



# Sanchay Syntactic Annotation Interface

**Anil Kumar Singh** <[anil@research.iiit.ac.in](mailto:anil@research.iiit.ac.in)>  
**LTRC, IIIT, Hyderabad, India.**



# Introduction

- Sanchay: A collection of APIs and tools for NLP
- Includes GUI based interfaces for annotation
  - Like Syntactic Annotation Interface



# Syntactic Annotation Interface

- Can view or edit:
  - POS tagging
  - Chunking
  - Feature structures of words or chunks
    - Morphological information
    - Kaaraka or dependency relationships
- Many more things in future (hopefully)



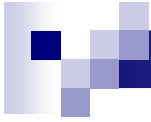
# Overview

- System requirements
- Installation
- Setting up the tasks
- Starting the interface
- Viewing annotation
- Editing annotation



# System Requirements

- JDK-1.5 (or JRE-1.5) installed on your system
- Preferably >1 GHz processor and >256 MB RAM
- You may like to specify the memory allocated for JVM (see later), if you have more RAM
- Tested on Linux and Windows



# Installation

- Untar the Sanchay.tgz file somewhere on your system
  - Or unzip the Sanchay.zip file
- There should be a **Sanchay** directory
- That's all for installation!



# Setting Up the Tasks-I

- Annotation performed on 'tasks'
- Task data located inside the **Sanchay/workspace/syn-annotation** directory
- Tasks grouped into task-sets
- One directory for each task-set inside the **syn-annotation** directory



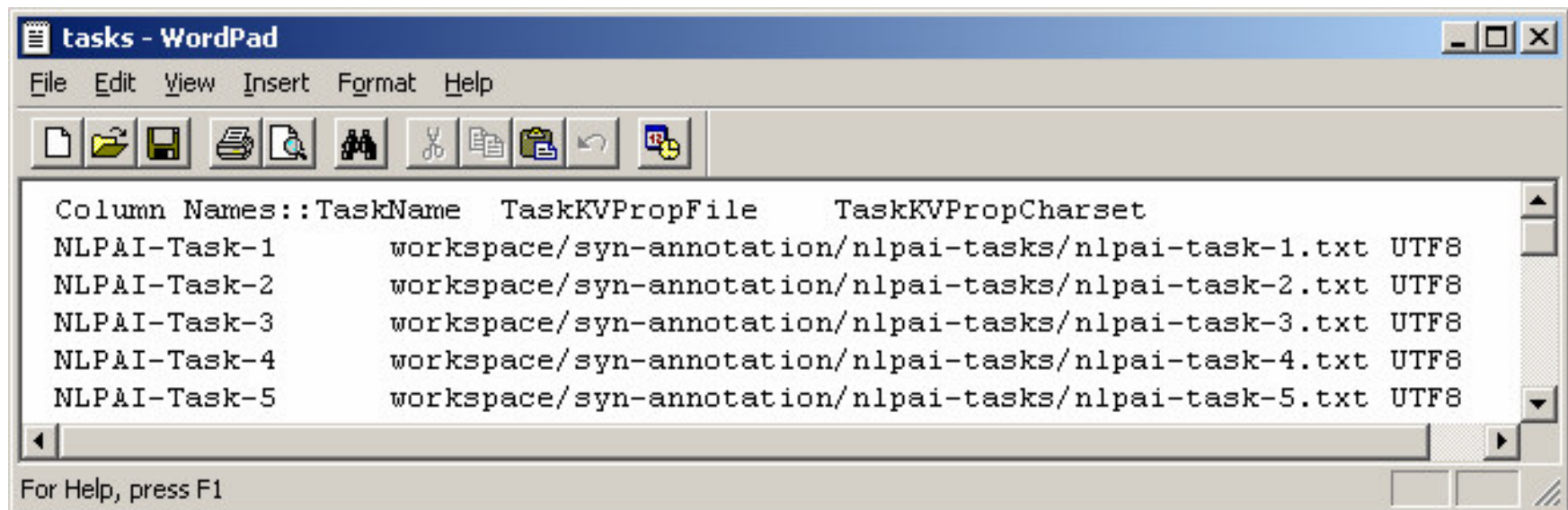
# Setting Up the Tasks-II

- E.g., an **nlpai-tasks** directory for NLPAL ML contest tasks
- Each 'task' represents a part of the (to be) annotated corpus
  - A 'story' or a part of the 'story'
- A list of currently loaded tasks is kept in a file **tasks.txt** in **syn-annotation** directory
  - The tasks displayed on the interface



