Using Minimum Description Length to Compare Different Logical Primitives

Idan Drori
205983406

1 Introduction

Understanding the cognitive and representational underpinnings of language acquisition and processing is one of the foundational challenges in linguistics. A critical aspect of this question is determining the logical and set-theoretic primitives that underlie our linguistic representations. Building on the framework proposed by Katzir et al., 2020, this paper attempts to compare different formalisms that represent quantifiers.
The choice of primitive operations within a formal framework is an implicit claim about how linguistic knowledge is structured and stored. We would want such claims to be tested empirically, as two different sets of primitive operations can be equally expressive-capable of expressing the same quantifiers, but still make divergent predictions regarding learnability.
We'll use a formalism that we'll call Building Blocks (BB), which claims that linguistic knowledge is encoded via a set of simple logical and set operators, that are then combined in some way in order to generate more complex units of meaning. To evaluate these frameworks, we'll use Minimum Description Length (MDL), a metric that balances representational simplicity with empirical fit, offering insights into how learners might select among alternative hypotheses during language acquisition.
Because our method of evaluation focuses on learning, we'll focus on quantifiers that need to be learned and stored, and ignore expressions that function as quantifiers but are syntactically complex, and therefore don't need to be lexically stored.

2 Background

2.1 Building Blocks

First we'll discuss the representational model that we're calling Building Blocks, loosely inspired by Keenan and Stavi, 1986, and assumed in later works (Piantadosi et al., 2012, Piantadosi et al., 2016, Hackl, 2009). This approach assumes that people have a set of simple logical operators (like $\wedge$, $\neg$, $\vee$, etc), set operators (like $\cap$, $\cup$, etc), which are then combined in some way in order to form more complex quantificational expressions. In our case we'll assume that the way these primitives are combined is with a context-free grammar. For example:
SimpleBoolean

$START$	$\rightarrow \lambda x.BOOL$
$BOOL$	$\rightarrow (BOOL \wedge BOOL) \mid (BOOL \vee BOOL) \mid (\neg BOOL) \mid True \mid False$
	$\rightarrow (F\ x)$
F	$\rightarrow COLOR \mid SHAPE \mid SIZE$
$COLOR$	$\rightarrow blue? \mid green? \mid red?$
$SHAPE$	$\rightarrow circle? \mid square? \mid triangle?$
$SIZE$	$\rightarrow small? \mid medium? \mid large?$

Each rule in a context-free grammar is of the form $A \rightarrow B$, meaning we turn A into B. The $\mid$ symbol is shorthand for denoting multiple rules, so a rule of the form $A \rightarrow B \mid C$ is actually a rule $A \rightarrow B$ and a rule

$A \rightarrow C$.
Using SimpleBoolean we can generate expressions like: $\lambda x.(blue?x) \wedge (circle?x)$, which gives us a predicate function that returns true if and only if the input is blue and is a circle.
SimpleBoolean here is universal, meaning all propositional formulas can be written using SimpleBoolean. However, there are more universal grammars, for example:
NAND

$START$	$\rightarrow \lambda x.BOOL$
$BOOL$	$\rightarrow (BOOL\ NAND\ BOOL) \mid True \mid False$
	$\rightarrow (F\ x)$
F	$\rightarrow COLOR \mid SHAPE \mid SIZE$
$COLOR$	$\rightarrow blue? \mid green? \mid red?$
$SHAPE$	$\rightarrow circle? \mid square? \mid triangle?$
$SIZE$	$\rightarrow small? \mid medium? \mid large?$

Where $NAND$ is not-and.
As mentioned, both SimpleBoolean and NAND are universal, all propositional formulas can be written using both, the only difference between them are the primitives used. NAND is more compact in terms of number of rules, but in order to produce even seemingly simple expressions we need to apply our rules more times than had we used SimpleBoolean. For example, let's take the set of blue or red entitites:

