An ACT-R based left-corner style DRT parser:
General design considerations and an implementation in Python ACT-R

Adrian Brasoveanu, Jakub Dotlačil

May 9, 2015

Contents

Note: this is a very early draft, with many parts still missing. It was posted online mainly as a proof of concept associated with the authors' SALT 25 talk. Please email the authors for a more recent version if you are interested in this work, e.g., you want to cite it.

1 Short introduction

This is a left-corner style parser for both syntactic structures and DRSs. The parsing style for the syntactic component is very close in spirit to Lewis and Vasishth (2005).

The parser moves between three main states:

- *scanning*: gets the next word of the input sentence and/or starts a cataphora resolution search
- *parsing*: builds phrases / parse trees, adds them to declarative memory and adds new goals to goal stack; the *scanning* and *parsing* states are interspersed
- *END*: the end of the input sentence is reached and the parser stops

The production rules of the ACT-R agent, i.e., the parsing actions, are usually conditioned on:

- the current chunk in the lexical memory buffer, which stores the lexical representation of the most recently scanned word
- the current top parsing goal, which is (basically) the most recently parsed / predicted non-terminal node (together with its DRT semantics).

When semantics is added, we add a θ buffer for argument linking:

- the dref introduced/retrieved by the subject NP needs to be linked to / identified with the subject position argument of the following verb, so the dref chunk will be stored in the θ buffer
- for transitive verbs, the object position argument of the transitive verb needs to be linked to / identified with the dref introduced/retrieved by the object NP; in this case, it is simpler to place the entire VP chunk in the theta buffer

Production rules have 3 parts (in this order):

- add chunks to lexical memory (`LexM`) and declarative memory (`DM`)
- set goal (buffer and stack)
- print statements summarizing everything above

We already hinted at the fact that we use 2 kinds of drefs:

- `new_dref` stores discourse referents in the traditional DRT / dynamic semantics sense. These drefs are introduced by indefinites and proper names (PNs) and they get stored in the context DRS
- `arg_dref` are used for argument linking / 'theta identification'; they are introduced by lexical items and they are the argument position placeholders that need to be resolved as a matter of sub-clausal level composition; they are NEVER stored in the context DRS, they are only stored in the lexical items or in incrementally constructed phrases that are still under development; they are exclusively used to link the new drefs introduced by indef. determiners and PNs and they corresponding argument positions in N, V (intrans. or trans.), etc.

The code itself (only sparsely commented for now) is included in the appendix.

The following two sections run 2 simulations, one for a sentence with cataphora + *if*, the other for the same sentence except *if* is replaced with *and*. We see that we predict a difference in the correct direction: *and* takes longer than *if*.

Note however that this is obtained with particular values for the ACT-R subsymbolic parameters. A comprehensive investigation of alternative ACT-R models and alternative settings for the subsymbolic parameters is ongoing. We hope to report its results in the near future, as well as provide an in-depth discussion of / introduction to the code included in the appendix of this document.