# Setting Up the Tasks-III

- **tasks.txt** has tab separated fields
  - Task name, task properties file path and task properties file charset



tasks - WordPad

File Edit View Insert Format Help

Column Names::TaskName TaskKVPPropFile TaskKVPPropCharset

NLP AI-Task-1	workspace/syn-annotation/nlpai-tasks/nlpai-task-1.txt	UTF8
NLP AI-Task-2	workspace/syn-annotation/nlpai-tasks/nlpai-task-2.txt	UTF8
NLP AI-Task-3	workspace/syn-annotation/nlpai-tasks/nlpai-task-3.txt	UTF8
NLP AI-Task-4	workspace/syn-annotation/nlpai-tasks/nlpai-task-4.txt	UTF8
NLP AI-Task-5	workspace/syn-annotation/nlpai-tasks/nlpai-task-5.txt	UTF8

For Help, press F1



# Task Properties Files-I

- Each task has a task properties file
- Key-value pairs
- Three relevant keys for setting up new tasks
  - TaskName
  - SSFCorpusStoryFile
  - TaskPropFile

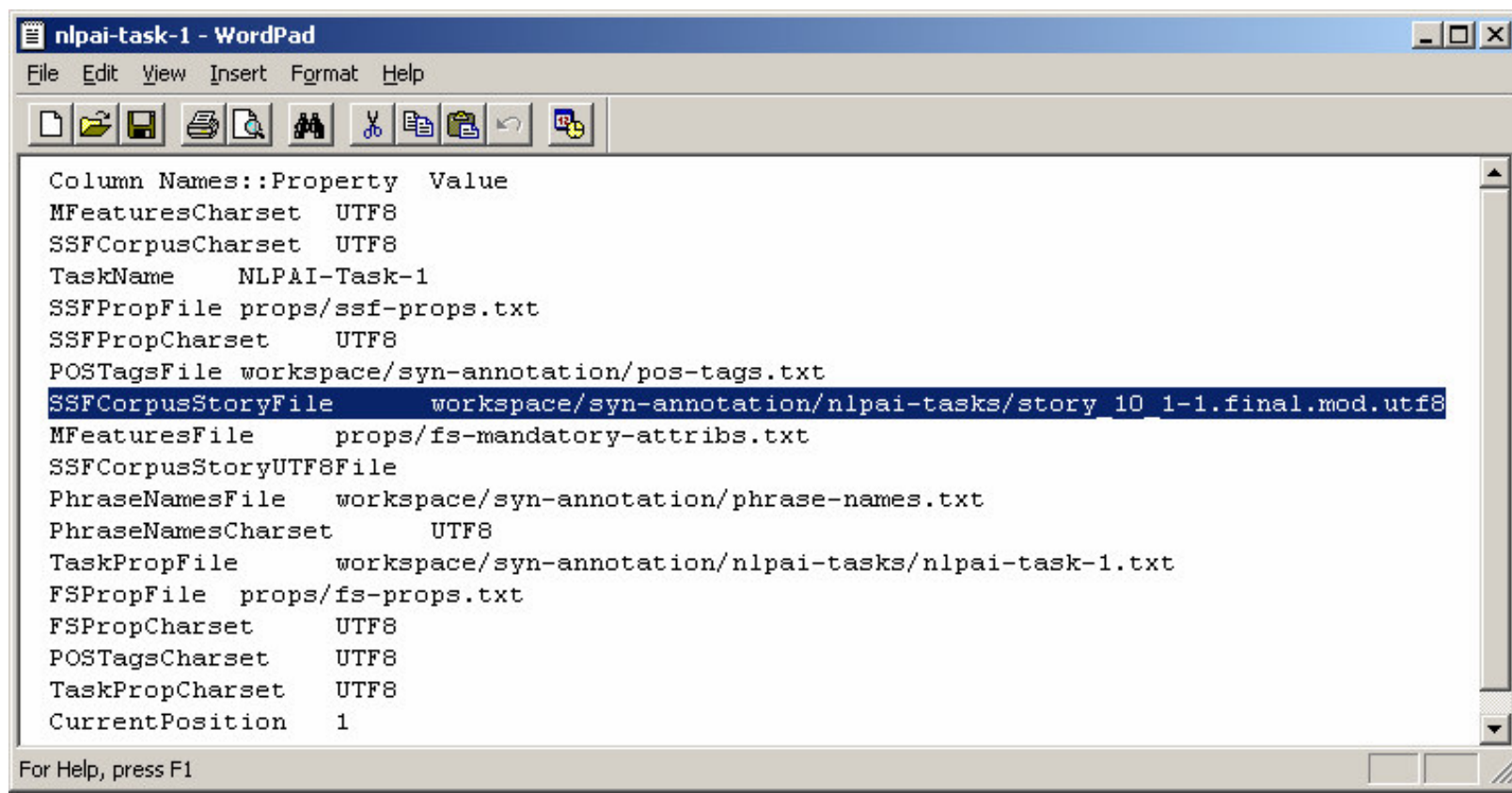


# Task Properties Files-II

- E.g., for nlpai-task-1
  - There is an entry in the **tasks.txt**
  - There is a task properties file, say, **nlpai-task-1.txt**
  - There is the corpus 'story' part, say, **story\_10\_1-1.final.mod.utf8**
    - The actual corpus data file

# Task Properties Files-III

- **nlpai-task-1.txt** has key-value pairs



```
Column Names::Property Value
MFeaturesCharset UTF8
SSFCorpusCharset UTF8
TaskName NLPai-Task-1
SSFPropFile props/ssf-props.txt
SSFPropCharset UTF8
POSTagsFile workspace/syn-annotation/pos-tags.txt
SSFCorpusStoryFile workspace/syn-annotation/nlpai-tasks/story_10_1-1.final.mod.utf8
MFeaturesFile props/fs-mandatory-attrs.txt
SSFCorpusStoryUTF8File
PhraseNamesFile workspace/syn-annotation/phrase-names.txt
PhraseNamesCharset UTF8
TaskPropFile workspace/syn-annotation/nlpai-tasks/nlpai-task-1.txt
FSPropFile props/fs-props.txt
FSPropCharset UTF8
POSTagsCharset UTF8
TaskPropCharset UTF8
CurrentPosition 1
```

For Help, press F1



