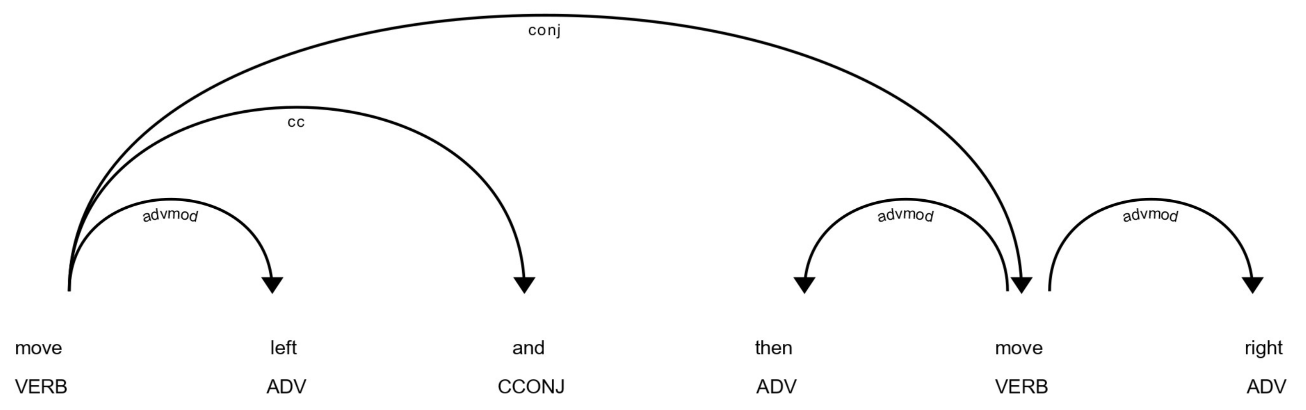
In [ ]:
```python
import spacy
from spacy.symbols import *
from spacy import displacy
import numpy as np

# spacy download en_core_web_sm
# nlp = spacy.load("en_core_web_sm")

# spacy download en_core_web_md
# nlp = spacy.load ("en_core_web_md")

# spacy download en_core_web_lg
# nlp = spacy.load ("en_core_web_lg")

# spacy download en_core_web_trf
nlp = spacy.load ("en_core_web_trf")


from gensim.models import KeyedVectors

model = "C:\\Users\\Unkow\\Downloads\\GoogleNews-vectors-negative300.bin.gz"
import gensim
from gensim import models
from gensim.models import Word2Vec

# Load pretrained model (since intermediate data is not included, the model cannot be refined with additional data)
#model = Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True, norm_only=True) -> Deprecated
# model = gensim.models.KeyedVectors.load_word2vec_format(model, binary=True) #without *norm_only* param

# model.init_sims(replace=True)
# model.save("model/GoogleNews")

model = KeyedVectors.load ("model/GoogleNews", mmap='r')

command_map = {
            'move': {
                'stop': ['move 0', 'strafe 0'],
                'forward': 'move 1', 'back': 'move -1', 'backwards': 'move -1', 'right': 'strafe 1', 'left': 'strafe -1',
                'north': 'movenorth 1', 'south': 'movesouth 1', 'east': 'moveeast 1', 'west': 'movewest 1',
                'to': 'LIST OF ENTITIES/OBJECTS'
            },
            'jump': {
                '': 'jump 1', 'up': 'jump 1', 'stop': ['jump 0'], 'forward': 'jumpmove 1', 'back': 'jumpmove -1',
                'backwards': 'jumpmove -1', 'right': 'jumpstrafe 1', 'left': 'jumpstrafe -1',
                'north': 'jumpnorth 1', 'south': 'jumpsouth 1', 'east': 'jumpeast 1', 'west': 'jumpwest 1'
            },
            'strafe': {'right': 'strafe 1', 'left': 'strafe -1', 'stop': ['strafe 0']},
            'look': {'up': 'look -1', 'down': 'look 1'},
            'pitch': {'up': 'pitch -1', 'down': 'pitch 1', 'stop': ['pitch 0']},
            'turn': {'right': 'turn 1', 'left': 'turn -1', 'stop': ['turn 0', 'pitch 0'],
                     'up': 'pitch -1', 'down': 'pitch 1'},
            'crouch': {'': 'crouch 1', 'stop': ['crouch 0']},
            'attack': {'': 'attack 1', 'stop': ['attack 0']},
            'use': {},
            'stop': ['move 0', 'jump 0', 'turn 0', 'strafe 0', 'pitch 0', 'crouch 0', 'attack 0'],
            'get': {'': 'LIST OF ENTITIES/OBJECTS'},
            'discard': {'': 'discardCurrentItem'},
            'quit': {'': 'quit'}
        }

def word_similarity_score (word1, word2):
    score = model.similarity (word1, word2)
    return score

def get_best_match (word, commands_map, threshold):
    scores = []
    for commands in commands_map:
        score = word_similarity_score (word, commands)
        scores.append (score)
    keys = list (commands_map.keys())
