10_top_down_parsing_running_the_model

April 8, 2021

Model up to this point:

[1]:
```
import pyactr as actr
```

[2]:
```
actr.chunktype("parsing_goal", "stack_top stack_bottom parsed_word task")
actr.chunktype("sentence", "word1 word2 word3")
actr.chunktype("word", "form, cat")
```

[3]:
```
parser = actr.ACTRModel()
dm = parser.decmem
g = parser.goal
imaginal = parser.set_goal(name="imaginal", delay=0.2)
```

[4]:
```
g.add(actr.chunkstring(string="""
    isa parsing_goal
    task parsing
    stack_top S
"""))

imaginal.add(actr.chunkstring(string="""
    isa sentence
    word1 Mary
    word2 likes
    word3 Bill
"""))
```

[5]:
```
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
"""))
```

[6]:
```
parser.productionstring(name="expand: S ==> NP VP", string="""
    =g>
    isa parsing_goal
    task parsing
    stack_top S
    ==>
    =g>
    isa parsing_goal
    stack_top NP
    stack_bottom VP
""")

parser.productionstring(name="expand: NP ==> ProperN", string="""
    =g>
    isa parsing_goal
    task parsing
    stack_top NP
    ==>
    =g>
    isa parsing_goal
    stack_top ProperN
""")

parser.productionstring(name="expand: VP ==> V NP", string="""
    =g>
    isa parsing_goal
    task parsing
    stack_top VP
    ==>
    =g>
    isa parsing_goal
    stack_top V
    stack_bottom NP
""")

parser.productionstring(name="retrieve: ProperN", string="""
    =g>
    isa parsing_goal
    task parsing
    stack_top ProperN
```

```
    =imaginal>
    isa sentence
    word1 =w1
    ==>
    =g>
    isa parsing_goal
    task retrieving
    +retrieval>
    isa word
    form =w1
""")

parser.productionstring(name="retrieve: V", string="""
    =g>
    isa parsing_goal
    task parsing
    stack_top V
    =imaginal>
    isa sentence
    word1 =w1
    ==>
    =g>
    isa parsing_goal
    task retrieving
    +retrieval>
    isa word
    form =w1
""")

parser.productionstring(name="scan: word", string="""
    =g>
    isa parsing_goal
    task retrieving
    stack_top =y
    stack_bottom =x
    =retrieval>
    isa word
    form =w1
    cat =y
    =imaginal>
    isa sentence
    word1 =w1
    word2 =w2
    word3 =w3
    ==>
    =g>
    isa parsing_goal
```

```
    task printing
    stack_top =x
    stack_bottom None
    parsed_word =w1
    =imaginal>
    isa sentence
    word1 =w2
    word2 =w3
    word3 None
    ~retrieval>
""")

parser.productionstring(name="print parsed word", string="""
    =g>
    isa parsing_goal
    task printing
    =imaginal>
    isa sentence
    word1 ~None
    ==>
    !g>
    show parsed_word
    =g>
    isa parsing_goal
    task parsing
    parsed_word None
""")

parser.productionstring(name="done", string="""
    =g>
    isa parsing_goal
    task printing
    =imaginal>
    isa sentence
    word1 None
    ==>
    !g>
    show parsed_word
    ~imaginal>
    ~g>
""");
```

0.1 Running the model

We run the model as before: we first instantiate a simulation of the model and then run it.

