits a far more basic feature of thought: concepts can include **indices**.
In particular, there is least one conceptual analog of 'that' as used in 'that person'—a concept **:_(_)** such for each context **K** and index *i*, the concept :*i*(_) applies to whatever **K** assigns to *i*.
Think of contexts as determining which (possible) assignments of values to indices are *germane*.

—There are also conceptual analogs of 'I', 'here', and 'now': **SPEAKER**(_), **PLACE**(_), **TIME**(_).
But here, all that matters is that there are concepts like **:1(_)**^PERSON(_) and :**2**(_)^PERSON(_).
So a sentential expression like 'a poet saw that person' can correspond to a polarized concept like
⇑{Past(_)^∃[ExperiencedBy(_, _)^Poet(_)]^∃[SeeingOf(_, _)^[**:1(_)**^PERSON(_)]]}
And a sentential expression like 'that person saw a poet' can correspond to a polarized concept like
⇑{Past(_)^∃[ExperiencedBy(_, _)^[**:2(_)**^PERSON(_)]]^∃[SeeingOf(_, _)^Poet(_)]}

—Concepts that include **indices** have Kaplanian "characters" that determine contents in contexts.
One can still think of meanings as *instructions for how to build* concepts, as opposed to mappings from contexts to contents. But the *concepts built* will often have contents that are context-dependent.

—Think of 'wh'-abstraction as the grammatical correlate of a Tarskian operation that converts a concept of the form '⇑…:*i*(_)…' (or '⇓…:*i*(_)…') into an abstracted monadic concept of the form
'*i*(_):⇑…*i*…' (or '*i*(_):⇓…*i*…'); where relative to any assignment $\mathcal{A}$ of values to indices, the abstracted monadic concept applies to x iff the (embedded) polarized concept applies to x relative to the assignment that is just like $\mathcal{A}$ except for assigning x to the index *i*.

—Then relative to any assignment $\mathcal{A}$, the abstracted concept
1(_):⇑{Past(_)^∃[ExperiencedBy(_, _)^Poet(_)]^∃[SeeingOf(_, _)^[**:1(_)**^PERSON(_)]]}
applies to x if and only if the polarized concept
⇑{Past(_)^∃[ExperiencedBy(_, _)^Poet(_)]^∃[SeeingOf(_, _)^[**:1(_)**^PERSON(_)]]}
applies to x relative to the assignment that is just like $\mathcal{A}$ except for assigning x to the index **1**. So the abstracted concept applies to x iff (something/everything/x is such that) x is a person and a poet saw x.

This recodes Lambda Abstraction (as in Church 1941) for the *very special case* of *polarized* concepts, targeting an *indexed monadic* concept. This special case is a slight variant of Lambda Abstraction on expressions of type <t>, targeting an index of type <e>. But at least as envisioned by Church—and Frege, whose ideas Church was recasting in light of Tarski's treatment of quantifiers and variables—abstraction isn't limited to indices of type <e>. On the contrary, Frege and Church wanted abstraction on expressions of type <e, <e, t>>. So while it's useful to *compare* the limited operation posited by (**H3**) to the more familiar <t>-to-<e, t> abstraction, we'll also need to think about *overgeneration*. But first, comparison.

