

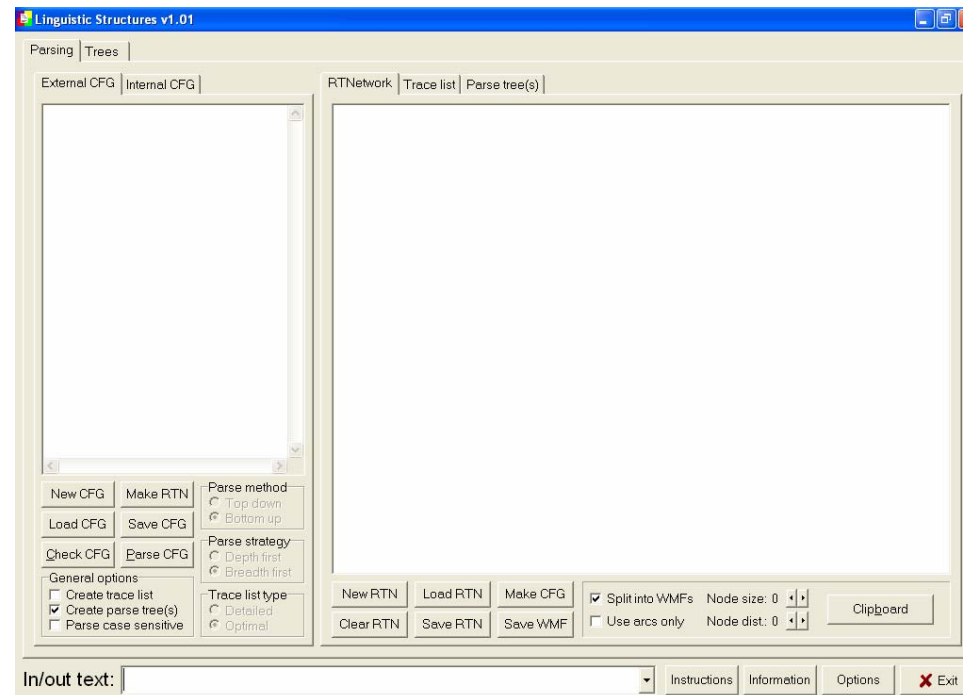
Testing a phrase structure
grammar with
Jürgen Reischer's
phrase structure program

(1) Download the program

- The program is available either directly from Jürgen Reischer's website (www.lingua-ex-machina.de) or from the local copy kept on our course website
- Copy this file to your own computer
- Unzip it (if you do not know how to do this, ask someone who does!) so that you have a single executable file, called "**LingStr.exe**"

(2) Start the program

- Click twice on the file “LingStr.exe”
- This should bring up a window looking like this:



(3) Typing in a simple grammar

The screenshot shows the 'Linguistic Structures v1.01' application window. The interface includes a menu bar with 'Parsing' and 'Trees', and sub-menus for 'External CFG', 'Internal CFG', 'RTNetwork', 'Trace list', and 'Parse tree(s)'. A central text area contains two instructional boxes: a yellow one on the left and a grey one on the right. A red arrow points from the grey box to the 'In/out text:' input field at the bottom. The bottom of the window features a toolbar with buttons for 'New CFG', 'Load CFG', 'Check CFG', 'Make RTN', 'Save CFG', 'Parse CFG', 'New RTN', 'Load RTN', 'Clear RTN', 'Save RTN', 'Make CFG', 'Save WMF', 'Split into WMFs', 'Use arcs only', 'Node size', 'Node dist', 'Clipboard', 'Instructions', 'Information', 'Options', and 'Exit'.

The simplest way of getting a grammar into the program is to type into **this window.**

A grammar is just as we have seen in class, but it uses '=' instead of the arrow: e.g., you must write $S = NP VP$ instead of $S \rightarrow NP VP$

When you have typed in your grammar, you can see how it analyses sentences by typing your sentence in the window In/out text at the bottom.

