How to use the Syntacticon

Raphael Finkel raphael@cs.uky.edu Daniel Kaufman dkaufman@qc.cuny.edu

April 7, 2025

Contents

1 Introduction

The Syntacticon is a web-accessible facility that applies rules to syntax trees to achieve a surface-form utterance. The input to the program includes a syntax tree, syntax rules to manipulate the tree, and morphophonological rules to convert the resulting tree to a surface form. This program is intended both as an aid in teaching syntax and to help researchers formalize syntax theories.

One can use the Syntacticon in several ways.

- Enter a syntax theory and have the Syntacticon compute its results.
- Choose a prepackaged syntax theory from the **Library**.
- Investigate the phonological features of IPA symbols.

The Syntacticon is the third in a series of facilities, building on the previous two: the Phonomaton (for phonological theories) and the Morphoton (for morphological theories). Many of the features of those facilities are also available in the Syntacticon.

This document is organized according the input sections of the Syntacticon. Each section has a show/hide button that lets the user expand or contract its contents. The Syntacticon preserves the contents of input sections as it creates output, so the user can make modifications and resubmit the input.

2 Metadata

This input section contains optional information about the syntax theory. Its fields, with sample values, are these:

```
Language = English
Date = 2024-12-3
Family = Germanic, Indo-European
Code = stan1293
Sources = various books
Implementation = Jane R. Doe
Details = active and passive forms
Complexity = 5
```

The `Code` is, by convention, the Glottolog code. The `Complexity` is a guess at the difficulty of constructing and understanding the theory. The **Library** section (See **??**) presents most of the metadata for each of the stored syntax theories.

3 Syntax patterns

This input section provides the starting tree (or rarely, trees) that the syntax theory manipulates. Each tree representation should be on a single line, or, for convenience, split into multiple lines with at the end of all but the last line. The symbol `%` introduces a comment that continues to the end of the current line.

3.1 Format

Figure 1 shows a simple syntax tree.

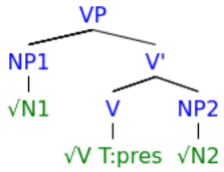

Figure 1: Syntax tree example

Syntax trees should follow a particular format. Nodes in the tree can be phrases, intermediate constituents, or heads. Each node has a **name**. All names in the tree must be unique.

The name of a **phrase** must end with `P`, optionally followed by digits to distinguish similar names. A phrase must have one or two children. Generally, one of the children is an intermediate constituent, and

the other is an optional specifier, which is another phrase. It is also valid for a phrase to only contain a surface expression, described below.

An **intermediate constituent** must have a name that ends with an apostrophe, optionally followed by digits. (The Syntacticon accepts either the ordinary `'` or the Unicode 02b9 `′`). It must have two children, a head and a phrase, which acts as a complement. The children may be in either order. As a special case, usually at the end of a long tree chain, the complement may be a phrase that is also a head.

A **head** may have any name that does not end with an apostrophe (followed by digits). It has no children; its content is a surface expression, which is called its **feature**.

A **surface expression** is meant to expand to part of the eventual surface form of the tree when it is spelled out. It contains one or more units, separated by space. A **morphological unit** names a morphological rule and provides morphological input. For instance, `D:def,3sg` names the morphological rule `D`, perhaps intended to express a determiner, with the morphological input for definiteness and a 3sg subject. A **lexeme unit** starts with √ and then names a lexeme (described in Section

In the tree shown in Figure 1, the root is a phrase (as it must be): `VP`. It has a specifier `NP1` and an intermediate constituent `V'`. The specifier `NP1` is a head, containing the surface expression `√N1`, referring to a particular lexeme, such as `dog`. The intermediate constituent `V'` has a head `V` and a complement `NP2`, each of which is a head. The head `V` contains a surface expression with two units, the lexeme unit `√V`, perhaps referring to the lexeme `has`, and a morphophonological rule `T`, perhaps expressing tense, with the morphological property `pres`. Finally, the phrase `NP2` is a head containing `√N2`, perhaps referring to `bone`.

3.2 Representation

A tree can be presented in either a bracket or a brace form. The **bracket** form is more conventional, although more verbose. Each node of the tree is represented in the form

```
[Name Content]
```

The tree shown in Figure 1 has this representation in bracket form:

```
[SP [NP1 √N1] [V′ [V √V T:pres] [NP2 √N2]]]
```

Because it is clumsy to count matching brackets, the Syntacticon accepts missing brackets at the start or at the end of the bracket representation, adding brackets implicitly as needed. The representation, which should be in a single line, may be split into multiple lines, with intermediate lines terminated by \.

The **brace form** takes advantage of the convention, visible in Figure 1, that an intermediate constituent has the same name as its parent phrase, but with the `P` converted to an apostrophe, and that its head has

the same name again, without the P or apostrophe. The brace form of the tree shown in Figure 1 is

```
{V {N1 √N1} √V_T:pres {N2 √N2}}
```

The Syntacticon automatically adds P to phrase names and apostrophes to intermediate constituents. It converts underscores to spaces in heads. It also inserts missing braces at the start and at the end of the brace representation. Again, long lines may be split with \ at the end of intermediate lines.