Suppose that 'Saw(i_2, i_1)' is a doubly indexed sentence of type <t>, and that relative to any assignment $\mathcal{A}$, 'Saw(i_2, i_1)' indicates truth or falsity, depending on whether or not the thing assigned to the second index saw the thing assigned to the first index. Abbreviating: $||\text{Saw}(i_2, i_1)||^{\mathcal{A}}$ = TRUE iff $\mathcal{A}$(2) saw $\mathcal{A}$(1). Then relative to any assignment $\mathcal{A}$, abstracting on 'i_1' corresponds to a function that maps each entity, x, to truth iff 'Saw(i_2, i_1)' indicates truth relative to an assignment $\mathcal{A}'$ that differs from $\mathcal{A}$ at most in that $\mathcal{A}'$ assigns x to the first index. Abbreviating: $||\mathbf{1}{:}\text{Saw}(i_2, i_1)||^{\mathcal{A}} = \lambda x.\text{TRUE}$ iff $\exists\mathcal{A}'{:}\mathcal{A}' \approx_1 \mathcal{A}[||\text{Saw}(i_2, i_1)||^{\mathcal{A}'} = \text{TRUE}]$. Likewise: $||\mathbf{2}{:}\text{Saw}(i_2, i_1)||^{\mathcal{A}} = \lambda x.\text{TRUE}$ iff $\exists\mathcal{A}'{:}\mathcal{A}' \approx_2 \mathcal{A}[||\text{Saw}(i_2, i_1)||^{\mathcal{A}'} = \text{TRUE}]$. Abbreviating more, as in Heim & Kratzer (1998): $||\mathbf{1}{:}\text{Saw}(i_2, i_1)||^{\mathcal{A}} = \lambda x.||\text{Saw}(i_2, i_1)||^{\mathcal{A}:1\rightarrow x}$; $||\mathbf{2}{:}\text{Saw}(i_2, i_1)||^{\mathcal{A}} = \lambda x.||\text{Saw}(i_2, i_1)||^{\mathcal{A}:2\rightarrow x}$.

The operation posited by (**H3**) can be viewed as a variant of <t>-to-<e, t> abstraction, at least if one adopts the simplifying (Tarskian, first-order) assumption that each assignment assigns *exactly one* thing to each index. Pietroski (2018) argues, following Schein (1993, 2002), that it's better to relax this assumption and let an assignment assign *one or more* things to an index if that index is *marked plural*; cp. Boolos (1998). But for now, let's not worry about clauses like 'people who gathered in the park'. The more important point is that while one can think of abstraction as an inversion of composition, one shouldn't assume that the basic principle of composition is Function Application, much less a *general* version of Function Application that allows for endlessly many higher-order types.

One can easily imagine a Fregean variant of English that allows for extraction of the sort indicated below.

$$[\text{Socrates}_{\langle e\rangle}\ [\text{saw}_{\langle e,\langle e,t\rangle\rangle}\ \text{Aristotle}_{\langle e\rangle}]_{\langle e,t\rangle}]_{\langle t\rangle} \rightarrow$$
$$[\text{whonk}_1\ [\text{Socrates}_{\langle e\rangle}\ [t_1\ \ \text{Aristotle}_{\langle e\rangle}]_{\langle e,t\rangle}]_{\langle t\rangle}]_{\langle\langle e,\langle e,t\rangle\rangle,t\rangle}$$

Abstraction on the expression of type <e, <e, t>> corresponds to a function of type <<e, <e, t>>, t> that maps a first-order relation **R** to truth or falsity, depending on whether or not Socrates bears **R** to Aristotle. Given two functions of type <e, <e, t>>, one can be represented as the transitive closure of the other.

$$[\text{Precedes}_{\langle e,\langle e,t\rangle\rangle}\ [\text{transits}_{\langle\langle e,\langle e,t\rangle\rangle,\langle\langle e,\langle e,t\rangle\rangle,t\rangle\rangle}\ \text{Predecessor}_{\langle e,\langle e,t\rangle\rangle}]_{\langle\langle e,\langle e,t\rangle\rangle,t\rangle}]_{\langle t\rangle}$$

So one can imagine abstractions of the sort indicated below.

$$[\text{Precedes}_{\langle e,\langle e,t\rangle\rangle}\ [\text{transits}_{\langle\langle e,\langle e,t\rangle\rangle,\langle\langle e,\langle e,t\rangle\rangle,t\rangle\rangle}\ \text{Predecessor}_{\langle e,\langle e,t\rangle\rangle}]_{\langle\langle e,\langle e,t\rangle\rangle,t\rangle}]_{\langle t\rangle} \rightarrow$$
$$[\text{whunk}_1\ [\text{Precedes}_{\langle e,\langle e,t\rangle\rangle}\ [t_1\ \ \text{Predecessor}_{\langle e,\langle e,t\rangle\rangle}]_{\langle\langle e,\langle e,t\rangle\rangle,t\rangle}]_{\langle t\rangle}]_{\langle\langle\langle e,\langle e,t\rangle\rangle,\langle\langle e,\langle e,t\rangle\rangle,t\rangle\rangle,t\rangle}$$