Using SimpleBoolean: $\lambda x.(blue?\ x)\ \vee\ (red?\ x)$

Using NAND: $\lambda x.((blue?\ x)\ \ NAND\ (blue?\ x))\ NAND\ ((red?\ x)\ NAND\ (red?\ x))$

Using SimpleBoolean we have 4 rule applications of some kind in order to reach our final expression, as opposed to NAND where we have 8 rule applications. Both expressions have the same truth conditions, but clearly the one using SimpleBoolean is easier to comprehend and "feels" more intuitive. We'll discuss this notion of intuitiveness more in the next section when we talk about MDL, our method of evaluation. But first we'll just show how to turn these propositional formulas into actual quantifiers, to do this we need to operate on sets.
We first note that these propositional formulas generated by SimpleBoolean and NAND are functions from entities (e) to truth values (t), i.e. they're of type $\langle e, t\rangle$. These are characteristic functions of sets, they map individuals that are elements of the set to $True$ and everything else to $False$. In other words, a function like $\lambda x.(red?\ x)$ is the characteristic function of all red entities, and so we can treat this function as denoting the set of all red entities. Similarly, any other expression we can generate using SimpleBoolean or NAND will ultimately be of type $\langle e, t\rangle$, and so will represent a set.
Now building on SimpleBoolean and NAND we can define a context-free grammar that will function on sets and output quantificational expressions:
SetOps

$START$	$\rightarrow \lambda A, B.\ BOOL$
$BOOL$	$\rightarrow True \mid False$
	$\rightarrow (card_{>}\ SET\ SET) \mid (card_{=}\ SET\ SET)$
	$\rightarrow SET \subseteq SET$
	$\rightarrow (empty?\ SET) \mid (nonempty?\ SET)$
SET	$\rightarrow (SET\ \cap\ SET) \mid (SET\ \cup\ SET) \mid (SET \setminus SET)$
	$\rightarrow (\neg SET) \mid A \mid B$

Some explanations regarding notation:

A, B are formed from SimpleBoolean or by NAND. Meaning, A and B are of type $\langle e, t\rangle$, they're sets.

$card_>$ X Y returns $True$ if and only if $|X| > |Y|$, and $card_=$ X Y returns $True$ if and only if $|X| = |Y|$.

($empty?$ SET) returns $True$ if and only if SET is an empty set. Similarly, ($nonempty?$ SET) returns $True$ if and only if SET is not an empty set.

Note that we have some redundancy in SetOps (we also have redundancy in SimpleBoolean), this is intentional. Using SetOps and SimpleBoolean we can now represent expressions like "Every circle is red":

$$\lambda A, B.(empty?(A \backslash B)), \text{ alternatively we can use } \lambda A, B.(A \subseteq B)$$

Where $A = \lambda x.(circle?\ x)$, $B = \lambda x.(red?\ x)$.

We can also represent expressions like "Most circles are red":

$$\lambda A, B.(card_>\ , A \cap B,\ A \backslash B)$$

We'll also define another CFG that will produce quantifiers:
SetNAND

$START$	$\rightarrow \lambda A, B.\ BOOL$
$BOOL$	$\rightarrow True \mid False$
	$\rightarrow (card_>,\ SET,\ SET) \mid (card_=,\ SET,\ SET)$
	$\rightarrow SET \subseteq SET$
	$\rightarrow (empty?\ SET) \mid (nonempty?\ SET)$
SET	$\rightarrow (SET\ \overline{\cap}\ SET)$
	$\rightarrow A \mid B$

$A\overline{\cap}B$ would be more properly written as $\overline{A \cap B}$, with the meaning of "the complement of the intersection of A and B". We're using a single character somewhat improperly for less cumbersome notation (perhaps it's also clearer) and for ease of calculation later on.

This is very similar to SetOps but the only set operation it has is the negation of set intersection. This is the set equivalent of only using NAND in the boolean case.
Using SetNAND and SimpleBoolean our representation for "Every circle is red" would be the same as using SetOps and SimpleBoolean, but if we take "Most circles are red", we get:

$$\lambda A, B.(card_>,\ ((A\overline{\cap}B)\overline{\cap}(A\overline{\cap}B)),\ (A\overline{\cap}(B\overline{\cap}B)))$$

Where $A = \lambda x.(circle?\ x)$, $B = \lambda x.(red?\ x)$.

