THEBENCH Guide

Cem Bozşahin|Ankara, Datça, İstanbul 2022–25
cem.bozsahin@gmail.com

For Linguistic Analysis, Description and Typological Exploration of Natural Grammars with Two Command Relations

Version 2.1 December 18, 2024 coloured text means changes from the previous release
Home: github.com/bozsahin/thebench (there are instructions here for install and use) Install

Contents

1 Introduction

THEBENCH is a tool to study linguistic structures through two command relations, intralinguistically and crosslinguistically. It is for writing monadic grammars, to explore analyses, to compare languages through their categories, and to fine-tune models of grammar from the form-meaning pairs where syntax is the latent (i.e. unobserved and unannotated) variable. It is based on the idea that linguistic categories in the grammar are the only devices to decide the grammaticality of an expression *and* its consequent sense of meaningfulness. The process of analysis accounts for both aspects. One such category landscape involving two particular command relations is offered by Bozşahin (2025) (hereafter MG, for Monadic Grammar). MG

In MG, the need for two command relations, the need for categories as correspondences of them, arises from trying to account for the sense of meaningfulness which is consequent to the grammaticality of for example the following senseless expression (due to Chomsky).

(1) Colorless green ideas sleep furiously.

Long story short, MG proposes that this meaningfulness is dependent on the well-formed syntactic command relations supporting certain semantic command relations which are not obvious from the physical event itself but clear once the reference in human practice is established, for example to 'torpid' for this example, from Roman Jakobson.

If we want to also explain under what conditions the following expression might be grammatical,

(2) *She played the piano in an hour

MG proposes that it would be because of a reference shift to another category by which the speaker-signer refers to an achievement or accomplishment (e.g. a genius mastering an instrument in a very short period), two human-specific social conceptions of events rather than the physical properties of the events themselves, which, via their own semantic command relations switches to another syntactic command relation to make it grammatical in the first place.

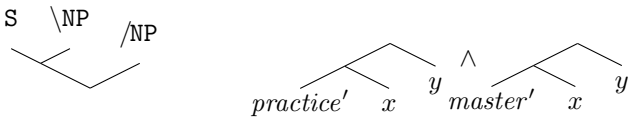
MG shows that the category landscape of command correspondences goes to considerable depth and breadth in understanding intralinguistic and crosslinguistic diversity in structural functions of case, agreement and grammatical relations.

Two command relations

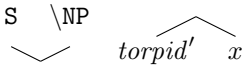
The two command relations are the syntactic command and the semantic command, respectively called the s-command and l-command in MG. A s-commands B means A compares later than B syntactically (e.g. comparing two arguments of a verb).[1] A asymmetrically brings more syntactic information than B in comparison because A can 'see what happens to' B but B cannot see what happens to A. A l-commands B means A compares later than B semantically (e.g. comparing the differences in the prominence of the arguments of a verb or their placeholding in meaning). A asymmetrically brings more information of that kind than B. They are put in a correspondence for every element of a monadic grammmar in the form of a *category*. THEBENCH helps to model that category space computationally.

For the two examples above, the categories as correspondences can be thought of as follows, denoted with ':'. We show the s-command and l-command as structured elements subsequently for (a) and (b). Upside-down trees are the s-command relations. Notice that *this* 'play' refers to mastery by experience, with subsequent placeholders for it, 'sleep' to torpid, its x referring to say a thing with some kind of uneasiness.

(3) a. **played** :: $(\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP} : \lambda x\lambda y.practice'xy \wedge master'xy$



b. **sleep** :: $\mathtt{S}\backslash\mathtt{NP} : \lambda x.torpid'x$



In (a), the \NP s-commands the /NP. In its l-command, the y's l-command the x's.

If you are more interested in the software specification and the tool's use, please skip to §4. In the next two sections we look into a bit of mathematics and linguistics behind THEBENCH.

2 The command monad

Monadic structures

The monadic structures are binary comparison of elements that employ semantics of composition only, in a hermetic seal, hence the name; see Mac Lane (1971); Moggi (1988) for that. Comparison of objects (the elements of grammar) is done with 'arrows' in category theory, which we can think of as functions. Analysis then is the structuring of comparison. This structure is the basis of interpreting an expression, or, building

[1] 'Compare' is the term used by category theory for analyzing all kinds of objects that we do not wish to take apart because they are assumed to be already put together—synthetic. They are presumed to be related. Therefore abstraction (as functions, 'arrows') is the only way to compare them. Binary composition is the only basis of their comparison. Comparison is therefore inherently structured because 'later than' is an abstract asymmetric relation; we can compare one at a time in binary composition. The objects in the case of natural languages would be the elements of grammar. They are visible to comparison only through their categories. Interpretation and production are both based on comparison. The first process compares an expression in the analysis to pick the elements of grammar than can support it. The second process compares the elements of grammar to form an expression in the analysis that is a reflection of them. Therefore analysis can mean 'untie' and 'through combination.'

one. Either can be done using the top-down or bottum-up parsing techniques. The structure projects the command relations (production) or takes an expression to the objects of grammar (interpretation). **The unfolding of the semantic command relation is consequent to the unfolding of the syntactic command relation.** They are free to vary within themselves. All variations have been attested in some language of the world according to MG.

Syntactic autonomy & semantics

THEBENCH is essentially old-school categorial grammar to syntacticize the idea of correspondences, with the implication that although syntax is autonomous (recall *colorless green ideas sleep furiously*), the treasure is in the baggage it carries at every step, viz. semantics, more narrowly, the predicate-argument structures indicating choice of categorial reference and its consequent placeholders for decision in such structures.

There are some new thoughts and gadgets that are brought to the old school in THEBENCH. Unlike traditional categorial grammars, application is turned into composition in the monadic analysis. Every correspondence requires specifying the two command relations, one on syntactic command and the other on semantic command. A monadic grammar of THEBENCH contains only synthetic elements (called 'objects' in category theory of mathematics) that are shaped by this analytic invariant, viz. composition. Both ingredients (command relations) of any analytic step must therefore be functions ('arrows' in category theory).

Having to have functions requires a bit of explaining on the linguistic side, which I do in the next section.

3 Linguistics and mathematics behind the tool

Mathematically, a monad composes two functions f and g as the only basis of object comparison to maintain the dependency of f on g, which we can write in extensional terms as $\lambda x.f(gx)$.

It does so by doing $g \circ f$. Computationally, it first gets g, then f. The ultimate element f is always the head function; see Gallier 2011:118 for reasons for the notation. In this representation, $g \circ f$, it is clearer that f depends on g in $\lambda x.f(gx)$, not the other way around. In the alternative notation, writing instead $f \circ g$ to mean $\lambda x.f(gx)$, and saying that $f \circ g$ means 'do g, then f' to get the dependency right, is somewhat counter-intutitive, especially if we are going to construct all structure from sequencing and reference categorization by a single chain of linguistic dependency. Note that $\lambda x.f(gx)$ dependency is a single chain: g depends only on x, and f depends only on (gx). The linguistic significance of these properties is that, because the monad always return the result of the ultimate element, the monad itself is oblivious to the heads and their directionality. The heads themselves must specify that information, one at a time.

The command monad of THEBENCH internally turns f and g into semantic functions even if one of them is syntactically not a function. (If none of them is a *syntactic* function, they would not be comparable, with implications—in my opinion—for morphological compounding.) It does so by deriving the case functions from the verbs and verb-like elements of a grammar, and by keeping intact the two command relations, which every element of grammar specifies: surface command (s-command) and predicate-argument command (l-command).

Command pairing

The s-command specifies in what order and directionality an element takes its syntactic arguments. It is asymmetrically structural; it is not a linear metric. It embodies the syntactic notion of combining later or earlier than other elements. The l-command specifies in what order and dominance the predicate-argument structure is revealed. It is similarly asymmetrically structural. Pairing of the two is always required in the monadic grammar of THEBENCH.

This pairing, in the form of a *procedural category*, is a concept we can think of as a culmination of all of the following in one representational package for singular comparison: Montague's (1973) basic rules associating every syntactic application with logical interpretation, Bach's (1976) rule-to-rule thesis, Klein and Sag's (1985) one-way trans-

Procedural categories

lation of syntactic and semantic types, Grimshaw's (1990) a-structure, and Williams's (1994) argument complex, without nonlocality, that is, without specification of argument's arguments—maintaining his locality of predication, and Hale and Keyser's (2002) syntactic configuration of a lexical item and its semantic components.

A procedural category has the added role of choice of reference *for the whole* reflected in its syntactic type to lay out decision points for truth conditions of the parts in the semantic type. The main implication for the grammarian is that, perforce, there is a need for language-variant linguistic vocabulary to explore case, agreement, grammatical relations and other structural functions given such invariant analytics, because only the variant vocabulary and preserved command relations seem to be able to make *any grammar* transparent and consistently interpretable with respect to the invariant (composition). Insisting on the transparency has dividends for the modelling and exploration of grammar acquisition.

Syntactic, phonological, morphological and predicate-argument structural reflex of composition-only analytics is the main practice of studying linguistic structures with a monad. THEBENCH is a tool implementing the idea. To sum up, it is an old-school categorial grammar, except that it uses composition only. That is, application is also turned into composition, and, multiple dependency projection in one step (such as CCG's **S** projection) is not analytic; it has to be specified by the head of a construction.

