CS161 SPR ’13 **HW-2** **due**: on CourseWeb before midnight, Monday, May 6

In this homework you will write a Lisp function, C-ANALYZE, which takes a SENTence and a *conceptual lexicon* C-LEX as input and returns a conceptual representation for SENT. The first SENT from *Lie Down With Pigs* is: PGS1 = (SALLY GRENTZ WARNED HER TEENAGE SON FREDDY ABOUT BECOMING FRIENDS WITH HANK BECAUSE HE HAD STOLEN A CAR)

(C-ANALYZE PGS1 C-LEX) will return the value of PIGCON1. PIGCON1 =

```
(MTRANS (AGENT (HUMAN (GENDER (FEMALE))
                      (F-NAME (SALLY))
                      (L-NAME (GRENTZ))))
        (RECIP (HUMAN (GENDER (MALE))
                      (REL (SON
                                (F-NAME (FREDDY))
                                (OF (HUMAN (GENDER (FEMALE))
                                           (F-NAME (SALLY))
                                           (L-NAME (GRENTZ))))))
                      (AGE (TEEN))
                      (F-NAME (FREDDY))))
        (OBJECT (C-CAUSE
                      (ANTE (ST-CHANGE
                                (AGENT (HUMAN (GENDER (MALE))
                                              (F-NAME (FREDDY))))
                                (FROM (ACQUAINT)
                                (TO (FRIEND
                                    (OF (HUMAN
                                              (GENDER (MALE))
                                              (F-NAME (HANK)))))))))
                      (CONSEQ
                          (GOAL-FAILURE
                                (AGENT (HUMAN
                                           (GENDER (MALE))
                                           (F-NAME (FREDDY))))
                                (TIME (FUTURE))))
                      (JUSTIF (STEAL (AGENT
                                           (HUMAN (GENDER (MALE))
                                                  (F-NAME (HANK))))
                                     (OBJECT
                                          (VEHICLE (REF (INDEF))))
                                     (TIME (PAST)))))))
```

Overview: C-ANALYZE will look for a word or phrase (***wph***) at the front of a sentence (***sent***) and remove it from **sent**. When found, it will access the associated *frame* and *demons*. For the frame it will create a unique *frame-instance* (***frami***) and place it in working memory (**wk-mem**). For each demon it will create a *unique demon-instance* (**demi**) and place it in an active-demons list (**actv-dems**). It will then execute all the **demi**s until they are quiescent. Demis will implement semantic expectations and will link up the frames in **wk-mem**. Once the demis are quiescent the process above will repeat, with the next recognized wph removed from **sent**. C-ANALYZE halts when **sent** is empty.

Once **sent** is empty, C-ANALYZE returns: (GAPVALS (TOP-CON **wk-mem**)). Recall that GAPVALS was defined in HW-1. TOP-CON returns the largest top frame-instance in **wk-mem** and will be defined below.

We will break up C-ANALYZE into a manageable set of sub-functions.

First we will need to associate a frame and demons with each wph in our story. To do this we will set up a conceptual lexicon, C-LEX, which is just a list. (We are not concerned with efficiency in this course project and so we will usually employ lists for different types of memories.)

The conceptual lexicon C-LEX is just a global variable whose value is a list of lists where each list contains 3 elements: 1. A list of one-or-more words, 2. a frame, and 3. a list of zero or more *demon-instances*.

C-LEX = (((word+) frame (demon-instance*))
((word+) frame (demon-instance*))
…
((word+) frame (demon-instance*)))

C-LEX will be created by using the Lisp function ADD-LEX, which will take 3 arguments: a wph, a frame, and a list of zero or more demons. ADD-LEX will simply add them to the global variable C-LEX.

<u>1. Write the ADD-LEX Lisp function</u> to build up C-LEX for the first sentence PGS1. Here are the lexical entries to use. (Note: we will use the convention of prefixing each demon with D-)

(Note: We will not add GRENTZ or SMYTHE to C-LEX and let C-ANALYZE figure out that these are last names.)

