# 19_left_corner_parsing_running_model_understanding_ouput

April 12, 2021

Open in Colab

The left-corner parsing model so far:

[1]:
```
import pyactr as actr

environment = actr.Environment(focus_position=(320, 180))

actr.chunktype("parsing_goal", "task stack_top stack_bottom\
                                parsed_word right_frontier")
actr.chunktype("parse_state", "node_cat mother daughter1\
                               daughter2 lex_head")
actr.chunktype("word", "form cat")

parser = actr.ACTRModel(environment, motor_prepared=True)

dm = parser.decmem
g = parser.goal
imaginal = parser.set_goal(name="imaginal", delay=0)

dm.add(actr.chunkstring(string="""
    isa  word
    form Mary
    cat  ProperN
"""))
dm.add(actr.chunkstring(string="""
    isa  word
    form Bill
    cat  ProperN
"""))
dm.add(actr.chunkstring(string="""
    isa  word
    form likes
    cat  V
"""))
```

```
g.add(actr.chunkstring(string="""
    isa             parsing_goal
    task            read_word
    stack_top       S
    right_frontier  S
"""))
```

[2]:
```
parser.productionstring(name="press spacebar", string="""
    =g>
    isa             parsing_goal
    task            read_word
    stack_top       ~None
    ?manual>
    state           free
    ==>
    =g>
    isa             parsing_goal
    task            encode_word
    +manual>
    isa             _manual
    cmd             'press_key'
    key             'space'
""")

parser.productionstring(name="encode word", string="""
    =g>
    isa             parsing_goal
    task            encode_word
    =visual>
    isa             _visual
    value           =val
    ==>
    =g>
    isa             parsing_goal
    task            get_word_cat
    parsed_word    =val
    ~visual>
""")

parser.productionstring(name="retrieve category", string="""
    =g>
    isa             parsing_goal
    task            get_word_cat
    parsed_word     =w
    ==>
    +retrieval>
    isa             word
    form             =w
    =g>
    isa              parsing_goal
    task             retrieving_word
""")

parser.productionstring(name="shift and project word", string="""
    =g>
    isa              parsing_goal
    task             retrieving_word
    stack_top        =t
    stack_bottom     None
    =retrieval>
    isa              word
    form             =w
    cat              =c
    ==>
    =g>
    isa              parsing_goal
    task             parsing
    stack_top        =c
    stack_bottom     =t
    +imaginal>
    isa              parse_state
    node_cat         =c
    daughter1        =w
    ~retrieval>
""")

parser.productionstring(name="project: NP ==> ProperN", string="""
    =g>
    isa              parsing_goal
    stack_top        ProperN
    stack_bottom     ~NP
    right_frontier   =rf
    parsed_word      =w
    ==>
    =g>
    isa              parsing_goal
    stack_top        NP
    +imaginal>
    isa              parse_state
    node_cat         NP
    daughter1        ProperN
    mother           =rf
    lex_head         =w
""")
```

```
parser.productionstring(
    name="project and complete: NP ==> ProperN",
    string="""
        =g>
        isa             parsing_goal
        stack_top       ProperN
        stack_bottom    NP
        right_frontier  =rf
        parsed_word     =w
        ==>
        =g>
        isa             parsing_goal
        task            read_word
        stack_top       None
        stack_bottom    None
        +imaginal>
        isa             parse_state
        node_cat        NP
        daughter1       ProperN
        mother          =rf
        lex_head        =w
""")

parser.productionstring(
    name="project and complete: S ==> NP VP",
    string="""
        =g>
        isa             parsing_goal
        stack_top       NP
        stack_bottom    S
        ==>
        =g>
        isa             parsing_goal
        task            read_word
        stack_top       VP
        stack_bottom    None
        right_frontier  VP
        +imaginal>
        isa             parse_state
        node_cat        S
        daughter1       NP
        daughter2       VP
""")

parser.productionstring(
    name="project and complete: VP ==> V NP",
    string="""
        =g>
        isa             parsing_goal
        task            parsing
        stack_top       V
        stack_bottom    VP
        ==>
        =g>
        isa             parsing_goal
        task            read_word
        stack_top       NP
        stack_bottom    None
        +imaginal>
        isa             parse_state
        node_cat        VP
        daughter1       V
        daughter2       NP
""")

parser.productionstring(name="finished", string="""
    =g>
    isa             parsing_goal
    task            read_word
    stack_top       None
    ==>
    ~g>
    ~imaginal>
""");
```

## 0.1 Running the model and understanding the output

Let us now run the left-corner parser on the sentence *Mary likes Bill* and examine its output. As shown by the stimuli variable below, the sentence is presented self-paced reading style with the words displayed one at a time in the center of the (virtual) screen (lines 1-3). We will specify that the simulation should be run for at most 1.5 s (the default time is 1 s, which would not suffice for this case).