2.2 Learning and Minimum Description Length

Having explained BB as a formalistic lens through which to look at quantifiers, we now move to discuss what such formalisms achieve and thereby tackle the question of grammar. The evaluation metric we'll be using is that of Minimum Description Length (MDL; Rissanen, 1978), which balances two competing factors: (a) the complexity of the grammar; and (b) its fit to the data. By doing so, MDL combines the perspectives of two other evaluation metrics that have been used within the generative tradition: the simplicity metric of Chomsky and Halle, 1991, which minimizes the complexity of the grammar, and the subset principle advocated in later work such as Dell, 1981, which maximizes the fit of the grammar to the data.
Given some grammar G, the MDL of G would simply be: $|G| + |D : G|$. Meaning we're measuring the length of the grammar $|G|$, and the length of the data as encoded by that grammar $|D : G|$. The length of the grammar is roughly a measure of simplicity, and the length of the data encoded by that grammar is roughly a measure of fitting the data.
The basic idea of the simplicity metric is to only measure $|G|$, but if this is our only metric, we'll end up

preferring overly general grammars, such that they can generate unattested outputs. And on the other hand, if we only look at fitting the data, i.e. only minimizing $|D : G|$, we'll end up preferring grammars that aren't general enough, such as grammars that simply memorize all the data.
As implied by the "Minimum" in Minimum Description Length, the idea is that learners prefer the grammar that minimizes $|G| + |D : G|$.
Stated more formally:

> MDL Evaluation Metric: If G and G' can both generate the data D, and if $|G|+|D : G| < |G'|+|D : G'|$, then prefer G to G'.

As an example, let's look at the following data:

```
pabikugolatudaropitibudogolatutibudogolatugolatupabikutibudodaro
pitibudopabikugolatudaropipabikutibudogolatu
```

Looking at the data, one can clearly make out "words", it's not just random letter, there's a pattern. We would ideally want our grammar to reflect this, and we'll see that when comparing grammars with MDL the one with the smaller $|G| + |D : G|$ is the one that captures this. We'll define the following grammar G_1:

$$\begin{aligned} S &\rightarrow W\ S \\ W &\rightarrow a \mid b \mid d \mid g \mid i \mid k \mid l \mid o \mid p \mid r \mid t \mid u \end{aligned}$$

This grammar is a maximally simple grammar that produces the above data. If we assume our alphabet is only 12 letters long then we can measure the length of our grammar to be:

$$|G_1| \approx \lceil \log_2 12 \rceil \cdot 12 = 48$$

The actual length is longer because we're overlooking things like seperators, nonterminals, etc, but this doesn't really matter here. The idea is that in order to encode a 12 letter alphabet we need $\lceil \log_2 12 \rceil = 4$ bits for each letter, and so the whole grammar is (roughly) $4 \cdot 12 = 48$ bits long. But in order to encode the data using this grammar, each letter in the data string must be encoded separately, then we need 4 bits per character, and our data has 108 characters, so we get $|D : G| = 432$.:

$$|D : G_1| = \lceil \log_2 12 \rceil \cdot |D| = 4 \cdot 108 = 432$$

Now for another grammar G_2:

$$\begin{aligned} S &\rightarrow W\ S \\ W &\rightarrow \textit{pabiku} \mid \textit{golatu} \mid \textit{daropi} \mid \textit{tibudo} \end{aligned}$$

Here the length of G_2 is longer than G_1:

$$|G_2| \approx \lceil \log_2 12 \rceil \cdot 6 \cdot 4 = 96$$

But the encoded data is significantly shorter than in our first grammar:

$$|D : G_2| = \lceil \log_2 4 \rceil \cdot \frac{|D|}{6} = 2 \cdot \frac{|D|}{6} = \frac{108}{3} = 36$$

Comparing the MDL of G_1 and G_2 we get:

$$|G_1| + |D : G_1| = 48 + 432 = 480 > 132 = 96 + 36 = |G_2| + |D : G_2|$$

Meaning G_2 is the "better" grammar, which fits our intuition that G_2 is the better generalization of the data.
Another argument in favor of MDL is that it's been shown that MDL-based learners actually work, see Rasin et al., 2018a and Rasin et al., 2018b.