    command = keys[np.argmax (scores)]
    return command

def get_similar_command (verb, commands_map):
    lemma_verb = verb.lemma_
    # synonyms?
    # left
    if verb.pos != VERB:
        return verb
    # get synonym?
    t = 0.5
    command = get_best_match (lemma_verb, commands_map, t)
    return command

def send_command (verb, commands_map):
    command = commands_map.get (verb).get('')
    print ("Command: ", command)

def send_command_option (verb, option, commands_map):
```

```python
def send_command_option (verb, option, commands_map):
    command = commands_map.get (verb).get (option.lemma_)
    print ("Command: ", command)

def send_prop_command (verb, prep, commands_map):
    if prep.lemma_ in commands_map.get ("move"):
        for r in prep.rights:
            if r.lemma_ in commands_map.get ("move"):
                send_command_option (verb, r, commands_map)

def send_object_command (verb, object, commands_map):
    # objString = getObjectString (object)

    print ("Command: TODO: ", verb, "object: ", object)

    if verb == 'use':
        # hotkey = getHotKeyForItem (objString)
        print
    elif verb == 'attack':
        pass
    elif verb == 'grab':
        pass

def send_stop_command (commands_map):
    command = commands_map.get ("stop")
    print ("Command: ", command)

def parse_commmand (verb, commands_map):
    malmo_command = get_similar_command (verb, commands_map)
    print ("malmo: ", malmo_command)
    for word in verb.rights:
        print (word, word.pos_)
        if word.pos == CCONJ:
            send_command (malmo_command, commands_map)
        if word.pos == ADV:
            # move forward
            # move backwards
            if word.lemma_ in commands_map.get (malmo_command):
                # print (command_map[malmo_command][word.lemma_])
                send_command_option (malmo_command, word, commands_map)
        elif word.pos == NOUN or word.pos == PROPN:
            # move 1 block forward
            # left registers
            if word.lemma_ == "left" and word.lemma_ in commands_map.get (malmo_command):
                send_command_option (malmo_command, word, commands_map)
            else:
                send_object_command (malmo_command, word, commands_map)
        elif word.pos == ADP:
            # preposition object
            # move to the left
            # move to the right
            # move to
            send_prop_command (malmo_command, word, commands_map)
        elif word.pos == VERB:
            # subsequent command
            # move forward and dig
            if verb.lemma_ == "stop":
                send_stop_command (commands_map)
            else:
                parse_commmand (word, commands_map)

def parse_command (doc, commands_map):
    for sentence in doc.sents:
            r = sentence.root
            parse_commmand (r, commands_map)
```

