NPFL075 Practical Class 00

Jiří Mírovský

Charles University

Institute of Formal and Applied Linguistics

PML

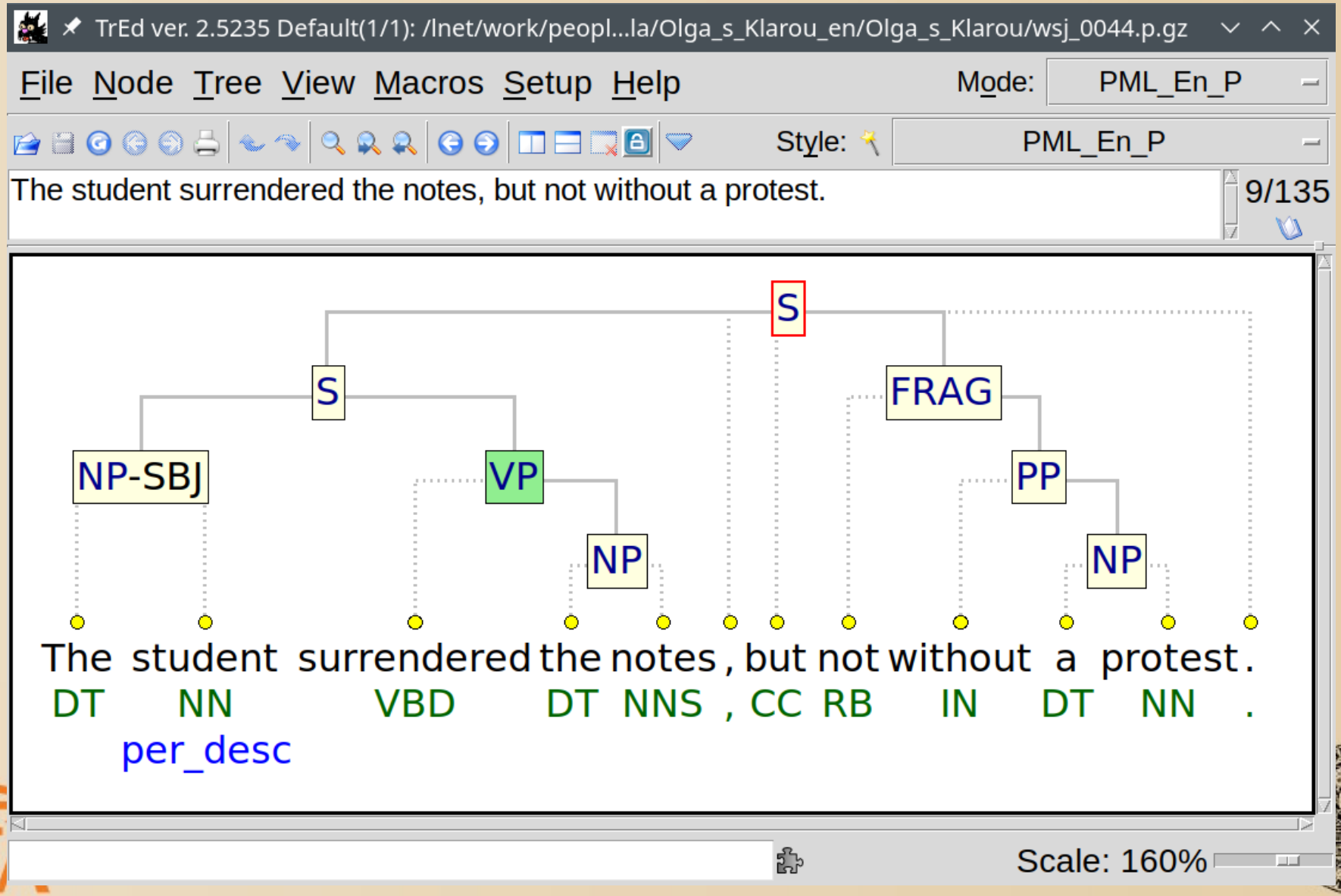
PML – **P**rague **M**arkup **L**anguage

PML is **a general format** (**XML**) for **all kinds** of linguistically annotated **treebanks**.