4 Lexeme selection

This input section provides the lexemes that the syntax patterns refer to with the √ notation. To supply the lexemes for the tree shown in Figure 1, this section could have

```
N1: dog
V: has
N2: bone
```

Each of these lexemes must be further defined in the **Roots** section (See section **??**) to show its stems and the morphosyntactic properties that determine which stem to use.

5 Syntax rules

The purpose of syntax rules is first to manipulate the syntax tree, then to create a resulting surface form of an utterance. Each rule is on a single line, although long lines may be interrupted by a \ at the end of an intermediate line. An empty line separates sets of syntax rules; the Syntacticon applies each set anew to each syntax pattern. The character % introduces a comment that continues to the end of the line. The syntax rules also allow **literate comments**, which start with %!, and which are meant to elucidate the purpose of the rules. The Syntacticon collects literate comments and presents them at the top of the output page.

5.1 Format

The format of all syntax rules is

```
RULENAME: COMMAND / CONDITION // FLAGS
```

A rule may omit the RULENAME and its colon. The purpose of the CONDITION is to restrict when the rule should apply. The purpose of the FLAGS is to modify how the rule applies. Both CONDITION and FLAGS are

optional; if they are omitted, the syntax rule should also omit the `/` character(s) that introduce them.

Commands generally modify the syntax tree. Some commands are purely for debugging purposes. One command changes the tree into a surface string.

5.2 Conventions

In the following, we use these conventions to describe syntax rules.

- `NAME`: The name of a particular node.
- `HEAD`: The name of a head node.
- `NODES`: A set of nodes, possibly empty. If there are several nodes in the set, they are sorted breadth-first, left-to-right, according to their position in the tree.
  - a `NAME`
  - a Perl regular expression, such as `/DP/`. All nodes with names matching the regular expression (anywhere in the name) are included.
  - A specification in braces, such as `{3sg}{past,fut}`. All head nodes with content matching the morphosyntactic specification are included. The rules for morphosyntactic specification are in Section **??**.
  - A relation between nodes, as discussed below.
- `FEATURE`: A morphosyntactic feature set and its morphosyntactic rule-set name, such as `Agr:apart,3sg`.
- `PATTERN`: a syntax tree in bracket form.
- `RELATIONS`: a comma-separated list of relations between nodes in the syntax tree. Each relation is a test that returns "true" or "false". Spaces are not allowed in relations. The relations are as follows.
  - `isSister(NAME, NAME)` Tests if the two named nodes have the same parent. In Figure 1, `NP1` and `V'` are sisters.
  - `contains(UPPER_NAME, LOWER_NAME)` Tests if the first named node is an ancestor of the second one. In Figure 1, `V'` contains `NP2`.
  - `ccommands(UPPER_NAME, LOWER_NAME)` Tests if the first named node is either a sister of the second node or if its sister is an ancestor of the second node. In Figure 1, `NP1` ccommands `NP2`.

- `isSpecifier(UPPER_NAME,LOWER_NAME)` Tests if the second named node is a specifier of the first node, that is, its parent is the first node and its sister is an intermediate constituent. In Figure 1, `NP1` is the specifier of `VP`.
- `containsFeature(NAME, REGEXP)` Tests if the named node or any of its descendants has contents matching the regular expression. In Figure 1, `V'` contains feature `/T:pres/`.
- `isOccupied(HEAD)` Tests if the given head node contains a surface expression. In Figure 1, `NP1` contains the expression √`N1`.
- `headPhrase(HEAD_NAME, PHRASE_NAME)` Tests if the given head node acts as the head of the given phrase. In Figure 1, `V` is the head of `VP`.
- `isHead(HEAD_NAME)` Tests if the given node is a leaf of the tree. In Figure 1, nodes `NP1`, `V`, and `NP2` are all heads.

5.3 Conditions

The condition optionally attached to a syntax rule is formed of a list of relations (no spaces allowed), each of which must hold for the syntax rule to apply. If a relation has `_` instead of a `NAME`, it refers to the source of the syntax rule. This facility lets a condition filter a set of source nodes to select only the ones of interest. If a relation has `=` instead of a `NAME`, it refers to the destination for syntax rules that have both source and destination, such as `CopyFeature`.

5.4 Flags

Some flags are specific to particular commands; they are discussed as appropriate. Some flags apply to any command.

- `echo`: Show the command in the “Errors and messages” section of the output, for purposes of debugging.
- `once`: Only run the command once if `NODES` expands to more than one node. Here, ”run” means ”make an effective change to the tree;” nodes where no effective change happens are silently skipped.

5.5 Commands

The commands generally modify the current syntax tree.

- `MoveFeature (NODES`$_1$`)(NODES`$_2$`)`

  The features in the first set of nodes (the sources) (restricted to heads) are moved to become or add to the features in the second set of nodes (the destinations). Proper use of this command requires that the CONDITION restrict the destinations to a single node, although `NODES`$_2$ may specify more than one node. The list that the Syntacticon generates for `NODES`$_2$ are sorted in reverse to the usual breadth-first, left-to-right order. Each source node acquires a special content, marked by `[t]`, indicating a trace remaining after its original content has moved.

