THEBENCH Guide

Cem Bozşahin Ankara, Datça, Şile 2022–23

Version: 1.0 April 10, 2023 coloured text means changes from the previous release
Home: github.com/bozsahin/thebench (there are instructions here to install and use the system)

1 Introduction

THEBENCH is a tool to study monadic structures in natural language. It is for writing monadic grammars to explore analyses and to train models.

Monadic structures are binary combinations of elements that employ semantics of composition only, in a hermetic seal, hence the name; see Mac Lane (1971), Moggi (1988) for that. A monadic grammar contains only synthetic elements that are shaped by this analytic invariant.

Computationally, a monad composes two functions f and g to maintain the dependency of f and g, which is $\lambda x.f(gx)$. It does so by doing $g \circ f$. 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 it means 'do g, then f' to get the dependency right in $f \circ g$, is somewhat counter-intutitive, especially if we are going to construct all structure from sequencing and reference categorization.

Linguistically, the natural language monad of THEBENCH internally turns f and g into semantic functions, even if one of them is not. It does so by keeping two command relations, which every element of grammar specifies, intact: surface command (s-command) and predicate-argument command (l-command). 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 variant vocabulary and preserved command relations can make *any grammar* transparent and consistently interpretable with respect to invariance. Syntactic, phonological, morphological and predicate-argument structural reflex of composition-only analytics is the main subject of studying monadic structures in natural language.

A proposal regarding the limits of elementary (synthetic) vocabulary and analytic structure in such grammars is described in Bozşahin (2023), hereafter MS. THEBENCH is a tool implementing that proposal. Briefly, it is a Polish School-style categorial grammar. It uses composition only. That is, application is also turned into composition. It has linguistic implications for case, agreement and grammatical relations. Unlike generative grammar, there are no top-down universals such as EPP, MLC or universal lexical categories such as N, V, A, P, or universal argument types such as those in X-bar theory. All of that is predicted from the ground up. 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),[1] and referentiality, together. MS's configurational approach to surface structure is inspired by Montague (1973), 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, 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.),

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

because different categorization due to different reference is expected to play the key role.[2] Also, syntactic types of second-order (structural) functions including those for case, agreement and grammatical relations require a linguistic theory (and a theory of cognition) rather than a logical theory, because we are studying choice rather than decision; see MS.

Consequently, distinct function-argument and argument-function sequencing of reference is the cause of categorial and structural asymmetries, which is an idea that goes back to Schönfinkel (1920/1924). He had used the idea to motivate his combinators, all but one considered to be too powerful in MS for monadic structures. What's left behind, composition, is anybody's composition, but with some exploratory power arising from this distinction. We don't really know whether choice is compositionally determined. We would be studying how far it can be transmitted compositionally starting with the whole, but not holistically. (This sounds like a distant memory in psychology too, but see Koffka 1936).

Empirically, morphology and syntax do not compete for the same task (e.g. for surface bracketing), morphology is not confined to the 'lexicon' or to some 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. And, referential differences in all kinds of arguments cause categorial differences including those in idioms and phrasal verbs (Bozşahin, 2022).

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 bottom-up view of grammar. Elements' morphology plays a crucial role in transparency of analytics.

Technically, the description functions of the THEBENCH, the functions that transform internal files to editable files, and interfaces of THEBENCH, are written in `Python`. Processor functions are written in `Common Lisp`. The installer (NB. top of 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, which you can use online, and offline (the '`@ command`', aka. 'batch mode'). There is no need to know either language to use THEBENCH. (Maybe this much is good to know to understand processor's output: `T` means true and `NIL` means false in Lisp.) Use the '`? command`' when you launch THEBENCH to recall the full list of commands.

Generally, the editable files of THEBENCH, either written by you or generated by the tool, reside in your working directory. Processor's internal files reside in `/var/tmp/thebench`. Occasional clean-up of both locations is recommended. 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.

2 Synthetics

There are three kinds of elements of grammar in THEBENCH:

(1) a. an elementary item,

b. an asymmetric relational rule, and

c. a symmetric relational rule.

Examples in THEBENCH notation are respectively:

[2] 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. MS is more compatible with the 1970 idea, however arguing instead from different subcategorization due to different referentiality.