In [ ]:

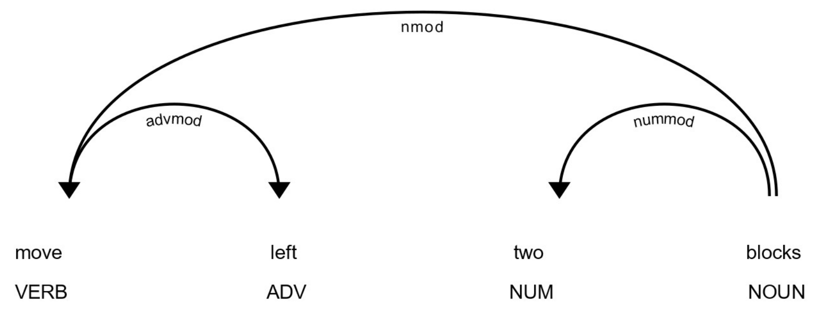
```python
doc = nlp ("move left and then move right")
displacy.render (doc, style = "dep")
parse_command (doc, command_map)
```

```
c:\Users\Unkow\miniconda3\envs\malmo3.6\lib\site-packages\torch\autocast_mode.py:141: UserWarning: User provided device_type of 'cuda', but CUDA is not available. Disabling
  warnings.warn('User provided device_type of \'cuda\', but CUDA is not available. Disabling')
```



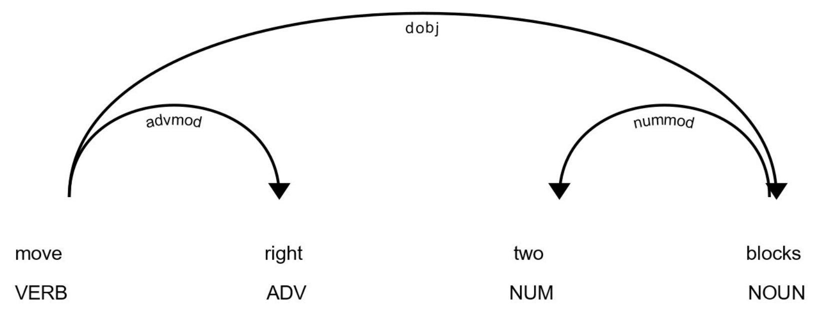
```
malmo:  move
left ADV
Command:  strafe -1
and CCONJ
Command:  None
move VERB
malmo:  move
right ADV
Command:  strafe 1
```

In [ ]:
```
doc = nlp ("move left two blocks")
displacy.render (doc, style = "dep")
parse_command (doc, command_map)
```



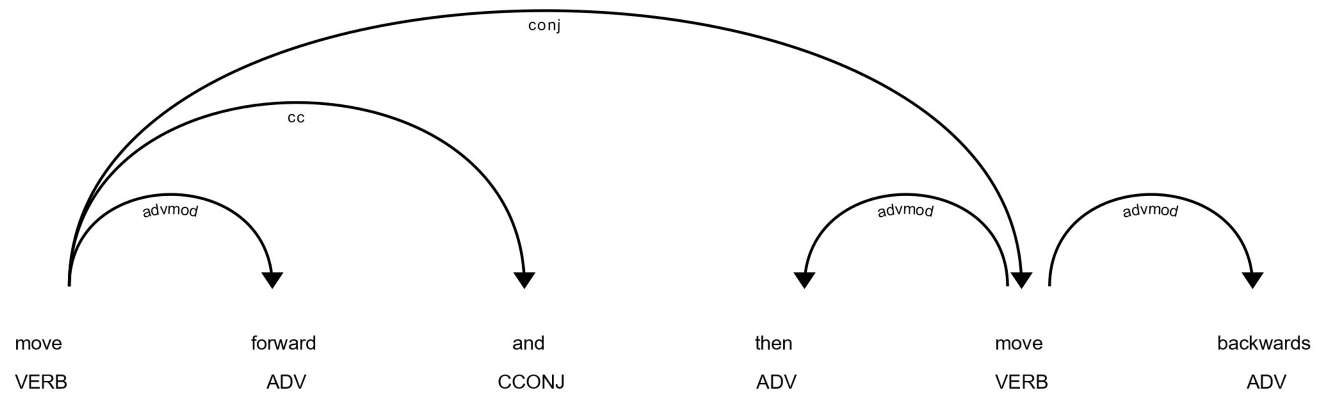
```
malmo:  blocks
```

In [ ]:
```
doc = nlp ("move right two blocks")
displacy.render (doc, style = "dep")
parse_command (doc, command_map)
```



