Your Time is Important to Us

Please wait while we check the grammatically of this sentence

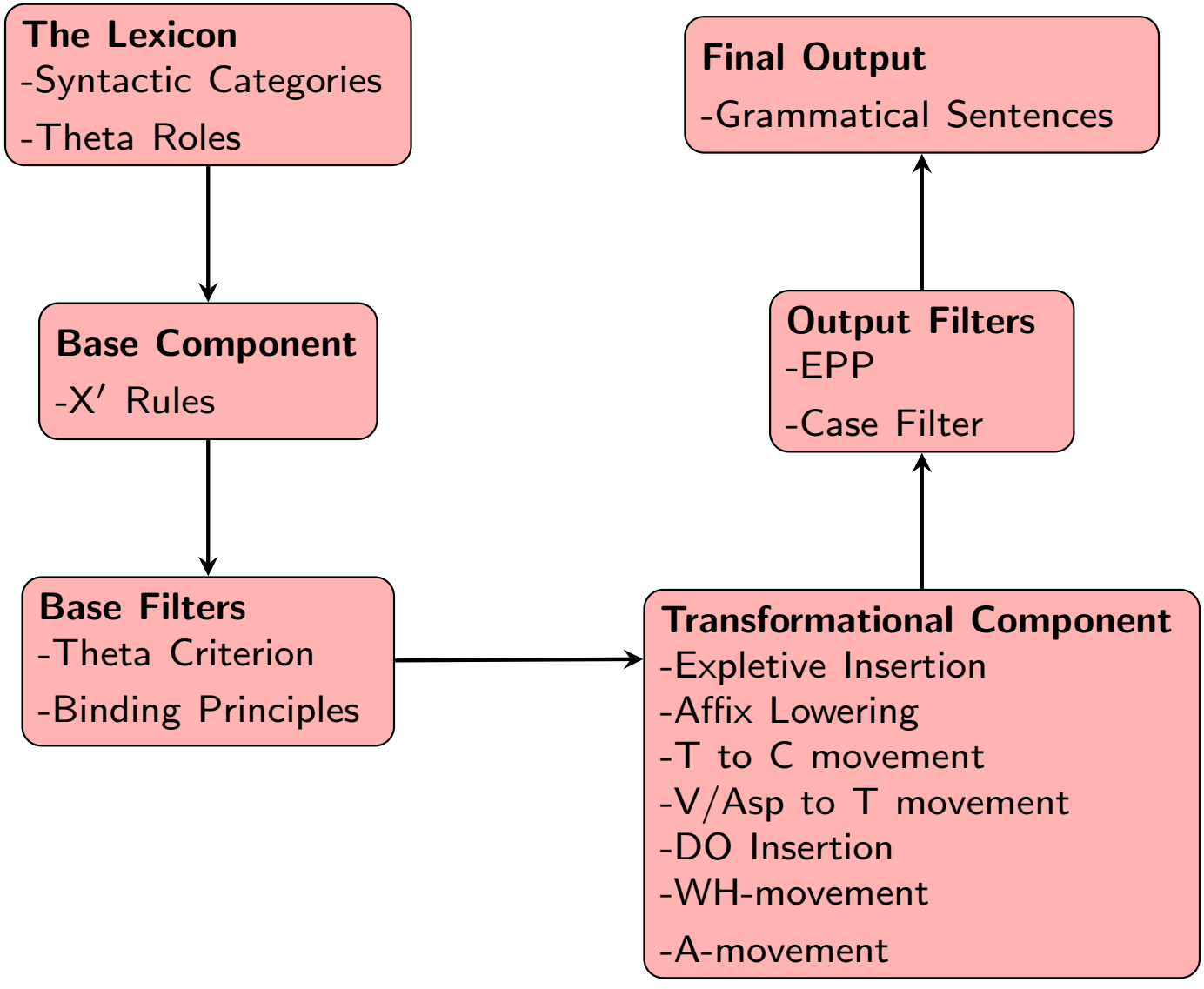
Logan Heath

Graduate Logic Seminar
University of Wisconsin-Madison
November 28, 2022

Some Observations

- Our model of the grammar changes regularly.
- Each revision of the model can be thought of as selecting a new model from among all possible models.
- New models may resolve some problems and introduce others.
- It would be nice to know what sort of problems are out there so that we can spot and avoid them as we revise our model of the grammar.

Our Current Model of the Grammar



How Accurate is this Model?

Recall that we have two notions of grammar:

(1) Unconscious knowledge about language that is represented in the mind.

(2) The linguist's hypothesis (model) about the knowledge in (1).

Is something that generates all and only the grammatical sentences a good model of the unconscious knowledge about language represented in the mind?

Some Definitions

Definition (Informal)

Call a set of strings of words from the lexicon a language if it is generated by some grammar.

- A language is said to be computable if there is an algorithm which can determine whether or not a given string of words from the lexicon is in the language.
- A language is said to be computably enumerable, or c.e., if there is an algorithm which generates a list of exactly the sentences of the language.

