The Frame Representation Language (FRL) is described. FRL is an adjunct to LISP which implements several representation techniques suggested by Minsky's concept of a frame: defaults, constraints, inheritance, procedural attachment, and annotation.
THE FRL MANUAL

R. Bruce Roberts and Ira P. Goldstein

The Frame Representation Language (FRL) is described. FRL is an adjunct to LISP which implements several representation techniques suggested by Minsky's [75] concept of a frame: defaults, constraints, inheritance, procedural attachment, and annotation.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. It was supported in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.
Table of Contents

1. WHAT IS A FRAME IN FRL? 3
2. FRAMES AND THEIR NAMES 4
3. ADDING AND REMOVING FRAMES 4
4. THE AKO AND INSTANCE SLOTS 5
5. ADDING AND REMOVING PARTS OF A FRAME 8
6. RETRIEVING PARTS OF A FRAME 8
   Evaluation 9
   The % Special Data Form 9
   Indirection 9
   The @ Special Data Form 9
   The Non-atomic Slot Convention 10
   Inheritance 10
   Frame Data retrieval 12
   Retrieval Functions in a Frame Environment 13
   The ! Special Form 13
   The & Special Form 13
7. THE SEVEN (PLUS OR MINUS TWO) FACETS OF KNOWLEDGE 13
   The $VALUE facet 13
   The $DEFAULT facet 14
   The $IF-ADDED and $IF-REMOVED facets 14
   The $IF-NEEDED facet 15
   The $REQUIRE facet 15
   Checking requirements 16
   Utility functions for predicates 16
8. ANNOTATING DATA IN FRAMES 17
9. SAVING FRAMES IN A FILE 17
Bibliography 19
Appendix A -- The FRL LISP Environment 20
Appendix B -- Frames are built out of FLISTS 23
Appendix C -- Index to FRL Functions 25
Appendix D -- Table of Data Retrieval Functions 26
Appendix E -- The FRL Trace Function 28
Appendix F -- The FRL Frame Editor 29
Figure 1 shows FRL from the larger perspective of intelligent support systems. FRL comprises the two bottom layers: a specialized data structure (the frame) and a collection of LISP functions for defining frames, storing and retrieving information. It has been used to implement NUDGE [Goldstein & Roberts 77], a system for maintaining a person’s schedule of activities in the face of individual preferences, conflicting constraints, and changing plans; PAL, a natural language front end for NUDGE [Bullwinkle 77]; a system to assist planning a birthday party [Clemenson 77]; TRIPPER, a knowledge base for places and travel around the country [Jeffery 77]; a representation for the discourse structure of news articles [Rosenberg 77]; and COMEX [Stansfield 77], a system for understanding discourse about the commodities market.

The intellectual issues surrounding the representation techniques provided by FRL are discussed in [Goldstein & Roberts 77]. A primer [Roberts & Goldstein 77] is available consisting of an extended example using FRL and an abridged version of this manual.

I. WHAT IS A FRAME IN FRL?

An FRL frame is implemented as nested association lists with at most five levels of embedding. The respective sub-structures of a frame are slot, facet, datum, comment and message. The overall structure of a frame is:

```
(frame1
  (slot1 (facet1 (datum1 (label1 message1 message2 ... more Messages ... )
      ... more Comments ...)
    (datum2 (label1 message ....))
      ... more Data ....)
    (facet2 (datum1 (label1 message1 message2 ....)))
      ... more Facets ....)
  (slot2 (facet1 (datum1 (label1 message ....) ....)...)
      ... more Slots ....)
```

We will refer to the first element in one of these sub-structures as the indicator (said to name the structure) and the remaining elements collectively as the bucket (in case of a slot, the bucket is a list of facets, for example). A path of indicators identifies a sub-structure in a frame. The order of sub-structures at any level in a frame is insignificant. In practice, facet names conventionally have a prefix "$"; labels, a suffix ".". This is simply to facilitate their recognition by the programmer.
2. FRAMES AND THEIR NAMES

(frame frame)

returns a frame structure of the kind shown on the preceding page. An error if frame is neither a frame name nor a frame structure.

Unless stated otherwise, a frame argument to any function can be either the name of a frame or the frame structure itself.

(frame? frame)

is like FRAME except returns NIL if frame doesn’t exist.