- `MoveToSpecifier (NODES)(NAME)`

  Move the first valid node in the source to the specifier position of `NAME` which must be an intermediate constituent that has no specifier. By default the node goes to the left of the destination, but if the destination has a right-hand sister with the special name `[Spec]`, then the source is moved to the right-hand position, replacing `[Spec]`. The flags implicitly include `once`.

- `DIR-Adjoin (NODES)(NAME)`

  Here, `DIR` is either `L` or `R`, indicating a left or right adjoin operation. The flags implicitly include `once`. The destination node is replace by a new node with the same name but with a superscript suffix, such as $_1$. The superscript is chosen to make the new name unique in the tree. The new node has the original destination node as one child and the source node as its other child, either to its left or right, as specified in the syntax command. The source node is replace by a trace `[t]`. For example, applying `L-Adjoin(V)(NP1)` to the tree in Figure 1 results in the tree in Figure 2.

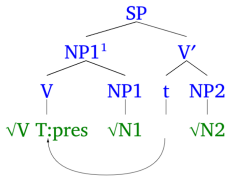

Figure 2: After `L-Adjoin(V)(NP1)`

- `AdjoinFeature(FEATURE)(NODES)`

  Adds the feature as a label on the destination nodes. A **label** is like the content of a head node, but it can apply to any node, and it forms a part of the node name, separated by a colon. The purpose of this rule is to let a subsequent syntax rule be conditional on the label. For instance, applying `AdjoinFeature(pres)(NP1)` to the tree in Figure 1 results in the tree in Figure 3.

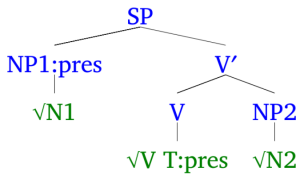

Figure 3: After `AdjoinFeature(pres)(NP1)`

- `MergeRoot(ROOT)(NODES)`

  Adds the given root (with a prefix √) at the start of the content of the NODES, restricted to heads. For instance, applying `MergeRoot(pres)(V)` to the tree in Figure 1 results in the tree in Figure 4.

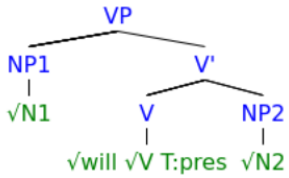

Figure 4: After `MergeRoot(pres)(V)`

- `MergeFeature(FEATURE)(NODES)`

  Adds the feature (such as `Agr:past`) to the given nodes, which should be heads that already have a surface expression. If the morphosyntactic rule set (such as `Agr`) is already represented in the destination node, then the morphosyntactic features are added to that surface expression, with duplicates suppressed. Otherwise, the entire feature is added to the surface expression. For instance, applying `MergeFeature(T:pres,active)(V)` to the tree in Figure 1 results in the tree in Figure 5.

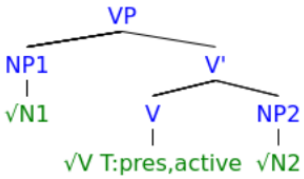

Figure 5: After `MergeFeature(T:pres,active)(V)`

- `CopyFeature(MP`$_1$`,NODES`$_1$`)(MP`$_2$`,NODES`$_2$`)`

  Copies the contents of the first morphosyntactic rule set collected from all of the first set of nodes into the second morphosyntactic rule set in all the second set of nodes. For instance, applying `CopyFeature(T,V)(D,NP1)` to the tree in Figure 1 results in the tree in Figure 6.

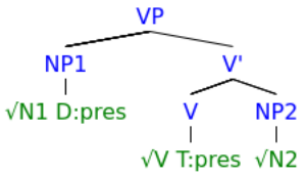

Figure 6: After `CopyFeature(T:V)(D,NP1)`

- `DeleteFeature(NODES)(FEATURE)`

  Modifies the surface expression of all the given nodes by removing the feature. If the feature is a morphosyntactic rule set, it is removed along with all its morphosyntactic features. If it is a single feature, it is removed from whatever morphosyntactic rule set it belongs to. For instance, applying `DeleteFeature(V)(T:pres)` to the tree in Figure 1 results in the tree in Figure **??**.

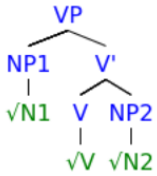

Figure 7: After `DeleteFeature(T:V)(D,NP1)`

6 Acknowledgements

The IPA charts are based on ones at `www.internationalphoneticalphabet.org/ipa-sounds/ipa-chart-with-sounds/`. The text-to-pronunciation tool is based on code at `ipa-reader.xyz/` by Katie Linero. The graphing tool for syntax patterns (`mshang.ca/syntree`) is by Miles Shang

The authors coded the Syntacticon in Perl (**?**). The code is available under a Creative Commons CC-BY license.