When a language is computable, then we can build a machine which answers questions about whether given strings of words are grammatical. When a language is c.e., we can build a machine which will write a list of exactly the grammatical sentences.

Computable vs c.e.

Are these things different?

Fact

Every computable language is c.e., but there are c.e. languages which are not computable.

- What sort of thing do we get when a language is c.e., but not computable?
- What sorts of languages do our grammars generate?

Turing Machines

In order to talk more precisely about the sorts of languages our grammars can generate we need the notion of a Turing machine:

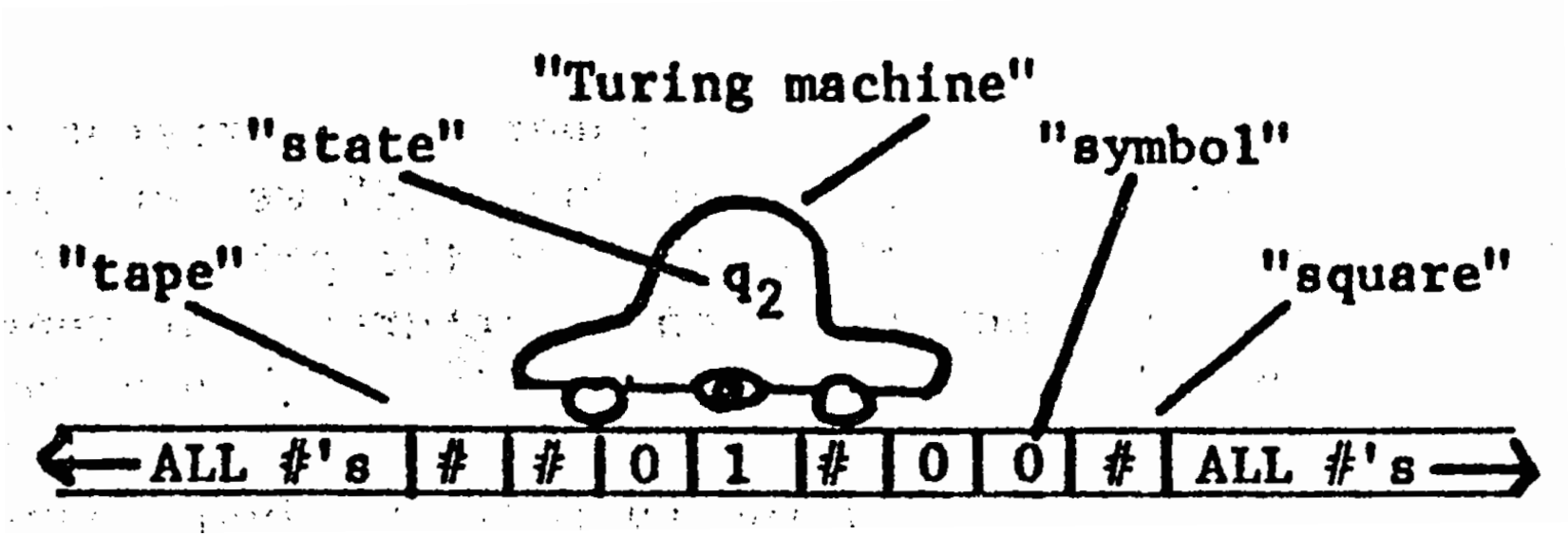


Figure: Bach and Marsh [1987]

Turing Machines

Definition

A Turing machine operates on an infinite sequence of cells, one cell at a time, consists of a finite number of states $q_1, \ldots, q_n$, and for each state contains and executes non-conflicting instructions of the form "if in state q_i and scanning symbol σ, then

- write symbol τ in the current cell and go to state q_j."
- move one cell to the right (left) and go to state q_j."
- halt."

We require that at any given point all but finitely many cells be blank and that the symbols allowed to be written in the cells be from a finite alphabet. We will usually represent a blank cell with the symbol #.

Definitions Again

It will be enough for us to recognize Turing machines as mathematical idealizations of computer programs. This being the case, we obtain the following definitions corresponding to those given earlier for languages:

Definition (Informal)

Let S be a set of strings from a given alphabet.

- S is computable if there is a Turing machine M which correctly answers "yes" or "no" when asked if a string σ is an element of S.
- S is c.e. if there is a Turing machine M which writes a list of exactly the elements of S.

As before, every computable set is c.e., but not every c.e. set is computable.

The Peters-Ritchie Theorem

Theorem (Peters and Ritchie [1973])

Every c.e. language is generated by some transformational grammar.