ÚFAL

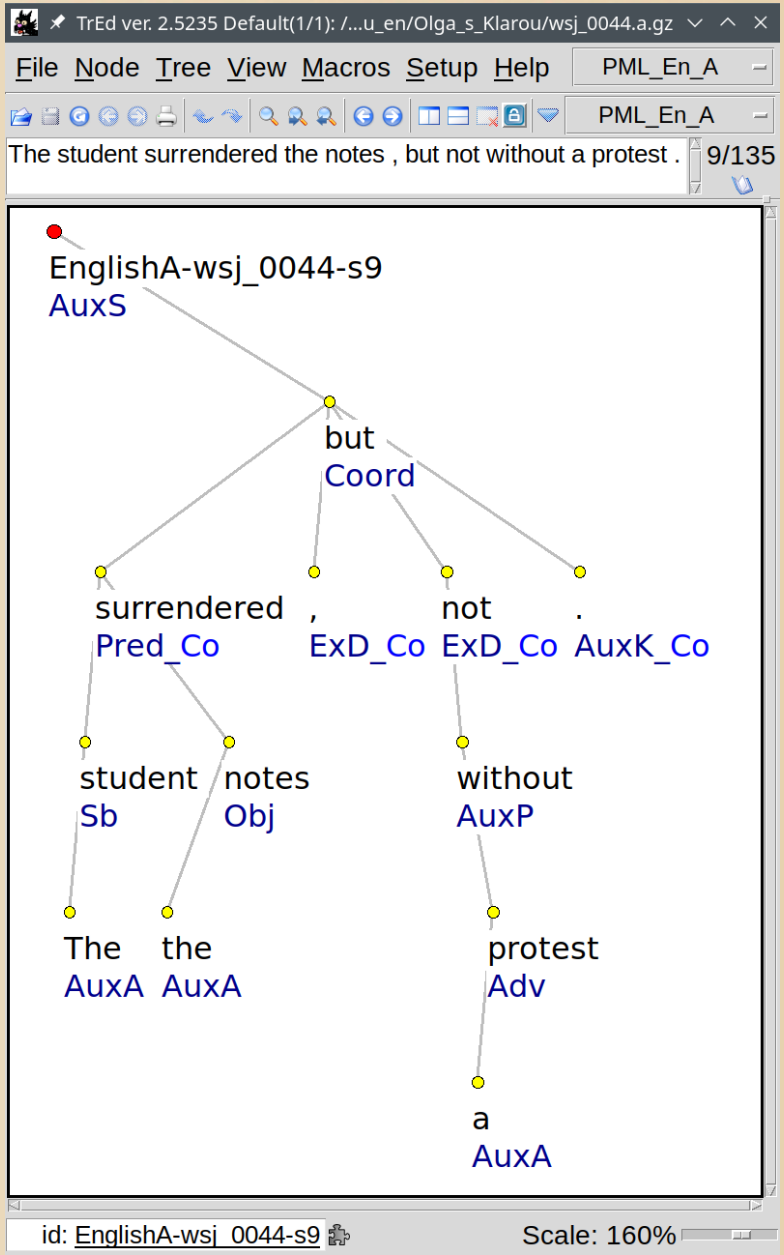
Phrase Structure Tree

Penn Treebank, file wsj0044



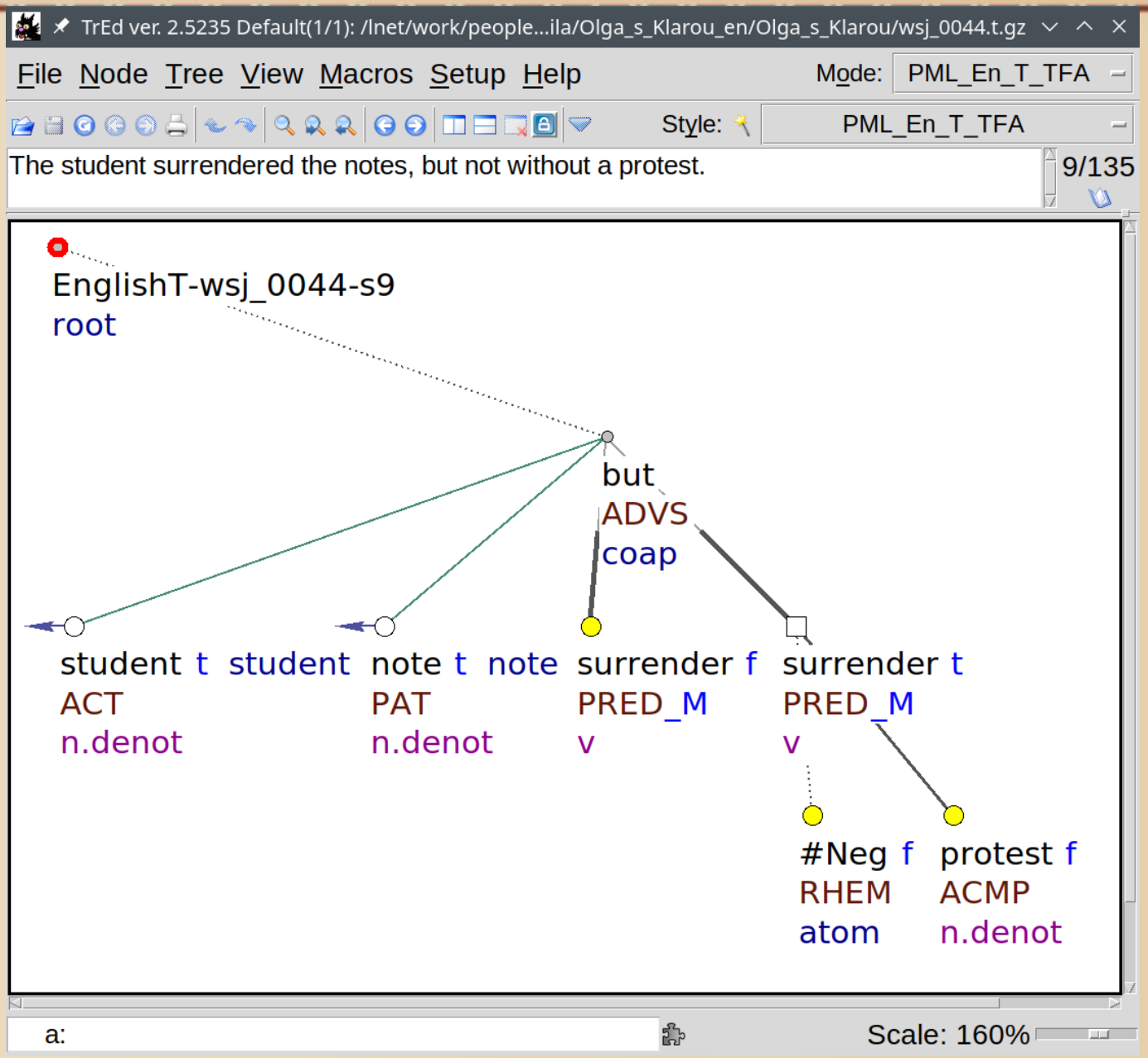
Dependency Tree

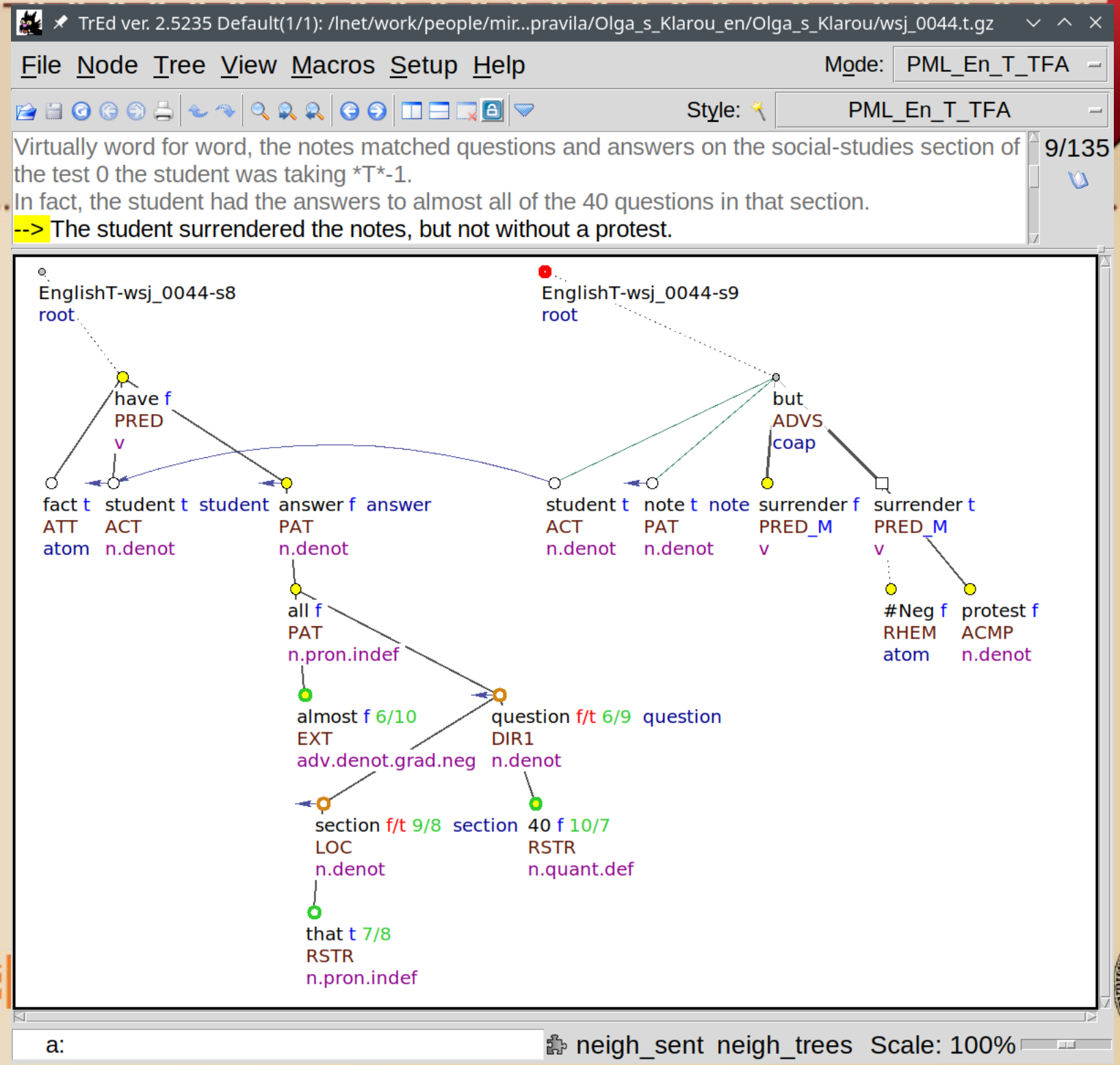
Analytical (surface syntax) layer



Dependency Tree

Tectogrammatical (deep syntax) layer




TrEd ver. 2.5235 Default(1/1): /lnet/work/people/mir...pravila/Olga_s_Klarou_en/Olga_s_Klarou/wsj_0044.t.gz
File Node Tree View Macros Setup Help
Mode: PML_En_T_TFA
Style: PML_En_T_TFA
Virtually word for word, the notes matched questions and answers on the social-studies section of the test 0 the student was taking *T*-1.
In fact, the student had the answers to almost all of the 40 questions in that section.
--> The student surrendered the notes, but not without a protest.
9/135
EnglishT-wsj_0044-s8
root
have f
PRED
v
fact t
ATT
atom
student t
ACT
n.denot
student
answer f
PAT
n.denot
answer
all f
PAT
n.pron.indef
almost f 6/10
EXT
adv.denot.grad.neg
question f/t 6/9
DIR1
n.denot
question
section f/t 9/8
LOC
n.denot
section
that t 7/8
RSTR
n.pron.indef
40 f 10/7
RSTR
n.quant.def
EnglishT-wsj_0044-s9
root
but
ADVS
coap
student t
ACT
n.denot
note t
PAT
n.denot
note
surrender f
PRED_M
v
surrender t
PRED_M
v
#Neg f
RHEM
atom
protest f
ACMP
n.denot
a:
neigh_sent neigh_trees Scale: 100%

Prague Markup Language

three components for your data

For a given (type of) treebank, you need to define a TrEd extension:

- **PML-schema**
  - structure of the data
- **Stylesheet**
  - how to display the data
- **Macros**
  - changing the data

Prague **M**arkup **L**anguage

PML-schema

Description of the **structure** of the data

- **types of nodes** in the data (root, node, terminal, non-terminal, ...)
- **relations** among nodes (child relation between non-terminal → non-terminal, non-terminal → terminal, coreference, discourse relations, ...)
- **names** and **types** (and special roles) **of attributes**
- **values** of enumerative attributes

Prague Markup Language

PML-schema

```
<type name="t-node.type"> <!-- simplified! -->
  <structure role="#NODE" name="t-node">
    <member as_attribute="1" name="id" role="#ID" required="1">
      <cdata format="ID"/>
    </member>
    <member name="is_generated" type="bool.type"/>
    <member name="t_lemma" required="1">
      <cdata format="any"/>
    </member>
    <member name="functor" required="1">
      <alt type="func.type"/>
    </member>
    <member name="deepord" role="#ORDER" required="1">
      <cdata format="nonNegativeInteger"/>
    </member>
    <member name="discourse" required="0">
      <list ordered="0" type="t-discourse-link.type"/>
    </member>
    ...
  </structure>
</type>
```

Prague Markup Language

PML-schema

```
<type name="t-discourse-link.type"> <!-- simplified! -->
  <structure>
    <member name="target_node.rf" required="0">
      <cdata format="PMLREF"/>
    </member>
    <member name="start_range" required="1"> ... </member>
    <member name="target_range" required="0"> ... </member>
    <member name="discourse_type"  type="t-discourse-type.type" required="0"/>
    <member name="a-connectors.rf" required="0">
      <list ordered="0"> <cdata format="PMLREF"/> </list>
    </member>
    <member name="t-connectors.rf" required="0">
      <list ordered="0"> <cdata format="PMLREF"/> </list>
    </member>
    <member name="connective" required="0"> <!-- for searching in PML-TQ only (not in the distributed data) -->
      <cdata format="any"/>
    </member>
    ...
  </structure>
</type>
```

Prague Markup Language

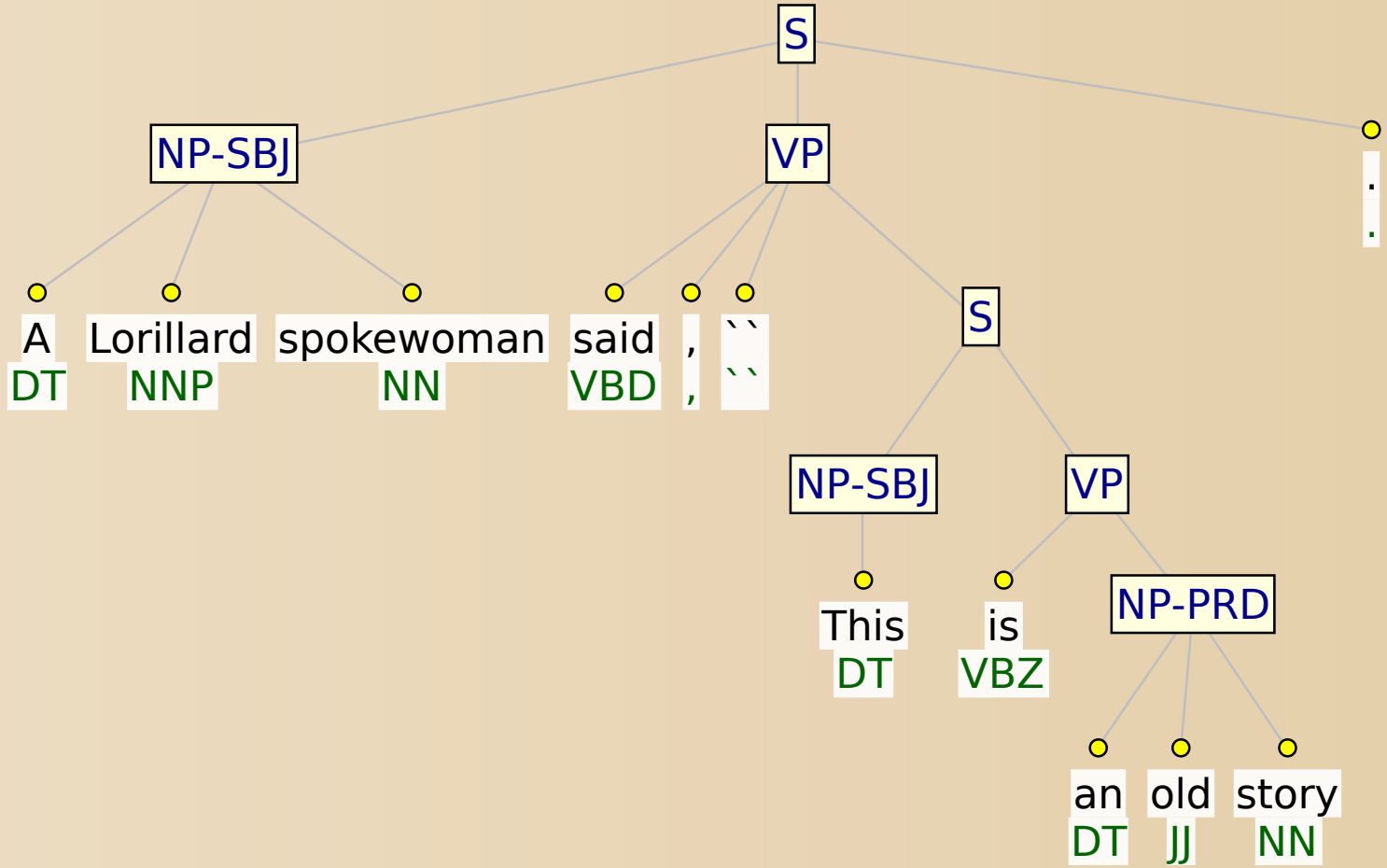
Stylesheet

How to **present** the data to the user

- **attributes** displayed at nodes
- **relations** displayed between nodes
- **shape** of nodes and edges
- **position** of nodes
- …