To make that more readable, let 'R' be an abbreviation for '<e, <e, t>>'.

$$[\text{Precedes}_{R}\ \ [\text{transits}_{\langle R,\langle R,t\rangle\rangle}\ \ \text{Predecessor}_{R}]_{\langle R,t\rangle}]_{\langle t\rangle} \rightarrow$$
$$[\text{whunk}_1\ [\text{Precedes}_{R}\ \ [t_1\ \ \text{Predecessor}_{R}]_{\langle R,t\rangle}]_{\langle t\rangle}]_{\langle\langle R,\langle R,t\rangle\rangle,t\rangle}$$

Such abstraction corresponds to a function that maps functions of type <R, <R, t>> to truth or falsity, depending on whether or not the "outer" relation is the transitive closure of the "inner" relation. Given two functions of type <<R, <R, t>>, t>.... Moreover, note the absence of natural relativizers like 'whut'.

$$[[\text{Plato talked}]_{\langle t\rangle}\ [\text{and}_{\langle\langle t,\langle t,t\rangle\rangle}\ [\text{Frege walked}]_{\langle t\rangle}]_{\langle t,t\rangle}]_{\langle t\rangle} \rightarrow$$
$$[\text{whut}_1\ [[\text{Plato talked}]_{\langle t\rangle}\ [\text{and}_{\langle\langle t,\langle t,t\rangle\rangle}\ \ t_1\ \ ]_{\langle t,t\rangle}]_{\langle t\rangle}]_{\langle t,t\rangle}$$

Fregean languages are *designed* to allow for expressions of endlessly many types: <e>; <t>; and <α, β> if <α> and <β> are types. Prima facie, this typology overgenerates *wildly*, compared with the lexical types and possible targets for abstraction in *natural* languages; cp. Chierchia (1984).

The Lowest Fregean Types (described iteratively)

0.	<e> <t>		(2) basic types
1.	<e, e> <e, t> <t, e> <t, t>		(4) of <0, 0>
2.	eight of <0, 1> sixteen of <1, 1>	eight of <1, 0>	(32), including <e, et> and <et, t>
3.	64 of <0, 2> 128 of <1, 2> 1024 of <2, 2>	64 of <2, 0> 128 of <2, 1>	(1408), including <e, <e, et>> and <et, <et, t>> and <<e, et>, t>
4.	2816 of <0, 3> 5632 of <1, 3> 45,056 of <2, 3> 1,982,464 of <3, 3>	2816 of <3, 0> 5632 of <3, 1> 45,056 of <3, 2>	(> 2M), including <e, <e, <e, et>>> and <<e, et>, <<e, et>, t>>
5.	...		> 5×10^{12}

By contrast, (**H3**) posits an abstraction operation that applies *only* to (polarized) concepts of type <**M**> and yields *only* (unpolarized) concepts of the same type.

The generable concepts do not include instances of type <t> or type <e>, much less instances of higher types like <<e, t>, t> or <<R, <R, t>>, t>.

Some facts (via Icard & Moss) about SMPL **plus** *the limited operation of abstraction*

—Polarizing provides an analog of Aristotelian predicate-denial, but no analog of predicate-complement.
The sentence 'That is not a cow' corresponds to ⇓[↜**:1(_)**^COW(_)], presupposing ⇑↜**:1(_)**.
However, *given abstraction*, one can build analogs of negative concepts—concepts that correspond to relative clauses like 'which is not a cow', in which 'not' indicates predicate-denial.
1(_):⇓[↜**:1(_)**^COW(_)]
In this sense, abstraction lets you use COW(_) to *build* a concept that is *equivalent* to ~COW(_).

—There is a sense in which the posited concept-generator, SMPL$^+$, is equivalent to first-order logic.
Given some atomic concepts, each complex concept is equivalent to some first-order concept that has exactly one free variable. More surprisingly, each such first-order concept is equivalent to a concept generated by SMPL$^+$. In this sense, adding abstraction to SMPL is a *significant but not wild* addition.