3 Comparison

We'll first explain the setup with which our learner is acquiring their representations: We assume that the learner observes some set of objects; circles, squares and triangles of 3 possible colors and 3 possible sizes, and then hears a speaker utter a quantified expression. The quantifier used is generated according to the correct lexicon of the language. The learner must then use the observed word usages in this context in order to infer word meanings. Now, we want to compare the MDL of BB's with different primitives, we'll compare between the following "languages": SimpleBoolean+SetOps, SimpleBoolean+SetNAND, NAND+SetOps and NAND+SetNAND.
Using these four combinations, we'll construct representations for a few quantifiers. Because MDL is ultimately a metric for learning, we'll be looking at quantifiers that need to be acquired and stored, as opposed to expressions that function as quantifiers but are syntactically complex (we'll be looking at quantifiers like "some", and not at quantificational expressions like "at least 4"). For each quantifier, our "grammar" will be the representation according to the specific language. So for each language we'll fix a quantifier, and represent it, then measure $|G| + |D : G|$.

3.1 Representations and computing $|G|$

Our "grammars" will be the representations of the specific quantifiers according to a specific language, so we need to measure the "size" of this grammar. What we'll do is count each of the rules in the grammar, say we have k rules, then count the number of rule applications it took in order to reach our representation, let it be m. Then $|G| = \lceil \log_2 k \rceil \cdot m$.
It should be noted that a representation of a quantifier using languages like ours is not singular, there are infinite ways to represent one quantifier, but there's generally only one representation that is compact and "reasonable". For example, if we use SimpleBoolean to create and expression for "the set of red objects", we would do $\lambda x.(red?\ x)$, but we can also do $\lambda x.(red?\ x) \vee False \vee False \vee \cdots \vee False$, but clearly the first representation is the reasonable one.

3.2 Constructing the test data

Our learning scenario is such that each quantifier is learned "in a vacuum", the learner knows the word they're ought to learn and the only thing that matters is whether the representation they currently have produces a truth condition that matches the context they observe. In order to decide whether some representation of a quantifier is correct, i.e. it fits the data, we only need our data to encode the context. We can thus represent our data as strings of the form: $\langle T, C, C, S, \# \rangle$. Where T, C, S represent a triangle, circle and square respectively, and the color of the character represents the color of the object. The # character simply denotes the end of the input.
For example, if a learner has two competing representations of the lexical item "Most":

(a) $\lambda A, B.(card_{>}\ , A \cap B,\ A \backslash B)$ - Correct

(b) $\lambda A, B.(A \subseteq B)$ - Incorrect (this is the correct representation of "Every")

And our learner hears the expression "Most circles are red", with the following context in front of them: $\langle C, C, C, C, \# \rangle$. Then representation (b) is a worse fit to the data, the truth conditions don't match the context, in MDL terms $|D : G|$ should suffer and increase in size.

3.2.1 Computing $|D : G|$

We now need to tackle the problem of actually computing $|D : G|$. As BB's don't give us a convenient way to parse input, we face the challenge of encoding data according to a specific grammar. So what we'll need to do is construct our data such that it will be unambiguous regardless of the representation (assuming vaguely reasonable representations). This way we can "neutralize" the impact of $|D : G|$, which is obviously not really what we want with MDL, but we're kind of forced to do this because of the way BB's are structured. Another option is to convert the representations into Deterministic Finite Automata and calculate $|D : G|$ using those, where it's very clear how to calculate $|D : G|$. But I will not do this for lack of time.