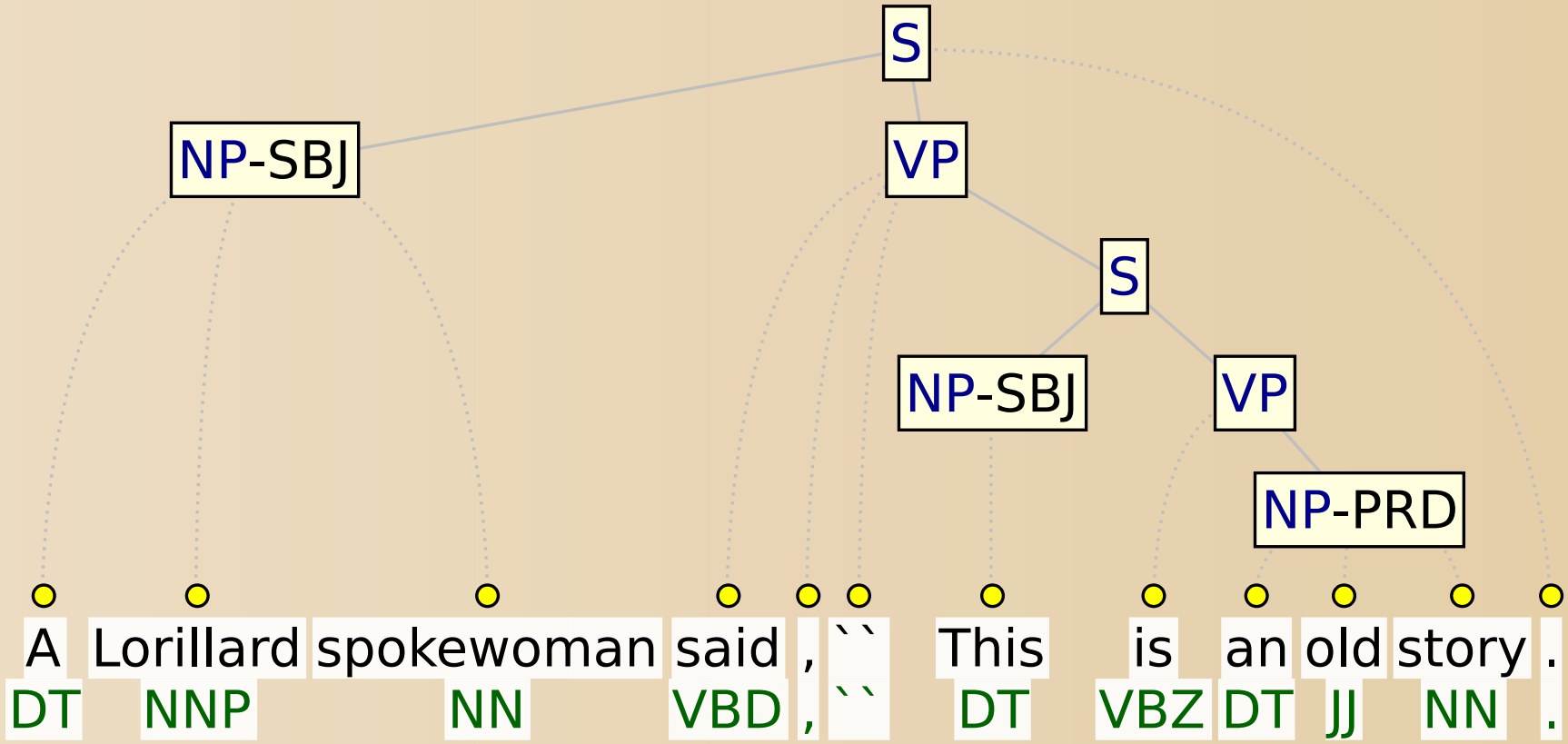
Prague Markup Language

Stylesheet



Prague Markup Language

Stylesheet



Prague Markup Language

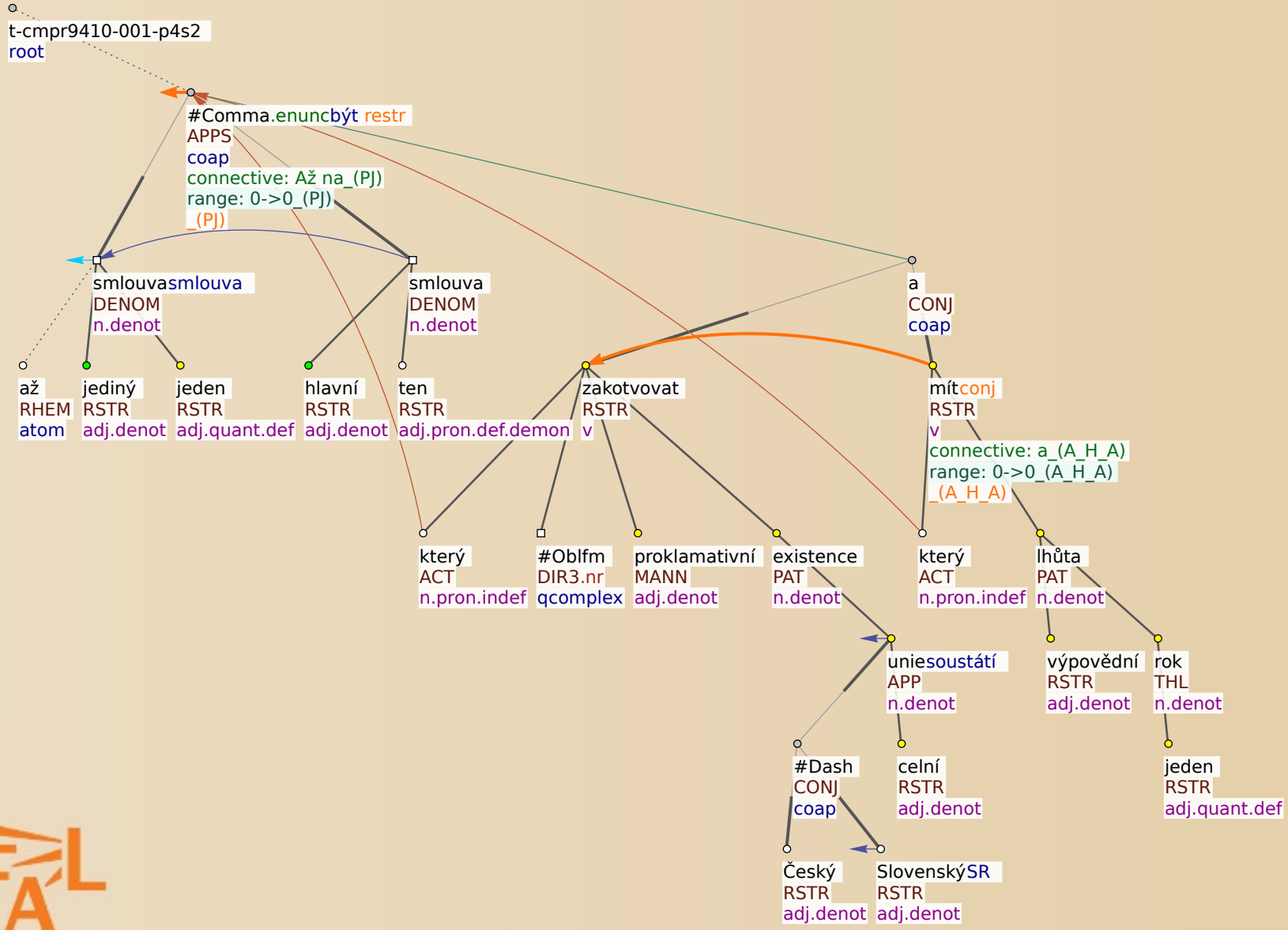
Macros

Perl code to **change** the **data or** their **appearence**

- run by a key stroke (or invoked from a script)
- **annotation** of the data
- various possibilities to **present** the same data
- …

Prague Markup Language

Macros



Prague Markup Language

Macros

Prague Markup Language

treebanks

Which treebanks do we have in PML?

- Family of Prague treebanks
  (PDT, PCEDT, PDTSC, ...)
- Tiger Corpus, BNC, Penn Treebank, ...
- HamleDT
- (Universal Dependencies)

ÚFAL

Prague Markup Language

application framework

Once you have **data in PML**, you can use:

- editor **TrEd** to **browse** and **manually edit** the data
- **btred** for **batch processing** the data **from command line** – apply **scripts in Perl/btred** to the data
- **PML-Tree Query** for **graphical search** in the data

ÚFAL

Prague Markup Language

application framework

Once you have **data in PML**, you can use:

- editor **TrEd** to **browse** and **manually edit** the data
- **btred** for **batch processing** the data **from command line** – apply **scripts in Perl/btred** to the data
- **PML-Tree Query** for **graphical search** in the data

Prague **M**arkup **L**anguage

application framework

Now, let's install TrEd

NPFL075 Practical Class 01

Jiří Mírovský

Charles University

Institute of Formal and Applied Linguistics

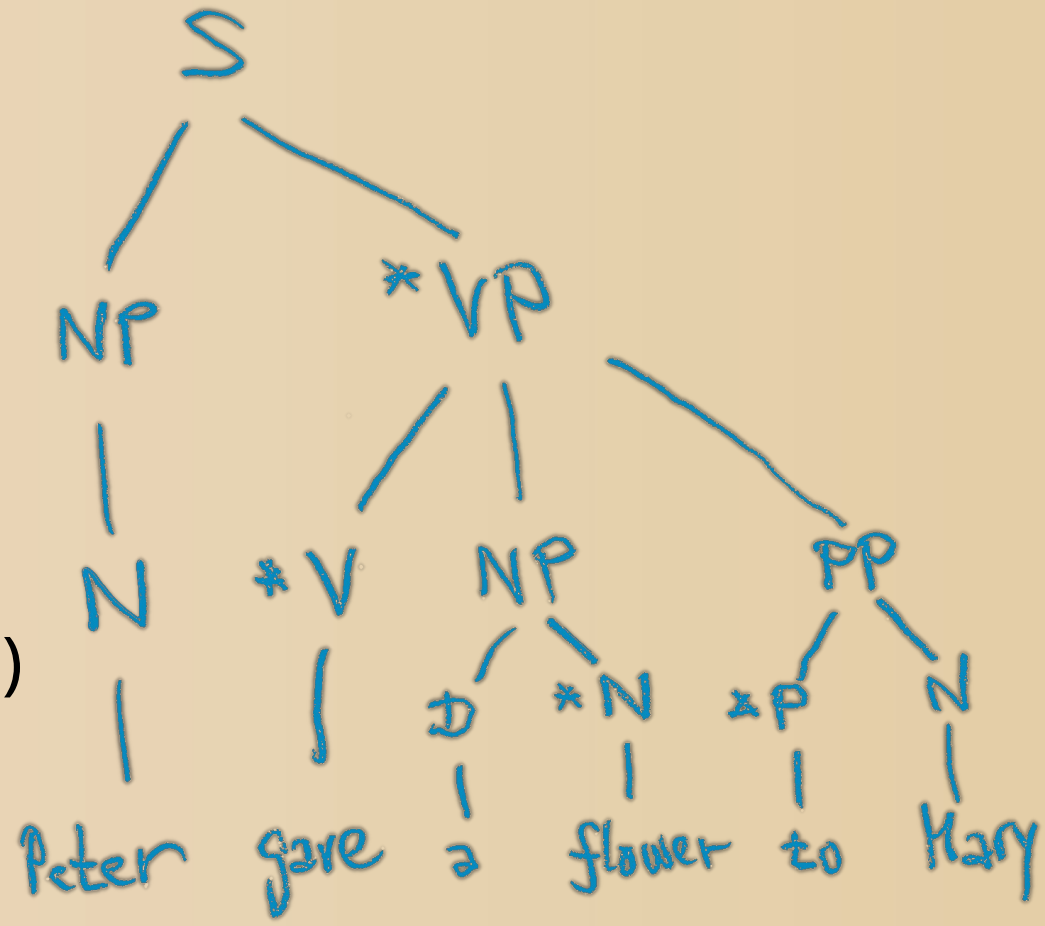
Phrase Structure Trees

```
S (
    NP ( N ( 'Peter' ) )
  * VP ( * V ( 'gave' )
           NP ( D ( 'a' )
                * N ( 'flower' ) )
           PP ( * P ( 'to' )
                  N   ( 'Mary' ) )
       )
   )
```