# Encoding or (Font) Disclaimer

- Sanchay currently assumes that you will only be using corpus files in UTF-8 (8-bit Unicode) encoding
- You might also use files in some ASCII based notation like WX or Roman
- Sorry, but ISCI won't work currently
  - Let's hope it will soon
- So you might need to convert corpus files to UTF-8 or WX etc.



# Corpus Data Format Disclaimer

- The interface only accepts data in correct SSF format
  - Actually, raw data is also accepted, but is converted into SSF (see the next slide)
- SSF is Shakti Standard Format
  - A CFG (Context Free Grammar) tree like format with the addition that every node can have feature structures for carrying morphological and dependency relation information, among other things



# **\*WARNING\* - Using Raw Data**

- The task data files can also be in raw form
  - **ONE SENTENCE ON ONE LINE**
- If you set up a task with raw data
  - You get a message about raw data
  - You are asked to make sure each sentence is on a separate line
  - You are presented with a simple editor for this
- **IF YOU DON'T BREAK UP SENTENCES BEFORE YOU START, YOUR DATA MAY BE UNUSABLE**

# SSF

## Shakti Standard Format

		chunk-name	
		/	
1	((	NP	
1.1	children	NNS	<af=child,n,m,p,3,0,,>
	)		
			root      pers \
	lex	pos	cat   number case
			gender
2	((	VG	
2.1	are	VBP	<af=be,v,m,p,3,0,,>
2.2	watching	VBG	<af=watch,v,m,s,3,0,, /aspect='PROG'>
	)		
3	((	NP	
3.1	some	DT	<af=some,D,m,s,3,0,,> <af=some,det,m,s,3,0,,>
3.2	programmes	NNS	<af=programme,n,m,p,3,0,,>
	)		
4	((	PP	
4.1	on	IN	<af=on,p,m,s,3,0,,> <af=on,n,m,s,3,0,,>
4.1.1	((	NP	
4.1.2	television	NN	<af=television,n,m,s,3,0,,>
	)		
	)		
5	((	PP	
5.1	in	IN	<af=in,p,m,s,3,0,,> <af=in,D,m,s,3,0,,>
5.2	((	NP	
5.2.1	the	DT	<af=the,det,m,s,3,0,,>
5.2.2	house	NN	<af=house,n,m,s,3,0,,> <af=house,v,m,s,3,0,,>
	)		
	)		





