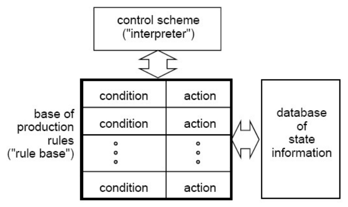
Production Systems Architecture and Implementing Conversational Agents

CSE 415: Introduction to Artificial Intelligence
University of Washington
Spring, 2017

Outline

- Production Systems Architecture
- Production System Form
- Example: Conversion to Roman Numerals
- Ordered vs. Unordered Production Systems
- Discrimination Nets
- Weizenbaum's ELIZA and the SHRINK.
- Implementation Issues, Using Python

Production System Architecture



Production System Form

A rule of the form G **if [condition] then [action]** is sometimes called a *production rule*.

Python's *if* statement can be used to implement rule testing and application.

```
If input=='Why?':  return explain()
```

Example: Conversion to Roman Numerals

0 =	11 = XI	30 = XXX
1 = I	12 = XII	40 = XL
2 = II	13 = XIII	50 = L
3 = III	14 = XIV	60 = LX
4 = IV	15 = XV	70 = LXX
5 = V	16 = XVI	80 =
6 = VI	17 = XVII	LXXX
7 = VII	18 = XVIII	90 = XC
8 = VIII	19 = XIX	100 = C
9 = IX	20 = XX	500 = D
10 = X		1000 = M

Example Production System

If x is null, then prompt the user and read x.
If x is not null and is bigger than 39, then print ``too big''
  and make x null.
If x is not null and is between 10 and 39, then print ``X''
  and reduce x by 10.
If x is not null and is equal to 9, then print ``IX''
  and reduce x to 0.
If x is not null and is between 5 and 8, then print ``V''
  and reduce x by 5.
If x is not null and is equal to 4, then print ``IV''
  and reduce x to 0.
If x is not null and is between 1 and 3, then print ``I''
If x is not null and is between 1 and 3, then print ``I''
  and reduce x by 1.
If x is not null and is equal to 0,
  then print an end-of-line and make x null.

Ordered vs Unordered Production Systems

We've just seen an "unordered" production system. Permuting the rules does not change the behavior of an unordered system.

- Easy to add new rules, remove rules
- Condition testing typically is redundant

In an ordered system, the rule order is important and errors might occur if the rules are permuted.

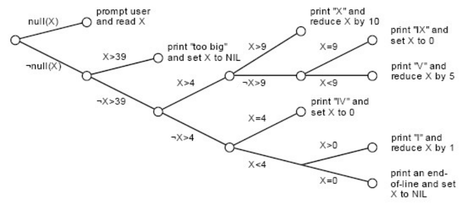
- Harder to add and remove rules without breaking the system
- Condition testing is less redundant

Python Implementation of an Ordered Production System

```
# Roman2.py
def Roman2():
  """Roman numeral conversion with ordered Production System"""
  x = ''; ans = ''
  while True:
      if x=='': x = input('Enter number: ')
      elif x > 39:
          print('too big'); x = ''
      elif x > 9:
          ans += 'X'; x -= 10
      elif x == 9:
          ans += 'IX'; x = 0
      elif x > 4:
          ans += 'V'; x -= 5
      elif x == 4:
          ans += 'IV'; x = 0
      elif x > 0:
          ans += 'I'; x -= 1
      elif x == 0:
          print(ans); x = ''; ans = ''
      else: print('bad number'); break
```

Discrimination Nets

In order to speed up condition testing,
Structure the search for a matching condition as a tree search rather than a linear search.


null(X)
prompt user and read X
¬null(X)
X>39
print "too big" and set X to NIL
¬X>39
X>4
X>9
print "X" and reduce X by 10
¬X>9
X=9
print "IX" and set X to 0
X<9
print "V" and reduce X by 5
¬X>4
X=4
print "IV" and set X to 0
X<4
X>0
print "I" and reduce X by 1
X=0
print an end-of-line and set X to NIL

Application to Dialog-Style Interaction

The ELIZA program was written by Joseph Weizenbaum at the Massachusetts Institute of Technology's Artificial Intelligence Lab in the late 1960s.

It was written in Lisp, and demonstrated how, through clever pattern matching and phrase transformation, an illusion of an intelligent agent could be created.

The Shrink

The Shrink is a small program modeled after Weizenbaum's ELIZA program to demonstrate production-system programming.

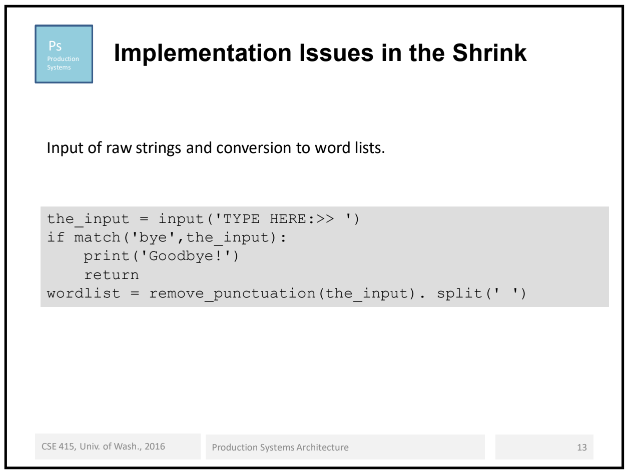
This version of the Shrink is written in Python, and works by converting the user's raw string inputs into lists of words. It then looks for particular words in these word lists.