2 Simulation: *John won't-eat it if a hamburger is-overcooked.*

```
[py1] >>> from DRT_parser import *

      >>> example = "John won't-eat it if a hamburger is-overcooked"
      >>> simulation = DRTparser(example)

      >>> simulation.run()

      >> INITIALIZING lex. mem. (adding lex. info for words)
      >> INITIALIZING goal stack
         -- top goal: scan first word
         -- next goal: predicting an S & an empty DRS
```

```
>> [TIME: 50 ms] Scanned and requested lex. info for ’ John ’; moving to next goal


>> Attaching NP, the word ’ John ’, and predicting VP
>> Top goal: scan next word; next goal: the predicted VP
>> [TIME: 150 ms] Current parse tree and context DRS in declarative mem.:

-S /NP/-John
 _________
| u       |
|---------|
| john(u) |
|_________|


>> [TIME: 200 ms] Scanned and requested lex. info for ’ won’t-eat ’; moving to next goal


>> Placing VP in the theta buffer for linking with the object dref
>> VP not in DM yet, so VP info missing in the parse tree and DRS below
>> Attaching V (trans.), the word ’ won’t-eat ’, and predicting NP
>> Top goal: scan next word; next goal: the predicted NP
>> [TIME: 300 ms] Current parse tree and context DRS in declarative mem.:

-S /NP/-John
 _________
| u       |
|---------|
| john(u) |
|_________|


>> [TIME: 350 ms] Scanned and requested lex. info for ’ it ’; moving to next goal


>> [TIME: 450 ms] Marked the DRS of the pronoun in the old goal.
>> Top goal: recall pronoun antecedent in current DRS


>> Placed DM request for a pronoun antecedent in current DRS
>> [TIME: 500 ms] Top goal: Check if any antecedent was found in current DRS


>> [TIME: 550 ms] No antec. found; no parent DRS to move to; marking the antec. as unknown


>> Marking the result of pron. resolution in the lex. buffer
>> [TIME: 600 ms] Top goal: the goal before the pron. resolution
>> Top goal: scan next word
>> [TIME: 650 ms] Current parse tree and context DRS in declarative mem.:

      /NP/-it
   /VP
-S|   \V /-won’t-eat
  |
   \NP/-John
 -------------------
| u v               |
|-------------------|
| will_not_eat(u,v) |
| john(u)           |
| unknown(v)        |
|___________________|


>> [TIME: 700 ms] Scanned and requested lex. info for ’ if ’; moving to next goal


>> Attaching CP, C, the word ’ if ’, and predicting S
>> Top goal: scan next word; next goal: the predicted S
>> [TIME: 800 ms] Current parse tree and context DRS in declarative mem.:

      /-S
   /CP
  |   \C /-if
-S|
  |      /NP/-it
  |   /VP
   \S|   \V /-won’t-eat
     |
      \NP/-John
 ---------------------------------
|                                 |
|---------------------------------|
|   __      -------------------   |
|  |  |    | u v               |  |
| (|--| -> |-------------------|) |
|  |__|    | will_not_eat(u,v) |  |
|          | john(u)           |  |
|          | unknown(v)        |  |
|          |___________________|  |
|_________________________________|


>> [TIME: 850 ms] Scanned and requested lex. info for ’ a ’; moving to next goal
>> Attaching Det, the word ’ a ’, and predicting N and VP
>> Top goal: scan next word; next goal: the predicted N
>> [TIME: 950 ms] Current parse tree and context DRS in declarative mem.:

      /S /NP/Det-a
   /CP
  |   \C /-if
-S|
  |      /NP/-it
  |   /VP
   \S|   \V /-won’t-eat
     |
      \NP/-John
 ----------------------------------
|                                  |
|----------------------------------|
|   ___      ___________________   |
|  | w |    | u v               |  |
| (|---| -> |-------------------|) |
|  |___|    | will_not_eat(u,v) |  |
|           | john(u)           |  |
|           | unknown(v)        |  |
|           |___________________|  |
|__________________________________|


>> [TIME: 1000 ms] Scanned and requested lex. info for ’ hamburger ’; moving to next goal


>> Attaching N, and the word ’ hamburger ’
>> Top goal: scan next word
>> [TIME: 1100 ms] Current parse tree and context DRS in declarative mem.:

            /N /-hamburger
      /S /NP
   /CP      \Det-a
  |  |
  |   \C /-if
-S|
  |      /NP/-it
  |   /VP
   \S|   \V /-won’t-eat
     |
      \NP/-John
 ---------------------------------------------
|                                             |
|---------------------------------------------|
|   ______________      ___________________   |
|  | w            |    | u v               |  |
| (|--------------| -> |-------------------|) |
|  | hamburger(w) |    | will_not_eat(u,v) |  |
|  |______________|    | john(u)           |  |
|                      | unknown(v)        |  |
|                      |___________________|  |
|_____________________________________________|


>> [TIME: 1150 ms] Saved current goal; placed DM request for the cataphoric NP
>> Crucial START time: 1.15


>> [TIME: 1426 ms] Placed DM request for the original DRS of the cataphora


>> [TIME: 1526 ms] Setting parent DRS as current DRS; DM request for a cata antec. in current


>> Found cata antec.; marking it in the lex. buffer and adding it to DM
>> Crucial STOP time: 1.62616356782
>> [TIME: 1626 ms] Top goal: scan next word


>> [TIME: 1676 ms] Scanned and requested lex. info for ’ is-overcooked ’; moving to next goal


>> Attaching VP, V (intrans.), and the word ’ is-overcooked ’
>> Top goal: scan next word
>> [TIME: 1776 ms] Current parse tree and context DRS in declarative mem.:

         /VP/V /-is-overcooked
      /S|
     |  |   /N /-hamburger
   /CP   \NP
  |  |      \Det-a
  |  |
-S|   \C /-if
  |
  |      /NP/-it
  |   /VP
   \S|   \V /-won’t-eat
     |
      \NP/-John
 ----------------------------------------------
|                                              |
|----------------------------------------------|
|   _______________      ___________________   |
|  | w             |    | u v               |  |
| (|---------------| -> |-------------------|) |
|  | overcooked(w) |    | (v = w)           |  |
|  | hamburger(w)  |    | will_not_eat(u,v) |  |
|  |_______________|    | john(u)           |  |
|                       |___________________|  |
|______________________________________________|


>> DONE!

>> FINAL parse tree and context DRS in declarative mem.:

         /VP/V /-is-overcooked
      /S|
     |  |   /N /-hamburger
   /CP   \NP
  |  |      \Det-a
  |  |
-S|   \C /-if
  |
  |      /NP/-it
  |   /VP
   \S|   \V /-won't-eat
     |
      \NP/-John
 ______________________________________________
|                                              |
|----------------------------------------------|
|   _______________      ___________________   |
|  | w             |    | u v               |  |
| (|---------------| -> |-------------------|) |
|  | overcooked(w) |    | (v = w)           |  |
|  | hamburger(w)  |    | will_not_eat(u,v) |  |
|  |_______________|    | john(u)           |  |
|                       |___________________|  |
|______________________________________________|

Crucial STOP-START time (rounded to 3 digits): 0.476
>>> ccm.finished()
```