# Corpus Data File-I

- Everything between **<Story></Story>** tags
- Story can have an id: **<Story id="1">**
- A sentence between **<Sentence></Sentence>** tags
- Sentences must have an id: **<Sentence id="1">**



# Corpus Data File-II

- Four fields in SSF format
  1. The node id (in way deprecated since the SSF API works on the basis of brackets)
  2. Starting (“(”) or ending (“)”) brackets for chunks or the word for a lexical item
  3. The chunk name for a chunk or the POS tag for a lexical item
  4. The feature structure for the node (chunk or lexical item)

# Corpus Data File-III

- A sample data file
  - story\_35\_1-2.final.mod.utf8

```
<Story id="11">
<Sentence id="31">
0  ((  VG
0  समझता VFM
0  हूँ      VAUX
0  ))      VG
0  ((      NP
0  आप     PRP
0  ))      NP
0  ((      NP
0  इस     PRP
0  बात    NN
0  से      PREP
0  ))      NP
0  ((      VG
0  सहमत   JVB
0  होंगे    VFM
0  ))      VG
0  ((      BLK
0  और     CC
0  ))      BLK
0  ((      NP
0  इसे     PRP
0  ))      NP
0  ((      VG
0  पारित    JVB
0  करेंगे    VFM
0  .         SYM
0  ))      VG
</Sentence>
```

Page 1    Sec 1    1/41    At 4.9"    Ln 24    Col 8



# Starting the Interface-I

- On Linux

1. On the console, go to the Sanchay directory
2. Run the shell script  
**sh run-syntactic-annotation.sh**

- On Windows

1. Go to the Sanchay directory
2. Double click on  
**run-syntactic-annotation.bat**  
Or **run-syntactic-annotation** (if DOS extensions are not displayed)