[3]:
```
stimuli = [{1: {'text': 'Mary', 'position': (320, 180)}},
            {1: {'text': 'likes', 'position': (320, 180)}},
            {1: {'text': 'Bill', 'position': (320, 180)}}]
parser_sim = parser.simulation(
    realtime=True,
    gui=False,
    environment_process=environment.environment_process,
    stimuli=stimuli,
    triggers='space')
parser_sim.run(max_time=1.5)
```

```
(0, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0, 'PROCEDURAL', 'RULE SELECTED: press spacebar')
****Environment: {1: {'text': 'Mary', 'position': (320, 180)}}
(0, 'visual_location', 'ENCODED LOCATION: _visuallocation(color= , screen_x=
320, screen_y= 180, value= )')
(0.0086, 'visual', 'AUTOMATIC BUFFERING: _visual(cmd= , color= , screen_pos=
_visuallocation(color= , screen_x= 320, screen_y= 180, value= ), value= Mary)')
(0.05, 'PROCEDURAL', 'RULE FIRED: press spacebar')
(0.05, 'g', 'MODIFIED')
(0.05, 'manual', 'COMMAND: press_key')
(0.05, 'manual', 'PREPARATION COMPLETE')
(0.05, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.05, 'PROCEDURAL', 'RULE SELECTED: encode word')
(0.1, 'manual', 'INITIATION COMPLETE')
(0.1, 'PROCEDURAL', 'RULE FIRED: encode word')
(0.1, 'g', 'MODIFIED')
(0.1, 'visual', 'CLEARED')
(0.1, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.1, 'PROCEDURAL', 'RULE SELECTED: retrieve category')
(0.15, 'PROCEDURAL', 'RULE FIRED: retrieve category')
(0.15, 'g', 'MODIFIED')
(0.15, 'retrieval', 'START RETRIEVAL')
(0.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.15, 'PROCEDURAL', 'NO RULE FOUND')
(0.2, 'manual', 'KEY PRESSED: SPACE')
(0.2, 'retrieval', 'CLEARED')
****Environment: {1: {'text': 'likes', 'position': (320, 180)}}
(0.2, 'retrieval', 'RETRIEVED: word(cat= ProperN, form= Mary)')
(0.2, 'visual_location', 'ENCODED LOCATION: _visuallocation(color= , screen_x=
320, screen_y= 180, value= )')
(0.2, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.2, 'PROCEDURAL', 'RULE SELECTED: shift and project word')
(0.2087, 'visual', 'AUTOMATIC BUFFERING: _visual(cmd= , color= , screen_pos=
_visuallocation(color= , screen_x= 320, screen_y= 180, value= ), value= likes)')
(0.25, 'PROCEDURAL', 'RULE FIRED: shift and project word')
(0.25, 'g', 'MODIFIED')
(0.25, 'retrieval', 'CLEARED')
(0.25, 'imaginal', 'CLEARED')
(0.25, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= Mary, daughter2= ,
lex_head= , mother= , node_cat= ProperN)')
(0.25, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.25, 'PROCEDURAL', 'RULE SELECTED: project: NP ==> ProperN')
(0.3, 'PROCEDURAL', 'RULE FIRED: project: NP ==> ProperN')
(0.3, 'g', 'MODIFIED')
(0.3, 'imaginal', 'CLEARED')
(0.3, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= ProperN, daughter2= ,
lex_head= Mary, mother= S, node_cat= NP)')
(0.3, 'PROCEDURAL', 'CONFLICT RESOLUTION')
```

```
(0.3, 'PROCEDURAL', 'RULE SELECTED: project and complete: S ==> NP VP')
(0.35, 'manual', 'MOVEMENT FINISHED')
(0.35, 'PROCEDURAL', 'RULE FIRED: project and complete: S ==> NP VP')
(0.35, 'g', 'MODIFIED')
(0.35, 'imaginal', 'CLEARED')
(0.35, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= NP, daughter2= VP,
lex_head= , mother= , node_cat= S)')
(0.35, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.35, 'PROCEDURAL', 'RULE SELECTED: press spacebar')
(0.4, 'PROCEDURAL', 'RULE FIRED: press spacebar')
(0.4, 'g', 'MODIFIED')
(0.4, 'manual', 'COMMAND: press_key')
(0.4, 'manual', 'PREPARATION COMPLETE')
(0.4, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.4, 'PROCEDURAL', 'RULE SELECTED: encode word')
(0.45, 'manual', 'INITIATION COMPLETE')
(0.45, 'PROCEDURAL', 'RULE FIRED: encode word')
(0.45, 'g', 'MODIFIED')
(0.45, 'visual', 'CLEARED')
