12_HOMEWORK1_extending_the_top_down_parser

April 21, 2021

0.1 Homework 1: extending the grammar and the top-down parser

Consider extending our grammar and the top-down parser with intransitive verbs like *sleeps*, as well as sentential-complement taking verbs like *believe*. That is, add the following phrase-structure and lexical-insertion rules to the grammar:

$$
\begin{array}{lcl}
\text{VP} & \rightarrow & \text{V} \\
\text{VP} & \rightarrow & \text{V CP} \\
\text{CP} & \rightarrow & \text{C S} \\
\text{V} & \rightarrow & \text{sleeps} \\
\text{V} & \rightarrow & \text{believes} \\
\text{C} & \rightarrow & \text{that}
\end{array}
$$

Add the new lexical items to declarative memory and add new production rules to procedural memory to encode the new phrase-structure rules.

Once your model is in place, parse the sentence *Mary believes that Bill sleeps*.

You can probably already see that the new parser might run into problems.

For example, the parser might get stuck when parsing the target sentence *Mary believes that Bill sleeps* if it decides to

- expand the first (matrix-clause) VP into V and NP, i.e., if it incorrectly expects a transitive verb instead of a sentential-complement taking verb
- or expand that same VP into just V, i.e., if it incorrectly expects an intransitive verb.

As already discussed, this is a typical issue with top-down parsing: categories and structures are hypothesized / predicted before seeing any evidence for them.

The extended top-down parser has several ways to expand VPs and it fails to parse the input if it uses a VP expansion rule that happens to be incompatible with the sentence to be parsed.

0.2 Important hints

To enable random rule selection so that we can see how the top-down parser runs into problems, we have to enable the subsymbolic component of ACT-R/`pyactr`. We will discuss this subsym-

bolic component later on, for now, just initialize your parser as shown below, and make sure you run the simulation for at least 2 seconds:

[ ]:
```
import pyactr as actr

actr.chunktype("parsing_goal", "stack_top stack_bottom parsed_word task")
actr.chunktype("sentence", "word1 word2 word3 word4 word5")
actr.chunktype("word", "form, cat")
```

[ ]:
```
parser = actr.ACTRModel(subsymbolic=True, utility_noise=0.1,␣
 ↪retrieval_threshold=-10, decay=0.001)

dm = parser.decmem
g = parser.goal

imaginal = parser.set_goal(name="imaginal", delay=0.2)
```

- now add:
  - the necessary word chunks to declarative memory
  - the necessary production rules to declarative memory
  - the correct starting chunk to the goal buffer
  - the correct starting chunk (that is, sentence) to the imaginal buffer
- when you're done, run the simulation as shown below

[ ]:
```
parsing = parser.simulation()
parsing.run(2)
```

[ ]:
```

```