Example:

```
(ADD-LEX  '(SALLY)
          '(HUMAN (GENDER (FEMALE))
                  (F-NAME (SALLY))
                  (L-NAME L-NAME))
          '((D-LAST-NAME L-NAME)))

returns:  C-LEX = (   ((SALLY)
                       (HUMAN (GENDER (FEMALE))
                              (F-NAME (SALLY))
                              (L-NAME L-NAME))
                       ((D-LAST-NAME L-NAME)))      )

(ADD-LEX '(FREDDY)
          '(HUMAN (GENDER (MALE))
                  (F-NAME (FREDDY))
                  (L-NAME L-NAME))
          '((D-LAST-NAME L-NAME)))

returns:  C-LEX = (   ((FREDDY)
                        (HUMAN (GENDER (MALE))
                               (F-NAME (FREDDY))
                               (L-NAME L-NAME))
                        ((D-LAST-NAME L-NAME)))
                       ((SALLY)
                        (HUMAN (GENDER (FEMALE))
                               (F-NAME (SALLY))
                               (L-NAME L-NAME))
                        ((D-LAST-NAME L-NAME)))      )
```

Here are frames & demons for the remaining first names:

```
(HANK)    (HUMAN (GENDER (MALE))
                 (F-NAME (HANK))
                 (L-NAME L-NAME))
          ((D-LAST-NAME L-NAME))

(HAROLD) (HUMAN (GENDER (MALE))
```

(**F-NAME** (HAROLD))
(**L-NAME** L-NAME))
((D-LAST-NAME L-NAME))

Here are frames & demons for the other words/phrases that appear in the first sentence:

(WARNED) (MTRANS
(**AGENT** AGENT)
(**RECIP** RECIP)
(**OBJECT** (C-CAUSE
(**ANTE** ANTE)
(**CONSEQ** (GOAL-FAILURE
(**AGENT** AGENT)
(**TIME** (FUTURE))))
(**JUSTIF** JUSTIF))))
((D-FILL (AGENT) BEF HUMAN)
(D-FILL (RECIP) AFT HUMAN)
(D-AFT-PRED (OBJECT ANTE) INVOLVE (ACT ST-CHANGE))
(D-AFT-PRED (JUSTIF) REASON-FOR (ACT))
(D-SAME-BINDING (RECIP) (OBJECT CONSEQ AGENT)))

(HER) (PRONOUN (TYPE (POSS))
(REF REF))
((D-POSS-PRO-REF REF BEF (GENDER) FEMALE))

(TEENAGE) (AGE (**VAL** (TEEN)))
((D-ATTACH-SF AFT HUMAN AGE (TEEN)))

(SON) (HUMAN (**GENDER** (MALE))
(**REL** (SON (**OF** OF)))
(**F-NAME** F-NAME))
((D-POSS-PRO BEF (REL OF))
(D-IMM-AFT F-NAME))

(ABOUT) (INVOLVE)
( )

(BECOMING FRIENDS) (ST-CHANGE

(**AGENT** AGENT)
(**FROM** (ACQUAINT))
(**TO** (FRIEND
(**OF** OF)))
((D-FILL (AGENT) BEF HUMAN)
(D-AFT-PRED (TO OF) WITH (HUMAN))))

(WITH) (WITH)
( )

(BECAUSE) (REASON-FOR)
( )

(HE) (HUMAN (**GENDER** (MALE))
(**F-NAME** F-NAME))
((D-PRO-REF BEF GENDER MALE F-NAME))

(HAD STOLEN) (STEAL
(**AGENT** AGENT)
(**OBJECT** OBJECT)
(**TIME** (PAST)))
((D-FILL (AGENT) BEF HUMAN)
(D-FILL (OBJECT) AFT PHYS-OBJ))

(A) ( )
((D-ATTACH-SF AFT PHYS-OBJ REF (INDEF)))

(CAR) (VEHICLE)
( )

********** end of lexical entries for PGS1 and all first names **********

2. Write a Lisp function RECALL-WPH This function takes a sentence S and conceptual lexicon CLX as input. It finds the longest phrase in CLX that matches the front of S and returns the list: (frame demons S1) where frame and demons are those associated with the recognized wph and where S1 is the NTHCDR of S (N is the length of the matched phrase).
If a word w is not recognized, then it returns ((UNKNOWN (**IS** (w))) NIL S1)