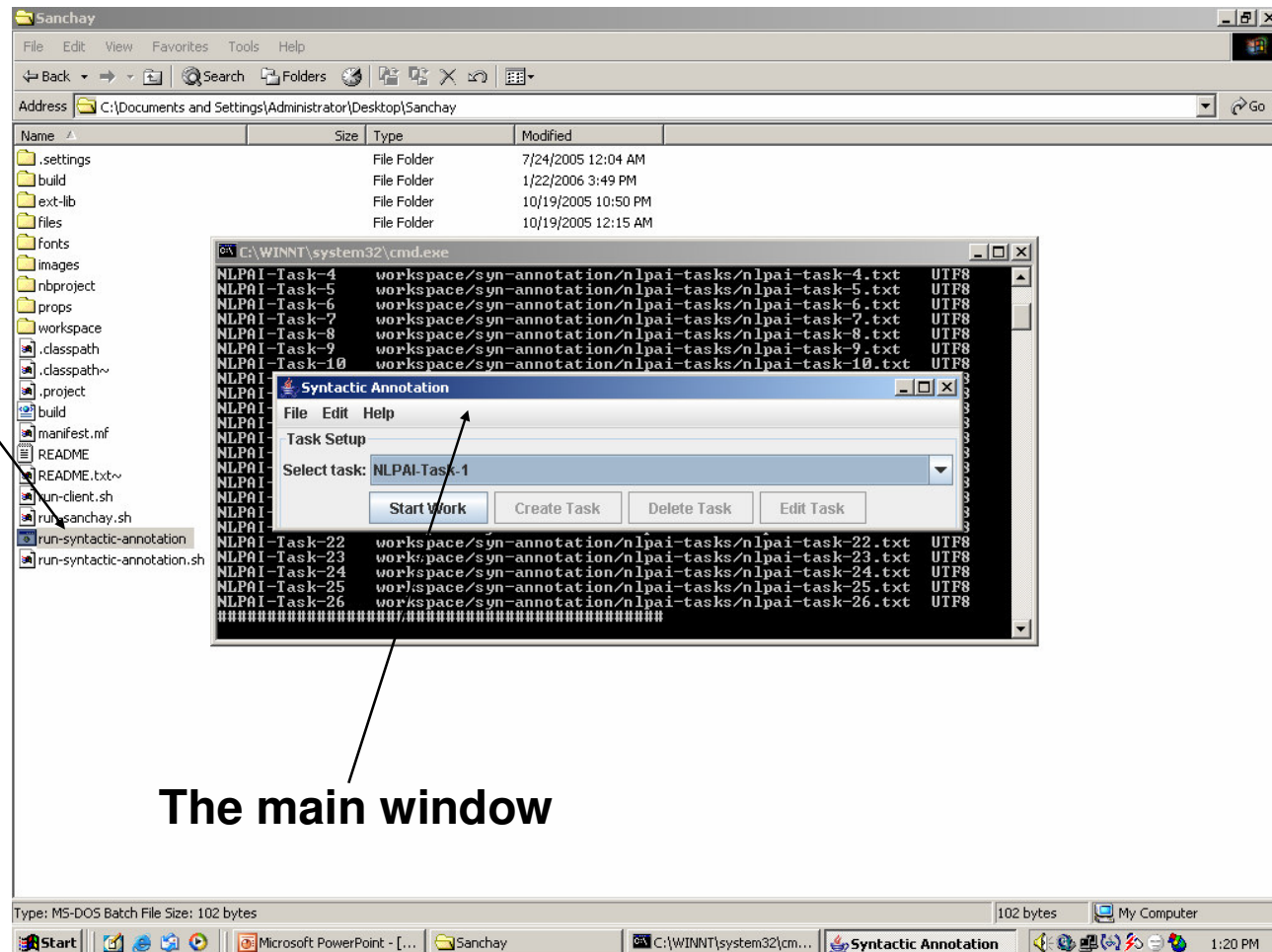


# Starting the Interface-II

- You will see the main interface window
- You can select the task you want to view/edit from here
- After selecting the task, click on the **Start Work** button
- You will see the work window where one sentence from the task data file will be displayed
- You can create your own annotation

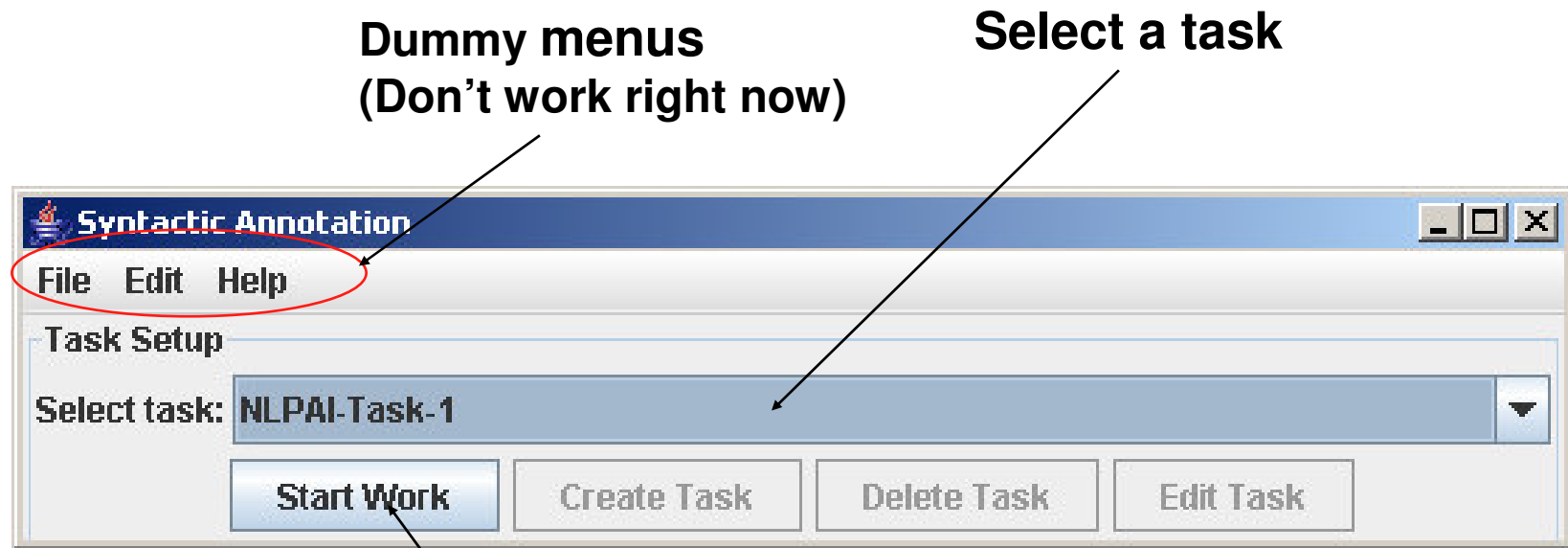
# Starting the Interface-III

Double click



The main window

# The Main Window



Dummy menus  
(Don't work right now)