Turning application into composition has implications for case, consequently for agreement and grammatical relations. It works as follows. Consider the following analysis in the context of applicative categorial grammar, much like Ajdukiewicz (1935); Bar-Hillel (1953); Montague (1973):

(4)
$$\overset{\mathtt{S}}{:\ admire'\,john'\,sincerity'}$$

$$\overset{\mathtt{NP_{3s}}}{:\ sincerity'}\quad \overset{\mathtt{S\backslash NP_{3s}}}{:\ \lambda y.admire'\,john'\,y}$$

Sincerity

$$\overset{\mathtt{(S\backslash NP_{3s})/NP}}{:\ \lambda x\lambda y.admire'\,x\,y}\quad \overset{\mathtt{NP}}{:\ john'}$$

admires **John**

Here, the verb applies to its arguments one at a time, first to its syntactic object, then to its subject. Application is evident on the semantic side, written after ':'.

What takes places inside the monad is the following, where both applications are via function composition, in the template of $\lambda z.f(gz)$. And, the f function, the head, is always the ultimate element:

Inside the monad

(5)
$$\overset{\mathtt{S}}{:\ admire'\,john'\,sincerity' = \lambda z.[\lambda\psi.\psi(\lambda y.admire'\,john'\,y)][(\lambda\phi.\phi sincerity')z]}$$

$$\overset{\mathtt{NP_{3s}}}{:\ sincerity' \nearrow \lambda\phi.\phi sincerity'}\quad \overset{\mathtt{S\backslash NP_{3s}}}{:\ \lambda y.admire'\,john'\,y = \lambda z.[\lambda\psi.\psi\,john'][(\lambda\phi.(\lambda x\lambda y.admire'\,x\,y)\phi)z] \nearrow \lambda\psi.\psi(\lambda y.admire'\,john'\,y)}$$

Sincerity

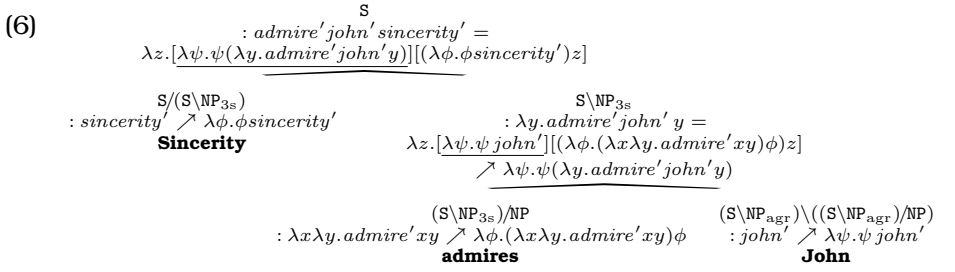
$$\overset{\mathtt{(S\backslash NP_{3s})/NP}}{:\ \lambda x\lambda y.admire'\,x\,y \nearrow \lambda\phi.(\lambda x\lambda y.admire'\,x\,y)\phi}\quad \overset{\mathtt{NP}}{:\ john' \nearrow \lambda\psi.\psi\,john'}$$

admires **John**

A monad always composes this way. The head function, the one that determines the result, is always the ultimate function in the input to combination. It is underlined at every step in the preceding example.

The implication for linguistic theory is that, the semantic lifting that takes place to engender composition, which is represented in the example using $\nearrow$, must be accompanied by a lifted syntactic category, for all elements. For the arguments of the verbs, these are the case functions, for any kind of argument including the clausal arguments of *think*-like verbs. Category theory suggests that case is a natural transformation which keeps the comparison associative, in effect supplying the theoretical basis for the following empirically motivated case functions for the same example—empirical because they are all derived from the verbal subcategorizations in a grammar.

(6)



The narrowing of reference can be observed from the case functions, for example for agreement. The subject agreement reduces the options for reference from $\mathtt{S}/(\mathtt{S}\backslash\mathtt{NP})$ to $\mathtt{S}/(\mathtt{S}\backslash\mathtt{NP}_{3s})$ for third-person singular subject, which can be read off the verbal subcategorization if distinct verb forms make the distinction in their categories. For example, the following verb forms can bear the distinct but related categories—notice their l-commands— to account for the grammaticality and ungrammaticality by case functions alone:

Case & Category

(7) a. *studies* :: $\mathtt{S}\backslash\mathtt{NP}_{3s} : \lambda x.study' x$ (finite form)
b. *to study* :: $\mathtt{VP} : \lambda x.study' x$ (stem form)
c. *study* :: $\mathtt{IV} : \lambda x.study' x$ (root form)

(8) a. Mary /studies/*study/*to study.
b. Mary wants /to study/*studies/*study.
c. Mary may /study/*to study/*studies.

Such case functions refer to different kinds of events of the same predicate, (a) to the actual or imaginary worlds in which Mary studies, Montague-style, (b) to the choice of potential worlds where Mary could be considered to study, which can be assumed to be constructed by the root forms, (c). The ungrammaticality above therefore could be the result of event-referential mismatch. For categories to do all the linguistic work on (un)grammaticality, such distinctions must be categorized. We can mark these three different semantic actions (refer, potentially refer, construct to refer) in the l-commands as well; see MG, chapter 5.

Cross-categorial generalizations are possible, also from verbal subcategorization, for example grammatical relations and systems of accusativity and ergativity. Such categorizations are not external to grammar. For example the Dyirbal control verb subcategorizes for a $\mathtt{VP}_{abs}$, meaning a VP with implicit absolutive argument. In an ergative language, it would be the single argument of the intransitive and the more patient-like argument of the transitive, which is the hallmark of ergative 'alignment' . It is captured categorially without linking theories of grammatical roles as primitives.

Notice that in (6) the verb itself is also 'lifted' semantically inside. Although, formally speaking, its lifted type is eta-equivalent to its unlifted type, the implication is that the lifted type is a function over and above who does what to whom.

For the verb itself, and for similarly complex categories, their lifting can be associated empirically with spatiotemporality. Since ϕ in the example is a transparent function arising from analytics, tense and aspect must be language-particular functions over verbal types (see Klein 1994; Klein, Li and Hendriks 2000). For the arguments of verbs, the lifting of their basic syntactic type relates to case—because they are subcategorized for. Such functions are crosslinguistically inferrable in THEBENCH from verbal subcategorizations; NB. the '`c command`'.

For case, all cases are structural (i.e. second-order) functions because they can be read off the first-order functions such as the verb. Identifying different subcategorizations of distinct verb forms (even for the same verb, as its root, stem and finite subcategorizations) leads to a bottom-up theory of 'universals', that is, categorizations available to everyone, from the ground up.

Unlike generative grammar, there are no grammar-external devices for studying grammaticality such as EPP, minimal link condition (MLC) or universal lexical categories such as N, V, A, P, or universal argument types such as those in X-bar theory. All grammaticality and ungrammaticality are decided by the local categories in a grammar.

Grammaticality, intensions, distributions

The invariant circumscribes what they can be. The Husserlian and Polish-school idea is there, that segments arise because analysis-by-categories seeks out reference fragments in an expression. We do not assume that analysis arises from (or rules 'generate' from) Harris-style distributional segments.

THEBENCH is meant to explore configurationality, narrowly understood here as studying the limits on surface distributional structure, *pace* Hale (1983),[2] and referentiality, together. THEBENCH's configurational approach to surface structure is inspired by Lambek (1958); Steedman (2020). Its referential approach to constructing elements is inspired by Sapir (1924/1949); Swadesh (1938); Montague (1970, 1973); Schmerling (2018). The following aspects can help situate the proposal in the landscape of theories of grammar.

Formally, as stated earlier, unlike categorial grammars such as that of Steedman (2000), all analytic structures handle one dependency at a time (i.e. there is no **S**-style analytic structure taking care of multiple dependencies at once). Unlike Montague grammar, predicate-argument structures are not post-revealed (*de re, de dicto*, scope inversion, etc.), because different categorization due to different reference is expected to play the key role in the explanation.[3] Moreover, the syntactic types of the second-order (structural) functions including those for case, agreement and grammatical relations require a linguistic theory precisely because they are all based on grounded (first order) elements.