Examples:
Here is a small conceptual lexicon:
C-LX-1 =

```
(       ((THE) ( )
                ((D-ATTACH-SF AFT PHYS-OBJ REF (DEF))))
        ((BIG) (SIZE (VAL (>NORM)))
                 ((D-ATTACH-SF AFT PHYS-OBJ SIZE (<NORM))))
         ((APPLE)  (FRUIT (TYPE (APPLE)))   ( ) )
        ((APPLE PIE)  (DESSERT)  ( ) )
         ((THE BIG APPLE) (NYC) ( ) )      )
```

Here are some sentences:
APS-1 = (THE APPLE WAS FOUND IN THE BIG APPLE)
APS-2 = (APPLE WAS FOUND)
APS-3 = (APPLE PIE IS TASTY)
APS-4 = (THE BIG APPLE IS AN EXCITING PLACE)
APS-5 = (CRAB APPLE)

Here is RECALL-WPH at work:
(RECALL-WPH APS-1 C-LX-1)
returns: (( ) ((D-ATTACH-SF AFT PHYS-OBJ REF (DEF)))
(APPLE WAS FOUND IN THE BIG APPLE))

(RECALL-WPH APS-2 C-LX-1)
returns: ((FRUIT (**TYPE** (APPLE))) ( ) (WAS FOUND))

(RECALL-WPH APS-3 C-LX-1)
returns: ((DESSERT)   ( )   (IS TASTY))

(RECALL-WPH APS-4 C-LX-1)
returns: ((NYC) ( ) (IS AN EXCITING PLACE))

(RECALL-WPH APS-5 C-LX-1)
returns: ((UNKNOWN (**IS** (CRAB)))   ( ) (APPLE))

3. Write a Lisp function TOKENIZE. TOKENIZE takes a frame as input and turns it into a unique frame-instance (frami) by uniquely renaming its gaps. To do this it calls on the Lisp function UNIQUE-GAP, which we have provided in the skeleton. For example, (UNIQUE-GAP ‘HUMAN) returns HUMAN001. A second call to (UNIQUE-GAP ‘HUMAN) returns HUMAN002 and so on. (The actual numbers will

vary, but they are guaranteed to be unique within a particular Lisp session.) TOKENIZE goes through a frame recursively and replaces each gap with a tokenized version of that gap and assigns the value NIL to each new tokenized gap.

Examples:

```
(TOKENIZE '(ST-CHANGE
                    (AGENT AGENT)
                    (FROM (ACQUAINT))
                    (TO (FRIEND
                          (OF OF)))))
returns:  (ST-CHANGE
                    (AGENT AGENT001)
                    (FROM (ACQUAINT))
                    (TO (FRIEND
                          (OF OF001))))   where AGENT001 = NIL and OF001 = NIL

(TOKENIZE '(MTRANS
                    (AGENT AGENT)
                    (RECIP  RECIP)
                    (OBJECT (C-CAUSE
                                        (ANTE ANTE)
                                        (CONSEQ (GOAL-FAILURE
                                                            (AGENT AGENT)
                                                            (TIME (FUTURE))))
                                        (JUSTIF JUSTIF)))))
returns:  (MTRANS
                    (AGENT AGENT002)
                    (RECIP  RECIP001)
                    (OBJECT (C-CAUSE
                                        (ANTE ANTE001)
                                        (CONSEQ (GOAL-FAILURE
                                                            (AGENT AGENT003)
                                                            (TIME (FUTURE))))
                                        (JUSTIF JUSTIF001))))
where AGENT002, RECIP001, ANTE001, AGENT003, JUSTIF001 all = NIL

(TOKENIZE 'AGENT) returns:  AGENT004
where AGENT004 = NIL
```

4. Write a Lisp function EXT-WK-MEM. This function takes a frame-instance **frami** as input. It creates a new token by calling UNIQUE-GAP with a base name of CON, and assigns **frami** as the value of that token. As a side-effect it adds that token to the global variable WK-MEM.