3.3 Details of the comparison

We'll divide our comparison into two parts, first we'll compare SimpleBoolean and NAND, and then we'll compare SetOps and SetNAND. This is because any difference between SetOps and SetNAND will be identical whether we're using SimpleBoolean or NAND, and likewise, the difference between SimpleBoolean and NAND will be identical whether we use SetOps or SetNAND. So we might as well compartmentalize things for clarity.

3.3.1 SimpleBoolean compared to NAND

For the comparison between SimpleBoolean and NAND we'll fix our quantifier to be "Every" and use SetOps. We'll represent the following: "Every square or triangle is red and large" The reason for testing this is that the difference between our boolean languages only emerges with more complex sets.

Language	Representation	$\|G\|$
SimpleBoolean	$A = \lambda x.(square?\ x) \vee (triangle?\ x)$, $B = \lambda x.(red?\ x) \wedge (large?\ x)$	40
NAND	$A = \lambda x.((square?\ x)\ \ NAND\ (square?\ x))\ NAND\ ((triangle?\ x)\ NAND\ (triangle?\ x))$ $B = \lambda x.(red?\ x)\ NAND\ ((red?\ x)\ NAND\ (large?\ x))$	70

Table 1: SimpleBoolean compared with NAND - "Every square or triangle is red and large"

We'll show the calculation used to reach the $|G|$'s given:

Number of rules in SimpleBoolean - 19, number of rules to reach output - 8

$$|G| = \lceil \log_2 19 \rceil \cdot 8 = 5 \cdot 8 = 40$$

Number of rules in NAND - 17, number of rules to reach output - 14

$$|G| = \lceil \log_2 17 \rceil \cdot 14 = 5 \cdot 14 = 70$$

It seems like NAND is worse, let's try and give NAND some better chances and look at: "Every non-square or non-triangle is not red or not large":

Language	Representation	$\|G\|$
SimpleBoolean	$A = \lambda x.\neg(square?\ x) \vee \neg(triangle?\ x)$, $B = \lambda x.\neg(red?\ x) \vee \neg(large?\ x)$	60
NAND	$A = \lambda x.(square?\ x)\ NAND\ (triangle?\ x)$ $B = \lambda x.(red?\ x)\ NAND\ (large?\ x)$	40

Table 2: SimpleBoolean compared with NAND - "Every non-square or non-triangle is not red or not large"

Substantially better for NAND, but the sets we used are somewhat unnatural. Perhaps there's also something to be said about the margin of the first comparison as opposed to the second (30 as opposed to 20), but I don't know if that's enough of a size difference to matter.

3.3.2 SetOps compared with SetNAND

For this comparison, we'll use SimpleBoolean with both languages and compare $|G|$ between the quantifiers: Every, Some and Most.
We're assuming $A = \lambda x.(circle?\ x), B = \lambda x.(red?\ x)$.
Both representation are identical, and SetNAND has less rules than SetOps (11 as opposed to 14), but $\lceil \log_2 11 \rceil = 4 = \lceil \log_2 14 \rceil$, so both $|G|$'s are the same.
It seems like SetNAND is worse than SetOps in pretty much every situation, and unlike with NAND, it's not as simple to find a situation where SetNAND comes out better than SetOps.

Language	Representation	$\|G\|$
SetOps	$\lambda A, B.(A \subseteq B)$	16
SetNAND	$\lambda A, B.(A \subseteq B)$	16

Table 3: SetOps compared with SetNAND - "Every circle is red"

Language	Representation	$\|G\|$
SetOps	$\lambda A, B.(nonempty?\ A \cap B)$,	20
SetNAND	$\lambda A, B.(nonempty?\ ((A \overline{\cap} B) \overline{\cap} (A \overline{\cap} B)))$	28

Table 4: SetOps compared with SetNAND - "Some circle are red"

Language	Representation	$\|G\|$
SetOps	$\lambda A, B.(card_{>}\ , A \cap B,\ A \backslash B)$,	32
SetNAND	$\lambda A, B.(card_{>},\ ((A \overline{\cap} B) \overline{\cap} (A \overline{\cap} B)),\ (A \overline{\cap} (B \overline{\cap} B)))$	56

Table 5: SetOps compared with SetNAND - "Most circle are red"

4 Conclusion

This paper has explored the implications of different choices of logical and set-theoretic primitives for representing quantifiers, focusing on what MDL can tell us about this choice. The central question is what each choice of primitives predicts about the efficiency of representations and their learnability when evaluated through MDL.