(2)
```
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 in the screen. Everything starting with '`%`' till the end of a line is considered to be a comment. Capitalization is significant only in phonological elements. Whitespacing is not important anywhere. Empty and comment-only lines are fine. The order of specification for elementary items and symmetric relational rules in the grammar's text file is not important. Asymmetric relational rules apply in the order they are specified. THEBENCH home repository contains sample grammars as a cheat sheet. (My personal convention is to write the commands that process or generate the files in these folders in '`.commands`' files, which are input to the '`@ command`'.)

The first kind of element in (2) 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.)

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

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 is the most common convention in monads; it is called the monadic type constructor.) The part before the '`:`' is called the syntactic type, which is the domain of s-command. The part after '`:`' is called the predicate-argument structure, the domain of l-command.

Left-associativity is assumed for content in both command materials. Right-associativity is assumed for lambda bindings. If you don't like associativity and like to write the structure yourself, please be careful with the proliferating numbers of parentheses. Together they constitute a syntax-semantics correspondence for the element.

The correspondence is explicit in the order of syntactic slashes and lamba bindings. For this example, '`\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 '`\x.\y.like x y`'.

The syntactic type consists of basic and complex categories. (The term 'category' is quite commonly used also for the 'syntactic type' when no confusion arises.) The first one is 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 basic categories only (unlike many other phrase-structure theories). Multiple features are comma-separated. A variable value for a feature is prefixed with '`?`'; for example, '`np[person=1,number=sing,agr=?a]`' means the agreement feature is underspecified. Complex categories 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]`' . The surface directionality would be different. In the first case, the function would look to the left, in the latter, to the right, in surface structure (i.e. phonological order).

Modal control on the directionality is optionally written right after the syntactic slash, such as '`^`' above. 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 semantic composition in the monad. There are also the 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.) There is no modal control on them. They are applicative because they cannot be the range of a complex category, they are domains only ; see MS and Bozşahin (2022).

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

The second kind of element in (2) is an asymmetric relational rule. It means that the element bearing the category on the left of '`-->`' at surface structure *also* has the category on the right. The token immediately after '`#`' is the name of the rule, in this case '`np-raise`'. For such rules 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 rules are not associated with a particular substantive element.

The third kind of element in (2) is a symmetric relational rule. It means that during analysis either form is eligible for consideration along with its particular category. The grammarian 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 rule name is right after '`#`'. The material on the left and right of '`<-->`' must also contain some phonological material (ending with a comma) so that we can reflect the substantive adjustment on the categories.

There are no elements or rules in 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 representations from surface asymmetries alone.

3 Analytics

The tripartite organization of every synthetic element is

(3)`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 these domains are obviously not isomorphic, but, equally obviously, related, in conveying reference. One claim of MS is that they are homomorphic to analytic structure because of that.

The analytic structure is universal. This means that these domains are homomorphic to analytic structure also 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.

Knowledge 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 s-command is the following: If we have the category sequence (a) below, we get (b) by composition, not (c).

(4) 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 combination 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).

Meta-categories such as $(X\backslash X)/X$, written in THEBENCH as `(@X\@X)/@X`, are allowed with application only. Only coordination and coordination-like structures need them, which are syntactic islands with Ross (1967) escape hatches, that is, things with same case, therefore same category:

(5) `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 universal analytic structure. (Recall also that structural-reentrant unification is in fact semi-decidable.)

For l-command, one important property is that its realization in analysis arises from the evaluation of the surface correspondents in 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:

(6) *`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, 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. (6) is not an analytically interpretable synthetic element. If the intention here were to capture topic-dropped sleep, for example *slept all day, what else could I do?*, it might be say `slept :: s : sleep topic`.

4 Exploration

1. There are some aids in THEBENCH to explore monadic surface structures. MS makes a prediction about what kind of syntactic case and similar structural functions arise given a grammar. These can be conceived as asymmetric relational rules, 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`' in the tool generates these functions from a grammar (loaded by the '`g command`'), taking a list of parts of speech from which to generate the second-order functions. (Presumably, these are the parts of speech of the verbs and verb-like elements.) These functions are saved in a separate textual grammar file with extension '`.sc.arules`'. The name designates that these are consequences of synthetic case of MS, given the current grammar. The name also designates that all of them are asymmetric relational rules.

These rules 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 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 lot more surface configurations as a consequence of synthetic case (see MS) 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.

2. We can view the surface-syntactic skeleton of a grammar. The '`k command`' goes over all the elements of the currently loaded grammar, and reports how many distinct syntactic categories are there in it. For brevity it does not report features of a basic category. This much is certain though: if two categories reported by the command look the same without basic category features, 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 feature-less version looks common: `s\np`.

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

This 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 possible values.

3. There is no morphophonological analysis built in to THEBENCH. Morphological boundedness of a synthetic form can be designated with '+' in `a/r commands`. It goes as far allowing in analysis and ranking commands entering surface tokens including 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 operating on phonology to construct reference, Montague-style.

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`'.

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 modeling sense, characterizing a data point. To do this, THEBENCH turns a textual grammar into a 'source form', which is a `Common Lisp` data structure. Such files are automatically generated and carry the file extension of '`.src`'. 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.[3] 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.

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 supervision data. Therefore it is important to know that entries with the same phonological form compete with their weights in making themselves into 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. (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.)

For example, when you inspect the re-generated textual grammar containing (2), you may see something like:

(7)
```
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 (2). The symmetric relational rules are compiled into separate entries in (7), and their link is preserved by using the name of the rule as the '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 surface structures therefore need parameter values of their own. Using the preserved link (same tag) we can explore re-construction of the paradigm intended by '`<-->`'. The possibility of using the same rule name to capture a paradigm facilitates this exploration.

