cs161 SPR '13 **HW #1**

Due: Monday, April 15 by 11:59 PM. Submit a *single* Lisp file to CourseWeb, named hw1.lsp.

For this homework you are to implement the 7 Lisp functions listed below. Make sure that your function names and arguments match the ones given here *exactly*, or the grading script might give you a 0! To help with this, we have provided a code skeleton which you should download from CourseWeb. Feel free to define your own helper functions, but *do not* change the names or signatures of the top-level functions. You will be graded both on the examples given here *and* some additional cases, so test your code thoroughly! You may assume that the test case inputs conform to the specifications--input validation is not necessary.

These functions will operate on a Lisp list structure we will call a *frame*, which has the following syntax:

frame → (*pred (slot filler)*)* | ( ) where ()* indicated zero or more of ()
pred → *atom*
slot → *atom*
filler → *frame* | *gap*
gap → *atom* | *variable*
variable → *(V atom)*

So a *frame* consists of a *predicate* followed by zero or more *slot-filler* structures. A *slot* is just a Lisp atom. A *filler* can, recursively, itself be a *frame*. A *filler* can also be a *gap*, which will be either a Lisp atom or a *variable*. Variables are preceded by V.

Common *slots* will include: **AGENT, OBJECT, RECIP, SRC, DEST, ANTE, CONSEQ, LOC, TIME, F-NAME, L-NAME, GENDER.**

Common *pred* instances will include: INGEST, HUMAN, ACT, FOOD, MALE, FEMALE, CAUSE, STATE, PTRANS, ATRANS, MTRANS.

To make it easier to spot a *slot* I have indicated them here in **bold**. I've also done some indenting, but notice that the only thing that distinguishes different types of atoms in a frame is their relative position in a list.

We will add more slots and predicates as needed. In general, there would a finite number of *slots*, but an unlimited number of *preds*.

We will represent concepts by filling in (instantiating) the slots of frames with different content. We will refer to instantiated frames as "concepts".
Here is an example of how the meaning of sentence S1 (below) can be represented as an instantiated frame, here assigned as the value of the atom C1.

Note: "atom = val" below indicates that atom has been assigned a value of val.

S1 = (A pretty young girl tells John that she ate the big apple)

```
C1 =
(MTRANS
   (AGENT (HUMAN (GENDER (FEMALE))
                 (APPEARANCE (>NORM))
                 (AGE (<NORM))
                 (REF (INDEF))))
   (RECIP (HUMAN (GENDER (MALE))
                 (F-NAME (JOHN))))
   (OBJECT (INGEST (AGENT (HUMAN (GENDER (FEMALE))))
                   (OBJECT (FOOD (TYPE (APPLE))
                                 (SIZE (>NORM))
                                 (REF (DEFINITE))))
                  (TIME (PAST)))))
```

Note: In functional and logic notations, the order of the arguments matters, but in our frame notation, the order of the slot-fillers does <u>not</u> matter.

For example, these frames are semantically equivalent:

```
(pred1 (AGENT (pred2))   (OBJECT (pred3)))
(pred1    (OBJECT (pred3)) (AGENT (pred2)))
```

Here is another example:
S2 = Fred makes Betty ecstatic by telling Betty that she's pretty.

C2 =

```
(CAUSE
   (ANTE (INFORM (AGENT (HUMAN (F-NAME (FRED))
                               (GENDER (MALE))))
                 (RECIP (HUMAN (GENDER (FEMALE))
                               (F-NAME (BETTY))))
                 (OBJECT (STATE (VALUE (>NORM))
                                (TYPE  (APPEARANCE))
                                (AGENT (HUMAN
                                         (GENDER (FEMALE))))))))
   (CONSEQ (STATE (AGENT (HUMAN (GENDER (FEMALE))
                                (F-NAME (BETTY))))
                  (VALUE (>>NORM))
                  (TYPE (HAPPY)))))
```

Frames will get linked to one another by binding a *gap (*that appears in the *filler* of a *frame*) to an atom that is bound to another frame. This can be accomplished by SET or SETQ in Lisp.