(frame frame)

is like FRAME except that if frame is a name for a nonexistent frame, a frame is F CREATED and returned.

(fname frame)

returns the name of frame. An error if frame is neither a frame name nor a frame structure.

(fname? frame)

is like FNAME except returns NIL if frame doesn’t exist.

(fslots frame)

returns a list of the slot names in frame.

(fcopy frame)

returns a copy of frame.

3. ADDING AND REMOVING FRAMES

(fassert name slot1 slot2 ... slotN)

creates a frame called name (if it doesn’t already exist) containing the slots slot1 ... slotN. If the name frame exists, the new information in the slots is merged with the existing slots. The frame is stored as the FRAME property of name and name is added to FRAMES, the list of known frames. FASSERT is a FEXPR.

The FASSERT switch. If FASSERT is nil, FASSERT forms are not interpreted. This is convenient for selectively reading just the code in a file containing intermixed code and frame definitions.
(FERASE frame)
removes frame from the FRAME property of its name and its name from the list «FRAME».

FASSERT and FERASE can cause side-effects if the frame being added or removed contains slots with values. These values are added or removed individually using FPUT and FREMOVE respectively and may trigger the execution of attached procedures. These issues will be considered in greater detail shortly. DEFRAME and FDESTROY, unlike FASSERT and FERASE, have no side-effects.

(DEFRAME name slot1 slot2 ... slotN)
creates a frame name containing precisely the slots slot1 ... slotN. If the name frame already exists, its previous definition is FDESTROYed. DEFRAME is a FEXPR.

The DEFRAME switch. If DEFRAME is nil, DEFRAME forms are not interpreted. This is convenient for selectively reading just the code in a file containing intermixed code and frame definitions.

(FDESTROY frame)
removes frame from the FRAME property of its name and deletes its name from the list «FRAME».

(FRESET)
removes all frames from the system. It uses FDESTROY.

(FCREATE {name})
creates an empty frame and returns its name. The name will be unique. Name, if given, will be used instead, although it may be modified by FGENAME to guarantee uniqueness. Frame names must be atomic.

Note: Bracketed expressions -- { ... } -- are used in this manual to denote optional arguments. An unbalanced right bracket -- } -- denotes the possible end of the argument list for a LEXPR.

(FGENAME name)
returns a guaranteed unique frame name by adding a numerical suffix to name.

4. THE AKO AND INSTANCE SLOTS

A slot is a generalization of the attribute-value pair in the traditional LISP property list representation. $VALUE$ is the slot "facet" which indicates its values. Five
other facets indicate other types of knowledge associated with the slot. Data in the $DEFAULT facet supplies defaults. Data in $IF-ADDED and $IF-REMOVED facets are procedures triggered whenever a slot value is added or removed. $IF-NEEDED data are procedures which may compute a slot value. The $REQUIRE facet holds predicates which describe and restrict the value.

Two slots are recognized by FRL system functions: AKO (A Kind Of) and INSTANCE. These define a relation between frames along which data is inherited. FRL maintains AKO and INSTANCE as inverses. The AKO relation can be used to establish a conceptual hierarchy of frames in which general information stored higher in the hierarchy is inherited by more specialized concepts lower in the hierarchy. Functions like FGET implement this inheritance mechanism.

A relation between frames is defined by making the name of one frame the value of a slot in another frame. The slot names the relation. A tree of frame relations is possible since a slot can have many values. Several functions are provided to examine these relations.

(FCHILDREN frame slot)

returns a list of the immediate inferiors of frame along the relation named by slot. This is just a list of values of slot.

(FTREE frame slot)

returns a tree of the form (root subtree1 subtree2 ... ) with frame at the root; each subtree's root is a child of frame along the relation named by slot.

(FDESCENDANTS frame slot)

returns a list of all inferiors of frame along the relation slot defines. That is, it includes all the frames occurring in the "tree" of FTREE except the root frame.

(FRINGE frame slot)

returns a list of all "leaves" on the tree of (FTREE frame slot).

(FLINK? slot f1 f2)

returns T only if a path exists from f1 to f2 following only the slot "link"; i.e., if one of the values of slot of f1 is f2, or FLINK? is true for any of these values.

(AKO? f1 f2)

returns T only if f1 is a kind of f2. Equivalent to (FLINK? 'AKO f1 f2). Similar definitions are possible for any slot whose value is another frame.