Distinct function-argument and argument-function sequencing of reference is the cause of categorial and structural asymmetries in the monadic grammar of THEBENCH, which is an idea that goes back to Schönfinkel (1920/1924). He had used the idea to motivate his combinators, all but three (binary **B**, **A** and **T**) considered to be too powerful in THEBENCH for monadic structures. In their simplest forms they are:

(9) a. $\mathtt{X/Y}\ \ \mathtt{Y/Z} \Rightarrow \mathtt{X/Z}$ (forward **B**)
b. $\mathtt{Y\backslash Z}\ \ \mathtt{X\backslash Y} \Rightarrow \mathtt{X\backslash Z}$ (backward **B**)
c. $\mathtt{X/Y}\ \ \mathtt{Y} \Rightarrow \mathtt{X}$ (**A**)
d. $\mathtt{Y}\ \ \mathtt{X\backslash Y} \Rightarrow \mathtt{X}$ (**T**)

Steedman and Baldridge (2011) provide more information about them. When successful in comparison, they carry semantics with them, driven by syntactic category, because they are in fact doing the following:

(10) a. $\mathtt{X/Y}\colon f\quad \mathtt{Y/Z}\colon g \Rightarrow \mathtt{X/Z}\colon \lambda x.f(gx)$ (forward **B**)
b. $\mathtt{Y\backslash Z}\colon g\quad \mathtt{X\backslash Y}\colon f \Rightarrow \mathtt{X\backslash Z}\colon \lambda x.f(gx)$ (backward **B**)
c. $\mathtt{X/Y}\colon f\quad \mathtt{Y}\colon a \Rightarrow \mathtt{X}\colon fa$ (**A**)
d. $\mathtt{Y}\colon a\quad \mathtt{X\backslash Y}\colon f \Rightarrow \mathtt{X}\colon fa$ (**T**)

What's left behind by disallowing multiple dependency passing, composition, including internal composition with **A** and **T**, is anybody's composition, but with some exploratory power arising from function-argument and argument-function distinction because it allows transparent transmission of reference. We don't really know whether category choice by the speaker-hearer-signer is compositionally determined. We would be studying how far it can be transmitted compositionally in syntax starting with the whole, but not holistically. (This sounds like a distant memory in psychology too; see Koffka 1936).

One dependency at a time

Empirically, morphology and syntax do not compete for the same task in monadic analysis, for example for surface bracketing (morphology constructs and syntax transmits reference). Morphology is not confined to the 'lexicon', to 'operators' or to some

Morphology does not compete

[2] From this perspective, there is no such thing as a non-configurational language in the sense of Hale (1983).

[3] Cf. *every professor wrote a book* and *every student missed a meal*. Not all predicates allow for inversion, therefore it is not a far-fetched idea to replace post-reveal of scope with differences in the referentiality of the predicate and consequent change in type-raising of the arguments. In other words, neither direct compositionality of Montague (1970) nor quantifying-in of Montague (1973) are suggested as alternatives, relying instead on the verb's referential properties and the category of the quantifiers and non-quantifiers. THEBENCH is more compatible with the 1970 idea, but arguing instead to support different subcategorization due to different referentiality.

```
------------------------------------------------------------------------
Welcome to The Bench
    A workbench for studying NL structures built by two command relations
        Bench version:       2.0 Dated: March 25, 2024
        Python version:      3.10.10
        Common Lisp version: SBCL 2.2.9

        Pre/post processing by Python (grammar development, interfaces)
        Processing by Common Lisp     (analysis, training, ranking)

    Today: June 02, 2024, 12:41:36
Type x to exit, ? to get some help
------------------------------------------------------------------------
lisp     : bench.lisp loaded, version 8.0, encoding UTF-8
python   : bench.py   loaded, version 2.0, encoding utf-8
ready
```

Figure 1: Welcome screen of THEBENCH (without UTF codes for terminal prompts).

other component, or to leaves of a tree. It is an autonomous structure, obviously not isomorphic to phonological or syntactic structure but also not subserving them either, projected altogether homomorphically in analytic structure. Morphology is considered to be the category constructor for the form, therefore all languages have morphology. And, referential differences in all kinds of arguments cause categorial differences including those in the idioms and phrasal verbs (Bozşahin, 2023).

The verb

Typologically, any language-particular difference in elementary (i.e. synthetic) vocabulary can make its way into invariant analytic structure presuming it is the empirical motive to make argument-taking transparent. It is a very verb-centric view of grammar. Elements' morphology plays a crucial role in transparency of analytics. This is one reason we avoid commitment to certain ways of doing morphology in THEBENCH.

4 THEBENCH's organization

The App

The only user executable for THEBENCH is called `thebench`. To make it accessible from any directory, it is placed by the installer in `/usr/local/bin`. When launched, it will display the welcome screen in Figure 1.

All the files of THEBENCH are text files. There are two kinds: editable files and uneditable files. The editable files are grammar files, those created by the user using some plain text editor, and those generated by the system for the benefit of the user. The latter kind can be cut and pasted to plain-text grammars. A list of commonly-used operations can be saved as macro files to avoid redundant typing. These too are plain text editable files. The location of all editable files is your working directory. The uneditable files are not saved in your working directory to avoid clutter.

File types

Technically, the editing functions of THEBENCH, the functions that transform internal files to editable files, and the interfaces of THEBENCH, are all written in `Python`. The processor functions are written in COMMON LISP. The installer (NB. the top of the first page) takes care of the software requirements to work with all popular personal computer platforms (well, almost all). The command interface, which is given at the back, has its own syntax combining the two aspects for editing and processing, which you can use online and offline (the '`@ command`', aka. 'batch mode'—see the basic glossary at the back).

There is no need to know either COMMON LISP or Python to use THEBENCH. Maybe this much is good to know to understand the processor's output: `T` means true, and `NIL` means false in all Lisps. You will encounter them quite often in using the interface. Use the '`? command`' when you launch THEBENCH to recall the full list of commands. Figure 1 shows what you see when you launch THEBENCH. (Software versions may vary.)

The processor's internal files go into the directory created by THEBENCH installer for you; they reside in `/var/tmp/thebench`. Occasional clean-up of your working directory and this directory is recommended.[4] Transformation of an internal file to an editable grammar is possible, which takes a file from `/var/tmp/thebench` to save the editable file in your working directory.

File locations

The training command '`t`' loads an extra processor file called `bench.user.lisp` which is available in the repository. You can change that file, but please do not rename, relocate or delete it. If it is not loadable, the '`t command`' would complain and exit. No internal code depends on the code contained in this file. It is useful for experimenting on model parameters without changing THEBENCH code if you are going to do modelling.

Direct IPA support is too unwieldy for the tool; it seemed best to use UTF-8 tools for it in Python and Lisp. When you launch THEBENCH, please check the encoding reported by the processor. Both `Python` and COMMON LISP in your computer must report UTF-8, as in Figure 1.

5 Synthetics: The elements of grammar

There are three kinds of elements of grammar in THEBENCH:

Entry types

(11) a. an elementary form bearing a functional category,
b. an asymmetric relational category relating the two categories of one form, and
c. a symmetric relational category relating the categories of two forms.

Examples in THEBENCH notation are respectively:

THEBENCH notation

(12)
```
likes  | v :: (s\^np[agr=3s])/^np : \x\y.like x y     % ^: object can topicalize
#np-raise np[agr=?x] : lf  --> s/(s\np[agr=?x]) : \lf\p. p lf
#tense runs, s[t=pres,agr=3s]\np:\x.pres run x <--> ran, s[t=past]\np:\x.past run x
```

Every entry must start in a new line and it must be on one line, without line breaks. This allows us to minimize non-substantive punctuation. Let the longer lines wrap around on the screen. Everything starting with '`%`' until the end of a line is considered to be a comment. Capitalization is significant only in phonological elements (the material up to the '`|`' for entries of type (a), the stuff separated from the category by the comma in type (c).).

Whitespacing is not important anywhere. Empty and comment-only lines are fine. The order of specification of elementary items and symmetric relational categories in the grammar's text file is not important. Asymmetric relational categories apply in the order they are specified. THEBENCH home repository contains sample grammars as a cheat sheet. (My personal convention is to put the commands that process or generate THEBENCH commands in '`.tbc`' files, for 'the bench command'. The '`@ command`' can take them as macros to run.)

Whitespace

Order of elements

The first kind of element in (12) starts with space-separated sequence of UTF-8 text, ending with '`|`'. Multi-word entries, non-words, parts of words, punctuation and diacritics are possible. (This is provided so that we can avoid single or double-quoted strings at any cost; there is no universal programming practice about strings, and the concept is overloaded.)

The form & MWEs

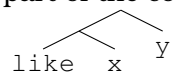
The material before '`|`' is considered to be the textual proxy of the phonological material of the element. The next piece is the part of speech for the element, which can be any token (see basic glossary at the back). The material after '`::`' is the category of the element. (This token is the most common convention in monads; it is called the monadic type constructor.) The part before the single '`:`' is called the syntactic type, which is the domain of s-command. The part after '`:`' is called the predicate-argument structure, which is the domain of l-command.

tokens

[4] See the end of §8 for when not to do a clean-up.

Left-associativity is assumed for the bodies of both s-command and l-command. Right-associativity is assumed for lambda bindings. Together they constitute a syntax-semantics correspondence for the element. If you don't like such associativity and want to write the structure yourself, please be careful with the proliferating numbers of parentheses. The system will warn you about mismatches, but the process of getting them right can be painful. In training and exploration with complex elements it's easy to make judgment errors in parenthesization. With associativity, we can see that for example the l-command of first entry in (12) is equivalent to `(like x)y`, in asymmetric tree notation as follows. Its lambda binders are not part of the command relation.

l-command



The correspondence is explicit in the order of syntactic slashes and lamba bindings. For the first example in (12), '`\x`' in the lambda term corresponds to '`/^np`', and '`\y`' to '`\^np[agr=3s]`'. The '`\`' in a predicate-argument structure is for 'lambda'. Several lambdas can be grouped to write a single '`.`' before the start of a lambda body, as we did in the example. We could also write them individually, as '`\x.\y.like x y`'.

correspondence

The syntactic type can be a (i) basic or (ii) complex category. (The term 'category' is quite commonly used also for the 'syntactic type' when no confusion arises.) A basic category is one without a slash, for example '`np[agr=3s]`'. Here, '`np`' is the basic category and '`[agr=3s]`' is its feature. Features are optional; they can be associated with the basic categories only (unlike many other phrase-structure theories). Multiple features are comma-separated. An unknown value for a feature is prefixed with '`?`'. For example '`np[person=1,number=sing,agr=?a]`' means the agreement feature is underspecified.