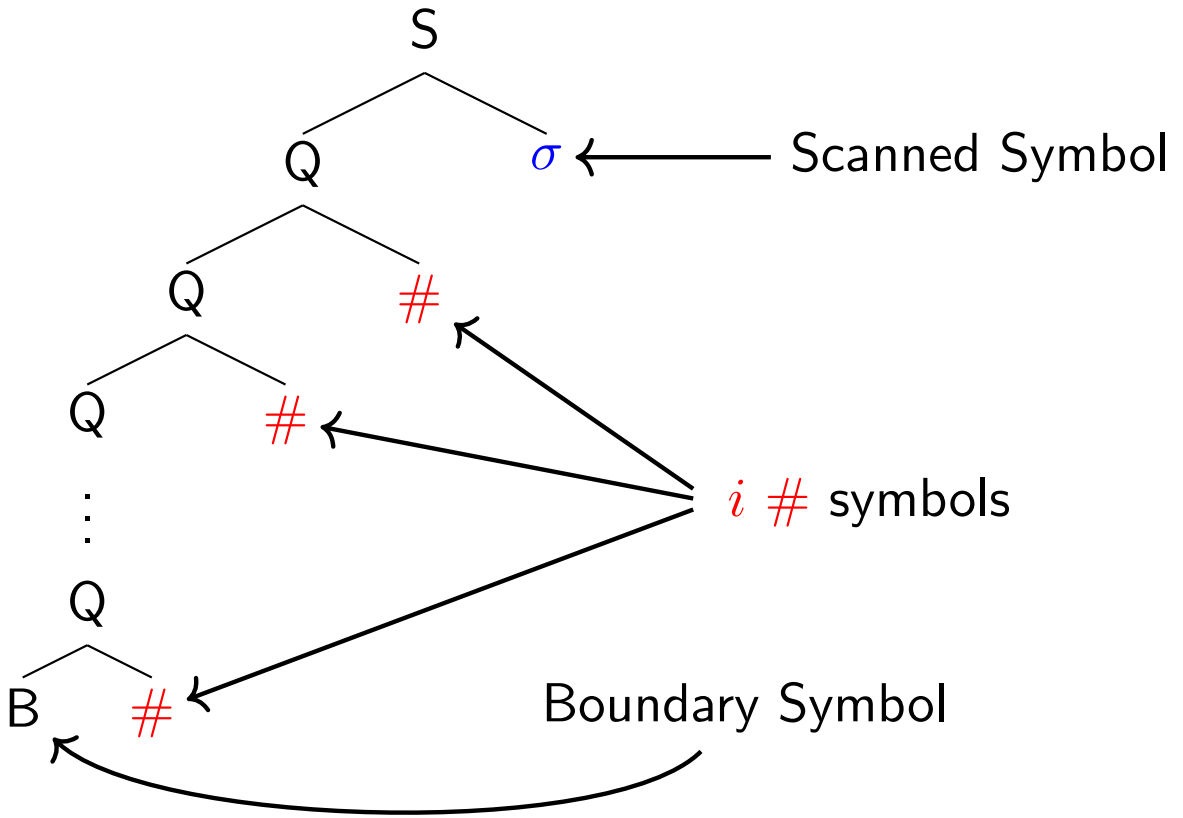
Proof (Bach and Marsh [1987]).

Given any Turing machine M, we can construct a transformational grammar which mimics the computations of M: Represent all stages of all computations of M as deep structures and define transformations so that they map the DS representing one stage of a computation of M to the DS representing the next stage of the computation. Surface structures will then be all the strings which M computes. Since every Turing machine defines a c.e. set, every c.e. set is generated by some transformational grammar. Taking lexical categories to be the finite alphabet over which our computations occur, we see that every c.e. language must be generated by some transformational grammar. □

Deep Structures for Computations

Example

The DS for a stage of a computation where our machine is in state q_i and scanning symbol σ would have something like the following as a subtree:



Some Confessions

- Having a c.e. language might not be problematic if one thinks that the human capacity for language is unlike the workings of a Turing machine.
- The Peters-Ritchie Theorem is nearly 50 years old and analyzed transformational grammars of the sort found in Chomsky's *Aspects of the Theory of Syntax* written nearly 20 years beforehand.
- Transformational grammars from at least 40 years ago were capable of avoiding the conclusions of the Peters-Ritchie Theorem (Berwick [1984]).

What's the Point?

A careful mathematical analysis of our theory can

- help us to see when something might be amiss in our theory
- pinpoint what goes wrong in the formalism of an unsatisfactory model
- suggest avenues of linguistic research to aid in the revision of the theory in an empirically motivated way.

References

Emmon Bach and William Marsh. An elementary proof of the peters-ritchie theorem. In *The Formal Complexity of Natural Language*, pages 41–55. Springer Netherlands, 1987. doi: 10.1007/978-94-009-3401-6_3.

Robert C. Berwick. Strong generative capacity, weak generative capacity, and modern linguistic theories. *Comput. Linguist.*, 10(3–4):189–202, jul 1984. ISSN 0891-2017.

P. S. Peters and R. W. Ritchie. On the generative power of transformational grammars. *Information Sciences*, 6:49–83, 1973. doi: https://doi.org/10.1016/0020-0255(73)90027-3.