(0.45, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.45, 'PROCEDURAL', 'RULE SELECTED: retrieve category')
(0.5, 'PROCEDURAL', 'RULE FIRED: retrieve category')
(0.5, 'g', 'MODIFIED')
(0.5, 'retrieval', 'START RETRIEVAL')
(0.5, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.5, 'PROCEDURAL', 'NO RULE FOUND')
(0.55, 'manual', 'KEY PRESSED: SPACE')
(0.55, 'retrieval', 'CLEARED')
****Environment: {1: {'text': 'Bill', 'position': (320, 180)}}
(0.55, 'retrieval', 'RETRIEVED: word(cat= V, form= likes)')
(0.55, 'visual_location', 'ENCODED LOCATION: _visuallocation(color= , screen_x=
320, screen_y= 180, value= )')
(0.55, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.55, 'PROCEDURAL', 'RULE SELECTED: shift and project word')
(0.556, 'visual', 'AUTOMATIC BUFFERING: _visual(cmd= , color= , screen_pos=
_visuallocation(color= , screen_x= 320, screen_y= 180, value= ), value= Bill)')
(0.6, 'PROCEDURAL', 'RULE FIRED: shift and project word')
(0.6, 'g', 'MODIFIED')
(0.6, 'retrieval', 'CLEARED')
(0.6, 'imaginal', 'CLEARED')
(0.6, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= likes, daughter2= ,
lex_head= , mother= , node_cat= V)')
(0.6, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.6, 'PROCEDURAL', 'RULE SELECTED: project and complete: VP ==> V NP')
(0.65, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> V NP')
(0.65, 'g', 'MODIFIED')
(0.65, 'imaginal', 'CLEARED')
(0.65, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= V, daughter2= NP,
lex_head= , mother= , node_cat= VP)')
(0.65, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.65, 'PROCEDURAL', 'NO RULE FOUND')
(0.7, 'manual', 'MOVEMENT FINISHED')
(0.7, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.7, 'PROCEDURAL', 'RULE SELECTED: press spacebar')
(0.75, 'PROCEDURAL', 'RULE FIRED: press spacebar')
(0.75, 'g', 'MODIFIED')
(0.75, 'manual', 'COMMAND: press_key')
(0.75, 'manual', 'PREPARATION COMPLETE')
(0.75, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.75, 'PROCEDURAL', 'RULE SELECTED: encode word')
(0.8, 'manual', 'INITIATION COMPLETE')
(0.8, 'PROCEDURAL', 'RULE FIRED: encode word')
(0.8, 'g', 'MODIFIED')
(0.8, 'visual', 'CLEARED')
(0.8, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.8, 'PROCEDURAL', 'RULE SELECTED: retrieve category')
(0.85, 'PROCEDURAL', 'RULE FIRED: retrieve category')
(0.85, 'g', 'MODIFIED')
(0.85, 'retrieval', 'START RETRIEVAL')
(0.85, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.85, 'PROCEDURAL', 'NO RULE FOUND')
(0.9, 'manual', 'KEY PRESSED: SPACE')
(0.9, 'retrieval', 'CLEARED')
(0.9, 'retrieval', 'RETRIEVED: word(cat= ProperN, form= Bill)')
(0.9, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.9, 'PROCEDURAL', 'RULE SELECTED: shift and project word')
(0.95, 'PROCEDURAL', 'RULE FIRED: shift and project word')
(0.95, 'g', 'MODIFIED')
(0.95, 'retrieval', 'CLEARED')
(0.95, 'imaginal', 'CLEARED')
(0.95, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= Bill, daughter2= ,
lex_head= , mother= , node_cat= ProperN)')
(0.95, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.95, 'PROCEDURAL', 'RULE SELECTED: project and complete: NP ==> ProperN')
(1.0, 'PROCEDURAL', 'RULE FIRED: project and complete: NP ==> ProperN')
(1.0, 'g', 'MODIFIED')
(1.0, 'imaginal', 'CLEARED')
(1.0, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1= ProperN, daughter2= ,
lex_head= Bill, mother= VP, node_cat= NP)')
(1.0, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(1.0, 'PROCEDURAL', 'RULE SELECTED: finished')
(1.05, 'manual', 'MOVEMENT FINISHED')
(1.05, 'PROCEDURAL', 'RULE FIRED: finished')
(1.05, 'g', 'CLEARED')
(1.05, 'imaginal', 'CLEARED')
(1.05, 'PROCEDURAL', 'CONFLICT RESOLUTION')
```

```
(1.05, 'PROCEDURAL', 'NO RULE FOUND')
```