Features

The complex categories must have a slash, e.g. '`s\^np[agr=3s]`' in the example. It is a syntactic function onto '`s`' from '`\^np[agr=3s]`'. The result is always written first, so for example '`s/np[agr=3s]`' would be a syntactic function onto '`s`' from '`/^np[agr=3s]`' . Surface directionality would be different. In the first case, the function would look to the left, in the latter, to the right, in surface order.

s-command

Modal control on the directionality is optionally written right after the syntactic slash, such as '`^`' in the first entry. If omitted, it is assumed to be the most liberal, that is '`.`'. Slash modalities are from Baldridge (2002); Steedman and Baldridge (2011). THEBENCH notation for the modalities are '`.`' for '`.`', '`^`' for '$\diamond$', '`*`' for '$\star$', and '`+`' for '$\times$'. They are for surface-syntactic control of composition in the monad.

Slash modality

Two more tools of THEBENCH are worth mentioning; they are not in every categorial grammar's toolbox. There are double slashes '`\\`' and '`//`' which are similarly backward and forward. They take and yield potential elements of grammar. (Therefore they are unlike Montague's multiple slashes, and they are not necessarily morphological.) They are called *extending categories* in MG, because of potentially extending the object set of a grammar. There is no modal control on them.

Double slashes

Basic categories can be singletons, that is, they can stand for their own value only. For example, `(S\NP)//'the bucket'` has the singleton '`the bucket`' which stands *as category* for the phonological sequence `the bucket`; cf. `NP`, which stands for many expressions that are distributionally NPs. Singletons can be single- or double-quoted as long as they are atomic (i.e. two successive single quotes do not make a double quote). Singletons are applicative because they cannot be the range of a complex category, they are domains only ; see Bozşahin (2023).

Singleton categories

In summary, elementary items bear categories that are functions of their phonological form.

The second kind of element in (12) is an asymmetric relational category. It means that the element bearing the category on the left of '`-->`' at surface form *also* has the category on the right. The token immediately after '`#`' is the name of the relation, in this case '`np-raise`'. For such relations to make sense, the first lambda binding on the right must be on the whole predicate-argument structure on the left. This is not checked by the system, we can write any category in principle; but, this is why such relations are not associated with a particular substantive element. Notice that there is

no phonological element associated with the relation.

The third kind of element in (12) is a symmetric relational category. It means that during analysis either form is eligible for consideration along with its particular category. The analyst apparently considered them to be grammatically related so that they are not independently listed. (This is one way to capture morphological paradigms but not the only way.) The relation's name is right after '#'. The material on the left and right of '<-->' must also contain some phonological material (whitespace-separated words ending with a comma) so that we can reflect the substantive adjustment on the categories.

Empty categories are not comparable

There are no elements, functions or relations in the monadic analysis which can alter or delete material in surface structure because that would not be always semantically composable. (It follows that IA/IP/WP morphology of Hockett must be sprinkled across the three kinds of specification above. Judging from the fact that no morphologically involved language is exclusively IA, IP or WP, this should not be surprising.)

No element type can depend on or produce phonologically empty elements, because such elements cannot construct categories. The monad's hermetic seal, that every analytic step is atomic, is also consistent with these properties. Together they allow us to adhere to the Schönfinkelian idea of building hirerarchical structures from sequential asymmetries alone.

6 Analytics: Comparing the elements of grammar

The tripartite organization of every synthetic element is reflected in THEBENCH notation:

(13) `phonology :: s-command : l-command`

It is transmitted unchanged in analytics. Self-distribution of morphology as it sees fit in a grammar is based on the common understanding that syntax, phonology, morphology and semantics are obviously not isomorphic, but, equally obviously, related, in conveying reference.

The analytic structure is invariant. This means that these domains are homomorphic to analytic structure, because there is only one such structure. It is function composition on the semantic side. How the forms (syntax, phonology and morphology) cope with that structure and limit it is the business of a linguistic theory.

Information transmission in analytics therefore needs a closer look. In building structures out of elements one at a time, the method of matching basic and complex categories of the s-command is the following.

If we have the two-category sequence (a) below, we get (b) by composition, not (c); cf. `f2` passing.

Term unification

(14) a. `s[f1=?x1,f2=v2]/s[f1=?x1]  s[f1=v1,f2=?x2]/np[f2=?x2]`
b. `s[f1=v1,f2=v2]/np[f2=?x2]`
c. `s[f1=v1,f2=v2]/np[f2=v2]`

This avoids unwanted generation of pseudo-global feature variables, and keeps every step of the comparison local. Using this property we could in principle compile out all finite feature values for any basic category in a grammar and get rid of its features (but nobody did that, not even GPSG. Maybe encoder-decoder transformer nets can make a difference here).

Meta category

The meta-categories such as `(X\X)/X` are written in THEBENCH as `(@X\@X)/@X`. They are allowed with application only. Linguistically, only coordination and coordination-like structures seem to need them, which are syntactic islands with Ross (1967) escape hatches on syntactic identity, that is, things having the same case, therefore same category:

(15) `and | x ::  (@X\*@X)/*@X : \p\q\x. and (p x)(q x)`

This way we can avoid structural unification to do linguistic work; all of it has to be done by the invariant analytic structure. (Recall also that structural-reentrant unification is in fact semi-decidable. We simply use term unification.)

For the l-command, one important property is that its realization in analysis arises from the evaluation of the surface correspondents in the s-command and l-command, that is, surface structure and predicate-argument structure, therefore we cannot have a complex s-command (i.e. one with a slash) and simplex l-command (i.e. no lambda abstraction to match the complex s-command). For example, in the following, the '`\np`' has no l-command counterpart (some lambda), therefore unable to keep the correspondence going:

Category well-formedness

(16) *`slept :: s\np : sleep someone`

However, the inverse, that is, having a complex l-command and simplex s-command, is fine, and common, for example `man :: n : \x.man x`. This is because the analytics is driven by the syntactic category, that is, by the s-command. Not all lambdas have to be syntactic, but all syntactic argument-taking must have a lambda to keep the correspondence going in an analysis. (16) is not an analytically interpretable synthetic element. If the intention here were to capture the topic-dropped 'sleep,' for example *slept all day, what else could I do?*, it might be say `slept :: s : sleep topic`.

We have covered all three types of elements that can enter a monadic grammar so that analytics can compare the objects of grammar compositionally. Essentially, having to compose in a monad forces the conditions on the elements of the well-formedness conditions we have discussed. Object comparison through functions and relations only is the hallmark of category theory in mathematics, the monadic grammar being its subspecies. Comparison can serve both interpretation and production, also bottom-up and top-down parsing.

7 Exploration: Case, agreement, grammatical relations, grammar modelling

1. There are some aids in THEBENCH to explore monadic analytic structures. We can explore what kind of syntactic case and similar structural functions arise given a grammar, from its verbs and verb-like elements. These functions can be conceived as asymmetric relational categories, showing the understanding that if we have one category for an argument in an expression we also have other categories for the argument, as a sign of mastery of argument-taking in various surface expressions.

The '`c command`' of THEBENCH generates these functions from a grammar (loaded by the '`g command`'), taking a list of parts of speech as input from which to generate the second-order functions. (Presumably, these are the parts of speech of the verbs and verb-like elements.) These extra functions are saved in a separate textual grammar file with extension '`.sc.arules`'. The name designates that all of them are asymmetric relational categories.

Case

These categories can be merged manually with the grammar from which they are derived. The reason for not automating the merge is because you may want to play around with them before incorporating them into an analysis. It is a textual file, therefore editable just like any grammar text file. Many surface structures follow from incorporating them into a grammar, which you can check with the '`a command`' (for 'analyze').

After the merge, analyses will reveal many more surface configurations as a consequence of synthetic case in the language under investigation. This way of looking at case renders generalizations about case, for example lexical, inherent and structural case (Woolford, 2006), as bottom-up typological universals arising from a range of class restrictions on the verb, from most specific to least. MG has more to say about these aspects.

2. We can preview the syntactic skeleton of a grammar in THEBENCH. '`k command`' goes over all the elements of the currently loaded grammar, and reports how many distinct syntactic categories we have in it.

For brevity it does not report features of a basic category. This much is certain though: if two categories reported by this command look the same but reported separately, it would mean that they are distinct in their features, to the extent that they would not match in analysis either. For example, `s\np[agr=3s]` and `s\np[agr=1s]` are distinct, although their featureless version looks common: `s\np`.