[7]:
```
parser_sim = parser.simulation()
parser_sim.run()
```

```
(0, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0, 'PROCEDURAL', 'RULE SELECTED: expand: S ==> NP VP')
(0.05, 'PROCEDURAL', 'RULE FIRED: expand: S ==> NP VP')
(0.05, 'g', 'MODIFIED')
(0.05, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.05, 'PROCEDURAL', 'RULE SELECTED: expand: NP ==> ProperN')
(0.1, 'PROCEDURAL', 'RULE FIRED: expand: NP ==> ProperN')
(0.1, 'g', 'MODIFIED')
(0.1, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.1, 'PROCEDURAL', 'RULE SELECTED: retrieve: ProperN')
(0.15, 'PROCEDURAL', 'RULE FIRED: retrieve: ProperN')
(0.15, 'g', 'MODIFIED')
(0.15, 'retrieval', 'START RETRIEVAL')
(0.15, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.15, 'PROCEDURAL', 'NO RULE FOUND')
(0.2, 'retrieval', 'CLEARED')
(0.2, 'retrieval', 'RETRIEVED: word(cat= ProperN, form= Mary)')
(0.2, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.2, 'PROCEDURAL', 'RULE SELECTED: scan: word')
(0.25, 'PROCEDURAL', 'RULE FIRED: scan: word')
(0.25, 'g', 'MODIFIED')
(0.25, 'imaginal', 'MODIFIED')
(0.25, 'retrieval', 'CLEARED')
(0.25, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.25, 'PROCEDURAL', 'RULE SELECTED: print parsed word')
(0.3, 'PROCEDURAL', 'RULE FIRED: print parsed word')
parsed_word Mary
(0.3, 'g', 'EXECUTED')
(0.3, 'g', 'MODIFIED')
(0.3, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.3, 'PROCEDURAL', 'RULE SELECTED: expand: VP ==> V NP')
(0.35, 'PROCEDURAL', 'RULE FIRED: expand: VP ==> V NP')
(0.35, 'g', 'MODIFIED')
(0.35, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.35, 'PROCEDURAL', 'RULE SELECTED: retrieve: V')
(0.4, 'PROCEDURAL', 'RULE FIRED: retrieve: V')
(0.4, 'g', 'MODIFIED')
(0.4, 'retrieval', 'START RETRIEVAL')
(0.4, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.4, 'PROCEDURAL', 'NO RULE FOUND')
(0.45, 'retrieval', 'CLEARED')
(0.45, 'retrieval', 'RETRIEVED: word(cat= V, form= likes)')
(0.45, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.45, 'PROCEDURAL', 'RULE SELECTED: scan: word')
```

```
(0.5, 'PROCEDURAL', 'RULE FIRED: scan: word')
(0.5, 'g', 'MODIFIED')
(0.5, 'imaginal', 'MODIFIED')
(0.5, 'retrieval', 'CLEARED')
(0.5, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.5, 'PROCEDURAL', 'RULE SELECTED: print parsed word')
(0.55, 'PROCEDURAL', 'RULE FIRED: print parsed word')
parsed_word likes
(0.55, 'g', 'EXECUTED')
(0.55, 'g', 'MODIFIED')
(0.55, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.55, 'PROCEDURAL', 'RULE SELECTED: expand: NP ==> ProperN')
(0.6, 'PROCEDURAL', 'RULE FIRED: expand: NP ==> ProperN')
(0.6, 'g', 'MODIFIED')
(0.6, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.6, 'PROCEDURAL', 'RULE SELECTED: retrieve: ProperN')
(0.65, 'PROCEDURAL', 'RULE FIRED: retrieve: ProperN')
(0.65, 'g', 'MODIFIED')
(0.65, 'retrieval', 'START RETRIEVAL')
(0.65, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.65, 'PROCEDURAL', 'NO RULE FOUND')
(0.7, 'retrieval', 'CLEARED')
(0.7, 'retrieval', 'RETRIEVED: word(cat= ProperN, form= Bill)')
(0.7, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.7, 'PROCEDURAL', 'RULE SELECTED: scan: word')
(0.75, 'PROCEDURAL', 'RULE FIRED: scan: word')
(0.75, 'g', 'MODIFIED')
(0.75, 'imaginal', 'MODIFIED')
(0.75, 'retrieval', 'CLEARED')
(0.75, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.75, 'PROCEDURAL', 'RULE SELECTED: done')
(0.8, 'PROCEDURAL', 'RULE FIRED: done')
parsed_word Bill
(0.8, 'g', 'EXECUTED')
(0.8, 'imaginal', 'CLEARED')
(0.8, 'g', 'CLEARED')
(0.8, 'PROCEDURAL', 'CONFLICT RESOLUTION')
(0.8, 'PROCEDURAL', 'NO RULE FOUND')
```

The parser runs as expected, and we successfully parse our three-word sentence.

The time course of the parsing is as follows: - the first word *Mary* is parsed at the 250 ms mark when the `scan: word` rule is fired for the first time (lines 22-25) - the word is printed after 300 ms of simulation time have elapsed (line 29) - the second word *likes* is parsed at the 500 ms mark when the `scan: word` rule is fired for the second time (lines 47-50) - the word is printed after 550 ms of total simulation time (line 54)
- the final word *Bill* is parsed at the 750 ms mark when the `scan: word` rule is fired for the third and final time (lines 72-75) - the word is printed after 800 ms of simulation time have passed (line 79)

Let's examine the content of the declarative memory module at the end of the simulation. It should contain: - the lexical items we added at the very beginning of the simulation - the chunks stored in the goal and imaginal buffers right before we cleared them at the end of the parsing process - **recall that clearing the buffers always moves their contents to declarative memory**

[8]:
```
dm
```

[8]:
```
{word(cat= ProperN, form= Mary): array([0.  , 0.25]), word(cat= ProperN, form=
Bill): array([0.  , 0.75]), word(cat= V, form= likes): array([0. , 0.5]),
sentence(word1= None, word2= None, word3= None): array([0.8]),
parsing_goal(parsed_word= Bill, stack_bottom= None, stack_top= None, task=
printing): array([0.8])}
```

[9]:
```
for idx, chunk in enumerate(dm):
    print(f"{idx+1}: {chunk}")
```

```
1: word(cat= ProperN, form= Mary)
2: word(cat= ProperN, form= Bill)
3: word(cat= V, form= likes)
4: sentence(word1= None, word2= None, word3= None)
5: parsing_goal(parsed_word= Bill, stack_bottom= None, stack_top= None, task=
printing)
```

- the goal chunk stored in declarative memory has an empty stack – since we're at the end of the parsing process
- the imaginal chunk has an empty sentence (no words) – since we're at the end of the parsing process
- both these chunks have been stored / activated in memory at the 800 ms mark, i.e., at the end of the simulation
- we also see the three lexical items *Mary*, *likes* and *Bill*, each of which has two activation time stamps:
  - one at 0 ms when they were added to declarative memory before running the simulation
  - one at 250, 500 and 750 ms respectively, when they were parsed during the simulation and the retrieval buffer was cleared by the three firings of the `scan: word` rule

Later on when we discuss the inner workings of declarative memory, we'll see how this schedule of activations for items in memory is a crucial component of determining the relative salience of the item in memory, and therefore how easy they are to retrieve.

[ ]:
```

```