Example:
WK-MEM = (CON001 CON002)
FRAME-1 = (STEAL
(**AGENT** AGENT88)
(**OBJECT** OBJECT99)
(**TIME** (PAST)))
(EXT-WK-MEM FRAME1) returns: CON003 = (STEAL
(**AGENT** AGENT88)
(**OBJECT** OBJECT99)
(**TIME** (PAST)))

and, as a side-effect, sets WK-MEM = (CON001 CON002 CON003)

5. Write a Lisp function SPAWN. SPAWN's job is to create a unique *demon-instance* (**demi**) of each demon associated with a frame. SPAWN receives, as input, a CON-atom and a list of demons. SPAWN inserts the CON-atom as the first argument of each demon and, as a side-effect, adds that **demi** to a global variable, which is the list of active demons: ACTV-DEMONS.

Example:
SALLY is the first word read. Let us assume now that, as a result:
WK-MEM = (CON99) where CON99 = (HUMAN (**F-NAME** (SALLY))
(**L-NAME** L-NAME001))
And the demons associated with SALLY are just this one:
DEMS = ((D-LAST-NAME L-NAME))

(SPAWN CON99 DEMS) will create the *demon-instance*:
(D-LAST-NAME CON99 L-NAME) (note: SPAWN inserted CON99)

This *demon-instance* of D-LAST-NAME will be working only for CON99 in WK-MEM. So there can be multiple *instances* of the same demon, each working for a different CON-atom in WK-MEM.

Finally, as a side-effect, SPAWN will add the demon-instances to the global variable ACTV-DEMONS.

6. Write a Lisp function SEARCH-WKM. This function searches in WK-MEM. It is a utility function that will be used by some of the demon-instances. Here are this function's parameters:
- WKM (a working memory, which is a list of atoms)
- MYCON (the CON-atom a demi works for),
- DIR (a direction in WK to look, which is going to be either BEFore or AFTer),
- PRED (the PREDicate that the demi is looking for in some other frame).

Starting at MYCON in WKM, SEARCH-WKM looks in the DIRection specified for the closest frame it can find with the specified PREDicate and returns the associated CON-atom of that frame.

Example:
WKM-1 = (CON33 CON44 CON55 CON66 CON77)
Where CON33 = (VEHICLE)
CON44 = (STEAL (**AGENT** AGENT22)
(**RECIP** RECIP11))
and CON55 = NIL
CON66 = (VEHICLE (**MODEL** (HONDA)))
CON77 = (VEHICLE)
MYCON = CON44

(SEARCH-WKM WKM-1 'CON44 'AFT 'VEHICLE) will return: CON66
(SEARCH-WKM WKM-1 'CON33 'BEF 'VEHICLE) will return: NIL
(SEARCH-WKM WKM-1 'CON55 'BEF 'VEHICLE) will return: CON33

7. Write an ISA? Lisp function. ISA? Is a utility function. It takes pred1, pred2 and an ONTology as input and returns T if pred1 is recursively a type of pred2 within that ONT.

Here is a partial ontology that we will use. It will be a global variable ONT

ONT =
((ISA ACT CONCEPT) (ISA ST-CHANGE CONCEPT)
(ISA STATE CONCEPT) (ISA C-CAUSE CAUSE)
(ISA CAUSE CONCEPT) (ISA MTRANS ACT)
(ISA STEAL ACT) (ISA HUMAN ANIMATE) (ISA CANINE ANIMATE)
(ISA ANIMATE PHYS-OBJ) (ISA VEHICLE PHYS-OBJ)
(ISA PAST TIME) (ISA D-LAST-NAME DEMON)
)

Examples:
(ISA? ‘MTRANS ‘CONCEPT ONT) returns: T
(ISA? ‘MTRANS ‘ANIMATE ONT) returns: NIL
(ISA? ‘HUMAN ‘PHYS-OBJ ONT) returns: T
(ISA? ‘HONDA ‘VEHICLE ONT) returns: NIL

Notice that ISA? makes a *closed-world assumption* (if it is not known then it is assumed to not be true).

C-ANALYZE needs a sub-function that executes the active demon-instances that have been spawned by SPAWN.