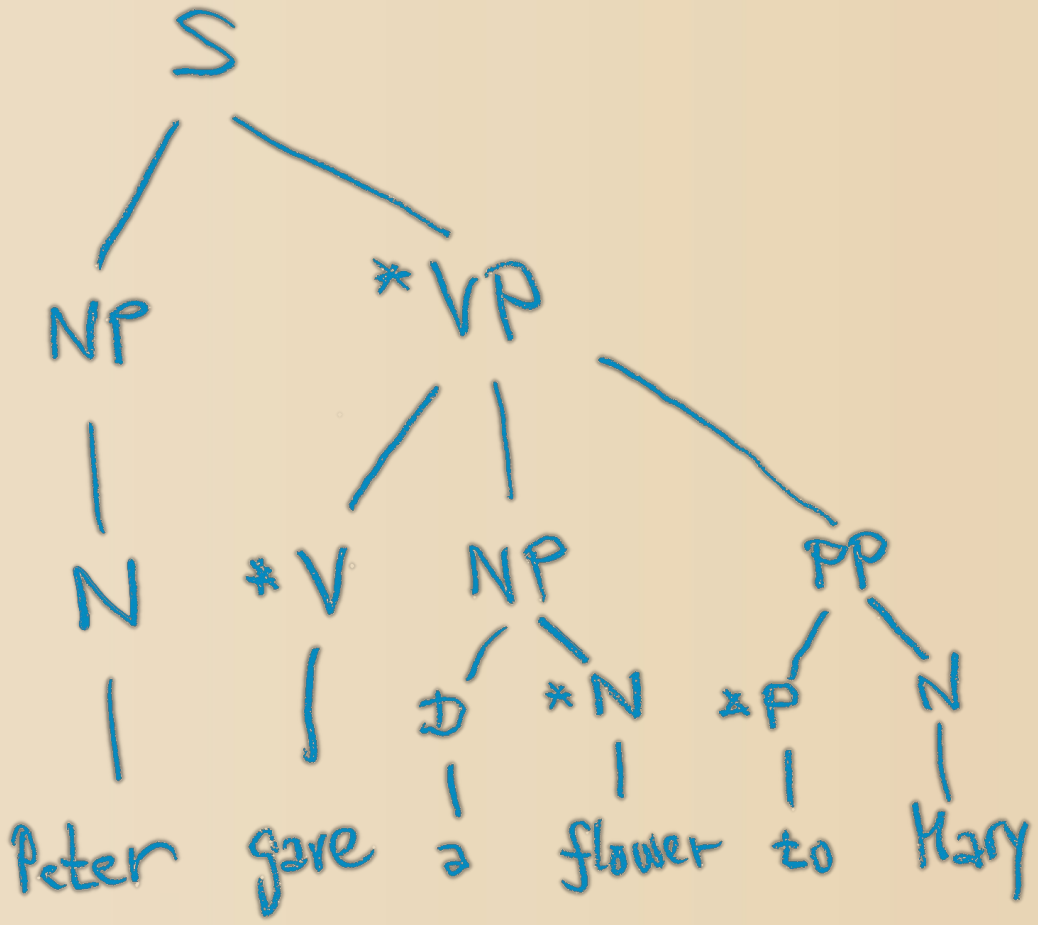
→ draw the tree

Phrase Structure Trees

```
S (
    NP ( N ( 'Peter' ) )
  * VP ( * V ( 'gave' )
           NP ( D ( 'a' )
                * N ( 'flower' ) )
           PP ( * P ( 'to' )
                  N   ( 'Mary' ) )
       )
   )
```

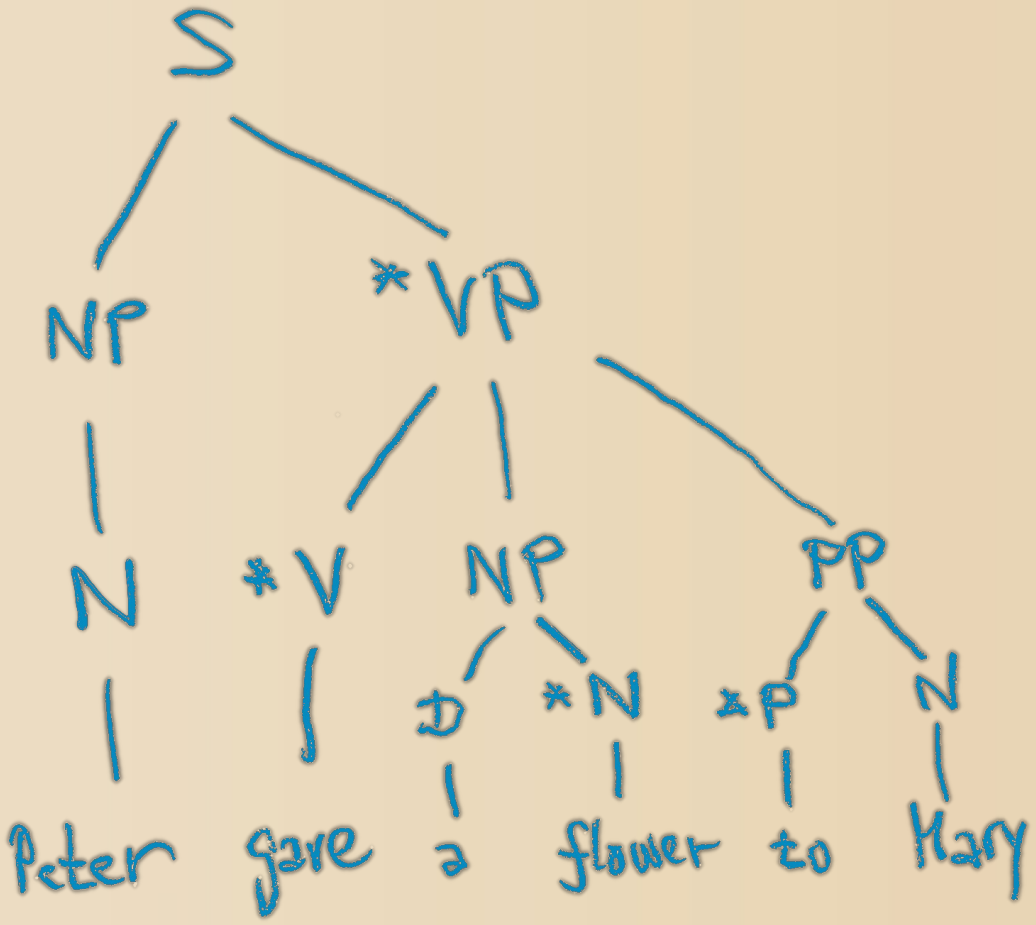


Phrase Structure and Dependency Trees



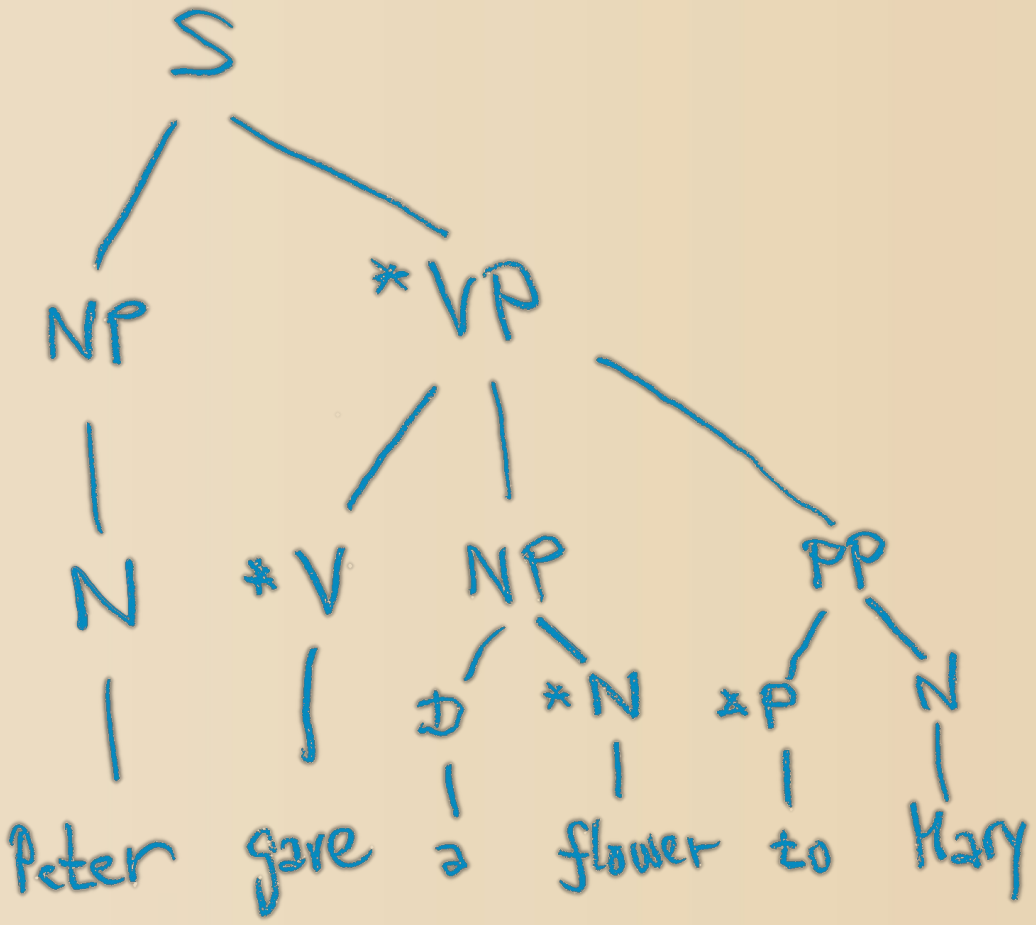
- properties?