3 Simulation: *John won't-eat it and a hamburger is-overcooked.*

[py2]
```
>>> from DRT_parser import *

>>> example = "John won't-eat it and a hamburger is-overcooked"
>>> simulation = DRTparser(example)

>>> simulation.run()

>> INITIALIZING lex. mem. (adding lex. info for words)
>> INITIALIZING goal stack
   -- top goal: scan first word
   -- next goal: predicting an S & an empty DRS


>> [TIME: 50 ms] Scanned and requested lex. info for ’ John ’; moving to next goal


>> Attaching NP, the word ’ John ’, and predicting VP
>> Top goal: scan next word; next goal: the predicted VP
>> [TIME: 150 ms] Current parse tree and context DRS in declarative mem.:

-S /NP/-John
 _________
| u       |
|---------|
| john(u) |
|_________|


>> [TIME: 200 ms] Scanned and requested lex. info for ’ won’t-eat ’; moving to next goal


>> Placing VP in the theta buffer for linking with the object dref
>> VP not in DM yet, so VP info missing in the parse tree and DRS below
>> Attaching V (trans.), the word ’ won’t-eat ’, and predicting NP
>> Top goal: scan next word; next goal: the predicted NP
>> [TIME: 300 ms] Current parse tree and context DRS in declarative mem.:

-S /NP/-John
 _________
| u       |
|---------|
| john(u) |
|_________|


>> [TIME: 350 ms] Scanned and requested lex. info for ’ it ’; moving to next goal


>> [TIME: 450 ms] Marked the DRS of the pronoun in the old goal.
>> Top goal: recall pronoun antecedent in current DRS


>> Placed DM request for a pronoun antecedent in current DRS
>> [TIME: 500 ms] Top goal: Check if any antecedent was found in current DRS


>> [TIME: 550 ms] No antec. found; no parent DRS to move to; marking the antec. as unknown


>> Marking the result of pron. resolution in the lex. buffer
>> [TIME: 600 ms] Top goal: the goal before the pron. resolution


>> Top goal: scan next word
>> [TIME: 650 ms] Current parse tree and context DRS in declarative mem.:

      /NP/-it
   /VP
-S|   \V /-won’t-eat
  |
   \NP/-John
 -------------------
| u v               |
|-------------------|
| will_not_eat(u,v) |
| john(u)           |
| unknown(v)        |
|___________________|


>> [TIME: 700 ms] Scanned and requested lex. info for ’ and ’; moving to next goal


>> Attaching Conj, the word ’ and ’, and predicting S
>> Top goal: scan next word; next goal: the predicted S
>> [TIME: 800 ms] Current parse tree and context DRS in declarative mem.:

     /-S
    |
    |-Conj-and
-ConjP
    |      /NP/-it
    |   /VP
     \S|   \V /-won’t-eat
       |
        \NP/-John
 -------------------
| u v               |
|-------------------|
| will_not_eat(u,v) |
| john(u)           |
| unknown(v)        |
|___________________|


>> [TIME: 850 ms] Scanned and requested lex. info for ’ a ’; moving to next goal


>> Attaching Det, the word ’ a ’, and predicting N and VP
>> Top goal: scan next word; next goal: the predicted N
>> [TIME: 950 ms] Current parse tree and context DRS in declarative mem.:

     /S /NP/Det-a
    |
    |-Conj-and
-ConjP
    |      /NP/-it
    |   /VP
     \S|   \V /-won’t-eat
       |
        \NP/-John
 -------------------
| u v w             |
|-------------------|
| will_not_eat(u,v) |
| john(u)           |
| unknown(v)        |
|___________________|


>> [TIME: 1000 ms] Scanned and requested lex. info for ’ hamburger ’; moving to next goal


>> Attaching N, and the word ’ hamburger ’
>> Top goal: scan next word
>> [TIME: 1100 ms] Current parse tree and context DRS in declarative mem.:

           /N /-hamburger
     /S /NP
    |      \Det-a
    |
-ConjPConj-and
    |
    |      /NP/-it
    |   /VP
     \S|   \V /-won’t-eat
       |
        \NP/-John
 -------------------
| u v w             |
|-------------------|
| will_not_eat(u,v) |
| john(u)           |
| hamburger(w)      |
| unknown(v)        |
|___________________|


>> [TIME: 1150 ms] Saved current goal; placed DM request for the cataphoric NP
>> Crucial START time: 1.15
>> [TIME: 1443 ms] Placed DM request for the original DRS of the cataphora


>> [TIME: 1543 ms] Setting parent DRS as current DRS; DM request for a cata antec. in current


>> [TIME: 1593 ms] No cata antec. found; placing DM request for the parent of the current DRS.


>> No parent DRS to move to.
>> Crucial STOP time: 1.64270994513
>> Moving to the goal before cata search started.


>> [TIME: 1693 ms] Scanned and requested lex. info for ’ is-overcooked ’; moving to next goal


>> Attaching VP, V (intrans.), and the word ’ is-overcooked ’
>> Top goal: scan next word
>> [TIME: 1792 ms] Current parse tree and context DRS in declarative mem.:

        /VP/V /-is-overcooked
     /S|
    |  |   /N /-hamburger
    |   \NP
    |      \Det-a
-ConjP
    |-Conj-and
    |
    |      /NP/-it
    |   /VP
     \S|   \V /-won’t-eat
       |
        \NP/-John
 -------------------
| u v w             |
|-------------------|
| overcooked(w)     |
| will_not_eat(u,v) |
| john(u)           |
| hamburger(w)      |
| unknown(v)        |
|___________________|


>> DONE!

>> FINAL parse tree and context DRS in declarative mem.:
        /VP/V /-is-overcooked
     /S|
    |  |   /N /-hamburger
    |   \NP
    |      \Det-a
-ConjP
    |-Conj-and
    |
    |      /NP/-it
    |   /VP
     \S|   \V /-won’t-eat
       |
        \NP/-John
 -------------------
| u v w             |
|-------------------|
| overcooked(w)     |
| will_not_eat(u,v) |
| john(u)           |
| hamburger(w)      |
| unknown(v)        |
|___________________|

Crucial STOP-START time (rounded to 3 digits): 0.493
>>> ccm.finished()
```

References

Lewis, Richard and Shravan Vasishth (2005). “An activation-based model of sentence processing as skilled memory retrieval”. In: *Cognitive Science* 29, pp. 1–45.