8. Write a Lisp function EXEC-DEMONS. This function takes a list of demon instances (**demis**) and a **wk-mem** as input and repeatedly attempts to execute each demi until all demis are quiescent. Whenever a demon successfully executes, EXEC-DEMONS removes that demon from the demon list and re-executes all the demons remaining in the list. (This is done because the successful execution of any one demi many change something that enables another demi to now successfully execute. A demi will indicate success to EXEC-DEMONS by returning T.)

Example:
WK-MEM-1 = (CON301 CON302)
where
CON301 = (HUMAN (**GENDER** (MALE))
(**F-NAME** (HANK))
(**L-NAME** L-NAME01))
CON302 = (STEAL
(**AGENT** AGENT01)
(**OBJECT** OBJECT01)
(**TIME** (PAST)))
CON303 = (UNKNOWN (**IS** (KRUGER)))

DEM-LIST-1 = ((D-FILL CON302 (AGENT) BEF HUMAN)
(D-FILL CON302 (OBJECT) AFT PHYS-OBJ)
(D-LAST-NAME CON302 L-NAME))

Let us refer to the above demis as: dem1, dem2 and dem3.

(EXEC-DEMONS DEM-LIST-1 WK-MEM-1)
will attempt to execute these demons. Only dem1 will execute successfully (thus return T) and so it will be removed from DEM-LIST-1.

Now let's define some demons:

9. Write your first demon function D-LAST-NAME. This demon takes as input its MYCON and its MYSLOT. It looks for the immediate next CON in WK-MEM that is right after its MYCON and if there is such a CON and also if the pred of that CON's frame = UNKNOWN, then D-LAST-NAME accesses the IS slot and binds that value to the gap associated with the demon's MYSLOT.

Example:
WK-MEM-1 = (CON123 CON456)
CON123 = (HUMAN (**F-NAME** (SALLY)) (**L-NAME** L-NAME234))
CON456 = (UNKNOWN (IS (GRENTZ)))
ACTV-DEMONS-1 = ((D-LAST-NAME CON123 L-NAME))

(EXEC-DEMONS WK-MEM-1 ACTV-DEMONS-1) will cause this demon to fire (i.e. execute successfully) and it will bind: L-NAME234 = (GRENTZ) and return T, which will cause EXEC-DEMONS to remove it from ACTV-DEMONS

Note: Whenever a demi binds a CON-atom in WK-MEM to the gap of some other slot of some other frame in WK-MEM, it should use the utility function BIND (defined next).

10. Write the utility function BIND. BIND simply uses SET or SETQ. It takes a GAP and a CON-atom as inputs and sets GAP = CON-atom. But it does *one more thing*: As a side-effect it adds this CON-atom to a global variable BOUND.

BOUND is a list that keeps track of every CON-atom that is bound to a gap in some other frame.

11. Write your second demon function D-FILL. This demon has parameters: MYCON, PATH, DIR, and PRED. It uses the utility function SEARCH-WKM to look in WK-MEM in direction DIR for a frame with predicate which ISA PRED. If it finds this CON, then it BINDs the gap of its MYSLOT (specified by PATH) to CON.

Example:
WK-MEM = (CON666 CON777 CON888)
CON777 = (STEAL

(**AGENT** AGENT999)
(**OBJECT** OBJECT888)
(**TIME** (PAST)))

CON666 = (HUMAN (**F-NAME** (HANK)))
CON888 = (VEHICLE)
ACTV-DEMONS = ((D-FILL CON777 (AGENT) BEF HUMAN)
(D-FILL CON777 (OBJECT) AFT PHYS-OBJ))

EXEC-DEMONS will fire (successfully execute) these demons.
The first D-FILL demon-instance will bind AGENT999 = CON666.
The second D-FILL demon-instance will bind OBJECT888 = CON888.

12. Write demon D-AFT-PRED. It takes the parameters: MYCON, MYPATH, PRED-MARKER, and ISA-CONSTRAINTS. It looks AFTer its MYCON in WK-MEM for a frame with predicate PRED-MARKER and then looks AFTer for a CON-atom that satisfies the ISA-CONSTRAINTS. If this CON is found, then D-AFT-PRED binds the gap (whose slot is specified by MYPATH) to CON.