Select a task

Click to start working



# Working Directory Disclaimer

- Currently Sanchay will work only if you start it from the **Sanchay** directory
- Thus, you can't create shortcuts somewhere else unless you take care that the current directory gets changed to **Sanchay** before the above script is run



## For chunking

## You should see this ‘work’ window

## The (partially) annotated sentence as a tree

## For POS tags and chunk names

## Comments

- **Navigation among sentences**

**Save etc.**



# Navigation

- At a time only one sentence is displayed in the work widow
- Using the buttons in the navigation bar, you can **go to** any sentence in the task
  - First, Previous, Next, Last
  - Or any specific sentence by selecting its number from the list (combo box)



# POS Tagging

- Select (say, by clicking) a lexical item (word) in the tree area
- You will see the list of possible POS tags in the **Node label** list (combo box)
- From the **Node label** list, select a tag
- The new POS tag for the word will be displayed in the tree area



# Chunks vs. Phrases

- The interface allows arbitrary nesting
  - PP inside an NP, which is itself inside an NP
- But by chunking we mean only one level
  - A chunk can be NP, VG etc. and can only contain lexical items inside it, not chunks
  - No recursion
  - NP -> The book on the table
    - But not
      - NP -> NP PP **or** PP -> Prep NP



# Phrases or Recursion Not Allowed

- It is assumed that only chunks will be annotated
- Everything (for the time being) will be in terms of
  - Words, POS tags, chunks, feature structures



# Forming Chunks

- You want to chunk “*The red book*”
- Select these words together in order
- You can do this by pressing the Ctrl or Shift and clicking on the words
- Click on **Join Nodes** button
- A new node will be created and the words you selected will become child nodes of this node



# Naming Chunks

- Select the chunk node (say, NP)
- You will see the list of possible chunk names in the **Node label** list (combo box)
- From the **Node label** list, select a chunk name
- The new chunk name for the chunk will be displayed in the tree area



# Removing Chunks

- Now suppose a chunk is wrongly formed
- Select the chunk node (say, NP)
- Click on Delete Layer button
- The chunk node will be removed
- The words will move up in the tree





# Forming and Removing

- For any rearrangement of words into chunks, you will have to use combinations of **form** and **remove** chunk operations
- Remember that words to be combined into a chunk should be on the same level
- You may have to bring them on the same level by removing those chunks they are currently parts, if any



# Combining Chunks

- You want to combine two chunks
- Remove the first chunk as explained above
- Remove the second chunk
- Combine the words of the two chunks into one chunk



# Splitting Chunks

- You want to split one chunk into two
- Remove the chunk
- Select some words and form a new chunk
- Select the rest of the words into another chunk



# Feature Structure Annotation-I

- Beyond tagging and chunking
- Every node (word or chunk) in the tree can have a feature structure



# Feature Structure

- A feature structure has attribute value pairs
- Some attributes may be mandatory
- The attribute value can itself be a feature structure
  - Usually it will just be a string
  - The current interface assumes only string values