In/out text:

(4) Example

- You could type in the following simple example grammar

S = NP VP

NP = Det N

VP = V NP

V = <"ate", "drank">

N = <"man", "cat">

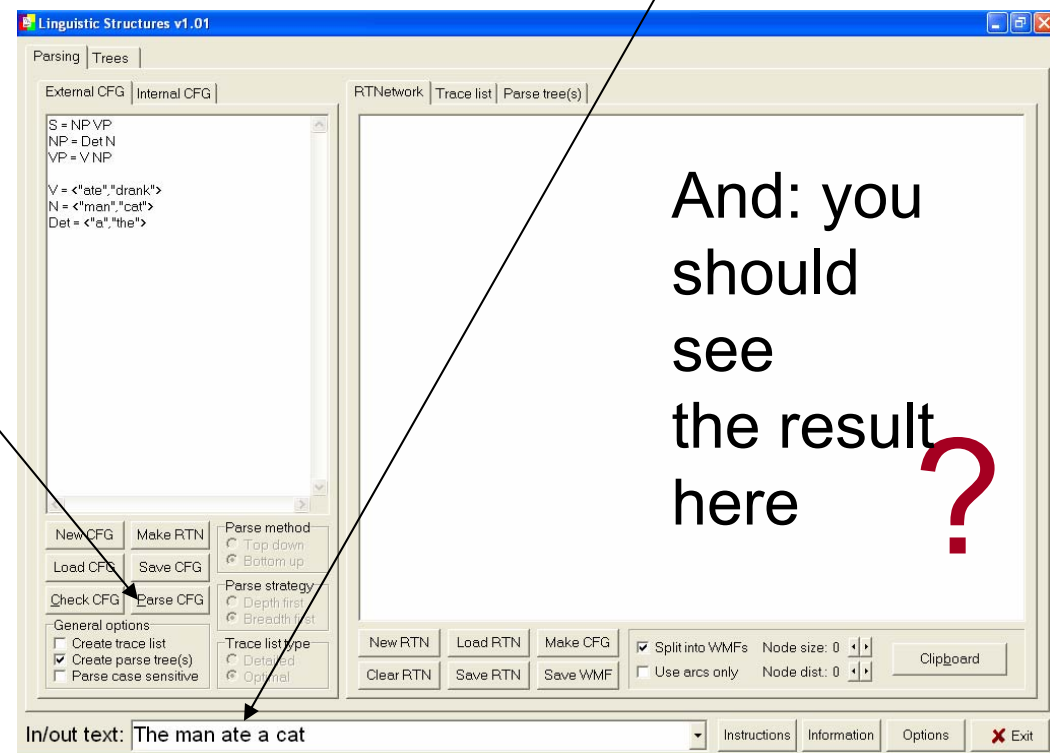
Det = <"a", "the">

(example-1)

- Note how words are introduced into the grammar: a list of alternatives within '<' and '>' brackets.

(4) Example

- After typing in the grammar, you can type in a sentence in the bottom window marked **In/out text** and press the **Parse CFG** button.



(5) A **parse tree** (the result)

The screenshot displays the 'Linguistic Structures v1.01' application window. The interface is divided into several sections:

- External CFG:** Contains the following rules:
 - S = NP VP
 - NP = Det N
 - VP = V NP
 - V = <"ate", "drank">
 - N = <"man", "cat">
 - Det = <"a", "the">
- Parse tree(s):** Shows a hierarchical parse tree for the sentence "The man ate a cat". The root node is S, which branches into NP and VP. The first NP branches into DET ("the") and N ("man"). The VP branches into V ("ate") and another NP. This second NP branches into DET ("a") and N ("cat").
- Parse method:** Radio buttons for Top down and Bottom up.
- Parse strategy:** Radio buttons for Depth first and Breadth first.
- Trace list type:** Radio buttons for Detailed and Optimal.
- General options:** Checkboxes for Create trace list, Create parse tree(s) (checked), and Parse case sensitive.
- Buttons:** New CFG, Load CFG, Check CFG, Make RTN, Save CFG, Parse CFG, Save tree, Transfer.
- Status:** Number of trees: 1, Current tree: 1.
- Bottom bar:** In/out text: The man ate a cat, with buttons for Instructions, Information, Options, and Exit.