A The main file

(1) File **DRT_parser.py**:

```python
import ccm
from ccm.lib.actr import *
from collections import deque


class MotorModule(ccm.Model):
    def __init__(self, input_sent=None):
        ccm.Model.__init__(self)
        self.current_wd_pos = -1
    def read_next_word(self):
        self.current_wd_pos += 1
        return self.parent.sent[self.current_wd_pos]


class DRTparser(ACTR):
    def __init__(self, input_sent=None):
        ACTR.__init__(self)
        self.sent = input_sent.split() + ["END"]

    START = 0
    STOP = 0

    goal = Buffer()
    goalstack = deque()

    motor = MotorModule()

    LexMBuffer = Buffer()
    LexM = Memory(LexMBuffer, latency=0.05, threshold=-2, maximum_time=1)

    DMBuffer = Buffer()
    DM = Memory(DMBuffer, latency=0.05, threshold=-2, maximum_time=1)

    #dm_n = DMNoise(DM, noise=0.05, baseNoise=0.05)
    dm_n = DMNoise(DM, noise=0.0, baseNoise=0.0)
    dm_bl = DMBaseLevel(DM, decay=0.5, limit=None)

    dm_spread = DMSpreading(DM, goal)
    # set strength of activation for buffers
    dm_spread.strength = 1.95
    # set weight to adjust for how many slots in the buffer; usually this is strength divided by num
    dm_spread.weight[goal] = 1.2

    # add a theta buffer to do the argument linking from SU to VP, and from trans. V to DO
    thetaBuffer = Buffer()

    ind_dref_list = deque(["z", "y", "x", "w", "v", "u"]) # drefs for individuals
    event_dref_list = deque(["k", "j", "i", "h", "g", "f", "c", "b", "a"]) # drefs for events

    from DRT_parser_chunks import pn_default, pro_default, conj_default, tv_default, p_default, iv_de
            NP_word, N_word, Det_word, V_word, Pro_word, Conj_word, \
            Proper_NP_into_DM, NP_into_DM, Dep_Proper_NP_into_DM, Dep_NP_into_DM, S_into_DM, VP_into_
            parsing_goal, scan_next_word, VP_into_goal, NP_into_goal, N_into_goal, recall_DRS_goal, r
            VP_chunk_theta_buffer, dref_chunk_theta_buffer, \
            recalled_DRS, recalled_antec, \
            S_new_root_into_DM, S_goal_second_conjunct, S_goal_antec, S_reanalyzed_as_first_conjunct
            start_cata_before_scan, start_cata_goal, resolving_cata, recall_cata_goal, recalled_cata_

    from DRT_parser_productions import init, \
            scan_word, \
            attach_NP_as_subject, attach_Det_as_subject, \
            attach_N, \
            attach_IV, attach_TV, \
            attach_NP_as_object, attach_Det_as_object, \
            change_goal_to_recalling, recall_DRS_antec, recall_pro_antec, change_drs, match_anteceden
            attach_and_as_S_conjunction, attach_if_as_S_conjunction, \
            stop, \
            get_new_ind_dref, get_new_event_dref, \
            start_cata, cata_resolution, recall_cata_antec_with_recalled_DRS, recall_cata_antec_no_re

    from DRT_parser_draw_tree_and_drs import draw_parse_and_drs
```

B Chunks

(2) File **DRT_parser_chunks.py**:

```
# these defaults are used for lexical entries
pn_default = "cat:NP pro:no new_ind_dref:d1 arg_dref:d1"
pro_default = "cat:NP pro:yes new_ind_dref:None arg_dref:None cond:None"
conj_default = "cat:Conj subcat:S new_ind_dref:None arg_dref:None"
tv_default = "cat:V new_ind_dref:None arg_dref:d1,d2"
p_default = "cat:P subcat:None new_ind_dref:None arg_dref:None cond:"
iv_default = "cat:V subcat:intrans new_ind_dref:None arg_dref:d1"
det_default = "cat:Det arg_dref:None cond:"
n_default = "cat:N new_ind_dref:None arg_dref:d1"
adv_default = "cat:Adv new_ind_dref:None arg_dref:None"

# these chunks are used in production rules
# 1. LexM chunks -- the new_ind_dref value is called dummy_ind_dref because it's still a place holde
NP_word = "phon:?word cat:NP gender:?g new_ind_dref:?dummy_ind_dref arg_dref:?arg_dref cond:!None?con
