Handout 6: Parsing Feature Grammars

1. A **category** consists of a **type** (symbol) and a list of **features**.

- **a.** Example: `V.sg.i.0`
- **b.** **Attributes** are implicitly associated with positions. Could more explicitly write `V(num=sg, tr=i, sel=0)`.
- **c.** Represent a category as a tuple: `('V', 'sg', 'i', '0')`

2. Variables

- **a.** Show up in rules, not in parse trees
- **b.** Example: `VP.$f -> V.$f.i.0`
- **c.** **Represent variables as ints.** Number them as we digest the rule.
- **d.** Example: `('VP', 0), ('V', 0, 'i', '0')`

3. Categories should be tuples, but we would like them to print like `V.$0.i.0`

- **a.** A trick

```
class Category (tuple):
    def __repr__ (self):
        ...
```

- **b.** Example of use

```
>>> from hw3 import *
>>> x = Category(['V', 0, 'i', '0'])
>>> x
V.$0.i.0
>>> x[0]
'V'
>>> x[1:]
(0, 'i', '0')
>>> len(x)
4
```

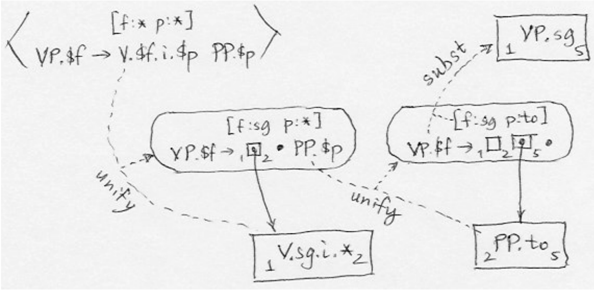
4. `parse_category`(x, t)

- **a.** t is a symbol table (dict). If omitted, variables are not allowed.
- **b.** `parse_category`(x, t) turns a category string x into a `Category` instance. Example of a category string: `'V.`$f.i.$`s'`
- **c.** If a feature string begins with dollar sign, it is a variable.
- **d.** If present in symbol table, use the stored integer. Otherwise, store and return the next integer (which is the size of the table!)

5. Using categories

a. Categories do not have to be identical to match

b. Example.



6. **Start:** rule + first child node

a. Categories as tuples:

```
('VP', 0) -> ('V', 0, 'i', 1) ('PP', 1)
('V', 'sg', 'i', '*')
```

b. Initial bindings: `['*', '*']`

c. **Unify** the categories

```
('V', 0, 'i', 1) + ('V', 'sg', 'i', '*')
```

d. Equivalent to:

```
('V', '*', 'i', '*') + ('V', 'sg', 'i', '*')
b[0] = '*' + 'sg'
b[1] = '*' + '*'
```

e. Result:

```
['sg', '*']
```

7. **Combine:** edge + second child

a. **Unify** edge after-dot category + child node category:

```
('PP', 1) + ('PP', 'to')
```

b. which is

```
('PP', '*') + ('PP', 'to')
b[1] = '*' + 'to'
```

c. Resulting (final) bindings:

```
['sg', 'to']
```

8. **Complete:**

a. **Substitute** the bindings into the lhs category:

```
('VP', 0) + ['sg', 'to'] = ('VP', 'sg')
```

9. Function **meet**(u, v) combines two values

- **a.** Return u if $u = v$
- **b.** Return u if $v =$ `*`; return v if $u =$ `*`
- **c.** Else fail

10. Function **unify**(x, y, b) takes two categories, returns updated bindings

- **a.** Fails if $x[0] \neq y[0]$
- **b.** Otherwise, compare each $u = x[i]$ to $v = y[i]$, for $i > 0$
- **c.** If u is a variable, call that "the variable," and replace u with the value of the variable: $b[u]$
- **d.** If v is a variable, signal an error
- **e.** Let the new value be `meet`(u, v, b); fail if `meet` fails
- **f.** If there is a variable, store the new value back into b
- **g.** (At the beginning, make a fresh copy of the bindings, so that the updates do not destructively modify the original bindings)
- **h.** Return value is the new bindings, or `None` on failure

11. Function **subst**(b, x)

- **a.** Makes a new category (tuple) in which each variable has been replaced by its value
- **b.** I.e., returns a new category containing no variables

12. `parse_category`(s, *symtab*)

- **a.** Takes a string like `"V.$v.i.0"` and turns it into a category `('V', 0, 'i', '0')`
- **b.** As it encounters symbolic variables (e.g., "`$v`"), it numbers them and makes an entry in the symtab (e.g., `{'v': 0}`)
- **c.** If *symtab* is `None`, then variables are not permitted.

Feature grammars

The definitions of `Lexicon`, `Rule`, and `Grammar` require a few modifications.

13. `parts()`, `expansions()`, `continuations()`

- **a.** If we ask for the continuations of `V.sg.i.0`, and the rule rhs begins with `V.$n.i.$p`, we still want it.
- **b.** I.e., rules are indexed by the category *type* ("`V`") and `continuations()` takes the type (`cat[0]`) as input.
- **c.** We may get too many rules back (e.g., `V.$n.t.$p`), but we will check them before we use them.

14. Add a `bindings` attribute to the `Rule` class. Holds the initial bindings: a list of "`*`"s, one for each variable in the rule.

15. Modify the functions that read lexicons and grammars from files.

- **a.** Use `parse_category()` wherever a category occurs
- **b.** There should be a single `symtab` shared by all categories in a given rule
- **c.** For parts of speech in the lexicon, `symtab` is `None`. (Variables are not allowed.)

Feature parser

Whenever we match a rule category against a node category, we must check that it unifies, and update the bindings.

16. Add a `bindings` attribute to the `Edge` class

17. `start()`

- **a.** After getting the continuations for the child node `cat` from the grammar, for each rule, unify `cat` with `rule.rhs[0]`.
- **b.** If unification succeeds, the resulting binding goes into the new edge.

18. `add_edge()`.

- **a.** Instead of indexing edge ($X \rightarrow {}_i \ldots {}_j \bullet Z \ldots$) by (j, Z), we index it by $(j, Z[0])$.
- **b.** By contrast: nodes are still indexed by the full category, not just the type symbol. E.g., ${}_2\texttt{NP.sg}_5$ and ${}_2\texttt{NP.pl}_5$ are two separate nodes.

19. `combine()`.

- **a.** Look up edges under $(j, Z[0])$, not (j, Z)
- **b.** Unify the category after the dot with the child category. Store the resulting bindings in the new edge (if unification succeeds).

20. `complete()`. The parent node category is not simply the rule lhs, but the result of `subst(edge.bindings, edge.rule.lhs)`.

Grammar development

21. Grammar and parser files: allow empty lines and comment lines (beginning with `#`)