One frame is predefined in FRL: THING. A partial definition of it follows:
(THING
  (AKO ($IF-ADDED ( (ADD-INSTANCE) ))
  ($IF-REMOVED ( (REMOVE-INSTANCE) )))
  (INSTANCE ($IF-ADDED ( (ADD-AKO) ))
  ($IF-REMOVED ( (REMOVE-AKO) ))))
)

(FINSTANTIATE frame {name})

creates an instance of frame (using FCREATE); i.e., it possesses only an AKO link to
frame. Its name is derived from name (using FGENAME) and will be unique. The
newly created frame is returned.

The existence of the AKO link implies a heritage for a frame (or any part of a
frame) consisting of all information both in that frame and in all the frames accessible
along the AKO link.

(FHERITAGE frame slot facet datum label)

returns a structure formed by merging the structure pointed to by the indicator path --
the arguments to FHERITAGE -- with all corresponding structures of the frames
accessible along the AKO link.

The IN Comment. Each datum in the heritage will have a comment -- (IN:
frame) -- added by FHERITAGE to record the frame in which the datum actually
occurs.

(FHERITAGE-SLOTS frame)

returns a list of slot names occurring in (FHERITAGE frame).

By convention, frames in an AKO hierarchy are distinguished as being either
 GENERIC or INDIVIDUAL by the value of their CLASSIFICATION slot. Two
predicates test the classification of a frame.

(INDIVIDUAL? frame)

returns T only if frame is marked as an individual. INDIVIDUAL? returns NIL if
frame is generic, and ? otherwise.

(GENERIC? frame)

is defined analogously to INDIVIDUAL?. 
5. ADDING AND REMOVING PARTS OF A FRAME

(FPUT frame slot) (facet) (datum) (label) (message)

adds the last argument at the point in frame named by the indicator path (the intervening arguments) and returns the modified frame. Adding new information to a frame is a merging process that retains the uniqueness of each indicator. FPUT is a LEXPR and can take from 2 to 6 arguments. It can be used to add an element anywhere in a frame; to add a slot name to frame or to put a message in a comment labeled label.

FPUT has a side-effect: Putting data items into a $VALUE facet triggers the execution of all procedures in the $IF-ADDED facet of the slot.

(FPUT-STRUCTURE frame)
(FPUT-STRUCTURE frame slot-structure)
(FPUT-STRUCTURE frame slot facet-structure)
(FPUT-STRUCTURE frame slot facet datum-structure)
(FPUT-STRUCTURE frame slot facet datum comment)

This family of FPUT-STRUCTURE functions differs from FPUT only in that the last argument is considered to be an entire sub-structure (rather than an indicator). The entire structure is merged into frame. Like FPUT, FPUT-STRUCTURE may trigger $IF-ADDED procedures.

(FREMOVE frame slot) (facet) (datum) (label) (message)

deletes the sub-structure of frame indicated by the path slot - facet - datum ... It returns the modified frame. FREMOVE is a LEXPR taking from 2 to 6 arguments. The structure deleted will have had as its indicator the final argument to FREMOVE.

FREMOVE has a side-effect: If any data in a $VALUE facet is deleted by this command, all procedures in the $IF-REMOVED facet of the slot are executed.

(FREPLACE frame slot) (facet) (datum) (label) (message)

like FREMOVEing all existing items following by FPUT with the arguments given. Following a call to this function, the only item present at the terminus of the indicator path is the final argument.

(FDELETE frame slot) (facet) (datum) (label) (message)

like FREMOVE except never triggers any side-effects. The portion of frame identified by the indicator path is simply excised.

6. RETRIEVING PARTS OF A FRAME

The following questions should be kept in mind when retrieving data from a facet.
What is the expected form of the data?
How is the data inherited?
How does it interact with other facets?
Are any special comments associated with the data?

But before considering individual facets, three general properties of facets and their data will be discussed: evaluation, indirection, and inheritance.

**Evaluation**

Normally, data in a frame is interpreted literally. The access functions return just what one sees in a frame if it were printed out. Data can be computed however, and to specify that a datum is to be evaluated whenever accessed, FRL provides:

*The % Special Data Form.* A percent sign prefixed to a datum causes the evaluated datum to be returned whenever it is accessed.

The implementation of % as a prefix character requires that it be defined as a readmacro in Lisp. See Appendix A for other changes to the standard MACLISP environment necessitated by FRL’s operation.