N_word = "phon:?word cat:N gender:?g new_ind_dref:?dummy_ind_dref arg_dref:?arg_dref cond:?cond"
Det_word = "phon:?word cat:Det new_ind_dref:?dummy_ind_dref arg_dref:?arg_dref cond:?cond"
V_word = "phon:?word cat:V new_ind_dref:?dummy_ind_dref arg_dref:?arg_dref cond:?cond"
Pro_word = "phon:?word cat:NP pro:yes cond:None gender:?g "
Conj_word = "cat:Conj cond:?cond"

# 2. DM chunks
Proper_NP_into_DM = "cat:NP id:?NP_id pos:0 gender:?g pro:no parent:?XP_id subcat:None new_ind_dref:
NP_into_DM = "cat:NP id:?NP_id pos:0 pro:no parent:?XP_id subcat:None new_ind_dref:?new_ind_dref DRS
Dep_Proper_NP_into_DM = "cat:NP id:?XP_id pos:1 gender:?g pro:no parent:?XP_parent subcat:None new_i
Dep_NP_into_DM = "cat:NP id:?XP_id pos:1 pro:no parent:?XP_parent subcat:None new_ind_dref:?new_ind_
S_into_DM = "cat:S id:?XP_id parent:?XP_parent pos:?XP_pos subcat:?XP_subcat DRS:?drs DRS_parent:?drs
VP_into_DM = "cat:VP id:?XP_id pos:1 parent:?XP_parent subcat:None arg_dref:?new_ind_dref cond:?cond
VP_from_DMB_into_DM = "cat:VP id:?tv_id pos:1 parent:?tv_parent subcat:None arg_dref:?new_ind_dref c
word_into_DM = "cat:?word pos:0 id:?word_id subcat:None"
Det_into_DM = "cat:Det pos:0 id:?det_id parent:?NP_id subcat:None new_ind_dref:?new_ind_dref DRS:?drs
N_into_DM = "cat:N pos:1 gender:?g pro:no id:?XP_id parent:?XP_parent subcat:None"
V_into_DM = "cat:V pos:0 id:?V_id parent:?XP_id subcat:None cond:?cond DRS:?drs"
ConjP_into_DM = "cat:ConjP id:?ConjP_id pos:?XP_pos parent:?XP_parent subcat:None cata:?cata"
Conj_into_DM = "cat:Conj pos:1 id:?Conj_id parent:?ConjP_id subcat:None"
CP_into_DM = "cat:CP id:?CP_id pos:1 parent:?S_id subcat:None cata:?cata"
C_into_DM = "cat:C pos:0 id:?C_id parent:?CP_id subcat:None"
S_new_root_into_DM = "cat:S pos:?XP_pos id:?S_id parent:?XP_parent subcat:?XP_subcat DRS:?drs_root D

# 3. General parsing goals
parsing_goal = "status:parsing id:?XP_id parent:?XP_parent pos:?XP_pos subcat:?XP_subcat DRS:?drs DRS

# 4. Specific parsing goals
# the value ?cata stores the DRS where the cataphora originated from
scan_next_word = "status:scanning cata:?cata cata_search:no"
start_cata_before_scan ="status:scanning cata:?cata cata_search:yes"
VP_into_goal = "status:parsing cat:VP pos:1 id:?VP_id parent:?XP_id subcat:None DRS:?drs DRS_parent:
NP_into_goal = "status:parsing cat:NP id:?NP_id parent:?XP_id subcat:None DRS:?drs DRS_parent:?drs_pa
N_into_goal = "status:parsing cat:N pos:1 id:?N_id parent:?NP_id subcat:None DRS:?drs DRS_parent:?drs
recall_DRS_goal = "status:DRS_recalling DRS:?drs"
recall_antec_goal = "status:antec_recalling DRS:?drs"
```

```
recalled_antec_goal = "status:antec_recalled DRS:?drs DRS_parent:?drs_parent"
S_goal_second_conjunct = "status:parsing cat:S pos:2 id:?S2_id parent:?ConjP_id subcat:None DRS:?drs
S_goal_antec = "status:parsing cat:S pos:1 id:?S2_id parent:?CP_id subcat:None DRS:?drs_antec DRS_par
start_cata_goal ="status:scanning cata:!None?cata cata_search:!no?cata_search"
resolving_cata ="status:resolving-cata cata:?cata"
recall_cata_goal = "status:cata_resolving DRS:?drs"
recalled_cata_goal = "status:cata_recalled DRS:?drs"
end_goal = "status:scanning cata:None cata_search:END"

#5. thetaBuffer chunks
VP_chunk_theta_buffer = "cat:VP id:?tv_id pos:1 parent:?tv_parent arg_dref:?tv_arg_dref cond:?tv_cond
dref_chunk_theta_buffer = "cat:None arg_dref:?new_ind_dref "

#6. DMBuffer chunks
recalled_DRS = "cat:S DRS:?drs DRS_parent:?drs_parent"
recalled_antec = "cond:?antec_cond arg_dref:?antec_dref"

#7. Reanalyzed chunks
S_reanalyzed_as_first_conjunct = "cat:S pos:0 id:?XP_id parent:?ConjP_id subcat:?XP_subcat DRS:?drs D
S_reanalyzed_as_conseq = "cat:S pos:0 id:?XP_id parent:?S_id subcat:?XP_subcat DRS:?drs DRS_parent:?d
```