The model output is prodigious since the parser runs for about 1 s – a fairly realistic time for a three-word sentence.

We will now briefly examine how the cognitive process of parsing unfolds in time, and then examine the final result of the process more closely.

- at the very beginning of the simulation (0 ms), the first word of the sentence (*Mary*) is already displayed on the screen, so the visual module immediately encodes its location and retrieves the visual value (word form) at that location soon after that
- at the same time, the `"press spacebar"` rule is selected, which will ultimately trigger a key press that will reveal the second word of the sentence; this rule fires 50 ms later, initiating a manual process in the motor module.
- since the visual module already retrieved the form of the first word, we can select the `"encode word"` rule at the 50 ms point
- at 100 ms, this rule fires, taking the retrieved visual value and encoding it in the goal chunk as the word to be parsed: the chunk in the `g` (goal) buffer is modified
- the visual buffer is cleared and made available for the next word
- we can now attempt the retrieval of the encoded word from declarative memory: rule `"retrieve category"` is selected at 100 ms and takes 50 ms to fire
- at this point, i.e., at 150 ms, a retrieval process from declarative memory is started that takes 50 ms to complete
- so at 200 ms, the word *Mary* is retrieved with its syntactic category ProperN
- meanwhile, the motor module executes the movement preparation and initiation triggered by the `"press spacebar"` rule fired at 50 ms
- this happens in parallel to the parsing steps triggered by the word *Mary*.
- with the syntactic category of the first word in hand, we start a cascade of parsing rules triggered by the availability of this 'left corner'
  - first, at 200 ms, we select the `"shift and project word"` rule that fires 50 ms later and creates a chunk in the imaginal buffer storing a unary branching tree with the syntactic category ProperN as the mother node and the word *Mary* as the daughter
  - at this point (250 ms), we select the project-NP rule , which fires 50 ms later and creates a chunk in the imaginal buffer storing the next part of our tree, namely the NP node built on top of the ProperN node we previously built
  - we are now at 300 ms; we select the project-and-complete-S rule, which fires at 350 ms and creates a binary branching chunk in the imaginal buffer with S as the mother node, the previously built NP as its first daughter, and a VP as its second daughter that we expect to identify later on in the parsing process
- in parallel to these parsing actions, the motor and visual modules execute actions that lead to the second word of the sentence being displayed and read
- at 200 ms, the motor module is finally ready to press the space key, which displays the word *likes* on the screen

- the visual location of this word is immediately encoded and the visual value is retrieved soon after that
- we are therefore ready at 350 ms to select the `"press spacebar"` rule once again, which fires 50 ms later and will eventually lead to displaying and reading the final word of the sentence
- after that, at 400 ms, we are ready to encode the word *likes* that we displayed and perceived around 130 ms earlier
- the `"encode word"` rule is selected at 400 ms, the rule fires at 450 ms, and triggers the selection of the `"retrieve category"` rule
- at the same time, the motor module prepares and initiates the second spacebar press much more quickly: the preparation is instantaneous and the initiation takes 50 ms, so the space key is pressed again at 550 ms
- the final word of the sentence (*Bill*) is displayed on the screen, triggering the same visual-location encoding and visual-value retrieval steps as before
- once again, these motor and visual processes happen in parallel, so at 500 ms we are able to fire the `"retrieve category"` rule that we selected 50 ms earlier
- the process of retrieval from declarative memory takes 50 ms, so at 550 ms we have the V category of our verb *likes*
- we shift and project the verb, which leads to the creation of a chunk in the imaginal buffer projecting a V node on top of the word *likes*
- we can now select the project-and-complete-VP rule, which fires at 650 ms, creating the VP node we were expecting (based on the previously triggered S rule) on top of the V node we just built, and adding a new expectation for an object NP
- at this point (700 ms), we are in a state in which `"press spacebar"` can be selected, but since there are no more words to be read, the application of this rule will not have any effect on further parsing
- after that, the `"encode word"` rule can be selected; the rule fires 50 ms later
- we then go through the retrieval process for the word *Bill*, after which we project a ProperN node on top of it
- finally, we trigger the project-and-complete-NP rule, which completes the object NP we were expecting by recognizing that the ProperN *Bill* is that NP
- we finished parsing the sentence, so the `"finished"` rule is triggered at 1050 ms and the parsing simulation ends with the clearing of the `g` (goal) and `imaginal` buffers into declarative memory