One aspect of the '`k command`' can be used to find out the patterns in the grammar's categories, and to find out why some categories are distinct eventhough they might look identical as reported by the '`k command`': for every distinct syntactic category reported, it enumerates the list of elements that bears it. These forms are presumably indicators of common syntactic, phonological and/or morphological properties, for example agreement classes and bound versus free elements.

THEBENCH is a tool for a very bottom-up view of grammar. Naturally, more than N, V, A, P distributions will be reported. (In fact, THEBENCH has no built-in universal categories or features. If you don't use these four categories at all, it is fine.) To see the basic category inventory, in addition to the inverted list display of the '`k command`', take a look at the output of '`! command`', which reports the basic categories in the grammar and the list of all features in them, and attested values.

No built-in category

3. There is no morphophonological analysis built in to THEBENCH. The morphological boundedness of a synthetic form can be designated with '+' in `a` and `r commands` for convenience. It goes as far allowing in analysis/ranking entering the surface tokens such as MWEs either as '`dismiss ed`' or '`dismiss+ed`', if you have '`ed`' as an entry in the grammar. ('+' is a special operator for this, i.e. the entry is not assumed to be '`+ed`'.) Keep also in mind that THEBENCH has no notion of 'morpheme'—you must model that if you want—but it has a notion of morphological structure: the category building on phonology to construct reference, Montague-style.

Multiword expressions (MWES)

The current '+' notation is just for some convenience until somebody comes up with a categorial theory of morphophonology for example along the lines of Schmerling (1981); Hoeksema and Janda (1988). Till then, we won't get '`insured`' from '`insure+ed`', or Turkish harmony distinction'`avlar`' (game-PLU) and '`evler`' (house-PLU) from e.g. '`av+ler`' and '`ev+ler`'.

The idea behind this self-distributed morphology is that languages of the world can distribute themselves across a grammar as they see fit by employing a mixture of functional and relational categories, rather than some pre-determined cascade of morphological and syntactic operations (e.g. lexical insertion). Morphology is not conceived as a convenient tokenizer; it is the category constructor.

Self-distributing morphology

4. We can turn the textual form of THEBENCH grammar into a set of data points each associated with a parameter. These parameters are parameters in the modelling sense, characterizing a data point. To do this, THEBENCH turns a textual grammar into a 'source form', which is a COMMON LISP data structure, basically Lisp source code. Such files are automatically generated and carry the file extension of '`.src`'.

Entry keys & parameters

This is an inspectable file, but it is much easier to inspect when it is turned into a textual grammar in THEBENCH notation, which the '`z command`' does.[5] The resulting textual file will be much like the original textual grammar, without comments, and with keys and initial value of the parameter added on the right edge of every entry, in the form of for example '`<314, 1.0>`' where 314 is the key and 1.0 is the parameter value. The keys are unique to each entry, therefore having multiple categories for the same form is possible; they would be distinct entries even if they are identical in text.

[5]The '`z command`' can also transform legacy '`.ded`', '`.ind`' and '`.ccg.lisp`' grammars of CCGlab to THEBENCH grammar format. Please change the top line of such grammars to '`(defparameter *current-grammar*`' and put the result in `/var/tmp/thebench` directory before running the command. The result will be put back in your working directory as a text file in the monadic format.

These values are not probabilities; they are weights. The ranker (the '`r command`') turns them into probabilities to arrive at most likely analyses, given the weights. The trainer adjusts these weights depending on the supervision data. Therefore it is important to know that entries *with the same phonological form* compete with their weights in making themselves enter an analysis. (In other words, there is no built-in determinism, mapping each phonological form to only one interpretation.)

The system makes the initial assignment automatically to ensure the uniqueness of the keys and completeness of parameter assignment.[6] For example, when you inspect the regenerated textual grammar containing (12), you may see something like:

(17)
```
likes  | v :: (s\^np[agr=3s])/^np : \x\y.like x y <120, 1.0>
#np-raise np[agr=?x] : lf  --> s/(s\np[agr=?x]) : \lf\p. p lf  <34, 1.0>
runs | tense :: s[t=pres,agr=3s]\np:\x.pres run x  <2, 1.0>
ran | tense :: s[t=past]\np:\x.past run x <76, 1.0>
```

This file is just as editable and processable as the original textual file containing the grammar. However, notice the difference from (12). The symmetric relational categories are compiled into separate entries in (17), and their link is preserved by using the name of the relation as 'part of speech' in both entries, in this case `tense`.

The point of creating different data points is that, when trained, these elements will participate in different analyses therefore need parameter values of their own. Using the preserved link (same tag) we can explore reconstruction of the paradigm intended by '`<-->`'. The possibility of using the same relation name to capture a paradigm facilitates this exploration.

There is one more change in the reconstructed text file: every entry's capitalization is normalized to lower case, except the phonological material, whose "orthographic case" is preserved. This is one lame attempt to indicate that COMMON LISP's default capitalization of values without us asking for it would not make any difference to an analysis.[7]

5. Taking the textual source of a developing grammatical analysis and turning to modelling with it is the idea behind the textual reconversion of a grammar source. The reconstructed source has the symmetric relations turned into separate entries with same 'part of speech': the relation's name. They will have their own parameter values. This is probably the last step after grammar development, when it is time to move on to grammar training to adjust belief in elements, using parameters. This is why the command that does this is called the '`z command`'.

Fine-tuning & modeling

Training a grammar with data updates the parameters, which can then be used for ranking the analysis, that is, for choosing from the possible analyses the most likely one after training. The method used is the sequence learning of Zettlemoyer and Collins (2005). It is basically a gradient ascent method. It places the whole trained grammar in probabilistic sample space of possible grammars.[8] Bozşahin (2021) (§4.3) explains the meaning of parameters increasing, decreasing or remaining unchanged.

The supervision pairs are correspondences of phonological forms and their presumed correct predicate-argument structures, sometimes called in computational work the 'gold annotation'. However, the term would be misleading, because there is no annotation or labeled data.

[6] If you use a specific key for a particular entry in your grammar text, say for easier tracking of a particular item during model training, the system respects your key choices and parameter initialization. I recommend not using this feature extensively to avoid key clashes. Uniqueness of personally assigned keys is not checked. If you do use the special key assignment feature, make sure that there is no keyed element after any unkeyed element, that is, pile unkeyed elements at the end of the file. This way your assigned keys will be seen before new key assignment begins.

[7] I could make it case-sensitive, to preserve both the analyst's and Lisp's cases, but this would be quite error prone: Is `AdvP` same as `advp`? I would think so.

[8] Linguistics alert: These are not probabilistic grammars, things in which a category would be uncertain. They are grammars with clear-cut categories which are situated by model selection in the probabilistic space of possible grammars. The idea is not too far from the continuity hypothesis in language acquisition by which a child moves from one possible grammar to another, apparently nondeterministically; see Crain and Thornton (1998).

The term 'supervision' needs clarification in the context of THEBENCH. It means that we *presume* that that data is known to hold some correspondence of expressions and predicate-argument structures, much like Brown (1973), Brown (1998) and Tomasello (1992) did when they started analyzing child data—it is indeed an adult assumption coming from psycholinguistic analysis. The predicate-argument structure is a specification of the presumed meaning of the phonological item, written in a text file one line per entry, for example:

(18)
```
Mary persuaded Harry to study : persuade (study harry) harry mary
Mary promised Harry to study : promise (study mary) harry mary
Mary expected Harry to study : expect (study harry) mary
```

Here, the material before the colon is the proxy phonological form (therefore "orthographic case"-sensitive). Multi-word expressions (MWEs) must be enclosed within "`|`' if they are considered to be referentially atomic, e.g. `|kicked the bucket|`. (After all, this is supervision in the sense above, so we assume its reference is fixed, we just need to find out which part of the predicate-argument structure corresponds to it, given a grammar with that MWE.)

MWEs

The material after the colon is the presumed predicate-argument structure of the whole expression. It has the same format as in grammar specification. The '`t command`' (for 'train') updates the parameters of a grammar-turned-into-a-model by this method. We can pick from the candidate models by looking at its output (known as model selection), which is a collection of plain text grammars (the number of candidates is specified by you), with parameters for every entry. We can then load the chosen grammar, and rank analyses with it using the '`r command`'.

The input to training is as many as number of entries in a file with lines like those in (18). We suggest minimal (required) parenthesization to avoid near misses in training; THEBENCH will internally binarize the predicate-argument structure anyway. Exact repetition of an entire line constitutes a separate supervision entry. This is useful—in fact required—in analyzing child-directed speech, where repetition is very common.[9]

6. The tools used for training a grammar on data such as (18) are triggered from THEBENCH command line, but they work offline, using the same processor code. These training sessions can last quite long, sometimes hours, sometimes days or weeks depending on the grammar size, the number of experiments attempted and the amount of supervision data. (That's because of the computer speed.) We don't have to stay in the originating session to keep the experimental runs going. They operate independently, using the `nohup` facility.[10]

Experiment files

During these runs, all information must be available offline. To do this, we prepare an 'experiment file'. It has a strict format, for example:

(19)
```
7000 4000 xp 1.2 1.0 nfp nfparse-off
4000 2000 10 0.5 1.0 bon beam-on
4000 2000 10 0.5 1.0 boff
```

This specification will fetch three processors if it can because three lines of experiments are requested, one using 7GB of RAM in which 4GB is heap (dynamic space needed for internal structures such as hash tables, etc.), the others with 4GB RAM and 2GB of heap. As a guideline, a larger heap helps to run analyses of longer expressions, which usually arises when case functions are added to a grammar.