(6) A failed parse

- If you give a sentence that the grammar cannot cover, then you get an error message. For example, if you try the sentence: '**the man ate**', the program will tell you:



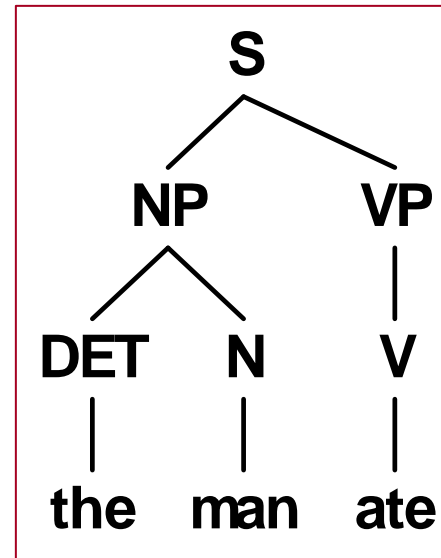
(7) Extending the grammar

- In order to cover the sentence 'the man ate', we have to change the grammar. This means, empirically, that we have discovered that our grammar is not sufficient (no surprise!)
- An appropriate extension to the grammar would be to add a further rule:

$$VP = V$$

(7) Extending the grammar

- With the extra rule, the program can produce a tree:




- A more concise way of writing the same thing is to use the rule: **VP = V [NP]**
The square bracket means *optional*.

Some more shorthand...

- $[X]$: symbol can occur **zero or one time**
(optional element)
- $\{X\}$: symbol can occur **0 or more times**
(repeatable Element)
- $\langle X, Y, \dots \rangle$: **at least one** of the given elements must occur
- (X, Y, \dots) : one of the elements **may** occur

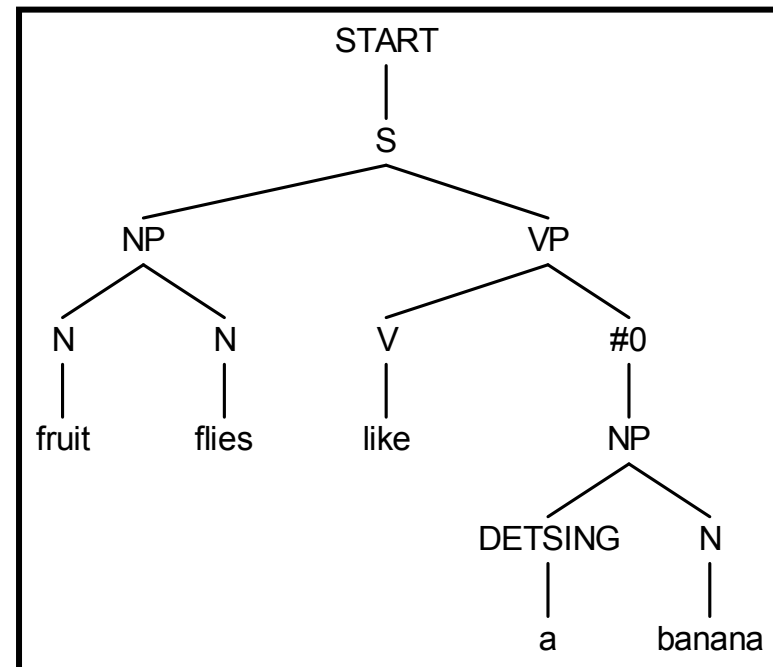
(8) Working with grammars

- When grammars get more interesting, it is better to write them first into a text file.
- Then you can save them and load them into the program whenever you want, without retyping, by using the  button.
- Your grammar files should have names like `'mygrammar.cfg'`
- The `cfg` stands for 'context-free grammar', which is the technical name for this kind of phrase structure grammar.

(9) Working with grammars

- For example, lets look now at a standard linguistic example
- What is the structure of the sentences:


