Yanker User Manual

Antonin Delpeuch

October 23, 2014

1 Introduction

This document describes the `yanker` software, how to use it, and how we could improve it.

2 Purpose

The categorical framework for compositional and distributional semantics requires the linguist or the engineer to define different word meaning shapes depending on the grammatical role of a word.

As a consequence, this framework has not been applied to arbitrary sentence structures, as it would require the definition of many (hundreds or thousands) different shapes for the meanings of the words, and probably as many learning strategies for them.

This tool is an attempt to reduce the effort needed to define such shapes. It provides a graphical editor where the string diagrams corresponding to the word meanings can be drawn in an intuitive way. Moreover, it allows to define word meanings not only for particular types, but also for *type skeletons*, i.e. polymorphic types.

2.1 Type system

This software currently uses the Lambek calculus as its type system. The Steedman notation and pregroups should be supported in the near future.

2.2 Polymorphic types: notation

We denote type variables by numbers. Hence `1` should not be confused with the monoidal unit: it is a variable ranging over all the types. Here are a few examples:

- `N/N` matches only itself (as it contains no variable)
- `1/1` matches any modifier applied to the left of the compound it modifies. For instance, it matches `N/N`, `S/S`, `(N/N)/(N/N)`, but not `S/N` nor `N\N`.
- `1` matches everything.
- `1/2` matches any type whose root operation is a forward slash.
- `N\(S/1)` matches the types `N\(S/N)`, `N\(S/S)`, `N\(S/(N/N))`, and so on.

3 Tutorial

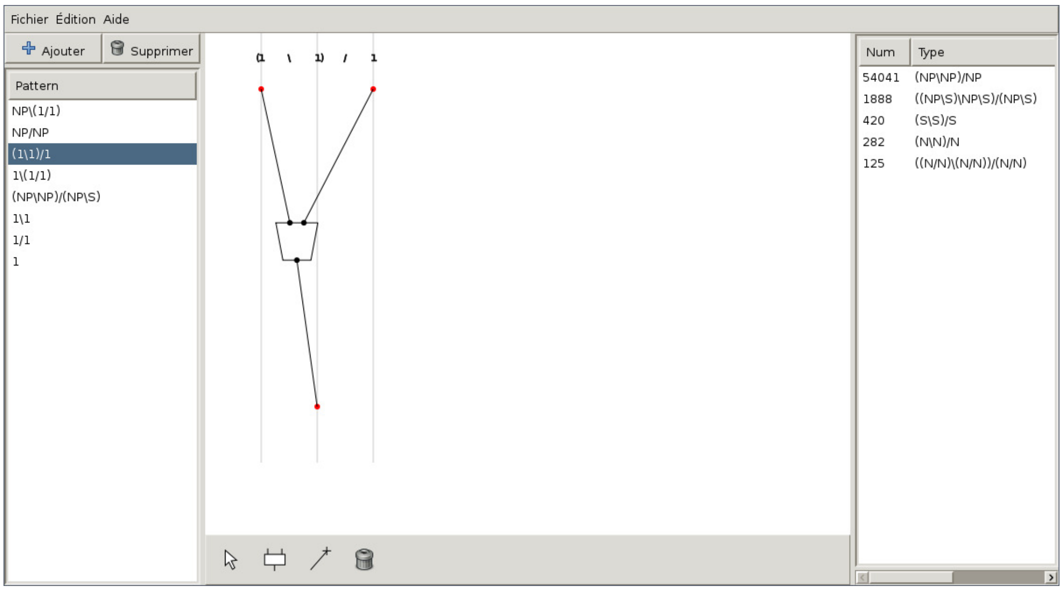
3.1 The graphical interface

Open Yanker by typing `yanker` in a terminal. A window opens and it is split vertically in three areas. From left to right, we have

1. The list of type skeletons, i.e. the polymorphic types to which we want to assign a string diagram representing the meaning.

2. The drawing area where we can draw our string diagrams. The string diagram currently being displayed corresponds to the skeleton selected in the list on the left.

3. The list of types we would like to cover. By default, this is filled with the types occurring in the CCGbank, sorted by decreasing number of occurrences. These types have been converted from Steedman notation to Lambek notation.