The data element is evaluated in a particular frame environment, as determined by the frame, slot, and facet named in the retrieval request. The global variables :FRAME, :SLOT, and :FACET at the time of evaluation can be assumed to be locally bound to the names of the "current" frame, slot, and facet. Because of indirection and inheritance, the frame environment may not be the one in which the datum actually lies. Situations may arise when the user will want to explicitly establish a frame environment for the evaluation of an expression. A function has been provided to facilitate this.

```lisp
(FEVAL s-expression frame slot facet)
```

binds :FRAME, :SLOT, and :FACET to the values supplied. It then evaluates the S-expression and returns the result. If an argument is missing or nil, the prior (higher) binding is unaffected.

**Indirection**

Datum in one frame can be retrieved indirectly by a request for datum in a different frame. This indirection is denoted by:

*The @ Special Data Form.* A datum with a prefix atsign is interpreted as an indirection pointer to all the data in another frame. The pointer is an indicator path: frame, slot, facet. When accessed, the data items pointed to by the indirection are copied and spliced together with any other items in the facet (generally, a facet can have many data items). The behavior of indirectly accessed items is equivalent to the local items.
A related convention allows one to define a slot in a frame to hold information accessed indirectly by another.

The Non-atomic Slot Convention. If a slot is created whose indicator is non-atomic, the CAR of the slot name is considered to name a frame and the CADR a slot in frame. An indirection pointer is put in each of the existing facets of the indicated slot in frame pointing back to the corresponding facet of the slot just created.

Comments:

- Each indirect datum returned will receive a comment of the form (IN: frame). Frame is the name of the target frame lying at the end of the indirection chain.

- Evaluation and indirection are mutually exclusive. A datum may be evaluated, expanded as an indirection pointer, or receive no special processing.

- How does evaluation differ from indirection? Evaluation returns a single datum. Indirection causes a list of data items to be appended to the list of structures returned from the local frame.

- The target of an indirection pointer can be another indirection pointer, in which case the process is repeated. If the target is to be evaluated (i.e., it is a % Special Data From) the evaluation is performed in the frame environment established by the original request.

- The elements of an indirection pointer are evaluated in the frame environment of the indirection pointer.

- Indirection pointers with less than three elements are extended using the :SLOT and :FACET of the current frame.

Inheritance

The AKO relation can be used to establish a hierarchy of frames in which general information stored higher in the hierarchy is inherited by more specialized concepts lower in the hierarchy. These three functions return data inherited along the AKO link of a frame.

(FINHERIT frame slot facet)

looks first for data in the slot and facet of frame. If data exists, a list of the datum structures is returned. If no data is found, the corresponding facet of the frame named in frame's AKO slot is inspected for data; and so on until a frame is found containing data -- which is then returned.

Comments:

- Inheritance stops at the first frame along the chain of AKO links whose selected facet
contains some data. This precedes any processing of special indicators for indirection and evaluation; hence, an indirect link and a to-be-evaluated datum are seen as non-empty data for the purpose of controlling inheritance. This fact can be used to construct a datum to "mask" the existence of data lying further along the AKO chain. The form -- %NIL -- as the datum element, being non-nil itself, will stop the inheritance of any data from AKO frames; and, assuming it is the only datum element in the facet, will subsequently be evaluated and return NIL.

If no data is found, FINHERIT returns NIL.

A frame can be A-Kind-Of more than one other frame; i.e., have more than one value in its AKO slot. FINHERIT traces each of the AKO paths, stopping at the first data encountered along each, and returns a list of all data thus found appended together.

The FINHERIT Comment. A comment -- (FINHERIT: CONTINUE) -- on any datum structures in a facet causes the inheritance to proceed further along the AKO link as if no data had been found; it returns the local data appended to that found further along the link.

The IN Comment. A comment -- (IN: frame) -- is inserted in each datum returned by the inheritance process, where frame is the name of the frame which actually held the datum.

Subsequent evaluation of inherited data is done in the Frame Environment of the original call to FINHERIT.

The inheritance process defined by FINHERIT is applicable to any facet. The following two variations treat the $VALUE facet specially. In both cases, the inheritance along the $VALUE facet interacts with the $DEFAULT facet.