Phrase Structure and Dependency Trees



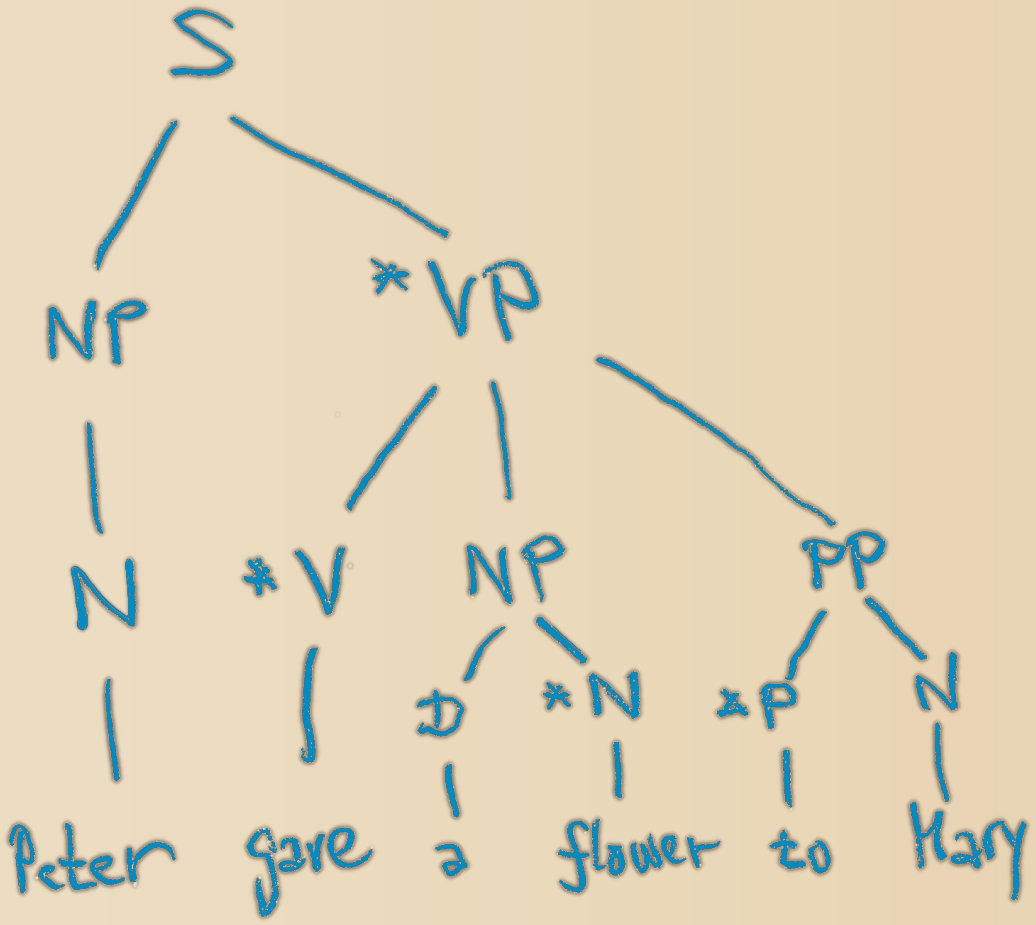
- oriented edges
- a single root
- each node has a single parent (except for the root)
- linear order

Phrase Structure Trees



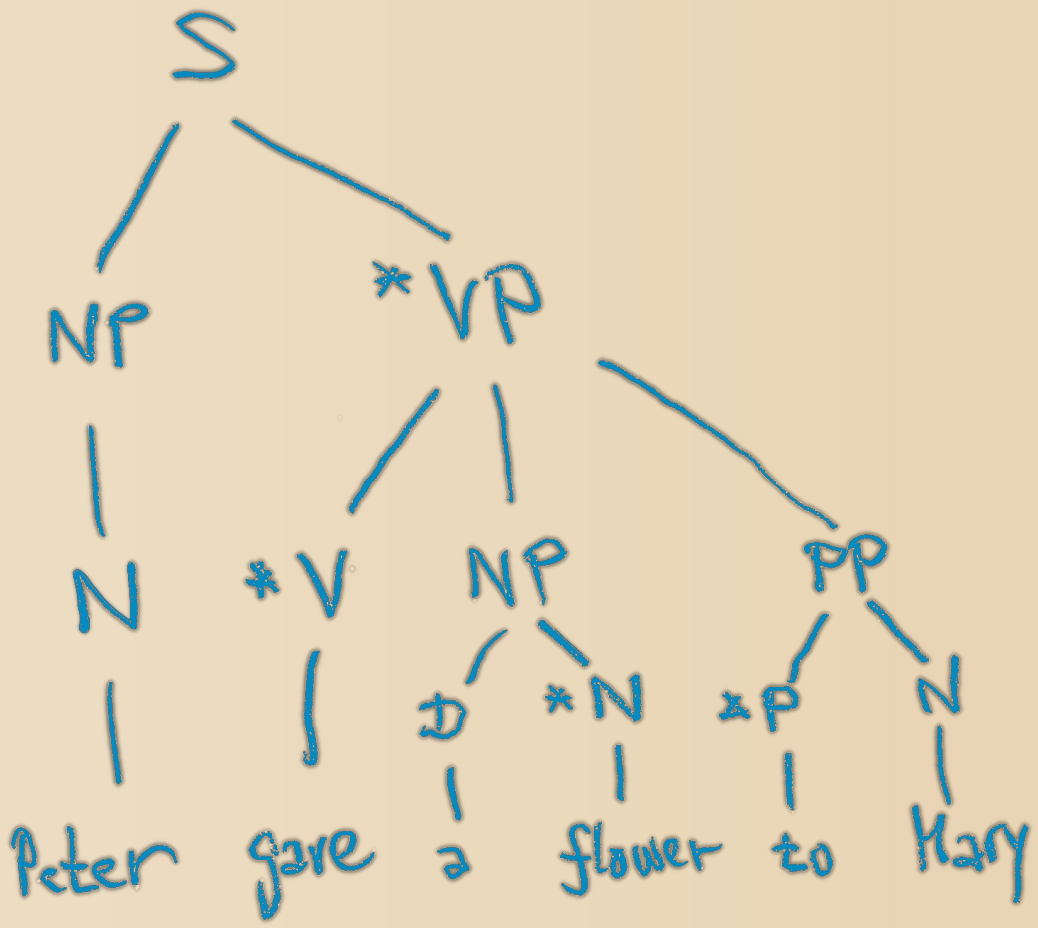
- additional properties?

Phrase Structure Trees



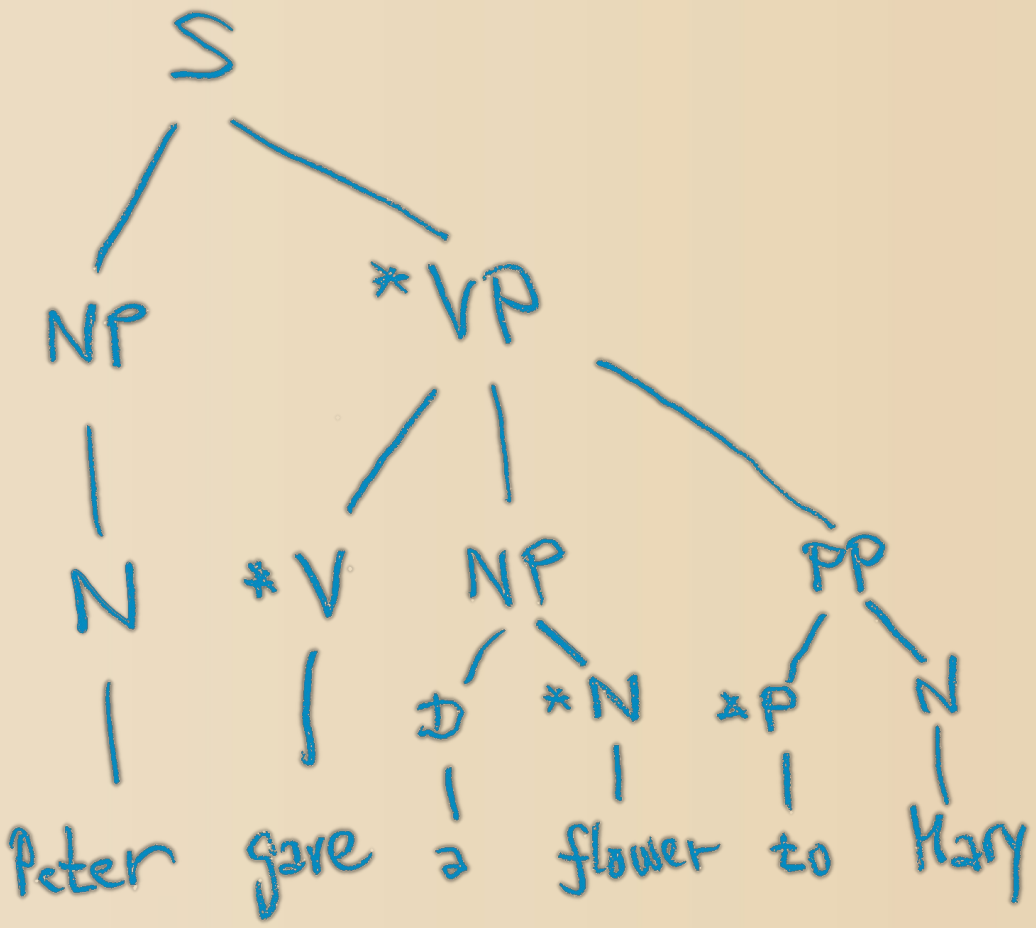
- projectivity (no crossing edges)
- context-free grammar