Pattern
NP\(1/1)
NP/NP
(1\1)/1
1\(1/1)
(NP\NP)/(NP\S)
1\1
1/1
1

Num	Type
54041	(NP\NP)/NP
1888	((NP\S)\NP\S)/(NP\S)
420	(S\S)/S
282	(N\N)/N
125	((N/N)\(N/N))/(N/N)

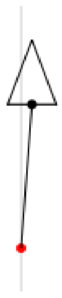
Our goal is that each of these types on the right is matched by a skeleton eventually, and hence that every type has an associated string diagram.

In the default state, there is only one skeleton, called 1. Hence it matches everything. We can give it a default string diagram, which should be simply a vector. To do so, select the unique type skeleton on the left. Then, click the *New Node* button, which looks like this: .

3.2 Our first diagram

Move your mouse to the drawing area. You can see that your mouse has grabbed a new node, that you can put anywhere you want with a click. You should put it above the red dot. The node has a funny shape: it is because by default, it has neither inputs nor outputs, hence it acts as a scalar in the category. Using the conventions introduced by Bob Coecke in analogy to the Dirac notation, we draw a scalar as a rotated rectangle, or more precisely as two concatenated triangles.

The red dot under our node is a *gate*. As it is drawn at the bottom, it corresponds to an output. The type associated with this gate is drawn on the same grey line, at the top. Now we need to connect the node to the gate. To do so, click the *New Edge* button, which looks like this: . Now, click on the gate, and then on the lower part of our node. This creates a link between the node and the gate. Horray, we have defined the meaning of our type skeleton!



But with this only type skeleton, we have not given any particular structure to word meanings.

3.3 More type skeletons

Let us add a new type skeleton. Click the *Add* button above the skeleton list. This gives you a text input where you can input a type skeleton, for instance `1 / 1`. Confirm the addition.

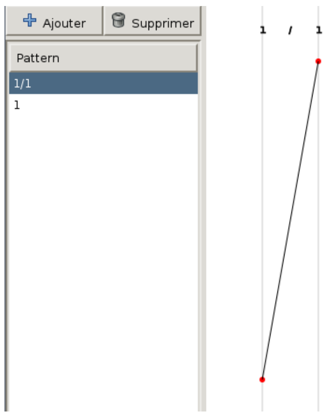
You will notice that the new type skeleton has been added at the beginning of the list. The order of that list matters, because the meaning of a type is defined by the graph associated to the first skeleton it matches. So we want the *1* type skeleton to be the last one in this list, so that types can match more specific type skeletons.

When you select one of the type skeletons, you will notice that the list of types on the right changes: the new types are the ones matched by the current type skeleton (and not matched by any other type skeleton higher up in the list).

3.4 A default meaning for `1/1`

The type skeleton `1/1` matches any modifier taking its argument to its right, and as you can see, there are many cases of such words. One simple baseline for all these modifiers would be to assign the identity as a meaning: with this baseline, there is nothing to learn, and the information flow is preserved.

Click the `1/1` skeleton and draw an edge between the gate at the top and the gate at the bottom. The gate at the top represents the input of our diagram (where the meaning of the argument comes from) and the gate at the bottom represents the output (where the meaning of the phrase goes out).



For now, this is the end of the tutorial, as we have roughly explored all the features of this current version. But you can still explore the possiblilities with other type skeletons and see what types from the CCGbank you get. Not that not all the types from the CCGbank appear though, because the current list has been cut for some reason.

A Installation

A.1 From the source code

The source code of `yanker` is available on `http://github.com/wetneb/yanker` and can be retrieved using Git. It is written in Haskell. To compile it, the simplest way is probably to use `cabal`, the standard build manager for Haskell programs. On Linux, type the following commands in a terminal, in the directory where the source code was fetched:

```
cabal configure
cabal build
```

It can be installed using the `cabal install` command. Note that it usually not necessary to run this command with superuser rights: `yanker` will be installed in `/.cabal/bin`. Be sure to add this path to your `$PATH` or to type the path in full when you run the program.

A.2 Using a Debian package

Debian packages have been compiled for both `i386` and `amd64` architectures. These can be found on the webpage of the program, `http://github.com/wetneb/yanker`.

References