(FINHERITI frame slot facet)

Like FINHERIT except if facet = $VALUE, before following the AKO path to look for a value, it inspects the $DEFAULT facet of slot. This process is repeated at each step up the AKO path. If no values are found, but defaults exist, they are returned instead.

(FINHERIT2 frame slot facet)

Like FINHERIT except if facet = $VALUE, it is equivalent to:

(OR (FINHERIT frame slot $VALUE) (FINHERIT frame slot $DEFAULT))

Frame Data retrieval

(FGET frame slot facet)

returns a list of all the data items in facet of slot in frame. The data is accessed using the function FINHERITI. Several data items are possible, thus a list is returned. Any
Special Forms are converted as described in section 4. Each element in the returned list is a complete data item; i.e., its bucket still contains the comments. FGET returns a list of all the indicators in the bucket addressed by the path of arguments. Usually, three arguments are given. The value of a slot is retrieved by (FGET frame slot $VALUE). FGET looks first in the slot of frame. If data exists, a list of the items is returned. If no data is found, the facet of the frame named in frame's AKO slot is inspected; and so on until a frame is found containing data, which is then returned.

An important special case is FGETting from a $VALUE facet. If still no value is found, FGET repeats, looking in the $DEFAULT facet instead.

The following questions represent useful distinctions to make in retrieving data from a frame database.

- How many items of data are expected?
- Should the data be returned with its Comments?
- Should data marked for evaluation be evaluated?
- Should indirection pointers be chased and the data thus found be included?
- If the frame and slot specified do not yield any data, should any attempt be made to inherit? And if so, what kind? i.e., NONE, FINHERIT, HERITAGE, and, in the case of $VALUE, FINHERIT1 or FINHERIT2.
- Should any $IF-NEEDED procedures be attempted? And if so, what kind? I.e., NONE, IMMEDIATE, REQUEST, DEFAULT, etc.

The FGET function can be parameterized along these dimensions as follows:

\[ \text{(FGET frame slot facet \{keywords\})} \]

returns data from the indicated facet according to the contents of the keywords list.

Allowable keywords are:

<table>
<thead>
<tr>
<th>A / O</th>
<th>All / One</th>
</tr>
</thead>
<tbody>
<tr>
<td>C / -C</td>
<td>Comments / NoComments</td>
</tr>
<tr>
<td>% %</td>
<td>Evaluation / NoEvaluation (Must be slashified)</td>
</tr>
<tr>
<td>@ @</td>
<td>Indirection / NoIndirection (Must be slashified)</td>
</tr>
<tr>
<td>0 / 1 / 2 / H / -H</td>
<td>FINHERIT, -1, -2 / Heritage / NoHeritage</td>
</tr>
</tbody>
</table>

The upper case letters in each keyword are useful abbreviations. As described, FGET without a retrieval key is equivalent to the specification: (A -C % @ l). Omitted keywords will be supplied from this default specification.

The choice of retrieval keys affects the form of the returned data. ONE and ALL imply a single item or a list of items is returned, respectively. COMMENTS requires that the returned object be in the form of a bucket; whereas the objects returned under NOCOMMENTS are indicators.
Appendix D lists numerous synonyms for common variations of FGET. Retrieving information from a frame database is also accomplished by matching a frame pattern against the frames in the database. The function FFIND will be presented in a forthcoming paper [Rosenberg & Roberts 77] which discusses matching frames in FRL.

Retrieval Functions in a Frame Environment

Frequently one writes a value retrieval expression to be evaluated in a frame environment (i.e., where :FRAME, :SLOT, and :FACET are externally bound); for example, inside an attached procedure. Two special abbreviation forms are recognized in this case to facilitate writing expressions for retrieving the value of a slot using FINHERIT.

The ! Special Form

!(frame slot) = (FGET frame slot '(0 -C /% 0))
!(slot) = (FGET :FRAME slot '(0 -C /% 0))
!slot = (FGET :FRAME slot '(0 -C /% 0))
!!<as above> = (FGET ... '(A -C /% 0))

The & Special Form

&(frame slot) = (FGET frame slot '(0 C /% 0))
&&<as above> = (FGET ... '(A C /% 0))

In both the ! and & special forms, ! forms can be substituted for the slot and frame. For example, if the MEETING frame has slots WHO and WHERE, an expression -- !(WHO OFFICE) -- appearing in the $IF-NEEDED procedure of WHERE means the value of the OFFICE slot in the frame for the participant (WHO) of MEETING.