Phrase Structure Trees



write a context-free grammar from the tree

Phrase Structure Trees



S → NP *VP

NP → N

VP → *V NP PP

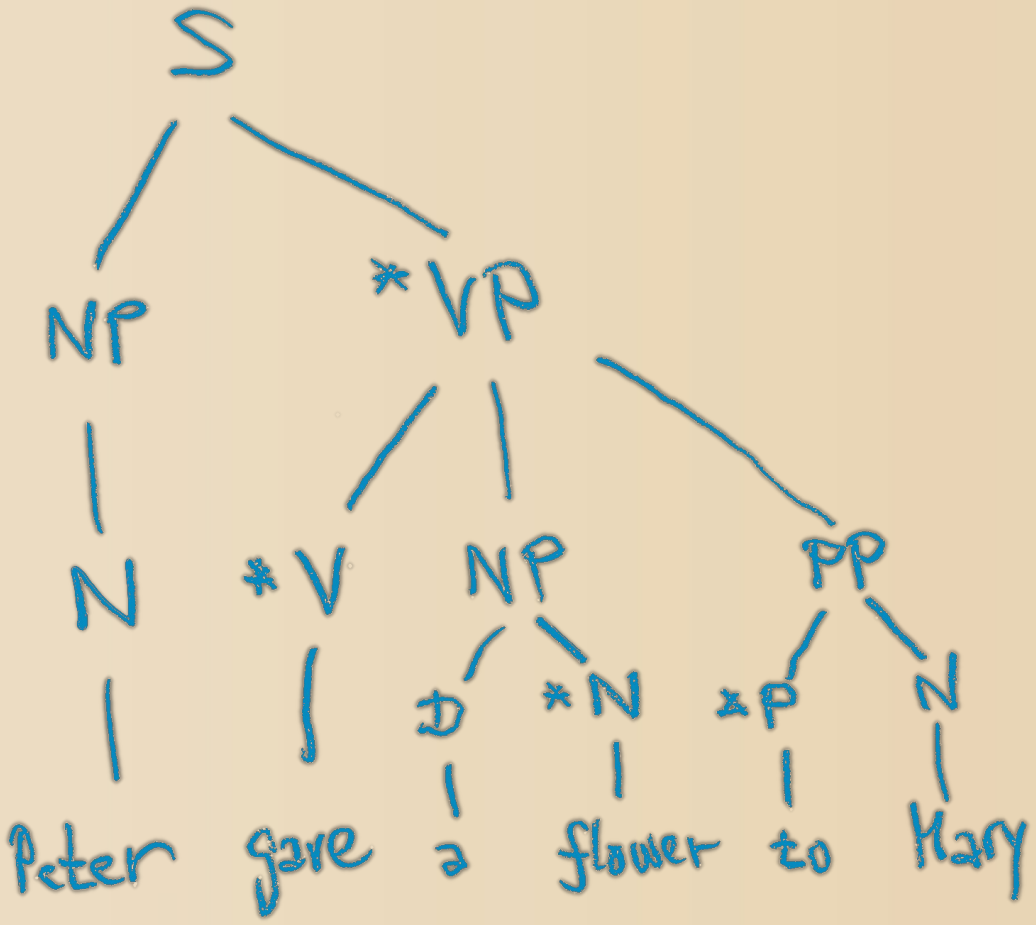
NP → D *N

PP → *P N

N → ‘Peter‘

…

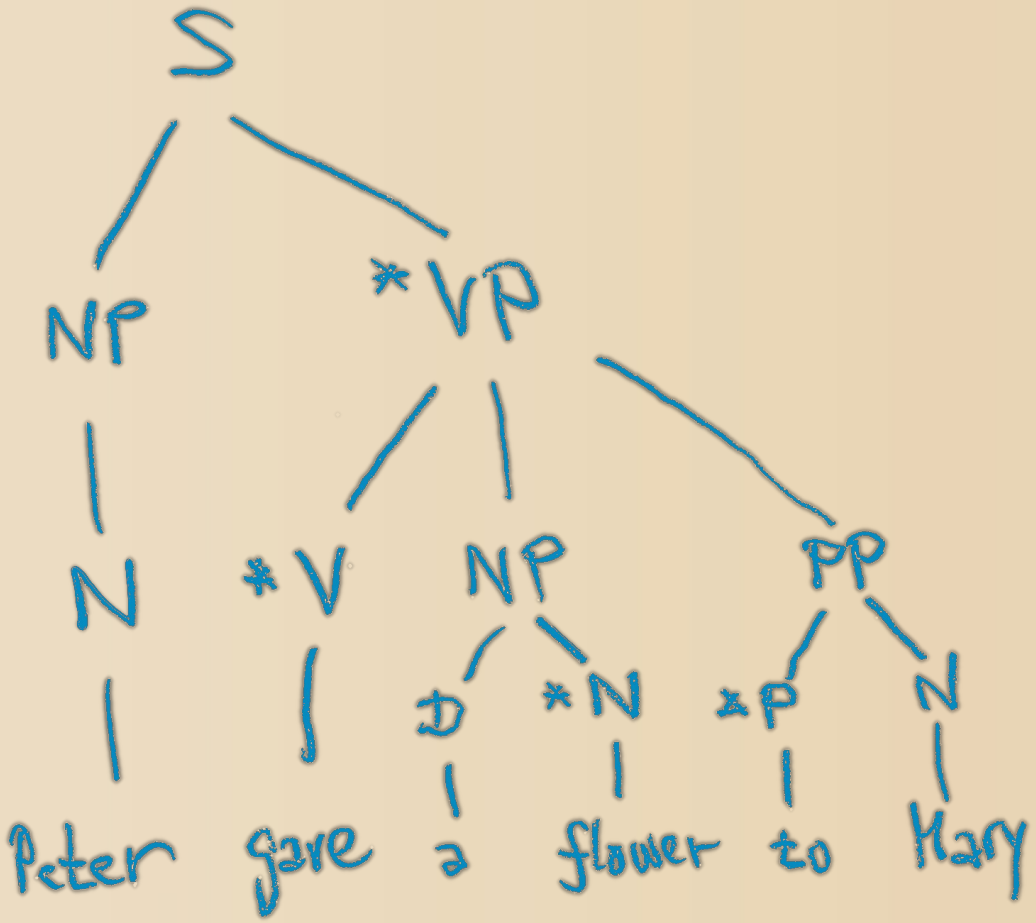
Phrase Structure Trees → Dependency Trees



bottom-up

- for each terminal node find its governor (head)
  - Peter → gave
  - gave = root
  - a → flower
  - …

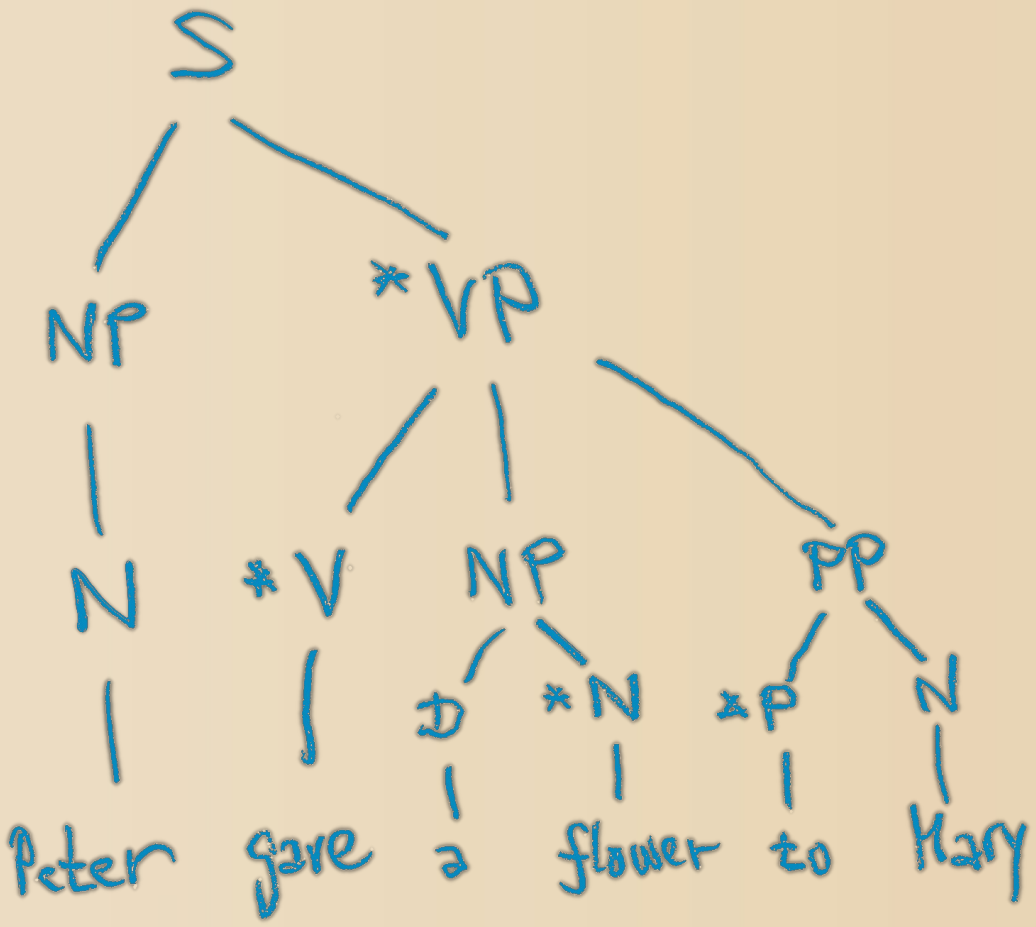
Phrase Structure Trees → Dependency Trees



from top to down

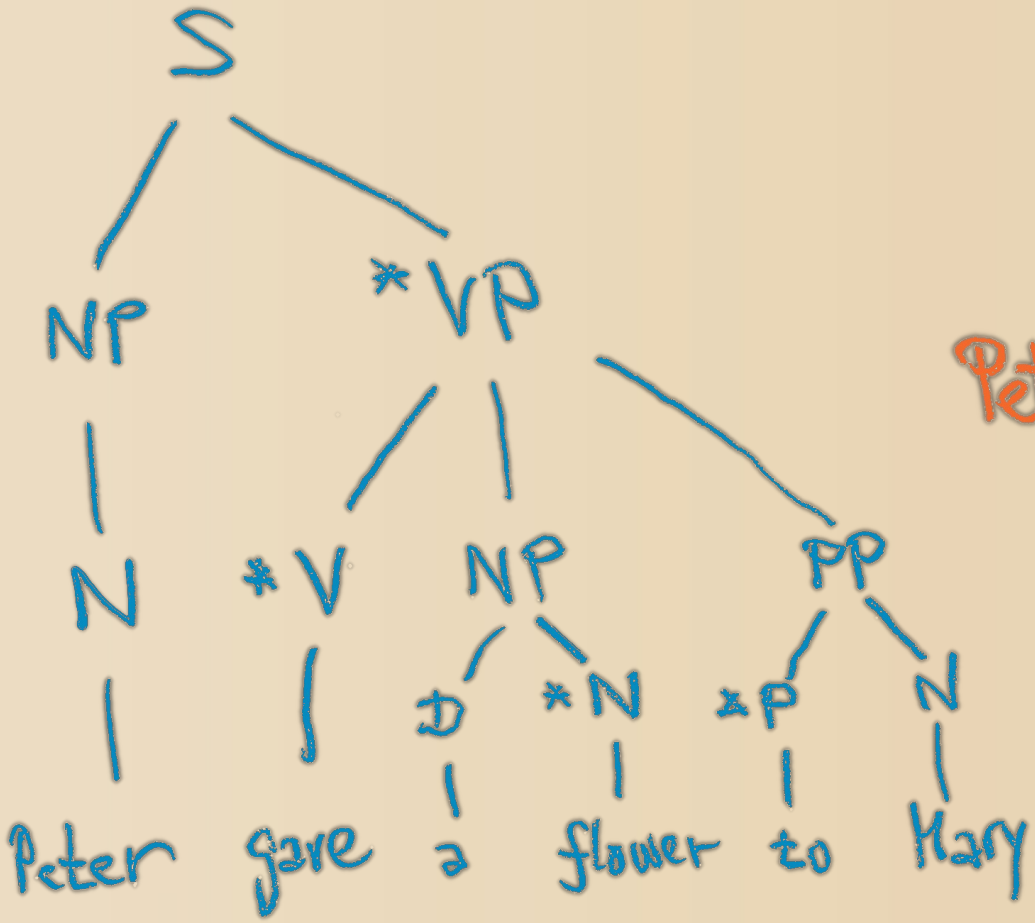
- start with S and go recursively:
  - process the head branch → the root of the subtree
  - process other branches and add them as sons of the subtree root