Application to Dialog-Style Interaction

User: "I am full of anticipation."
Shrink: "Please tell me why you are full of anticipation."

The Shrink's production rule, in Python:

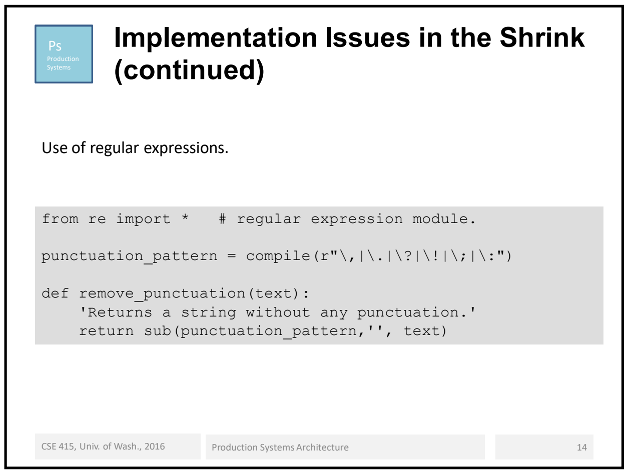
```
if wordlist[0:2] == ['i','am']:
          print("Please tell me why you are " +\
                stringify(mapped_wordlist[2:]) + '.')
          return
```



Implementation Issues in the Shrink

Input of raw strings and conversion to word lists.

```
the_input = input('TYPE HERE:>> ')
if match('bye',the_input):
    print('Goodbye!')
    return
wordlist = remove_punctuation(the_input). split(' ')
```



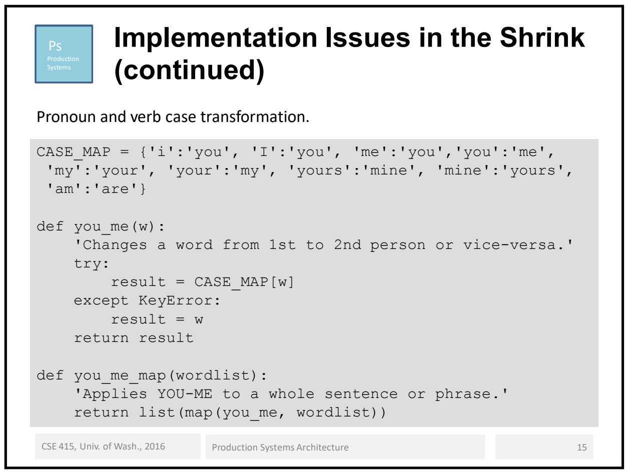
Implementation Issues in the Shrink (continued)

Use of regular expressions.

```
from re import *   # regular expression module.

punctuation_pattern = compile(r"\,|\.|\?|\!|\;|\:")

def remove_punctuation(text):
    'Returns a string without any punctuation.'
    return sub(punctuation_pattern,'', text)
```



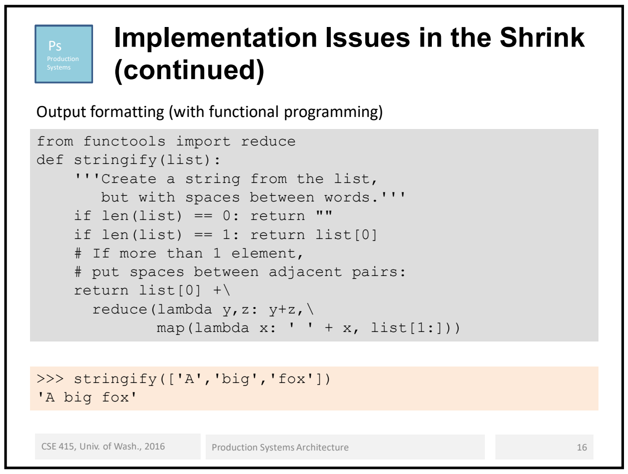
Implementation Issues in the Shrink (continued)

Pronoun and verb case transformation.

```
CASE_MAP = {'i':'you', 'I':'you', 'me':'you','you':'me',
 'my':'your', 'your':'my', 'yours':'mine', 'mine':'yours',
 'am':'are'}

def you_me(w):
    'Changes a word from 1st to 2nd person or vice-versa.'
    try:
        result = CASE_MAP[w]
    except KeyError:
        result = w
    return result

def you_me_map(wordlist):
    'Applies YOU-ME to a whole sentence or phrase.'
    return list(map(you_me, wordlist))
```



Implementation Issues in the Shrink (continued)

Output formatting (with functional programming)

```
from functools import reduce
def stringify(list):
    '''Create a string from the list,
       but with spaces between words.'''
    if len(list) == 0: return ""
    if len(list) == 1: return list[0]
    # If more than 1 element,
    # put spaces between adjacent pairs:
    return list[0] +\
      reduce(lambda y,z: y+z,\
             map(lambda x: ' ' + x, list[1:]))
```

```
>>> stringify(['A','big','fox'])
'A big fox'
```