Our findings show that systems with minimal primitives, such as our NAND-based grammars, often predict greater complexity in representing quantifiers due to the frequent application of rules. Conversely, grammars like SimpleBoolean+SetOps usually give us more compact representations for the same quantifiers, despite the fact that they have a more expansive set of primitive operators.

This suggests that learners should have a more expansive and expressive set of primitives as opposed to a more compact set of primitives. One could perhaps experimentally test this conclusion by testing whether learners have an easier time learning quantifiers (presumably constructed quantifiers that don't exist in the language) which have identical or similar representational lengths according to SimpleBoolean+SetOps and according to NAND+SetNAND. As a rough sketch: Construct 2 quantifiers, A, B such that according to SimpleBoolean+SetOps $|G_A| \ll |G_B|$, but according to NAND+SetNAND $|G_B| \ll |G_A|$. If learners do indeed have a more expansive set of primitives then we would expect learners to have an easier time learning quantifer A, than quantifier B. Once again, this is a rough sketch, and properly conducting an experiment such as this is beyond my skillset.

One issue that severely weakens this conclusion is that of $|D : G|$, which we have largely sidestepped. We neutralized the effect of $|D : G|$ and only measured $|G|$ of our grammars, and in doing so didn't use MDL properly. Rather, we used a simplicity metric. This suggests that our method has the same issues that Simplicity has, preferring overly general and overly permissive grammars. More work needs to be done that would properly tackle the challenge of computing $|D : G|$ of Building Block-like grammars, in order to properly measure MDL for these kinds of representations.

Bibliography

Chomsky, N., & Halle, M. (1991). *The sound pattern of english* (First edition.). MIT Press.

Dell, F. (1981). On the learnability of optional phonological rules. *Linguistic inquiry*, *12*(1), 31–37.

Hackl, M. (2009). On the grammar and processing of proportional quantifiers: Most versus more than half. *Natural language semantics*, *17*(1), 63–98.

Katzir, R., Lan, N., & Peled, N. (2020). A note on the representation and learning of quantificational determiners. *Proceedings of Sinn und Bedeutung*, *24*(1), 392–410. https://doi.org/10.18148/sub/2020.v24i1.874

Keenan, E. L., & Stavi, J. (1986). A semantic characterization of natural language determiners. *Linguistics and Philosophy*, *9*(3), 253–326. http://www.jstor.org/stable/25001246

Piantadosi, S. T., Tenenbaum, J. B., & Goodman, N. D. (2012). Modeling the acquisition of quantifier semantics: A case study in function word learnability. https://api.semanticscholar.org/CorpusID:29918589

Piantadosi, S. T., Tenenbaum, J. B., & Goodman, N. D. (2016). The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological review*, *123*(4), 392–424.

Rasin, E., Berger, I., Lan, N., & Katzir, R. (2018a). Learning phonological optionality and opacity from distributional evidence. In S. Hucklebridge & M. Nelson (Eds.), *Nels 48: Proceedings of the forty-eighth annual meeting of the north east linguistic society* (pp. 269–282, Vol. 2). GLSA, University of Massachusetts.

Rasin, E., Berger, I., Lan, N., & Katzir, R. (2018b). Learning rule-based morpho-phonology. https://api.semanticscholar.org/CorpusID:226253747

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*(5), 465–471. https://doi.org/https://doi.org/10.1016/0005-1098(78)90005-5