C Productions

(3) File **DRT_parser_productions.py**:

```
from DRT_parser_chunks import pn_default, pro_default, conj_default, tv_default, p_default, iv_defaul
    NP_word, N_word, Det_word, V_word, Pro_word, Conj_word, \
    Proper_NP_into_DM, NP_into_DM, Dep_Proper_NP_into_DM, Dep_NP_into_DM, S_into_DM, VP_into_DM, VP_i
    parsing_goal, scan_next_word, VP_into_goal, NP_into_goal, N_into_goal, recall_DRS_goal, recall_an
    VP_chunk_theta_buffer, dref_chunk_theta_buffer, \
    recalled_DRS, recalled_antec, \
    S_new_root_into_DM, S_goal_second_conjunct, S_goal_antec, S_reanalyzed_as_first_conjunct, S_reana
    start_cata_before_scan, start_cata_goal, resolving_cata, recall_cata_goal, recalled_cata_goal, en

def init():
    from uuid import uuid4
    # part 1: memory and buffers
    LexM.add("phon:Bob gender:m cond:bob " + pn_default)
    LexM.add("phon:John gender:m cond:john " + pn_default)
    LexM.add("phon:Dank-the-Donkey gender:n cond:dtd " + pn_default)
    LexM.add("phon:Mary gender:f cond:mary " + pn_default)
    LexM.add("phon:he gender:m " + pro_default)
    LexM.add("phon:him gender:m " + pro_default)
    LexM.add("phon:she gender:f " + pro_default)
    LexM.add("phon:her gender:f " + pro_default)
    LexM.add("phon:it gender:n " + pro_default)
    LexM.add("phon:and cond:AND " + conj_default)
    LexM.add("phon:if cond:IF " + conj_default)
    LexM.add("phon:owns subcat:trans cond:own " + tv_default)
    LexM.add("phon:won't-eat subcat:trans cond:will_not_eat " + tv_default)
    LexM.add("phon:likes subcat:trans cond:like " + tv_default)
    LexM.add("phon:beats subcat:trans cond:beat " + tv_default)
    LexM.add("phon:plays subcat:PP cond:play " + tv_default)
    LexM.add("phon:dances subcat:PP cond:dance " + tv_default)
    LexM.add("phon:sleeps cond:sleep " + iv_default)
    LexM.add("phon:is-overcooked cond:overcooked " + iv_default)
    LexM.add("phon:walks cond:walk " + iv_default)
    LexM.add("phon:brays cond:bray " + iv_default)
    LexM.add("phon:a new_ind_dref:d1 " + det_default)
    LexM.add("phon:the new_ind_dref:None pro:yes " + det_default)
    LexM.add("phon:donkey cond:donkey gender:n " + n_default)
    LexM.add("phon:hamburger cond:hamburger gender:n " + n_default)
    # part 2: goals
    goalstack.append("status:parsing cat:S pos:0 id:0 parent:None \
                      subcat:None DRS:0 DRS_parent:None cata:None")
    goal.set("status:scanning cata:None cata_search:no")
    # part 3: summary of what just happened
    print "\n>> INITIALIZING lex. mem. (adding lex. info for words)"
    print ">> INITIALIZING goal stack"
    print "   -- top goal: scan first word"
    print "   -- next goal: predicting an S & an empty DRS\n"
    # note that predicting a chunk means that it was added to the goal stack

def scan_word(goal=scan_next_word):
    word = motor.read_next_word()
    if word != "END":
        LexM.request("phon:?word")
        goal.set(goalstack.pop())
        goal.modify(cata=cata)
        print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] Scanned and requested lex. info fo
    else:
        goal.set(end_goal)

def attach_NP_as_subject(LexMBuffer=NP_word, \
        goal=parsing_goal + "cat:S"):
    # part 1: memory & buffers
    NP_id, VP_id, word_id = (str(uuid4()) for dummy_idx in range(3))
    new_ind_dref = self.get_new_ind_dref(dummy_ind_dref)
    DM.add(goal.chunk)
    DM.add(Proper_NP_into_DM)
    DM.add("parent:?NP_id " + word_into_DM)
    thetaBuffer.set(dref_chunk_theta_buffer)
    LexMBuffer.clear()
    # part 2: goals
    goalstack.append(goal.chunk) # push S goal
    goal.set(VP_into_goal) # create VP goal
    goalstack.append(goal.chunk) # push VP goal
    goal.set(scan_next_word) # scan next word
    # part 3: summary of what just happened
    print "\n>> Attaching NP, the word ’", word , "’, and predicting VP"
    print ">> Top goal: scan next word; next goal: the predicted VP"
    self.draw_parse_and_drs()

def attach_Det_as_subject(LexMBuffer=Det_word, \
        goal=parsing_goal + "cat:S"):
    # part 1
    NP_id, VP_id, det_id, N_id, word_id = (str(uuid4()) for dummy_idx in range(5))
    new_ind_dref = self.get_new_ind_dref(dummy_ind_dref)
    DM.add(goal.chunk)
    DM.add("pos:0 " + NP_into_DM)
    DM.add(Det_into_DM)
    DM.add("parent:?det_id " + word_into_DM)
    thetaBuffer.set(dref_chunk_theta_buffer)
    LexMBuffer.clear()
    # part 2
    goalstack.append(goal.chunk) # push S
    goal.set(VP_into_goal) # create VP
    goalstack.append(goal.chunk) # push VP
    goal.set(N_into_goal) # create N
    goalstack.append(goal.chunk) # push N
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Attaching Det, the word ’", word , "’, and predicting N and VP"
    print ">> Top goal: scan next word; next goal: the predicted N"
    self.draw_parse_and_drs()

def attach_N(LexMBuffer=N_word, \
        goal=parsing_goal + "cat:N", \
        thetaBuffer=dref_chunk_theta_buffer):
    # part 1
    word_id = str(uuid4())
    DM.add("cond:?cond arg_dref:?new_ind_dref DRS:?drs " + N_into_DM)
    DM.add("parent:?XP_id " + word_into_DM)
    LexMBuffer.clear()
    # part 2
    if cata != "None":
        goal.set(start_cata_before_scan) # start cata search
    else:
        goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Attaching N, and the word ’", word , "’"
    print ">> Top goal: scan next word"
    self.draw_parse_and_drs()

def attach_IV(LexMBuffer="subcat:intrans " + V_word, \
        goal=parsing_goal + "cat:VP ", \
        thetaBuffer=dref_chunk_theta_buffer):
    # part 1
    V_id, word_id = (str(uuid4()) for dummy_idx in range(2))
    DM.add(VP_into_DM)
    DM.add(V_into_DM)
    DM.add("parent:?V_id " + word_into_DM)
    thetaBuffer.clear() # We clear the thetaBuffer as soon as the info about the dref was used
    LexMBuffer.clear()
    # part 2
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Attaching VP, V (intrans.), and the word ’", word , "’"
    print ">> Top goal: scan next word"
    self.draw_parse_and_drs()

def attach_TV(LexMBuffer="subcat:trans " + V_word, \
        goal=parsing_goal + "cat:VP ", \
        thetaBuffer=dref_chunk_theta_buffer):
    # part 1
    NP_id, V_id, word_id = (str(uuid4()) for dummy_idx in range(3))