By design, first-order logic allows for expressions of the forms 'Δxy & Mx' and '∃x[Δxy & Mx]' and '∃y[Δxy & Mz]'—along with instances of 'Δxy & My' and '∃y[Δxy & My]'—and expressions with arbitrarily many variables, each of which can have arbitrarily many occurrences in a single expression. So a complex 1-place predicate can be formed by forming a 17-place predicate and quantificationally binding 16 variables, or more generally, by forming an *n*-place predicate and binding *n*-1 variables.

By contrast, SMPL allows for variable-free analogs of '∃y[Δxy & My]', *but not* analogs of '∃x[Δxy & Mx]'. Analogs of complex polyadic concepts—including 'Δxy & Mz', but also 'Δxy & Mx', 'Δxy & My', and 'Mx & My'—are beyond the pale. The '∃' in '∃Δ.M' *is not* a quantifier like '∃x', whose variable can appear in various positions with various predicates. Likewise, '⇑' *is not* a quantifier like '∃e', whose variable can appear in various positions with various predicates, and '⇓' *is not* a negated quantifier like '~∃e'. This already makes it a little interesting that given a stock of atomic concepts, the generable monadic first-order concepts can be mimicked by the concepts generable via the posited *limited* forms of conjunction, existential closure, polarization, and abstraction. But another fact adds to the interest.

SMPL$^+$ can be described as a context-free rewrite procedure. So given some atomic concepts, the corresponding set of SMPL$^+$y-generable concepts can be described in context-free terms. But not so for the corresponding sets of "first-order-logically-generable" monadic concepts, which can be formed by forming concepts that are *massively and intricately polyadic* and then *binding every variable except one*. And for each first-order concept **C**, of whatever adicity, there is also the concept ~**C**; but SMPL$^+$ can only generate negative concepts *cyclically*, by generating a polarized concept and abstracting in a limited way.

Chomsky (1986) distinguished I-languages (expression-generating *procedures*) from E-languages (*sets* of expressions or languages in some other sense). This echoed Church's (1941) contrast between functions-in-intension (computational procedures) and functions-in-extension (sets of input-output pairs), which echoed Frege's (1892) contrast between Functions (unsaturated mappings, or rules for determining a value given an argument) and "courses of values" (sets of ordered pairs in which the first element is mapped to the second by some Function). So unsurprisingly, Chomsky viewed the "Chomsky Hierarchy" as a way of describing *procedures* and the computational resources required to execute them, not as a way of describing *sets of strings* in terms of the procedures required to characterize them. (Though reference to sets can be useful in *proofs*, especially for negative results, even if the theorems concern procedures.)

So one might care about the contrast between (i) a concept-generator that works like a definition of well-formedness for first-order logic, but "filters out" formulae that don't have exactly 1 free variable, and (ii) a concept-generator that works like SMPL$^+$. If (i) and (ii) are interestingly *equivalent*, that's great. Faced with minds that generate concepts in this ballpark, we can ask if these minds generate the concepts in question by using the more powerful resources, the less powerful resources, or in some other way. (Compare: ContextFree, ContextSensitive, or other; if other… CF+transformations, TAG, or other; if other…) We can also contrast procedures that generate complex concepts of *endlessly many* types and *filter out endlessly many* of those types, with procedures that generate complex concepts of *one* type.

Hypothesis: human linguistic meanings are instructions for how to generate concepts, from a stock of lexical concepts, via simple operations that *don't include* Function Application or Lambda Abstraction.