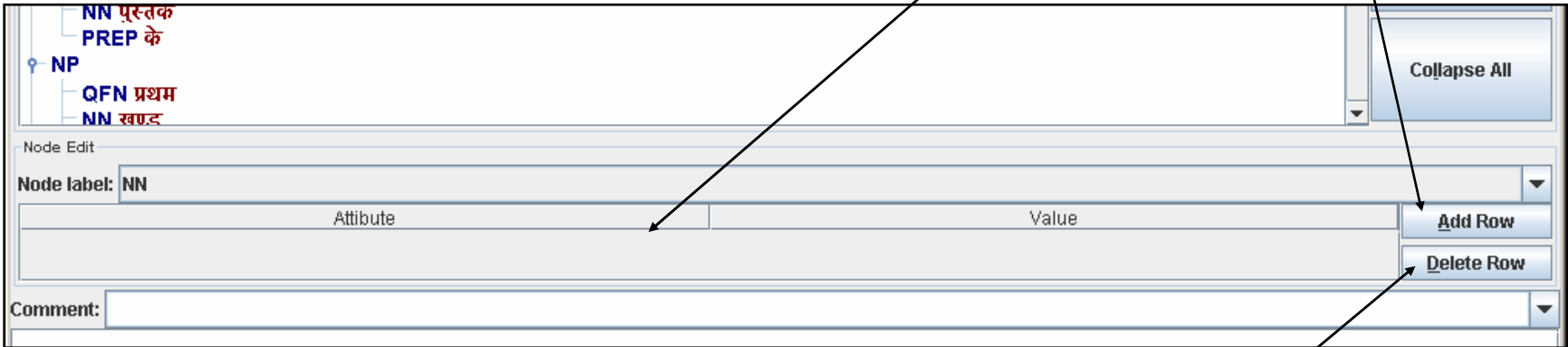
# Feature Structure Annotation-II

- When you select a node in the tree, you will see a feature table showing the attributes and values

☐ May be empty

Feature table

Add attribute



The screenshot shows a software interface for Feature Structure Annotation. On the left, a tree view displays a hierarchy of nodes: 'NP' (selected), 'NN प्रस्तक', 'PREP के', 'QFN प्रथम', and 'NN खण्ड'. Below the tree is a 'Node Edit' section with a 'Node label' field containing 'NN'. To the right of the node label is a table with two columns: 'Attribute' and 'Value'. Below this table are 'Add Row' and 'Delete Row' buttons. At the bottom left is a 'Comment' field. On the right side of the interface, there is a 'Collapse All' button and a vertical scroll bar. Three arrows point to specific elements: one from 'Feature table' to the table header, one from 'Add attribute' to the 'Add Row' button, and one from 'Delete attribute' to the 'Delete Row' button.

Attribute	Value
-----------	-------

Comment:

Delete attribute



# Adding Attributes

- Select the node for which you want to add an attribute
- You will see the attribute table below the **Node label**
- Click on the **Add Row** button
- There will be a new row representing a new attribute in the feature table



# Naming Attributes

- In the feature table, select the attribute (which may have no name if it was just created)
  - By clicking on the table cell under the **Attribute** column
- Type the new name and press enter
- When you click on a node other than the currently selected one in the tree, the new attribute will appear near the node





# Setting/Changing Attribute Values

- In the feature table, select the attribute value (which may be empty)
  - ☐ By clicking on the table cell under the **Value** column
- Type the new value and press enter
- When you click on a node other than the currently selected one in the tree, the new attribute value will appear near the node

# Attribute-Value Pairs

Node Edit

Node label: NP

Attribute	Value
drel	k1
name	2

Comment:

Add Row

Delete Row

Attribute names

Attribute values

```
SSF
├─ NP <drel=k1/name=2>
│   ├── NNP निराळा
│   └── PREP की
└─ NP
```

Displayed feature structure



# Deleting Attributes

- In the feature table, select the row for the attribute you want to delete
  - By clicking any cell in the row
- Click on the **Delete Row** button in the feature table
- When you click on a node other than the currently selected one in the tree, you can see that the attribute has been deleted



# Syntactic Annotation

- By syntactic annotation we mean marking up dependency relations
  - Say, kaaraka relations



# Kaaraka Relations-I

- The main thing is
  - Which node has what relation with which other node
- The node which 'has the relation' is given an attribute called **drel** whose value (say, **k1** or **kartaa**) will be the kaaraka relation
- The with which it has the relation is given an attribute called **name**
- The names should have unique values (across a sentence)



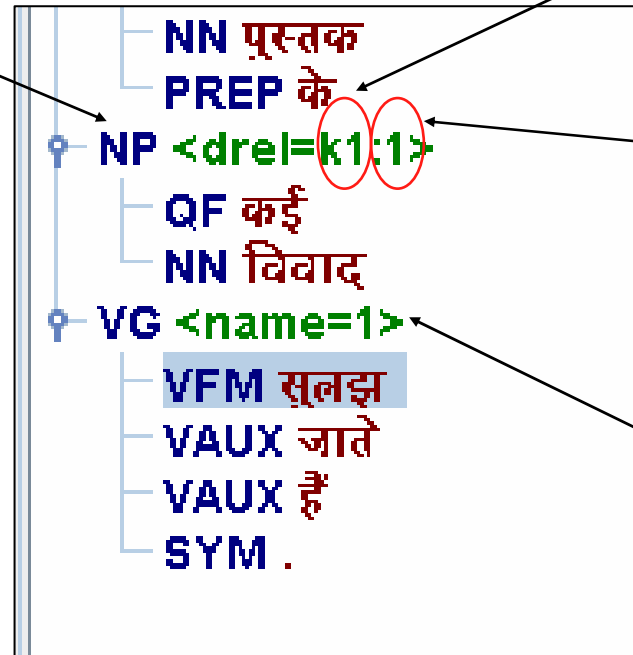
# Kaaraka Relations-II

- Name uniquely identifies a chunk with which other chunks are related
- This name is referred to in the value of the **drel** attribute of the chunk which has a relation with the named chunk
- This is done by adding the name to the value of the drel attribute, separated by a colon