These parameters are useful to run the experiments on machines with different powers and architecture (number of cores, amount of memory, size of heap) without changing the basic setup.

[9] Assigning different semantics to every repetition of an identical expression is the holy grail of child research. We have no theories for that as far as I know.

[10] `nohup` is for 'no hang up', which is one more reason why a `linux` system or subsystem is indispensable for tools such as this. One caution: Do *not* exit your session with control-D, which would kill the subprocesses in the linuxsphere. Just let it die or hang, or close the window normally, e.g. by clicking the red button in the red-yellow-green window menu.

The ‘t command’ specifies which grammar and supervision will take place in the experiments. (Therefore every experiment in one file runs on the same grammar-training data.) The supervision file must be in the format of (18). The rest are training parameters: xp means use of the extrapolator, which uses a predetermined number of iterations for the gradient ascent in implementing the Cabay and Jackson (1976) method.

The second experiment does not extrapolate; it iterates 10 times. Every iteration updates the gradient incrementally. Then comes the learning rate (1.2 in the first experiment), which is the distance the gradient travels in one step (‘the jump distance’), the learning rate rate (1.0 in the first experiment), which is how later iterations affect the jump distance, the prefix of the log file e.g. nfp (the actual name of the log file is prefixed with that, adding training parameters as suffixes of the name for easier identification of experiments that will be saved in the end), and the function to call before training starts, which is optional, as in the third experiment. In the first experiment it is nfparse-off, which turns normal form parsing off, which is a feature in the processor, implementing the algorithm of Eisner (1996) for reducing compositions after they come out of the monad. This affects the number of analyses generated and reported.

The full list of such functions is given at the back, which are callable from the command interface as well the ‘l command’. (For processor functions with more than one arguments, calling is a bit more complex; see the cl4py Python library documentation.) In the second experiment, it is beam-on. This function focuses the gradient on items with largest weight changes during iterations, avoiding the consideration of elements that change very little (no change is effectively handled by default). The results may be less precise because of this but it reduces the search space for the gradient. It is sometimes a lifesaver in eliminating excessive runtimes or out-of-memory errors.

7. Studying referentiality of all kinds along with configurationality in one package seems to be one way to avoid the purportedly dichotomic world of verbs-first and nouns-first approaches in language acquisition. THEBENCH is designed primarily to rise above these assumptions with the help of modelling.

By combining explorations **4–5–6** it is possible to have another look at event-and-participant attempts to explain language acquisition, for example Brown (1973); Tomasello (1992); Brown (1998); MacWhinney (2000); Abend et al. (2017), to compare with participant-centric approaches such as Gentner (1982).

Syntax as latent variable

In such an undertaking, the observables would be the phonological form on the left, much like in (18), and the analyst’s conclusion about what they would mean would be the predicate-argument structure on the right. There would be no labeled or hidden variables such as syntactic labels or dependency types. The mental grammar which is presumed to arise in the mind of the child would be proxied as a grammar much like in the earlier sections, which would be trained on the data such as above for model selection.

8 Recommended work cycles

Three work cycles are reported here from personal experience, for whatever their worth.

Writing, editing, exploring grammars

The first one is for the development of an analysis by checking its aspects with respect to the theoretical assumptions. This would be the simple cycle in (20a), with ‘k’ and ‘!’ commands sprinkled in between (not shown here).

(20) a. `edit - g command - a command - , command`
b. `edit - g command - c command - edit/merge - a command - , command`
c. `edit - t command - z command - g command - r command - # command`

It would load a grammar (g), analyze an expression with it (a), and display the results (,).

The second cycle (b) would be for studying a grammatical analysis in its full implications for surface structure when the class of verbs is large enough. ‘Merge’ here presumably merges case functions with the grammar. When we generate case functions

using '`c command`', the currently loaded grammar have access to them so we can do analyses with them in the current session. However, they are not added to the grammar's text file, so the grammar is left intact when the session is over. The merge is left up to you. The case functions are saved in a file to facilitate that.

The third one, (c), is for training a grammar to see how it affects the ranking of analyses. The '`z command`' here presumes that one trained grammar is selected for ranking, so the outputs of the '`t command`' have presumably gone through some kind of model selection after training, that is, choosing one of the grammars with updated weights, either manually or by a method of model selection.

The '`t command`' in (c) will fork as many processes as the number of experiments requested; NB. discussion around (19). It's best to use it offline. For that, put it in the commands files and use the '`@ command`' for batch processing. Waiting online can be painful depending on the number of experiments and data sizes. Just don't do control-D on THEBENCH interface if you use it online; it will kill the subprocesses. Let the terminal hang. Don't close the lid on a laptop; that would suspend all work. (Speaking of laptops, which are more and more designed toward efficient handling of video, audio and graphics using specific processors for them, it is best to use a more general high-speed multicore/multiprocessor system to do the training. Many experiments that ground a laptop to a halt worked effortlessly when I ran them on a powerful desktop.)

THEBENCH command macros

If you are training a grammar on a large supervision dataset, and want to find out how the top-ranked analyses fare with the gold pairings of expressions and meanings, one option of '`# command`' combined with the '`> command`' can be helpful. Before ranking, turn on logging with the '`> command`'. Then for '`# command`' in a batch command file use the '`bare`' option to eliminate verbosity. Minimal amount of extra text will be saved in the *processor's log file*, which you can easily eliminate. Don't forget to turn off logging at the end, using the '`< command`'.

Logging

One advice about the '`/ command`'. This command clears the `/var/tmp/thebench` directory where the internal results of THEBENCH are kept. Do not use this command if you started an experiment, either online from the interface or in batch mode (see glossary). These commands create files that are needed later on; see the end of §4 for file organization in THEBENCH.

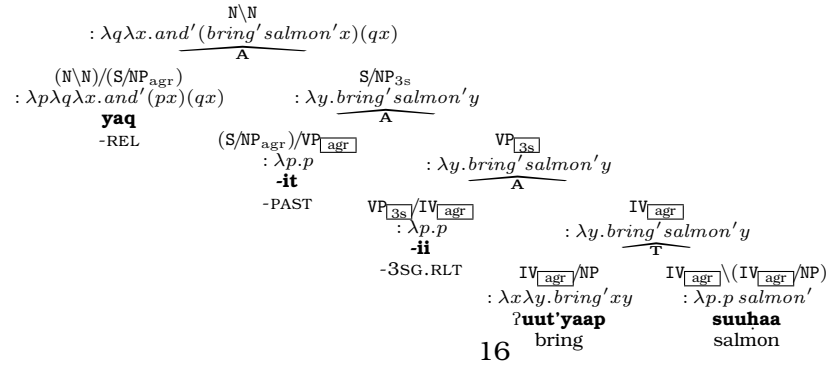
Clean-up

9 To do

I have a personal to-do list for collaborative work. I mention here the technical ones which require programming. For typology work, just drop me an email please.

1. Implementing the 'dashed boxes' of THEBENCH. 'Analysis' as understood in THEBENCH is a wholistic process, where every part-process is interpreted in the light of the whole in check all the time, at every step. Analyses can be displayed in the following full form.

(21) yaac-waas-it-ʔiš čakup-ʔi yaq-it-ii ʔuut'yaap suuḥaa
walk-outside-PAST-3SG man-DET -REL-PAST-3SG.RLT bring salmon
'The man who brought salmon left.' Nuu-Chah-Nulth; Wojdak 2000:274

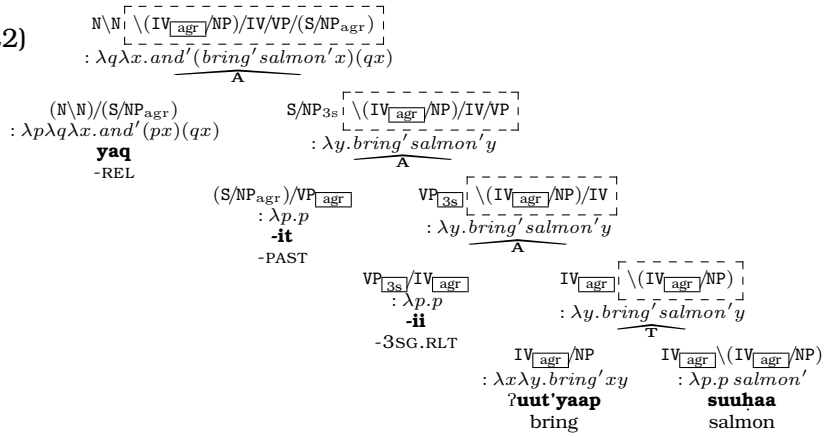


However, the original conception of categorial grammar was not just a better and simpler alternative to phrase-structure grammar, with or without movement. (I suppose this rather unfortunate overinterpretation emanated from Bar-Hillel, Gaifman and Shamir 1960/1964.) Every step of analysis is supposed to keep the whole in check, with parts showing *subprocesses* themselves. This was the connection to Gestalt Psychology, see for example Wertheimer (1924).