A Line of Thought that can Help Motivate the bold Hypothesis (cp. Chomsky 1957, 1959, 1964)

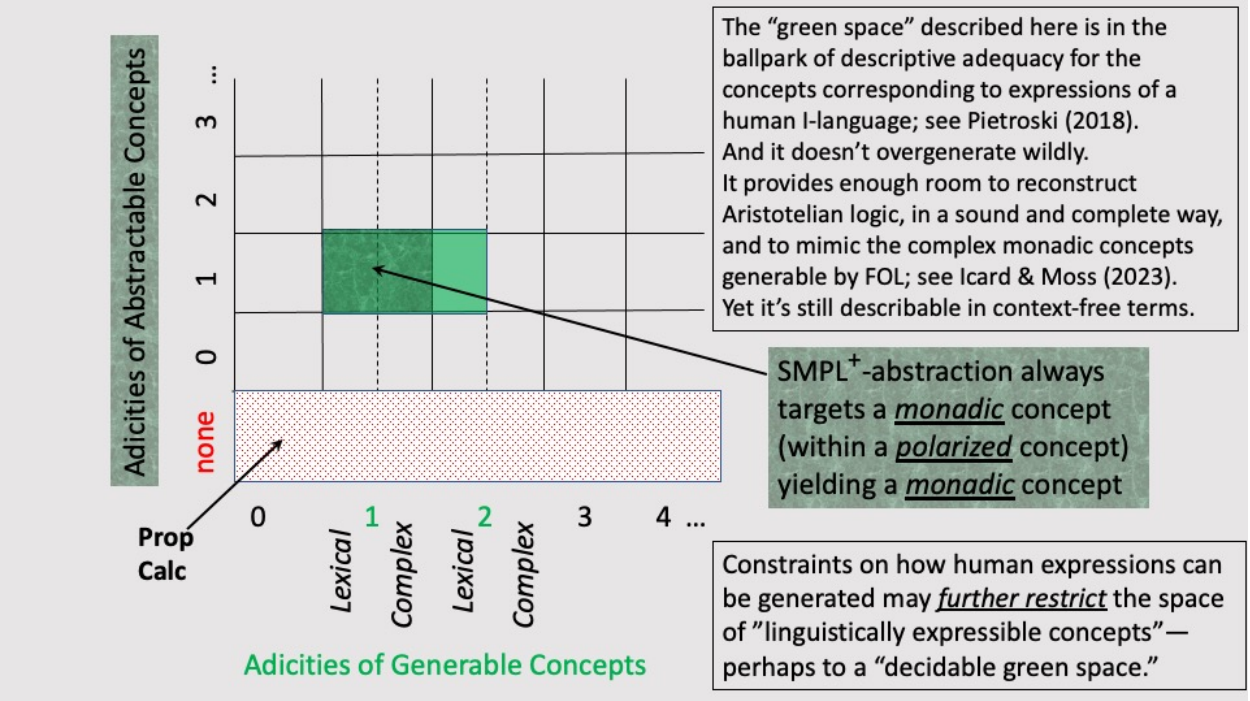
1. Humans naturally acquire grammars that generate expressions in ways that allow for homophony and polysemy, subject to substantive constraints.
2. Homophony, lexical or structural, reflects the fact that a grammar can generate distinct expressions that share a pronunciation. At least to a first approximation: a pronunciation π is homophonous in *n* (but not *n*+1) ways, relative to a grammar G, if and only if G generates *n* (but not *n*+1) expressions that have pronunciation π. If we describe expressions as *strings* of lexical expressions, allowing that a single string can have more than one "reading/meaning," then the readings/meanings correspond to *ways of deriving* strings from a grammar.
3. While human grammars are not context-free generative procedures, they generate expressions that exhibit *constituency structure* of an interesting kind; and while human grammars permit (and sometimes mandate) *transformations*, the combinatorial operations don't obliterate constituency. In particular, the cognitive systems that support the natural acquisition of human grammars do not support the acquisition of context-sensitive generative procedures that permit "shuffling" of the sort required to generate (all and only) the "$a^n b^n c^n$"-strings from an alphabet consisting of 'a', 'b', and 'c'.[1] Likewise, the cognitive systems that support human grammar acquisition do not support the acquisition of context-sensitive rules like "NP AUX VP ➔ AUX NP VP", even if such a rule would make it easy to generate both 'they will eat' and 'will they eat'. (Ditto, with a vengeance, for the AUX system itself: affix-hopping, yes; shuffling, no.)
4. Prima facie, linguistic meanings also exhibit constituency structure that can't be obliterated. And we shouldn't assume that the readings/meanings of homophonous strings *can't* be characterized in context-free terms. Absent evidence that more powerful computational resources are *needed* to describe these meanings, we should try to describe them in sparer (perhaps even SMPL$^+$) terms.
5. Of course, humans enjoy cognitive resources that go beyond those used in acquiring grammars. Maybe many animals have computational resources that go beyond those of Push-Down Automata. Some animals can even form the thought that any computable function can be computed by a Turing Machine; but so far as we know, all such animals acquire a grammar *and then* learn to count *and then* learn some math. So we shouldn't assume, at the outset, that humans *construct meanings* by employing computational resources that go beyond those of a Push-Down Automaton.