[3]The '`z command`' can also transform legacy '`.ded`', '`.ind`' and '`.ccg.lisp`' grammars 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.

There is one more change in the re-constructed 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. I could make it case-sensitive, to preserve both analyst's and Lisp's cases, but this would be quite error prone (is `AdvP` same as `advp`? I would think so.)

5. Taking the text source of a developing grammatical analysis and turning to modeling with it is the idea behind the textual re-conversion of a grammar source. The re-constructed source has the symmetric rules turned into separate entries with same 'part of speech': the rule'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`'.

Training a grammar with pre-analyzed 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 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.

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

(8)
```
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, for example `|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 predicatnfp.xp.1.2a.1.0c.srce-argument structure corresponds to it, given a grammar with that MWE.) 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 on the right for every entry. We can then load the chosen grammar, and rank analyses with it using the '`r command`'.

6. Combining **4–5** is one way 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). (Studying referentiality of all kinds and configurationality in one package seems to be one way to avoid the purportedly dichotomic world of verbs-first and nouns-first approaches.) The observables would be the phonological form on the left, 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.

The tools used for training in THEBENCH are triggered from its 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 grammar size, number of experiments attempted and amount of supervision data. We don't have to stay in the originating session to keep the experimental runs going. They operate independently, using the

nohup facility (for 'no hang up'; 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. Just let it die or hang, or close the window normally, e.g. by clicking the red button in the red-yellow-green menu.).

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

(9)
```
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 strings with many possibilities of analysis, which usually arises when case functions are added to a grammar. These parameters are useful to run experiments on machines with different powers and architecture (number of cores, amount of memory, size of heap) without changing the basic setup.

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 (8). 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), 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 are given at the back, which are callable from the command interface as well, using the '`l command`'. (For functions with one more arguments, calling is a bit more complex; see `cl4py` library documentation.) In the second experiment, it is `beam-on`. This function focuses the gradient on items with largest weight changes during iterations, avoiding 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, which is sometimes a lifesaver in eliminating excessive runtimes or out-of-memory errors.

Three work cycles are worth pointing out from personal experience. The first one is for the development of an analysis by checking its aspects with respect to theoretical assumptions. This would be the simple cycle in (10a), with '`k`' and '`!`' commands sprinkled in between. 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. The third one is for training a grammar to see how it affects ranking of analysis. This would be the cycle in (c).

(10) 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
```

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 lambda calculus in Lisp, which allows us to see the internal make-up of 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. Thanks to all four colleagues. I blame good weather, bad geography and my cats for the remaining errors.

Basic glossary of THEBENCH

analysis Combining synthetic elements of grammar in a given expression 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.

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 (2).

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 currently loaded grammar only.

grammar, text file A textual file containing a grammar with entries in the form of (2). 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 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 they are editable. The '`z command`' does this.

grammar, source 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 {`+, *,^, .`}. It may contain and begin with: a letter, number, tilde, dash or underscore. Examples are basic categories, feature names and values, rule 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 (2022), 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 feature values of basic syntactic categories.

!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. 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.

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 tokens before the part of speech is by definition an MWE, that is, they must be in `|..|` in a surface expression 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 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 also contain '<' or '>'. No universal set is assumed. For example, the '`c command`' uses these tags to derive case functions, because verbs cannot be discerned from their syntactic category alone. Take for example the category (S\NP)\VP: Is this a verb or an adverb? Somebody just has to just designate them as verb or verb-like, to be targeted by the '`c command`'.

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

relational rule, asymmetric A rule 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 rule, symmetric A rule 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 rule 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 (2022).

special category Any basic category prefixed with '`@`'. They are application-only, to avoid structured 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 (8). File extension is optional; it is up to you. Resides in your working directory.

supervision, source 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 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.

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

Bozşahin, Cem. 2022. Referentiality and configurationality in the idiom and the phrasal verb. *Journal of Logic, Language and Information*, 27:1–33.

Bozşahin, Cem. 2023. *Monadic Structures*. Leiden/Boston: Brill. 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.

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

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.

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.

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

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 *Non-transformational 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.

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 a phantom command, named 'pass', which is useful for commenting on the tasks in the '`@ command`' files. For example, '`pass the next command generates case functions`' does nothing but echoes itself.)

```
Letter commands are processor commands; symbol commands are for display or setup
Items in . must be space-separated; .? means optional .
 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 bench 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
 = .    | 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 experiment files and THEBENCH command line.

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