For example, here is a frame with *gaps* (indicated with uppercase italics)

S3 = (Frank walks to Frank's car)

```
C3 = (PTRANS  (AGENT AGENT0)
              (OBJECT OBJECT0)
              (SRC  LOC0)
              (DEST  LOC1)
              (INSTRU (MOVE (AGENT AGENT0)
                            (B-PART (LEGS)))))
```

Where each gap has the following value:
AGENT0 = HUM1
HUM1 = (HUMAN (**F-NAME** *F-NAME0)*
(**GENDER** (MALE)))
F-NAME0 = (FRANK)
OBJECT0 = HUM1
LOC0 = NIL
LOC1 = (LOC (**PROX** (AUTO (**OWNER** *AGENT1*))))
AGENT1 = HUM1
Note: The process of linking up gaps with their fillers is called instantiation.

You are to define the following 7 Lisp functions that will access frames:

1. (FILLER slot frame) takes a **slot** and a **frame** and returns the *filler* of that slot. Note that **slot** must be a top-level slot (i.e., cannot be some slot embedded within a subframe of that concept). If there is not that **slot** in the **frame**, then NIL is returned.

Examples:
(FILLER 'AGENT C1) returns:

```
                (HUMAN (GENDER (FEMALE))
                       (APPEARANCE (>NORM))
                       (AGE (<NORM))
                       (REF (INDEF)))
```

(FILLER ‘SRC C1) returns: NIL

(FILLER ‘OBJECT C1) returns:

```
          (INGEST (AGENT (HUMAN (GENDER (FEMALE))))
                  (OBJECT (FOOD (TYPE (APPLE))
                                (SIZE (>NORM))
                                (REF (DEFINITE))))
                  (TIME (PAST)))
```

(FILLER ‘DEST C3) returns: *LOC1*
(FILLER ‘OWNER LOC1) returns: NIL
(FILLER ‘PROX LOC1) returns: (AUTO (**OWNER** *AGENT1*))

2. (PATH-SL slots concept) takes a list of slots and uses them to path into a concept and return the frame that is being sought. Notice that the preds are ignored.

Examples:

```
(PATH-SL ‘(ANTE AGENT) C2) returns:  (HUMAN (F-NAME (FRED))
                                             (GENDER (MALE)))
```

(PATH-SL ‘(**CONSEQ AGENT F-NAME**) C2) returns: (BETTY)
(PATH-SL ‘(**OBJECT AGENT F-NAME**) C1) returns: NIL
(PATH-SL '(**DEST**) C3) returns: *LOC1*
(PATH-SL ‘(**F-NAME**) HUM1) returns: *F-NAME0*

3. (PATH-PS path concept) is like PATH-SL but the path now has a pred in between each slot-name. This list must contain alternating pred-slot pairs and start with a pred and end with a slot-name.   PATH-PS makes sure that the preds must conform to those in the path. If any pred fails to match, then the atom FAIL is returned. If any slot is missing, then NIL is returned.

<u>Examples</u>:
(PATH-PS '(CAUSE **CONSEQ** STATE **VALUE**) C2) returns: (>>NORM)
(PATH-PS ‘(CAUSE **TIME**) C2) returns: NIL
(PATH-PS ‘(INFORM **ANTE**) C2) returns: FAIL

4. **(GAPVALS frame)** takes a frame, or an atom that has a frame as its value. GAPVALS generates a new frame with <u>all</u> gaps replaced, recursively, by their values. (Note: A gap can be bound to an atom that is itself bound to a frame.)

<u>Examples:</u>
(GAPVALS ‘F-NAME0) returns: (FRANK)

```
(GAPVALS ‘AGENT0) returns:  (HUMAN (F-NAME (FRANK))
                                   (GENDER (MALE)))
(GAPVALS C3) returns:
   (PTRANS (AGENT (HUMAN (F-NAME (FRANK))
                         (GENDER (MALE))))
           (OBJECT (HUMAN (F-NAME (FRANK))
                          (GENDER (MALE))))
         (SRC ( ))
         (DEST (LOC (PROX
                      (AUTO (OWNER
                              (HUMAN
                                 (F-NAME (FRANK))
                                 (GENDER (MALE))))))))
         (INSTRU (MOVE (AGENT (HUMAN (F-NAME (FRANK))
                                     (GENDER (MALE))))
                       (B-PART (LEGS)))))
```

If a frame has no gaps in it, then GAPVALS returns a copy of the frame.