It is instructive to inspect the parse states stored in declarative memory at the end of the simulation, shown below (sorted by time of (re)activation).

- we first sort all the contents of declarative memory
- then, we display only the `parse_state` chunks

```
[4]: sortedDM = sorted(([item[0], time]\
                       for item in dm.items() for time in item[1]),
                     key=lambda item: item[1])

     for chunk in sortedDM:
         if chunk[0].typename == "parse_state":
             print(chunk[1], "\t", chunk[0])
```

```
0.3      parse_state(daughter1= Mary, daughter2= , lex_head= , mother= ,
node_cat= ProperN)
0.35     parse_state(daughter1= ProperN, daughter2= , lex_head= Mary, mother= S,
node_cat= NP)
0.6      parse_state(daughter1= NP, daughter2= VP, lex_head= , mother= ,
node_cat= S)
0.65     parse_state(daughter1= likes, daughter2= , lex_head= , mother= ,
node_cat= V)
0.95     parse_state(daughter1= V, daughter2= NP, lex_head= , mother= ,
node_cat= VP)
1.0      parse_state(daughter1= Bill, daughter2= , lex_head= , mother= ,
node_cat= ProperN)
1.05     parse_state(daughter1= ProperN, daughter2= , lex_head= Bill, mother=
VP, node_cat= NP)
```

We see here that the ACT-R architecture places significant constraints on the construction of deep hierarchical structures like syntactic trees:

- there is no global view of the constructed tree in declarative memory, only local snapshots recording
  - immediate domination relations, or at the most
  - two stacked immediate domination relations (when the `mother` slot is specified)

The time stamps of the parse states displayed above show how the parsing process unfolded over time: - we first parsed the subject NP *Mary*, which was completed after about 300 ms (the usual time in word-by-word self-paced reading) - based on this left-corner evidence, we were able to project the S node and add a VP expectation - this expectation was confirmed when the verb *likes* was parsed, after about 300 ms more - confirming the VP expectation added an object NP expectation, whih was confirmed when the final word *Bill* was parsed after yet another 300 ms

As external observers, we can assemble a global view of the syntactic tree based on the `parse_state` chunks stored in declarative memory and their time stamps.

It is similarly instructive to see the words in declarative memory at the end of the simulation (again sorted by time of (re)activation):

```
[5]: for chunk in sortedDM:
         if chunk[0].typename == "word":
             print(chunk[1], "\t", chunk[0])
```

```
0.0      word(cat= ProperN, form= Mary)
0.0      word(cat= ProperN, form= Bill)
```

```
0.0      word(cat= V, form= likes)
0.25     word(cat= ProperN, form= Mary)
0.6      word(cat= V, form= likes)
0.95     word(cat= ProperN, form= Bill)
```

All the words were added to declarative memory at the very beginning of the simulation, and then were reactivated as the parsing process unfolded.

- the reactivations are roughly spaced at 300 ms intervals, as expected for self-paced reading tasks

This concludes our basic introduction to

- the symbolic part of the ACT-R framework
- the kind of processing models for linguistic phenomena that can be developed in this cognitive framework

Up next:

- we introduce the basic subsymbolic components of ACT-R, which enable us to provide realistic models of performance and on-line behavioral measures
- these models and their numerical parameters can be fit to experimental data in the usual way, using frequentist or Bayesian methods for data analysis

Therefore, we will npw briefly introduce Bayesian methods for data analysis, which we will then be able to deploy in the remainder of the book to estimate the subsymbolic parameters of our ACT-R processing models.

[ ]:

```
```