The whole in check

So what we really want to implement in THEBENCH is showing how the whole is kept in check at every step. The dashed boxes in the following analysis of the same example show how it can be done. It has not been implemented in THEBENCH yet.

(22)



The overall result on the top in its full form does not mean that for example $(\mathtt{S/NP_{agr}})$ is followed in sequence by `VP`, then `IV`, and so on. It shows the structural unfolding of the entire expression, that is, the part-processes.

Such structures would be necessary but not sufficient to situate reference in context.

2. Production/generation. Categorial production can be seen as expressing an intended complex reference by choosing elements from grammar that would serve that reference. It would be more than expression of thought. It is more like environment control by language.

One way to start studying such processes is to see how far compositionality in grammar can be put to work on this task. To do this, it would be nice to have a "structural chunker," one that would take the triplet `<word, category, size>` and produce an expression, where `word` is the chosen phonological left edge of the chunk, `category` is the syntactic category of the overall result intended, and `size` is the upper limit on the chunk in terms of number of elements in it. The upper limit would be needed to make the problem decidable because the category can be endotypic, for example `S\S`, therefore the process is potentially recursive.

Generation is usually understood that way in language technology with phrase-structure grammars, because they tend to take care of distributional adequacy first and worry about reference later. In categorial grammar however, it is half the story, because reference itself would reveal the item choice in the first place.

3. Encapsulating the monad. A true monad is a hermetic seal. We cannot peek inside to see the intermediate results. In the current state of THEBENCH, we can. That's because it is easier to debug.

However, turning the current state of the implementation to a true monad requires more than employing a monad library in Python or Common Lisp. The categories that are input to the monad would have to be lifted in syntax and semantics.

Acknowledgments

I thank Python, Lisp, `StackOverflow` and `StackExchange` communities for answering my questions before I ask them. The predicate-argument structure evaluator in the processor is based on Alessandro Cimatti's abstract data structure implementation of the lambda calculus in Lisp, which allows us to see the internal make-up of the l-command. David Beazley's `sly` python library made THEBENCH interface description a breeze. Marco Heisig's `cl4py` python library made Python-Lisp communication easy, which allowed me to recycle some Lisp code. Luke Zettlemoyer explained to me his sequence learner in detail so that I can implement it on my own. Discussions with Chris Stone clarified some of the tasks in to-do list. Thanks to all five colleagues. I blame good weather, hapless geography and my cats for the remaining errors.

Basic glossary of THEBENCH

analysis Comparing the synthetic elements of grammar by composition.

auxiliary file A file internally generated by THEBENCH for the processor, and saved in `/var/tmp/thebench`. Examples are grammar source as Lisp code and processor-friendly supervision files.

batch mode Use of the '`@ command`'. You will see THEBENCH prompt doubled in the log file. THEBENCH begins to read your commands from a file rather than the terminal. The commands in the command files can themselves be the '`@ command`; useful for trying many models at batch mode.

editable file A grammar file that you create and save as text, or the grammar files that THEBENCH generates by '`c command`' and '`z command`'.

element A (synthetic) element of grammar; one of (12).

grammar, case functions file A file automatically generated by THEBENCH when you run the '`c command`'. File extension is '`.sc.arules`'. Such files are editable, therefore saved in your working directory. Merging these files with grammar text is up to you. The '`c command`' adds them to the currently loaded grammar only.

grammar, text file A textual file containing a grammar with entries in the form of (12). The file extension is optional, and it is up to you. Resides in your working directory.

grammar, re-text file A textual file generated by the system from a '`.src`' file. The result file is identical in structure to the edited grammar which generated the '`.src`' file. The system adds a default parameter value and a unique key to identify every entry. Such files are saved in your working directory after fetching the '`.src`' file from `/var/tmp/thebench`, indicating that they are editable. The '`z command`' does this.

grammar, source code file A textual grammar which is transformed to the processor notation. It is actually Lisp source code. Its extension is '`.src`' if generated by the system. THEBENCH saves them in the folder `/var/tmp/thebench` by default.

identifier, token An atomic element of ASCII symbols with at least one alphabetical symbol, not beginning with one of {`+, *, ^, .`}, which are the slash modalities. It may contain and begin with: a letter, number, tilde, dash or underscore. Examples are basic categories, feature names and values, relation names, parts of speech, predicate basic terms (names of predicates and arguments), lambda variable names.

identifier, predicate modality An identifier that consists of '`_`' only. It is a simple convention from Bozşahin (2023), nothing universal, to signify that the stuff before it is the predicate and the stuff right after it is a modality of the predicate rather than an argument. Used in l-command only.

identifier, variable An identifier that begins with '`?`'. Used in the feature values of basic syntactic categories to represent underspecification.

!identifier, !token An identifier/token in a lambda term prefixed with '`!`'. The Lisp processor converts such identifiers to a double-quoted string. For example, `!_name` becomes `"_name"` in the processor. It is kept for legacy.

intermediate representation Sometimes it is difficult to tell whether a textual element in grammar text has been converted to proper structural representation. Using the '`i command`' you can check the internal structure of all entries in a grammar in the form of Python code. The resulting file is editable (e.g. if you want to explore its dictionary data structure in `python`), therefore saved in your working directory unless specified otherwise. You can do the same check for one element only, without adding it to grammar. Use the '`- command`' for that.

item Any space-separated sequence of `UTF-8` text, for example the phonological material. In representation and processing, it is case-sensitive.

macro file See the batch mode and the '`@ command`.'

MWE Multiword expression. Something referentially atomic in a surface expression, marked as `|..|`, for example `|the bucket|`. Whitespacing is not important. In a grammar entry, multiple words before the part of speech is by definition an MWE. *In a surface expression* they must be within `|..|` to match that element in analysis.

modality, syntactic One of {`^`, `*`, `+`, `.`} after a syntactic slash as a further surface constraint. Assumed to be '`.`' if omitted. Controls the amount (none '`*`' to all '`.`') and the degree (harmony '`^`' disharmony '`+`') of composition.

part of speech tag An identifier which can be put to various use, e.g. morphological identification, grammatical organization. Such identifiers can be any sequence of ASCII non-space tokens. No universal set is assumed. For example, the '`c command`' uses these tags to derive case functions, because the verbs cannot be discerned from their syntactic category alone. Take for example the category `(S\NP)\VP`: Is this for a verb or an adverb? Somebody has to just designate them as verb or verb-like, to be targeted by the '`c command`'.

ranking Looking at the most likely analysis of an expression given a trained grammar.

relational category, asymmetric A category which maps a surface category to another surface category, to indicate that the surface form bearing the first one also bears the second one. In analysis, they apply in the order specified in grammar.

relational category, symmetric A category which maps two surface forms to each other if they are categorially related. In analysis, they are treated as independent elements; whichever takes part in surface structure will be used with its own category. Their order is not important, either in relation specification or in analysis.

singleton A basic category in single or double quotes, in basic and complex categories. For example `(s\np)/"the bucket"` treats `"the bucket"` as a category that can only take on one value, the surface string itself, as in *kicked the bucket* kind of idioms, so that a much narrower reference compared to the more general category `NP` can be discerned; see Bozşahin (2023).

special category Any basic category prefixed with '`@`'. They are application-only, to avoid nested term unification. Useful for coordinands and clitics, i.e. things that can take one complex category no matter what in one fell swoop.

supervision, text file A textual file containing entries in the form of (18). The file extension is optional; it is up to you. Resides in your working directory.

supervision, source code file A supervision file generated by the system from your supervision file text. It has the extension '`.sup`'. It is in the Lisp format that the processor uses. Such files are saved in the folder `/var/tmp/thebench`.

synthesis Putting together of an element of grammar. Although an element has components such as the phonological form, part of speech, the syntactic type and the predicate-argument structure, as far as analytics is concerned, this is a hermetic seal, and only the monad can manipulate the components.

References

Abend, Omri, Tom Kwiatkowski, Nathaniel Smith, Sharon Goldwater, and Mark Steedman. 2017. Bootstrapping language acquisition. *Cognition*, 164:116–143.

Ajdukiewicz, Kazimierz. 1935. Die syntaktische Konnexität. In *Polish Logic 1920–1939*. ed.Storrs McCall, 207–231. Oxford: Oxford University Press. Trans. from *Studia Philosophica*, 1, 1-27.

Bach, Emmon. 1976. An extension of Classical Transformational Grammar. In *Problems in Linguistic Metatheory: Proceedings of the 1976 Conference at Michigan State University*, 183–224. Lansing, MI: Michigan State University.

Baldridge, Jason. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Doctoral Dissertation, University of Edinburgh.

Bar-Hillel, Yehoshua. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Bar-Hillel, Yehoshua, Chaim Gaifman, and Eliyahu Shamir. 1960/1964. On categorial and phrase structure grammars. In *Language and Information*. ed.Yehoshua Bar-Hillel, 99–115. Reading, MA: Addison-Wesley.