<u>Example:</u>
(GAPVALS C1) returns:

```
   (MTRANS
     (AGENT (HUMAN (GENDER (FEMALE))
                    (APPEARANCE (>NORM))
                    (AGE (<NORM))
                    (REF (INDEF))))
     (RECIP (HUMAN (GENDER (MALE))
                   (F-NAME (JOHN))))
     (OBJECT (INGEST (AGENT (HUMAN (GENDER (FEMALE))))
                   (OBJECT (FOOD (TYPE (APPLE))
                                 (SIZE (>NORM))
                                 (REF (DEFINITE))))
                  (TIME (PAST)))))
```

5. (REPLACE-SF slot filler frame) replaces a top-level **slot** with **filler** in **frame**. If the **slot** does not appear at the top level of the **frame**, then REPLACE-SL <u>adds</u> the slot-filler pair at the top level.

<u>Example</u>:
(REPLACE-SF 'RECIP '(HUMAN) C1) returns:

```
(MTRANS
   (AGENT (HUMAN (GENDER (FEMALE))
                  (APPEARANCE (>NORM))
                  (AGE (<NORM))
                  (REF (INDEF))))
   (RECIP (HUMAN))
   (OBJECT (INGEST (AGENT (HUMAN (GENDER (FEMALE))))
                 (OBJECT (FOOD (TYPE (APPLE))
                               (SIZE (>NORM))
                                (REF (DEFINITE))))
                (TIME (PAST)))))
```

(REPLACE-SF 'TIME '(PAST) C3) returns:

```
   (PTRANS  (AGENT AGENT0)
            (OBJECT OBJECT0)
```

```
                (SRC  LOC0)
                (DEST  LOC1)
                (INSTRU (MOVE (AGENT AGENT0)
                              (B-PART (LEGS))))
                (TIME (PAST)))
```

6. (REPLACE-PTH path filler frame) – REPLACE-PTH is like REPLACE-SF but inserts **filler** as specified by the **path** of slot-names, in which the last slot-name in **path** gets **filler** as its value.

C2 =

```
(CAUSE
   (ANTE (INFORM (AGENT (HUMAN (F-NAME (FRED))
                               (GENDER (MALE))))
                 (RECIP (HUMAN (GENDER (FEMALE))
                               (F-NAME (BETTY))))
                 (OBJECT (STATE (VALUE (>NORM))
                                (TYPE  (APPEARANCE))
                                (AGENT (HUMAN
                                          (GENDER (FEMALE))))))))
   (CONSEQ (STATE (AGENT (HUMAN (GENDER (FEMALE))
                                (F-NAME (BETTY))))
                  (VALUE (>>NORM))
                  (TYPE (HAPPY)))))
```

<u>Example</u>:

(REPLACE-PTH '(CONSEQ AGENT) '(HUMAN) C2) returns:

```
(CAUSE
   (ANTE (INFORM (AGENT (HUMAN (F-NAME (FRED))
                               (GENDER (MALE))))
                 (RECIP (HUMAN (GENDER (FEMALE))
                               (F-NAME (BETTY))))
                 (OBJECT (STATE (VALUE (>NORM))
                                (TYPE (APPEARANCE))
                                (AGENT (HUMAN
                                          (GENDER (FEMALE))))))))
```

```
    (CONSEQ (STATE (AGENT (HUMAN))
                   (VALUE (>>NORM))
                   (TYPE (HAPPY)))))
```

7. (COMPARE-FRMS frame1 frame2) – returns T if frame1 and frame2 have the same slot-filler structure.

<u>Example:</u>

```
(COMPARE-FRMS
   (GAPVALS C1)
   ‘(MTRANS
      (RECIP (HUMAN (GENDER (MALE))
                    (F-NAME (JOHN))))
      (AGENT (HUMAN (GENDER (FEMALE))
                    (APPEARANCE (>NORM))
                    (AGE (<NORM))
                    (REF (INDEF))))
      (OBJECT (INGEST (AGENT (HUMAN
                               (GENDER (FEMALE))))
                      (OBJECT (FOOD (TYPE (APPLE))
                                    (SIZE (>NORM))
                                    (REF (DEFINITE))))
                      (TIME (PAST))))))
```

returns: T (because they have the same slot-filler structure. Remember, the order in which the slots appear at a given level is irrelevant.)

Note: Lisp is a *recursive* language, designed to operate on *recursive* data structures (lists). Life will be much easier for you if you avoid implementing these functions by means of iteration. Embrace the recursion! ☺