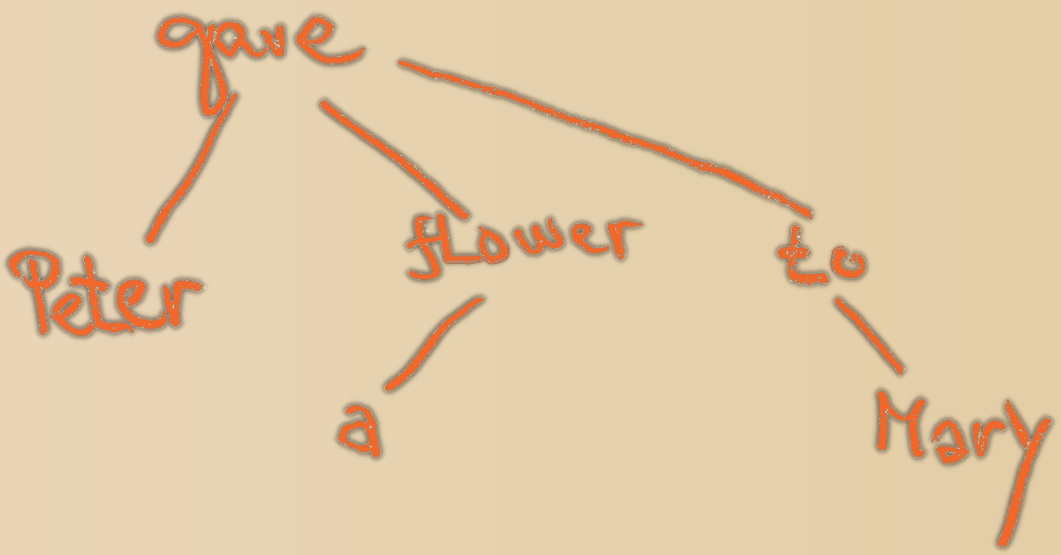
Phrase Structure Trees → Dependency Trees



Draw the dependency tree from the phrase-structure tree.

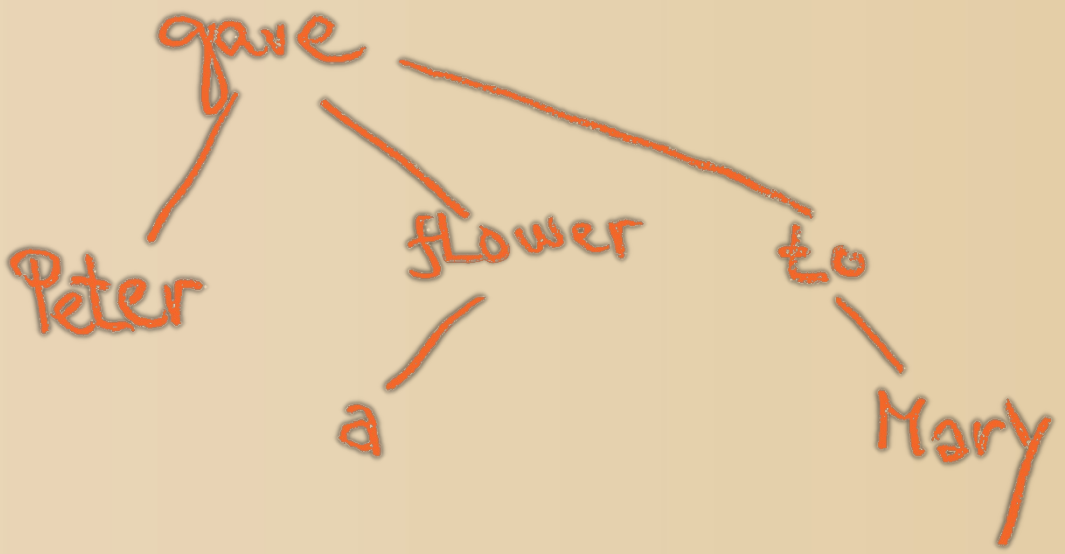
Phrase Structure Trees → Dependency Trees





Data Representation of Trees
1) a parent for each node

(**note:** for now, we do not care about linear order)



properties: simple, constant size for each node

Data Representation of Trees
1) a parent for each node

(**note:** for now, we do not care about linear order)

gave.

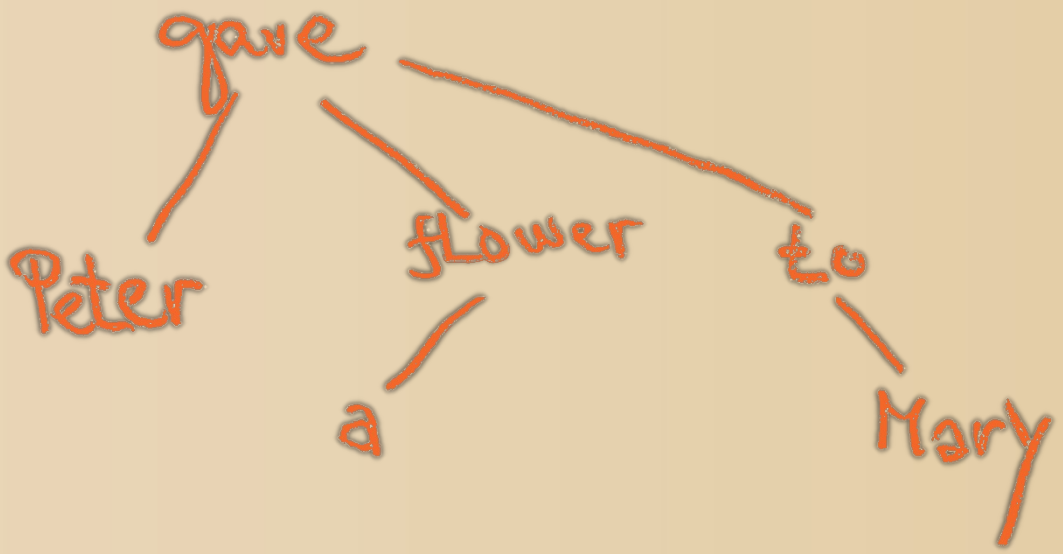
Peter.gave

flower.gave

a.flower

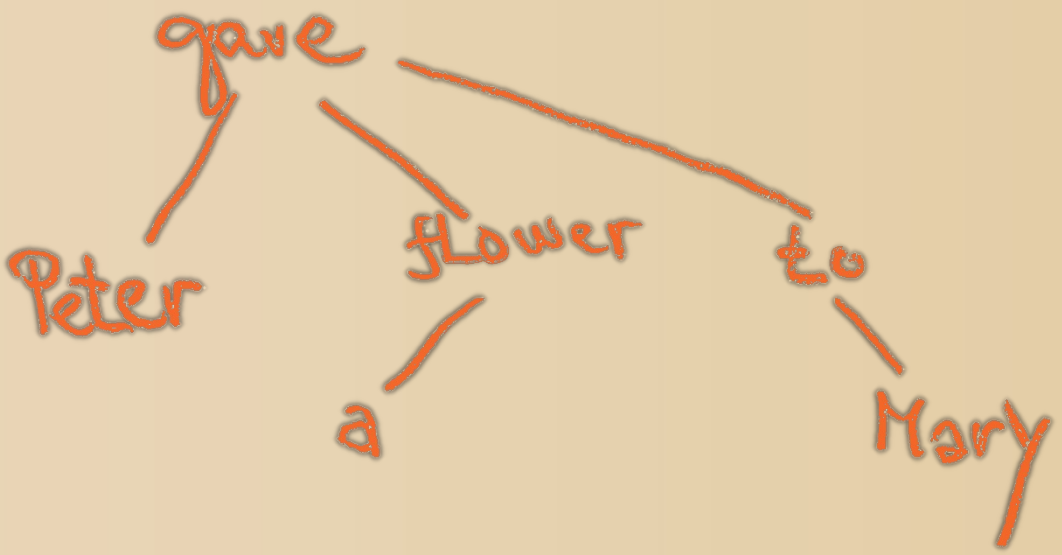
to.gave

Mary.to



properties: simple, constant size for each node

Data Representation of Trees
2) list of sons for each node



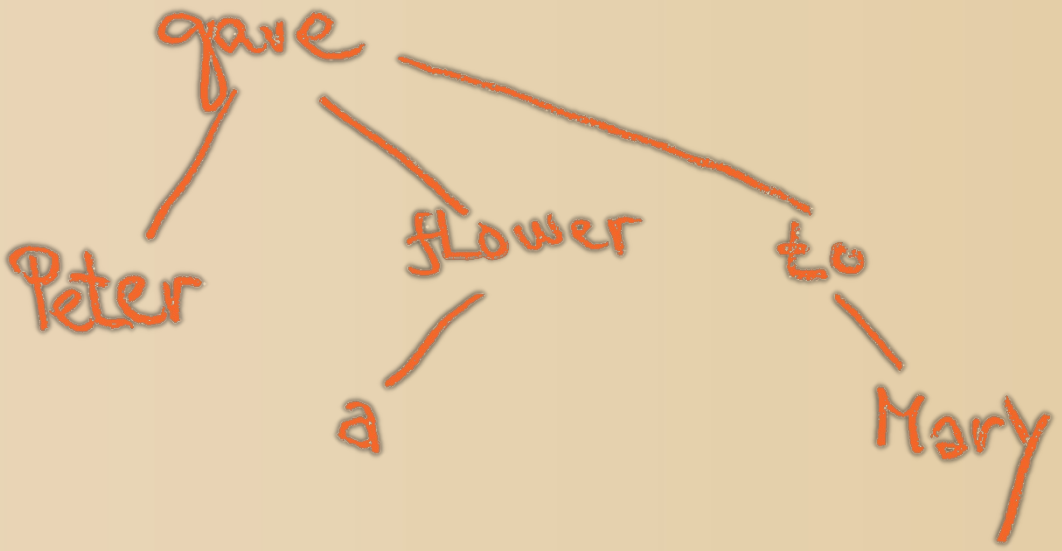
properties: simple, variable size for each node, good for top-down approach