– time flies like an arrow



Load CFG

fruit-flies.CFG

(10) Working with grammars

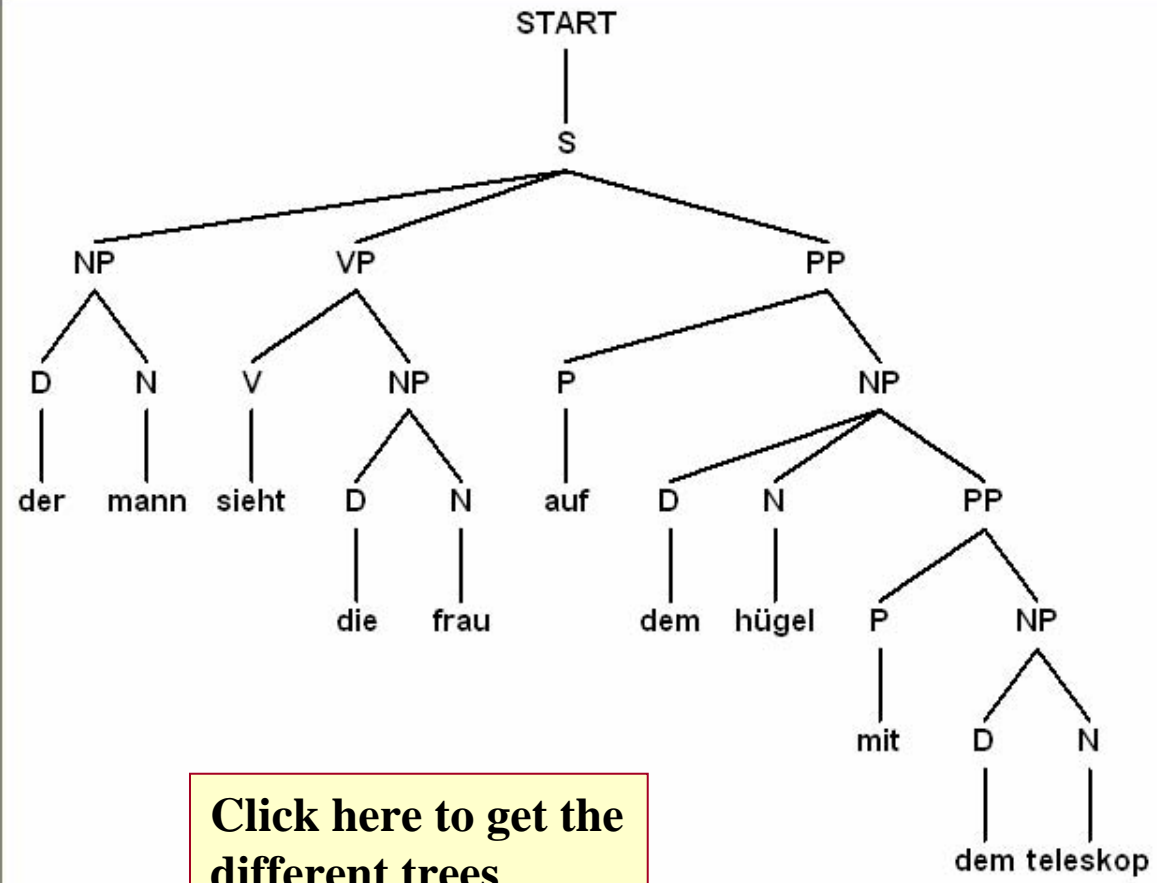
- For another example, try the example grammar given by Reischer called 'ambig2.cfg' given on the course website.
- Save this grammar somewhere on your computer and then load it with 
- You will see that if you give the test sentence: “Der Mann sieht die Frau auf dem Hügel mit dem Teleskop“ you get many result trees.
- Can you find the different meanings that each tree has?

Parsing | Trees

External CFG | Internal CFG

```
// Testsatz "Der Mann sieht die Frau auf  
// "dem Hügel mit dem Teleskop."  
  
// Startsymbol  
Start = S  
  
// nichtterminale Symbole  
S = NP VP [PP] ["."]  
NP = [D] N [PP]  
VP = V NP [PP]  
PP = P NP  
  
// terminale Symbole  
N = <"mann","frau","hügel","teleskop">  
V = "sieht"  
D = <"der","die","dem">  
P = <"auf","mit">
```

RTNetwork | Trace list | Parse tree(s)



Click here to get the different trees



New CFG | Make RTN | Parse method
 Top down
 Bottom up
Load CFG | Save CFG
Check CFG | Parse CFG
Parse strategy
 Depth first
 Breadth first
General options
 Create trace list
 Create parse tree(s)
 Parse case sensitive
Trace list type
 Detailed
 Optimal

Save tree | Transfer | Number of trees: 6 | Current tree: 3

In/out text:

Instructions | Information | Options | Exit