Examples: (WARNED … ABOUT … BECOMING FRIENDS)
Two D-AFT-PRED demon-instances work for the MTRANS frame associated with "warned". Let us refer to them as **d-aft1** and **d-aft2**. (Look at C-LEX above.)

d-aft1 looks for an ACT or ST-CHANGE following an INVOLVE. When it finds this CON-atom **d-aft1** binds the OBJECT→ ANTE gap to what it found.

d-aft2 looks for an ACT following a REASON-FOR. When it finds it **d-aft2** binds the RECIP gap to what it found.

13. Write demon D-SAME-BINDING. It takes the parameters: MYCON, PATH1, PATH2. If the gap value under PATH1 is non-NIL, then it BINDs the gap located at PATH2 to the PATH1 gap's value.

For example, in (WARNED), this demon waits until the RECIP gap's value is non-nil and then BINDs the gap, associated with the slot specified by the path (OBJECT CONSEQ AGENT), with the value of RECIP's gap.

(In “warn”, the one warned is usually the one who will suffer a goal failure, so this demon is setting the AGENT of the GOAL-FAILURE to be the same as the RECIP of the MTRANS.)

14. Write demon D-PRO-REF. This demon attempts to resolve a pronoun reference. It has the following parameters: MYCON, MYSLOT, DIR, PATH and PRED. D-POSS-PRO-REF looks in WK-MEM in direction DIR for a frame with PATH having PRED as its predicate. If it finds this CON, it binds it to the gap associated with MYSLOT.

Example:
Assume the word “her” has this *frami* in WK-MEM:

CON5544 = (PRONOUN (**TYPE** (POSS))
(**REF** REF987))

The D-PRO-REF demon working for CON5544 will look BEFore in WK-MEM for a frame with GENDER = FEMALE and bind that CON (assume Sally is CON1122) to REF987, then the result will be: REF987 = CON1122.

This is a very simple-minded type of pronoun reference resolution, since it assumes that the closest earlier female mentioned must be the referent to “her”.

15. Write demon D-IMMED-F-NAME. This demon looks for a F-NAME immediately after its MYCON and binds the first-name found to gap of F-NAME.

Example: D-IMMED-F-NAME is associated with the frame for the word “son”. It will find (HUMAN (F-NAME (FREDDY))…) immediately after (SON…) in WK-MEM and will bind the gap of F-NAME to (FREDDY).

(So, if we hear “Joan has a son Mitch” we assume that the name of this son is Mitch.)

16. Write the demon D-POSS-PRO. This demon works for a family relation (like “son”) and hunts for a possessive pronoun preceding it. It assumes that the referent for that pronoun will fill the **OF** slot. It has 3 parameters: MYCON, DIR, and a PATH.

Example: “her son”
Assume “her” created CON0808 = (PRONOUN (**TYPE** (POSS))

(**REF** REF0808))
and that a demon already fired and set REF0808 = (HUMAN (**F-NAME** (SALLY)))

Assume "Son" created CON0909 = (HUMAN (**GENDER** (MALE))
(**REL** (SON (**OF** OF0909)))
(**F-NAME** F-NAME0909))
Now the demi (D-POSS-PRO CON0909 BEF (REL OF)) will fire. This demi looks for a PRONOUN of **TYPE** = (POSS). If it finds it, it looks for the value of REF which, in this case, will be REF0808 = (HUMAN (**F-NAME** (SALLY)))
This demi then binds OF0909 = REF0808.

(Thus, this demon is figuring out that "her son" means that this son happens to be the son of whoever "her" refers to.)

17. Write a demon D-ATTACH-SF. This demon has the parameters: MYCON, MYDIR, PRED, SLOT, FILLER. D-ATTACH-SF looks in direction MYDIR for a frame with predicate PRED and when it finds that CON, it inserts the slot-filler pair: (**SLOT** FILLER).

D-ATTACH-SF is used to modify frames and is usually spawned by adjectives in English, such as "teenage" and "a".