# Kaaraka Relations-III

Chunk which has a relation

The kaaraka relation

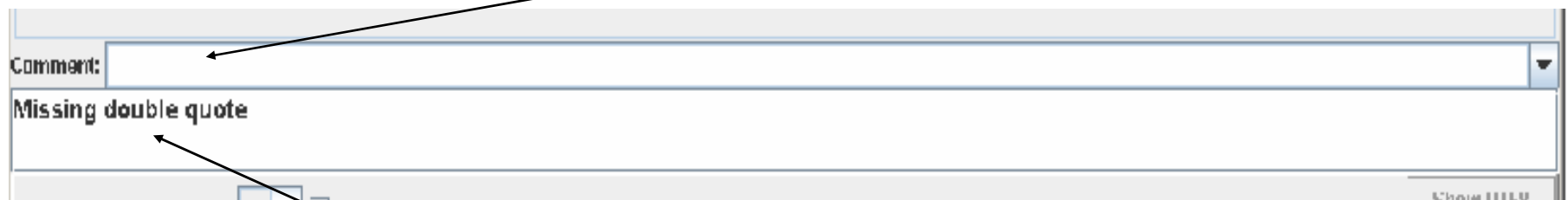


Reference to the  
named chunk

Named chunk with  
which it has a relation

# Comments

- For any sentence, if there is any (say, linguistic) issue which needs to be noted down, you can add a comment
  - Or a typing/grammatical mistake etc.



The screenshot shows a web form with a label 'Comment:' followed by a text input field. The input field contains the text 'Missing double quote'. Below the input field is a 'Show HTML' link. An arrow points from the text 'Frequent (reusable) comment' to the input field, and another arrow points from the text 'Comment' to the input field.

Frequent (reusable) comment

Comment





# Reusable Comments

- In the combo box above the comment text box, you can type comments which are needed frequently
  - So that you don't have to type them again and again
- Once you have entered them in the combo box, you can just select them later from the list (combo box) and they will appear in the comment text box



# Saving

- There are three buttons for ‘saving’
- **Save** actually saves the complete task
  - The one you will mostly use
- **Save Tree As** saves one sentence in (SSF format, text file) wherever you want
- **Save BForm As** saves the complete task in bracket form (text file) wherever you want



# Reset and Clear

- **Reset** will discard recent work and display the sentence read from the saved task
- **Clear** will clear tagging and chunking and will present the current sentence as a sequence of words
- **Reset All** will discard all the work you did after pressing **Save** and read the whole task again from the task file



# Caution

- Use **Reset**, **Clear** and **Reset All** buttons carefully



# Expand All and Collapse All

- These buttons expand or collapse the sentence tree
- Perhaps not very useful in most cases



# Bracket Form

- A tree based representation of sentence chunking or parsing
- (Possibly nested) brackets mark the chunks or phrases
  - For us, only chunks
- Allows you to see the chunking horizontally, rather than vertically (as in a tree)



# Increasing JVM Memory-I

- The memory allocated by default to JVM (heap size) may not be enough, given the RAM on your system
- Suppose you have 512MB RAM, you might want to allocate more memory to JVM to make Sanchay run faster
- Change the **run-syntactic-annotation.sh** or **run-syntactic-annotation.bat** script



# Increasing JVM Memory-II

- Use the `-Xms` option to specify the initial heap size
- Use the `-Xmx` option to specify the maximum heap size
- Example
  - `java -Xms128m -Xmx256m ...`





# Concluding Disclaimer

- **Sanchay** is at present continuously evolving
- Parts of it are being used for practical work, but the design has yet to stabilize
- There can be radical changes
- Fortunately, the data created by you will be usable even in future, as it is in a standard format



- Any comments/bugs

- Mail at

`anil@research.iiit.ac.in`