Bozşahin, Cem. 2021. *CCGlab Manual*. `http://github.com/bozsahin/ccglab`.

Bozşahin, Cem. 2023. Referentiality and configurationality in the idiom and the phrasal verb. *Journal of Logic, Language and Information*, 32:175–207.

Bozşahin, Cem. 2025. *Connecting Social Semiotics, Grammaticality and Meaningfulness: The Verb*. Newcastle upon Tyne: Cambridge Scholars. Forthcoming.

Brown, Penelope. 1998. Children's first verbs in Tzeltal: Evidence for an early verb category. *Linguistics*, 36:713–753.

Brown, Roger. 1973. *A First Language: the Early Stages*. Cambridge, MA: Harvard University Press.

Cabay, S, and LW Jackson. 1976. A polynomial extrapolation method for finding limits and antilimits of vector sequences. *SIAM Journal on Numerical Analysis*, 13:734–752.

Crain, Stephen, and Rosalind Thornton. 1998. *Investigations in Universal Grammar*. Cambridge, MA: MIT Press.

Eisner, Jason. 1996. Efficient normal-form parsing for Combinatory Categorial Grammar. In *Proceedings of the 34th Annual Meeting of the ACL*, 79–86.

Ellis, Willis D. 1938. *A Source Book of Gestalt Psychology*. London: Routledge and Kegan Paul.

Gallier, Jean. 2011. *Discrete Mathematics*. Springer.

Gentner, Dedre. 1982. Why nouns are learned before verbs: Linguistic relativity versus natural partitioning. In *Language Development, vol.2: Language, Thought and Culture*. ed.Stan A. Kuczaj II, 301–334. Hillsdale, New Jersey: Lawrence Erlbaum.

Grimshaw, Jane. 1990. *Argument Structure*. Cambridge, MA: MIT Press.

Hale, Ken, and Samuel Jay Keyser. 2002. *Prolegomenon to a Theory of Argument Structure*. Cambridge, MA: MIT Press.

Hale, Kenneth. 1983. Warlpiri and the grammar of non-configurational languages. *Natural Language and Linguistic Theory*, 1:5–47.

Hoeksema, Jack, and Richard D. Janda. 1988. Implications of process-morphology for Categorial Grammar. In *Categorial Grammars and Natural Language Structures*. eds.Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler. Dordrecht: D. Reidel.

Klein, Ewan, and Ivan Sag. 1985. Type-driven translation. *Linguistics and Philosophy*, 8:163–201.

Klein, Wolfgang. 1994. *Time in Language*. London: Routledge.

Klein, Wolfgang, Ping Li, and Henriette Hendriks. 2000. Aspect and assertion in Mandarin Chinese. *Natural Language and Linguistic Theory*, 18:723–770.

Koffka, Kurt. 1936. *Principles of Gestalt Psychology*. London: Kegan Paul.

Lambek, Joachim. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Mac Lane, Saunders. 1971. *Categories for the Working Mathematician*. Berlin/New York: Springer.

MacWhinney, B. 2000. *The CHILDES project: Tools for analyzing talk. Third edition*. Mahwah, NJ: Lawrence Erlbaum Associates.

Moggi, Eugenio. 1988. *Computational Lambda-Calculus and Monads*. LFCS, University of Edinburgh.

Montague, Richard. 1970. English as a formal language. In *Linguaggi nella Società e nella Technica*. ed.Bruno Visentini, 189–224. Milan: Edizioni di Communità. Reprinted as Thomason

1974:188-221.

Montague, Richard. 1973. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*. eds.J. Hintikka and P. Suppes. Dordrecht: D. Reidel.

Ross, John Robert. 1967. *Constraints on Variables in Syntax*. Doctoral Dissertation, MIT. Published as *Infinite Syntax!*, Ablex, Norton, NJ, 1986.

Sapir, Edward. 1924/1949. The grammarian and his language. In *Selected Writings of Edward Sapir in Language, Culture, and Personality*. ed.David G. Mandelbaum. Berkeley: University of California Press. Originally published in *American Mercury 1: (1924) 149–155*.

Schmerling, Susan. 1981. The proper treatment of the relationship between syntax and phonology. Paper presented at the 55th annual meeting of the LSA, San Antonio TX.

Schmerling, Susan. 2018. *Sound and Grammar: a Neo-Sapirian Theory of Language*. Leiden/Boston: Brill.

Schönfinkel, Moses Ilyich. 1920/1924. On the building blocks of mathematical logic. In *From Frege to Gödel*. ed.Jan van Heijenoort. Harvard University Press, 1967. Prepared first for publication by H. Behmann in 1924.

Steedman, Mark. 2000. *The Syntactic Process*. Cambridge, MA: MIT Press.

Steedman, Mark. 2020. A formal universal of natural language grammar. *Language*, 96:618–660.

Steedman, Mark, and Jason Baldridge. 2011. Combinatory Categorial Grammar. In *Nontransformational Syntax*. eds.R. Borsley and Kersti Börjars, 181–224. Oxford: Blackwell.

Swadesh, Morris. 1938. Nootka internal syntax. *International Journal of American Linguistics*, 9:77–102.

ed.Thomason, Richmond. 1974. *Formal Philosophy: Papers of Richard Montague*. New Haven, CT: Yale University Press.

Tomasello, Michael. 1992. *First Verbs: a Case Study in Early Grammatical Development*. Cambridge: Cambridge University Press.

Wertheimer, Max. 1924. Gestalt theory. English translation published in Ellis (1938).

Williams, Edwin. 1994. *Thematic Structure in Syntax*. Cambridge, MA: MIT Press.

Wojdak, Rachel. 2000. Nuu-chah-nulth modification: Syntactic evidence against category neutrality. In *Papers for the 35th International Conference on Salish and Neighbouring Languages (UBCWPL)*, vol. 3, 269–281.

Woolford, Ellen. 2006. Lexical case, inherent case, and argument structure. *Linguistic Inquiry*, 37:111–130.

Zettlemoyer, Luke, and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of the 21st Conf. on Uncertainty in Artificial Intelligence*. Edinburgh.

Table 1: The command interface of THEBENCH. There is also the phantom command called '`pass`', which is useful for commenting on the tasks in the '`@ command`'s batch processing files. For example, '`pass the next command generates case functions`' does nothing but echoes itself. Any command in the interface can be put in the batch files as long as it can be processed without online input.

```
______\ Letter commands are processor commands; symbol commands are for display or setup
_______\______ Dots are referred in sequence; .? means optional; .* means space-separated items
 a .*   | analyzes . in the current grammar; MWEs must be enclosed in |, e.g. |the bucket|
 c .*   | case functions generated for current grammar from elements with POSs .
 e .*   | evaluates the python expression . at your own risk (be careful with deletes)
 g .    | grammar text .  checked and its source made current (.src file goes to /var/tmp/thebench/)
 i .    | intermediate representation of current grammar saved (file . goes to /var/tmp/thebench/)
 k      | reports categorial skeleton of the current grammar---its distinct syntactic categories
 l ..?  | Lisp function . is called with args ., which takes them as strings
 o .*   | OS/shell command . is run at your own risk (be careful with deletes)
 r .*   | ranks . in the current grammar; MWEs must be enclosed in |, e.g. |the bucket|
 t ...  | trains grammar in file . on data in file . using training parameters in file .
 z .    | source . located in /var/tmp/thebench/ and saved as editable grammar locally (.txt)
 @ ..   | does (nested) commands in file . (1 command/line, 1 line/command); forces output to .log
 , .*?  | displays analyses for solutions numbered ., all if none provided
 # .?   | displays ranked analyses; outputs only [string likeliest-solution] pair if . is 'bare'
 = .*   | displays analyses onto basic cats in .
 ! .?   | shows basic cats and features of current grammar (optionally saves to file .log)
 $ .*   | shows the elements with parts of speech in .
 - .    | shows (without adding) the intermediate representation of element .
 + .    | processor adds Lisp code in file .
 > ..?  | Logs processor output to file .log; if second . is 'force' overwrites if exists
 <      | Logging turned off
 /      | Clears the /var/tmp/thebench/ directory
 ?      | displays help
_______/ Use UP and DOWN keys for command recall from use history
```

Table 2: Processor functions accessible from the experiment files and the command line.

`beam-on`	Turns the beam on. For training.
`beam-off`	Turns the beam off (default). For training.
`beam-value`	Shows current properties of the beam processor (status and exponent). For display.
`lambda-on`	Turns on display of lambda terms at every step of analysis (default). For display.
`lambda-off`	Turns it off. Final results are always shown. For display.
`monad-all`	Processor set to use all monad rules (default). For analysis and training.
`monad-montague`	Processor set to use only application in composition (**A** and **T**). For analysis and training.
`nfparse-on`	Turns normal-form parsing on (default). For display.
`nfparse-off`	Turns normal-form parsing off. For analysis and training.
`onoff`	List of on/off switches that control analysis.
`oov-on`	Turns on out-of-vocabulary treatment. Two dummy entries assumed for OOV items: One with the category `@X\@X` and the other with `@X/@X`. For analysis and training.
`oov-off`	Turns off out-of-vocabulary treatment (default). For analysis and training.
`show-config`	Shows current values of all the properties above.