Examples:

When "teenage" is encountered, a demi D-ATTACH-SF inserts the slot-filler: (**AGE** (TEEN)) into the top-level of the frame:
(HUMAN (**GENDER** (MALE))
(**REL** (SON …)))

When "a" is encountered, another demi D-ATTACH-SF inserts the slot-filler: (**REF** (INDEF)) into the top-level of the frame: (VEHICLE)

18. Write a demon D-PRO-REF. This demon has the parameters: MYCON, DIR, SLOT1, FILLER, SLOT2. It looks for frami in direction DIR that has slot SLOT1 with filler FILLER and if it finds this CON, D-PRO-REF binds the gap of SLOT2 in MYCON to CON.

Example: When the word "he" is encountered:

Assume that MYCON = CON9999. The demi (D-PRO-REF CON9999 BEF GENDER MALE F-NAME) will be looking BEFore CON9999 for a frami with a GENDER slot having a filler whose predicate is MALE and it will find the frame for HANK and bind this CON-atom to the gap of F-NAME in CON9999.

19. Write a demon D-IMM-AFT. This demon takes a SLOT as input. It looks for a CON-atom immediately after its MYCON and looks to see if there is a non-nil filler for that SLOT. If it finds this filler, then it fills that same SLOT in the frame of its MYCON with the same value.

Example: in the case of: "son Freddy" assume that the CON-atom for "son" in WK-MEM is CON7777. So (D-IMM-AFT CON7777 F-NAME) will look for a F-NAME in the frame immediately after CON777 in WK-MEM and bind that filler, in this case (FREDDY), to the gap associated with the F-NAME slot in the frame of CON7777. So when "her son Freddy" is read, C-ANALYZE will realize that the F-NAME of SON is FREDDY.

20. Write the utility function NUM-SLOTS. NUM-SLOTS takes a *frami* as an input and returns the number of slots at all levels in the GAPVALS of that *frami*.
For example:

```
CON3311 = '(C-CAUSE
                    (ANTE ANTE8787)
                    (CONSEQ (GOAL-FAILURE
                                    (AGENT AGENT8787)
                                    (TIME (FUTURE))))
                    (JUSTIF JUSTIF8787))
where ANTE8787 = NIL
      AGENT8787= NIL
      JUSTIF8787 = (STEAL (AGENT
                                    (HUMAN (GENDER (MALE))
                                                (F-NAME (HANK)))))
(NUM-SLOTS CON3311) returns:  8
(NUM-SLOTS '(INVOLVE)) returns: 0
```

21. Write the function TOP-CON. This function takes **wk-mem** as input and it goes through it looking for the largest CON-atom that do not appear in the global list BOUND.

Example: WK-MEM = (CON1 CON2 …) where CON1 = frame for Sally,

CON2 = frame with UNKNOWN, etc.

The demons that bind a found CON-atom into the gap of some other frame have been using the utility function BIND. As a result, all these CON-atoms will be in the list BOUND except for CON3 = (MTRANS…) and the CON-atoms for (REASON-FOR) and (INVOLVE) and (PRONOUN) since these frames were never bound to a slot in another frame. Since CON3 = (MTRANS…) has the most atoms, TOP-CON will return CON3.

22. Write C-ANALYZE and try it out by executing: (C-ANALYZE PGS1 C-LEX)

Here is what C-ANALYZE should do:

a. C-ANALYZE will use on RECALL-WPH to retrieve the longest phrase in PGS1 that matches a word or phrase in C-LEX. It will first recall the wph: (SALLY).

As each word or phrase is retrieved from the conceptual lexicon C-LEX, C-ANALYZE will:

b. TOKENIZE the associated frame (to create a unique frame instance)

c. Create a CON-atom whose value is that frame and add the CON-atom to the end of WK-MEM

d. call on SPAWN to create and spawn any demon instances associated with that frame instance

e. call EXEC-DEMONS to repeatedly execute all active demons until they are quiescent.

Once all demons in the active DEMONS list are quiescent, C-ANALYZE will again use RECALL-WPH and repeat the above process, until SENT is empty. C-ANALYZE will then call on TOP-CON to return the CON-atom whose value is the largest frame that is not itself bound to any gap in any other frame in WK-MEM. Finally, C-ANALYZE will return the GAP-VALS (see HW-1) of this top, largest CON-atom.

If everything worked correctly, then C-ANALYZE should return the value of PIGCON1 (on the first page of this homework).

Congratulations! You have a simple conceptual analyzer up and working! ☺