Data Representation of Trees
2) list of sons for each node

gave: Peter, flower, to

flower: a

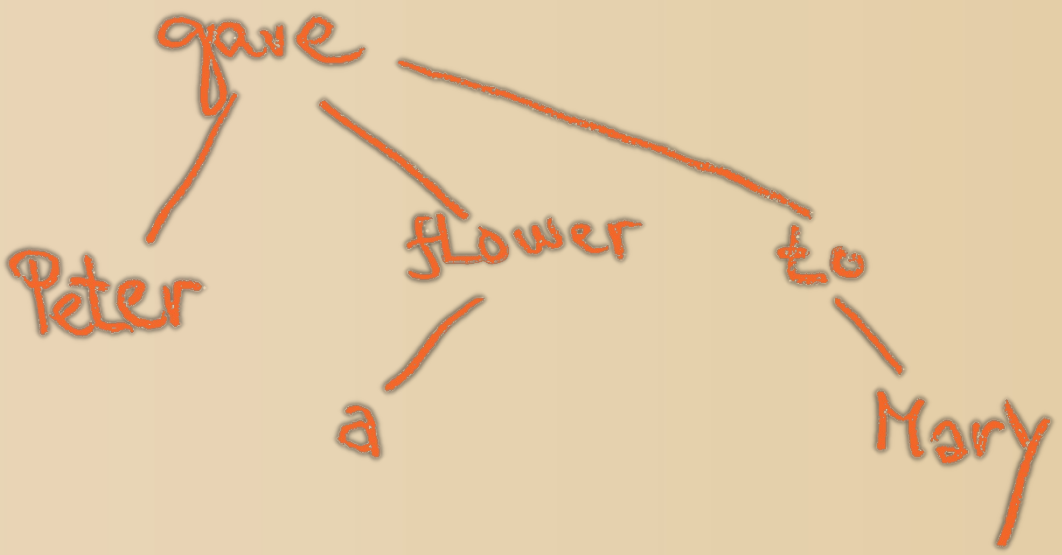
to: Mary



properties: simple, variable size for each node, good for top-down approach

Data Representation of Trees

3) first son and first brother for each node



properties: unintuitive, fixed size for each node, good for top-down approach

Data Representation of Trees

3) first son and first brother for each node

gave: Peter/-

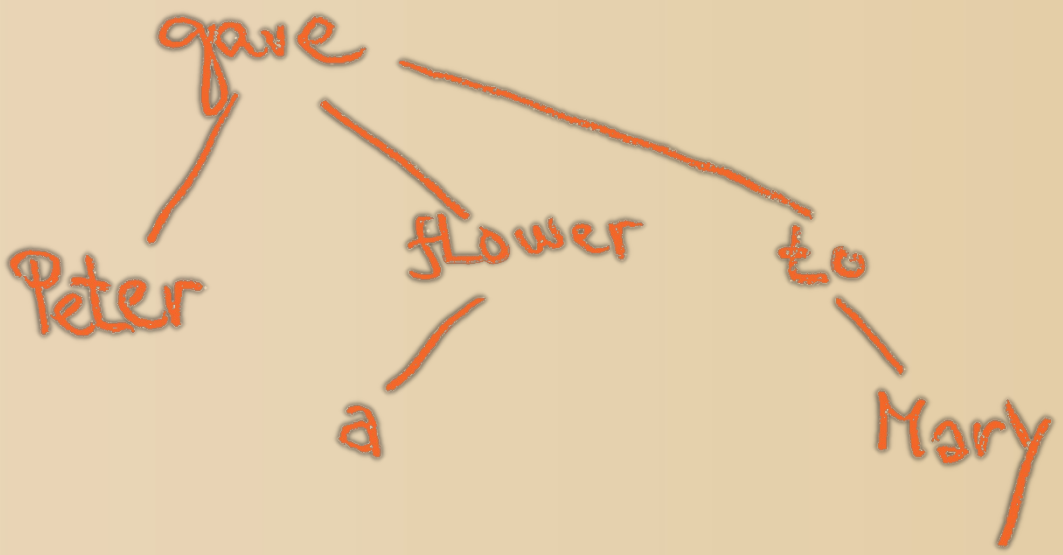
Peter: -/flower

flower: a/to

a: -/-

to: Mary/-

Mary: -/-



properties: unintuitive, fixed size for each node, good for top-down approach

Two File Formats for Trees

1) reference to parent

gave.

Peter.gave

flower.gave

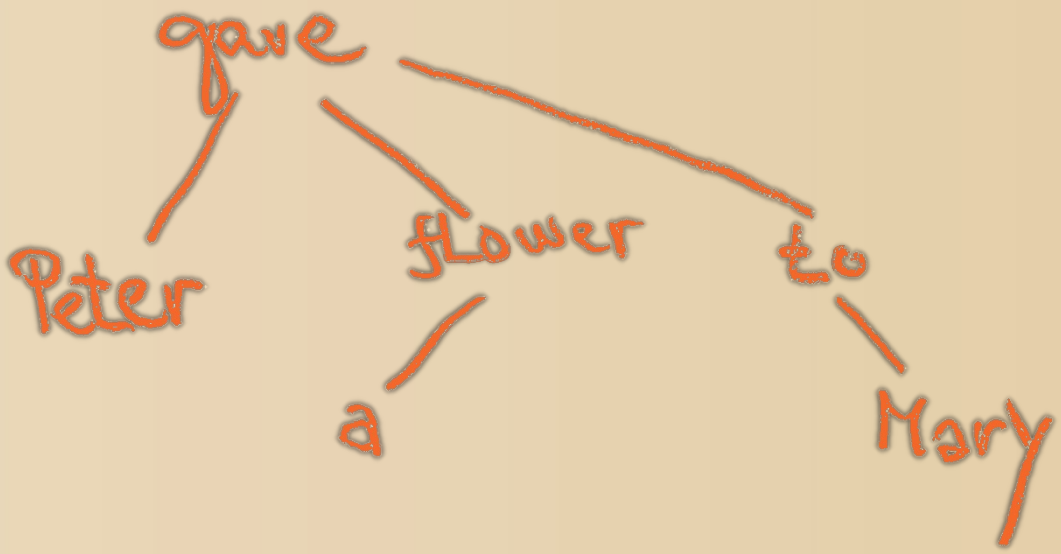
a.flower

to.gave

Mary.to

2) recursive list of sons

gave(Peter,flower(a),to(Mary))



(**note:** we still do not care about linear order)

Two File Formats for Trees

1) reference to parent

gave.

Peter.gave

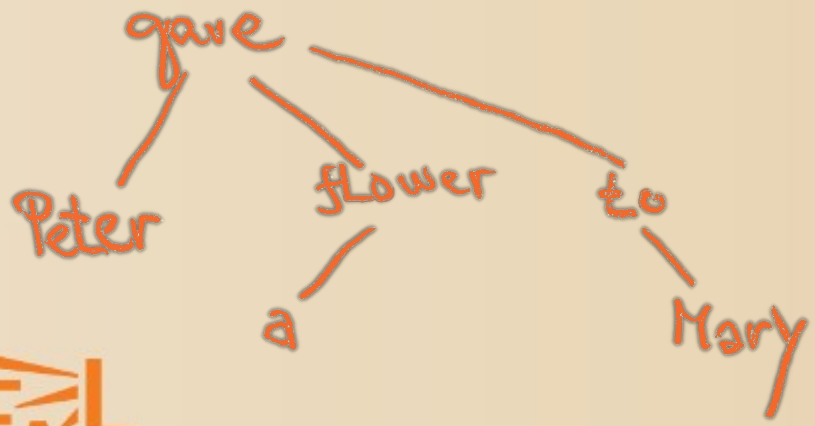
flower.gave

a.flower

to.gave

Mary.to

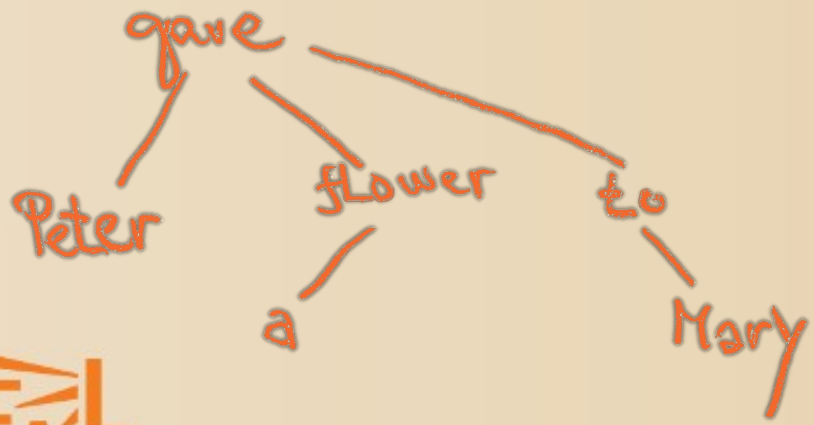
Possible errors:



Two File Formats for Trees

1) reference to parent

gave.
Peter.gave
flower.gave
a.flower
to.gave
Mary.to



Possible errors:

errors in syntax:
- no dot or more dots at a line
- forbidden characters

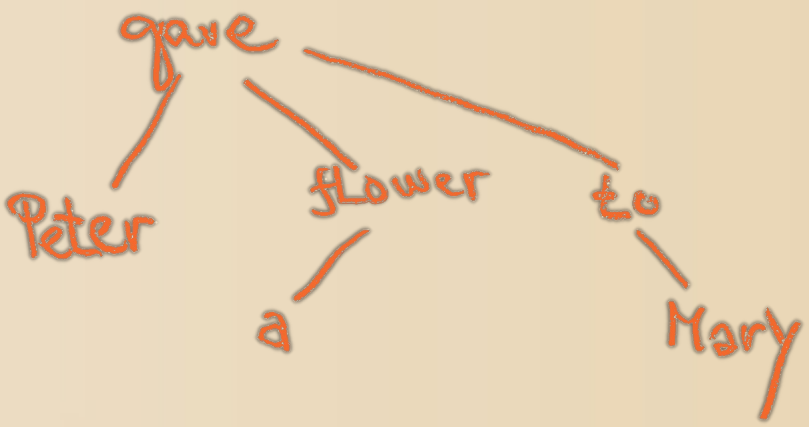
errors in semantics:
- missing root or more than one root
- several parents for a single node
- self-reference
- cycle in the references
- discontinuous tree
- empty file (maybe not an error)

Two File Formats for Trees

2) recursive list of sons

gave (Peter,flower (a),to(Mary))

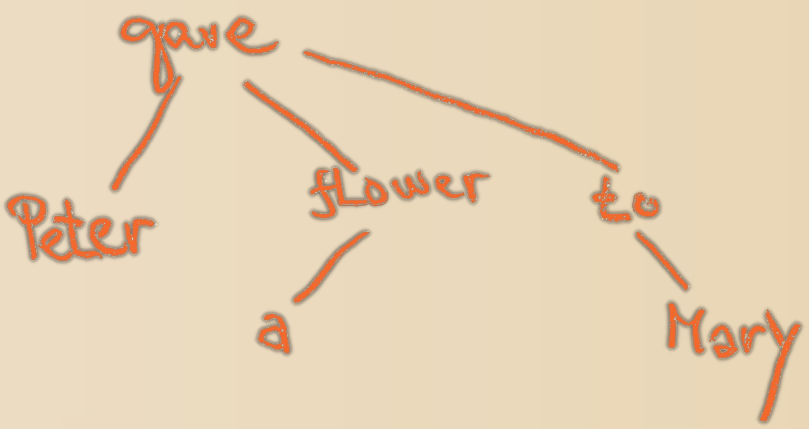
Possible errors:



Two File Formats for Trees

2) recursive list of sons

gave (Peter,flower (a),to(Mary))



Possible errors:

errors in syntax:

- many possible errors, e.g. mismatching parentheses

errors in sytax/semantics:

- repeated node
- multiple roots: a (b c) d (e f)
- no root: (b c)

NPFL075 Practical Class 00

Please:

- **submit your homeworks via svn to your personal directories**
- **send me your directory name via e-mail** (mirovsky@ufal.mff.cuni.cz)