Idsardi's (2018) Conjecture: while human grammars employ computational resources that go beyond Context Free Rewriting (Push-Down Automata), the generable expressions interface with phonological and semantic systems in computationally *sparer* ways; "sophisticated syntax" is how nature managed to forge systematic connections between *less sophisticated* meanings/concepts and pronunciations.

Generable expressions connect "phonological instructions" (PFs) with "semantic instructions" (LFs). But typically, PF-derivations and LF-derivations diverge, resulting in mismatches between the PFs that interface with Articulatory-Perceptual systems and the LFs that interface with Conceptual-Intentional systems; cp Chomsky (2000). Evidently, the interfacing systems impose different demands on the forms of generable expressions; and evidently, it's not an option to just "linearize" thoughts (that are un-SMPL) and pronounce them "directly." Perhaps this will be part of the story for why speech is unattested among non-human animals that can vocalize and exhibit verbal learning; cp. Berwick and Chomsky (2017).

---

[1] The following rewrite rules do the job: (i) S ➔ aSBC; (ii) S ➔ aBC; (iii) CB ➔ BC; (iv) aB ➔ ab; (v) bB ➔ bb; (vi) bC ➔ bc; (vii) cC ➔ cc. The context-free (i) and (ii) yield 'aaaBCBCBC'. But the context-sensitive (iii)—or more explicitly, the chain CB ➔ CX ➔ YX ➔ BX ➔ BC—licenses conversion to 'aaaBBBCCC'; and the context-sensitive (iv-vii) permit conversion to 'aaabbbccc.' More generally, rules like (iii) would permit arbitrary shuffling of segments that would otherwise be constituents, given rules like (i) and (ii).



References

Barwise, J. and Cooper, R. (1981). Generalized Quantifiers and Natural Language.

Berwick and Chomsky (2017). *Why Only Us*?

Boolos, G. (1998). *Logic, Logic, and Logic*.

Chierchia (1984). *Topics in the Syntax and Semantics of Infinitives and Gerunds*.

Chomsky (1957). *Syntactic Structures*.

Chomsky (1959). On Certain Properties of Formal Grammars.

Chomsky (1964). *Current Issues in Linguistic Theory.*

Chomsky (1986). *Knowledge of Language*.

Chomsky (2000). *The Minimalist Program*.

Church (1941). *The Calculi of Lambda Conversion*.

Frege (1892). Function and Concept.

Heim, I. & Kratzer, A. (1998). *Semantics in Generative Grammar*.

Horn (1989). *A natural history of negation*.

Icard & Moss (forthcoming). A simple logic of concepts. *Journal of Philosophical Logic*.

Idsardi, W. (2018). Why is Phonology Different? No recursion.

Lidz, et.al. (2011). Interface Transparency and the Psychosemantics of 'Most'.

Pietroski, et. al. (2009). The Meaning of 'Most': semantics, numerosity, and psychology.

Pietroski (2018). *Conjoining Meanings*.

—*see also* "Limiting Semantic Types" in *Language, Syntax, and the Natural Sciences* (R. Martin and A. Gallego, eds.), 192-211, Cambridge University Press (2018).

Post (1943). Formal Reductions of the General Combinatorial Decision Problem. *Am. Journal of Math*.

Schein (1993). *Events and Plurals*

Schein (2002). Events and the Semantic Content of Thematic Relations.

Tarski (1944). The Semantic Conception of Truth.