    DM.add(V_into_DM)
    DM.add("parent:?V_id " + word_into_DM)
    thetaBuffer.set(VP_into_DM)
    LexMBuffer.clear()
    # part 2
    goal.set("pos:1 " + NP_into_goal)
    goalstack.append(goal.chunk)
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Placing VP in the theta buffer for linking with the object dref"
    print ">> VP not in DM yet, so VP info missing in the parse tree and DRS below"
    print ">> Attaching V (trans.), the word '", word , "', and predicting NP"
    print ">> Top goal: scan next word; next goal: the predicted NP"
    self.draw_parse_and_drs()

def attach_NP_as_object(LexMBuffer=NP_word, \
        goal=parsing_goal + "cat:NP subcat:None", \
        thetaBuffer=VP_chunk_theta_buffer):
    # part 1
    word_id = str(uuid4())
    new_ind_dref = self.get_new_ind_dref(dummy_ind_dref)
    DM.add(Dep_Proper_NP_into_DM)
    DM.add("parent:?XP_id " + word_into_DM)
    new_ind_dref = tv_arg_dref + new_ind_dref # this is used for the verb so it modifies the correct
    DM.add("arg_dref:?arg_dref " + VP_from_DMB_into_DM)
    thetaBuffer.clear() # We clear the thetaBuffer as soon as the transitive verb chunk was used
    LexMBuffer.clear()
    # part 2
    goal.set(goalstack.pop())
    goal.modify(cata=cata)
    DM.add(goal.chunk)
    goalstack.append(goal.chunk)
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Top goal: scan next word"
    self.draw_parse_and_drs()

def attach_Det_as_object(LexMBuffer=Det_word, \
        goal=parsing_goal + "cat:NP subcat:None", \
        thetaBuffer=VP_chunk_theta_buffer):
    # part 1
    det_id, N_id, word_id = (str(uuid4()) for dummy_idx in range(3))
    new_ind_dref = self.get_new_ind_dref(dummy_ind_dref)
    DM.add(Dep_NP_into_DM)
    NP_id = XP_id
    DM.add(Det_into_DM)
    DM.add("parent:?det_id " + word_into_DM)
    thetaBuffer.set(dref_chunk_theta_buffer) #this is used for noun, so it modifies the correct dref
    new_ind_dref = tv_arg_dref + new_ind_dref #this is used for the verb so it modifies the correct
    DM.add("arg_dref:?arg_dref " + VP_from_DMB_into_DM)
    LexMBuffer.clear()
    # part 2
    goal.set(goalstack.pop())
    goal.modify(cata=cata)
    DM.add(goal.chunk)
    goalstack.append(goal.chunk)
    goal.set(N_into_goal)
    goalstack.append(goal.chunk)
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Top goal: scan next word"
    self.draw_parse_and_drs()

# pronoun resolution:
# -- we change goal to recalling first
# -- we then recall the antec. of the pronoun depending on its gender
# -- we look for an antec. in the current DRS
# -- if not found, we move to the next accessible DRS (change_drs rule)
# -- if the antec. search fails, we mark the antec. of the pronoun as unknown
# -- if an antec. is found, we match it to the pronoun
def change_goal_to_recalling(LexMBuffer=Pro_word, \
        goal=parsing_goal, \
        DMBuffer=None):
    # part 1
    DMBuffer.set(recalled_DRS)
    # part 2
    goal.modify(cata=drs) # mark the DRS of the pronoun in case it ends up unresolved
    goalstack.append(goal.chunk)
    goal.set(recall_antec_goal) # this takes you to the recall_pro_antec rule
    # part 3
    print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] Marked the DRS of the pronoun in the ol
    print ">> Top goal: recall pronoun antecedent in current DRS\n"

def recall_DRS_antec(LexMBuffer=Pro_word + "gender:?gender", \
        goal=recall_DRS_goal, \
        DMBuffer=None):
    # part 1
    DM.request("cat:S DRS:?drs") # requesting a potential antec. into DMBuffer
    # part 2
    goal.set(recall_antec_goal) # this takes you to the recall_pro_antec rule
    # part 3
    print "\n>> Placed DM request for the parent of the current DRS"
    print ">> [TIME:", int(round(self.now()*1000, 0)), "ms] Top goal: set parent DRS as current DRS a

def recall_pro_antec(LexMBuffer=Pro_word + "gender:?gender", \
        goal=recall_antec_goal, \
        DMBuffer=recalled_DRS):
    # part 2
    # set the new goal early so that you keep track of the drs and also drs_parent in DMBuffer
    goal.set(recalled_antec_goal) # this takes you to the change_drs rule if no antec. is retrieved
    # part 1
    DMBuffer.clear() # clear the DM buffer to make room for the next request
    DM.request("gender:?gender DRS:?drs") # requesting a potential antec. in the current DRS
    # part 3
    print "\n>> Placed DM request for a pronoun antecedent in current DRS"
    print ">> [TIME:", int(round(self.now()*1000, 0)), "ms] Top goal: Check if any antecedent was fou

def change_drs(LexMBuffer=Pro_word, \
        goal=recalled_antec_goal, \
        DMBuffer=None):
    if drs_parent == str(None): # No antec. found
        DMBuffer.set("cond:unknown arg_dref:unknown")
        print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] No antec. found; no parent DRS to r
    else: # No antec. in this DRS, moving to the parent DRS
```

```
        goal.set(recall_DRS_goal) # this takes you to the recall_DRS_antec
        goal.modify(DRS=drs_parent)
        print "\n>> No pronominal antecedent found in current DRS"
        print ">> [TIME:", int(round(self.now()*1000, 0)), "ms] Top goal: recall parent DRS\n"

def match_antecedent_to_pronoun(LexMBuffer=Pro_word, \
        goal=recalled_antec_goal, \
        DMBuffer=recalled_antec):
    if DMBuffer.chunk["cond"] != "unknown": # if there’s an actual antecedent
        DM.add(DMBuffer.chunk) # add retrieved antec. to DM to increase activation
        LexMBuffer.modify(cond=str("=" + antec_dref))
        LexMBuffer.modify(arg_dref="d1")
        LexMBuffer.modify(new_ind_dref="d1")
        # part 2
        goal.set(goalstack.pop())
        goal.modify(cata="None") # this has to be marked as None here because when we started the pr
    else: # if there’s no antecedent
        # part 2
        goal.set(goalstack.pop())
        LexMBuffer.modify(cond="unknown")
        LexMBuffer.modify(arg_dref="d1")
        LexMBuffer.modify(new_ind_dref="d1")
    DMBuffer.clear()
    # part 3
    print "\n>> Marking the result of pron. resolution in the lex. buffer"
    print ">> [TIME:", int(round(self.now()*1000, 0)), "ms] Top goal: the goal before the pron. reso