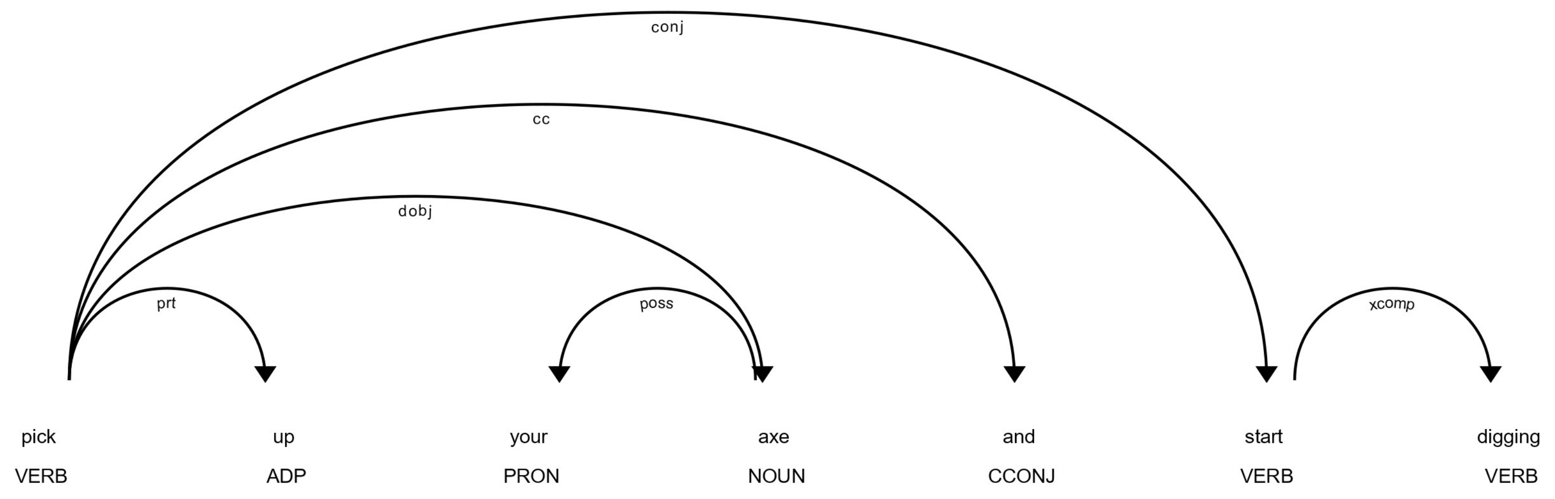
```
malmo:  move
right ADV
Command:  strafe 1
blocks NOUN
Command: TODO:  move object:  blocks
```

In [ ]:
```
doc = nlp ("move forward and then move backwards")
displacy.render (doc, style = "dep")
parse_command (doc, command_map)
```



```
malmo:  move
forward ADV
Command:  move 1
and CCONJ
Command:  None
move VERB
malmo:  move
backwards ADV
Command:  move -1
```

In [ ]:
```
doc = nlp ("pick up your axe and start digging")
displacy.render (doc, style = "dep")
parse_command (doc, command_map)
```



```
malmo:  get
up ADP
axe NOUN
Command: TODO:  get object:  axe
and CCONJ
Command:  LIST OF ENTITIES/OBJECTS
start VERB
malmo:  stop
digging VERB
malmo:  look
```