# adding AND and IF
# -- for AND, we just keep the same DRS and the same DRS_parent, we only do
# syntactic reanalysis
# -- for IF, we make the antec the parent of the conseq and the matrix drs (the
# one that contains the whole conditional) the parent of the antec
# -- the DRS child-parent relation is the accessibility relation used in
# pronoun resolution
def attach_and_as_S_conjunction(LexMBuffer="phon:and?word " + Conj_word, \
        goal=parsing_goal + "cat:S"):
    # part 1
    ConjP_id, Conj_id, S2_id, word_id = (str(uuid4()) for dummy_idx in range(4))
    DM.add(S_reanalyzed_as_first_conjunct)
    DM.add(Conj_into_DM)
    DM.add(ConjP_into_DM)
    DM.add("parent:?Conj_id " + word_into_DM)
    LexMBuffer.clear()
    # part 2
    goal.set(S_goal_second_conjunct)
    DM.add(goal.chunk)
    goalstack.append(goal.chunk)
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Attaching Conj, the word ’", word, "’, and predicting S"
    print ">> Top goal: scan next word; next goal: the predicted S"
    self.draw_parse_and_drs()

def attach_if_as_S_conjunction(LexMBuffer="phon:if?word " + Conj_word, \
        goal=parsing_goal + "cat:S"):
    # part 1
    CP_id, C_id, S_id, S2_id, word_id, drs_antec, drs_root = (str(uuid4()) for dummy_idx in range(7)
    DM.add(S_new_root_into_DM)
    DM.add(S_reanalyzed_as_conseq)
    DM.add(CP_into_DM) # this is the if-clause (with if included)
    DM.add(C_into_DM)
    DM.add("parent:?C_id " + word_into_DM)
    LexMBuffer.clear()
    # part 2
    goal.set(S_goal_antec)
    DM.add(goal.chunk)
    goalstack.append(goal.chunk)
    goal.set(scan_next_word) # scan next word
    # part 3
    print "\n>> Attaching CP, C, the word ’", word, "’, and predicting S"
    print ">> Top goal: scan next word; next goal: the predicted S"
    self.draw_parse_and_drs()

# cataphora resolution:
# -- we first recall the cataphoric pronoun based on its DRS of origin
# -- we then request the DRS of origin so that we can identify its parent
# -- we look for a cata antec. in the parent DRS
# -- if not found, we move to the next accessible DRS (recall_parent_DRS rule)
# -- if the antec. search fails, we move to the goal before the cata search started
# -- if an antec. is found, we match it to the pronoun, we mark that we do not
# have a cata subgoal anymore, and we move to the goal before the cata search
# started
def start_cata(goal=start_cata_goal, \
        DM="busy:False", \
        #DM="busy:False error:True", \
        DMBuffer=None, \
        LexMBuffer=None):
    # part 1
    DM.request("cat:NP DRS:?cata cond:unknown")
    # part 2
    if cata_search == "yes":
        goal.set(scan_next_word)
    elif cata_search == "END":
        goal.set(end_goal)
    goalstack.append(goal.chunk)
    goal.set(start_cata_goal)
    # part 3
    self.START = self.now()
    print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] Saved current goal; placed DM request 
    print ">> Crucial START time:", self.now(), "\n"

def cata_resolution(goal=start_cata_goal, \
        DMBuffer="DRS:?drs cat:NP cond:unknown", \
        LexMBuffer=None):
    # part 1
    LexMBuffer.set(DMBuffer.chunk)
    DMBuffer.clear()
    DM.request("cat:S DRS:?drs cata:?drs") # requesting the DRS where cataphora originated from
    # part 2
    goal.set(recall_cata_goal)
    # part 3
    print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] Placed DM request for the original DRS
```

```
def recall_cata_antec_with_recalled_DRS(LexMBuffer="cat:NP cond:unknown gender:?gender", \
        goal=recall_cata_goal, \
        DMBuffer=recalled_DRS):
    # part 1
    drs = drs_parent # setting the parent as the current DRS
    DMBuffer.clear() # clear the DM buffer to make room for the next request
    DM.request("gender:?gender DRS:?drs cond:!unknown") # requesting a potential antec. into DMBuffe
    # part 2
    goal.set(recalled_cata_goal)
    # part 3
    print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] Setting parent DRS as current DRS; DM

def recall_cata_antec_no_recalled_DRS(LexMBuffer="cat:NP cond:unknown gender:?gender", \
        goal=recall_cata_goal, \
        DMBuffer=None):
    LexMBuffer.clear() # clear the LexM buffer to make room for the next request
    goal.set(goalstack.pop())
    self.STOP = self.now()
    print "\n>> No parent DRS to move to."
    print ">> Crucial STOP time:", self.now()
    print ">> Moving to the goal before cata search started.\n"

def recall_parent_DRS(LexMBuffer="cat:NP cond:unknown gender:?gender", \
        goal=recalled_cata_goal, \
        DMBuffer=None):
    DM.request("cat:S DRS:?drs") # requesting parent DRS
    # part 2
    goal.set(recall_cata_goal)
    # part 3
    print "\n>> [TIME:", int(round(self.now()*1000, 0)), "ms] No cata antec. found; placing DM reques

def match_antecedent_to_pronoun_cata(LexMBuffer="cat:NP cond:unknown gender:?gender", \
        goal=recalled_cata_goal, \
        DMBuffer=recalled_antec):
    # part 1
    DM.add(DMBuffer.chunk) # add retrieved antec. to DM to increase activation
    LexMBuffer.modify(cond=str("=" + antec_dref))
    DM.add(LexMBuffer.chunk) # add resolved cata to DM
    DMBuffer.clear()
    LexMBuffer.clear()
    # part 2
    cata = "None"
    goal.set(goalstack.pop())
    self.STOP = self.now()
    # part 3
    print "\n>> Found cata antec.; marking it in the lex. buffer and adding it to DM"
    print ">> Crucial STOP time:", self.now()
    print ">> [TIME:", int(round(self.now()*1000, 0)), "ms] Top goal: scan next word\n"

def stop(goal=end_goal):
    # We are done parsing, so we clear the goal buffer and the goal stack
    goal.clear()
    goalstack.clear()
    print "\n>> DONE!"
    self.draw_parse_and_drs()
```

```python
    print "Crucial STOP-START time (rounded to 3 digits):", round(self.STOP-self.START, 3)

def get_new_ind_dref(self, new_ind_dref):
    """
    pop a new dref from the ind_dref_list stack; if no dref is introduced, return None
    """
    if new_ind_dref != "None":
        return self.ind_dref_list.pop()
    else:
        return None

def get_new_event_dref(self, new_event_dref):
    """
    pop a new dref from the event_dref_list stack; if no dref is introduced, return None
    """
    if new_event_dref != "None":
        return self.event_dref_list.pop()
    else:
        return None
```