7. THE SEVEN (PLUS OR MINUS TWO) FACETS OF KNOWLEDGE

Several facets have been mentioned so far as participating in the storage and retrieval of information in a frame. This section answers in detail the questions raised in section 6.

The $VALUE facet

Data: The data in a $VALUE facet is an arbitrary S-expression.

Inheritance: FINHERIT, FINHERIT1 or FINHERIT2

Interactions: The $VALUE facet interacts with all other facets.
The $DEFAULT facet

Data: The data in a $DEFAULT facet is an arbitrary s-expression.

Inheritance: FINHERIT.

Interactions: The $VALUE facet (via FINHERITI and FINHERIT2).

The $IF-ADDED and $IF-REMOVED facets

Data in the $IF-ADDED and $IF-REMOVED facets is treated as LISP forms. The forms in the $IF-ADDED facet will be evaluated whenever a value is added to the slot (i.e., in the $VALUE facet) by FASSERT or FPUT. The forms in the $IF-REMOVED facet will be evaluated whenever a value is deleted from a slot (i.e., from the $VALUE facet) by FERASE or FREMOVE.

- No $IF-ADDED procedure will be run if the value was already there. This serves to eliminate loops.

- No $IF-REMOVED procedure will be run if the value was not actually there to be removed.

- The order in which the procedures are run is not fixed.

- The procedures will be run in a frame environment in which the following free variables have been bound:
  - :FRAME = frame
  - :SLOT = slot
  - :FACET = $IF-ADDED or $IF-REMOVED (as appropriate).

In addition, the free variable "VALUE" will be bound to the datum whose addition or removal caused the execution of the attached procedures.

- IF-ADDED and IF-REMOVED procedures are inherited using FINHERIT.

- The APPLY Convention. Interpreting data in the $IF-ADDED and $IF-REMOVED facets as procedures permits the convention that if it is atomic, rather than EVAL'ing it, it is considered the name of a function of no arguments and APPLY'ed to NIL.

\[
(\text{FRUN } s\text{-expression (frame) slot facet})
\]

like FEVAL except for the manner in which it handles atoms. If $s\text{-expression}$ is atomic, (APPLY atom NIL) is evaluated and the result returned.

The $IF-NEEDED facet

Data: LISP procedures.
Inheritance: FINHERIT.

Interactions: The $VALUE facet.

No explicit functions are predefined to interact with $VALUE because personal conventions are so easily established. For example, a hypothetical:

\[
(FGET-AS-NEEDED \text{ frame slot})
\]

is equivalent to
\[
(\text{OR} \ (FGET \text{ frame slot}) \ (FNEED \text{ frame slot}))
\]

where FNEED is predefined:

\[
(F \cdot \cdot \cdot \text{FD frame slot \{types\}})
\]

runs the $IF-NEEDED procedures associated with frame and slot; and if one of them returns a value, FNEED returns it. Optionally, only those with a comment of the form (TYPE: type) are attempted, where type is an element of the types list. Suggested useful restricting comments are: request, immediate, and deduce.

Comments:

- The APPLY convention. [See $IF-ADDED]
- Frame Environment. [See $IF-ADDED]
- The $IF-NEEDED Convention. $IF-NEEDED procedures should be written to return nil if they fail to add a value to the slot.

The $REQUIRE facet

Data items in the $REQUIRE facet should be a LISP predicates which describe allowable values for the slot. There is an implicit conjunction between all data items present. Consistent with the view of specialization as involving additional restrictions on more general concepts, $REQUIRE data is inherited by taking the Heritage. The predicates are evaluated in the appropriate frame environment, as with the other procedural knowledge already discussed.

Checking requirements In FRL, requirement checking is done using the following function to maintain the so-called $VALUE convention.

\[
(F\text{APPLY-CONSTRAINTS constraints values})
\]

returns a poll (see FPOLL) produced by evaluating each of the constraints. A constraint is any S-expression with a Boolean value. FAPPLY-CONSTRAINTS binds
the free variables :VALUE and :VALUES, by which constraints can refer to potential values. If values has only one element, it is bound to :VALUE and values to :VALUES; otherwise, :VALUE = NIL.

(FPOLL predicates)

evaluates the predicates and records whether each was T, NIL, or caused an error. Returns a poll:

\[
\begin{align*}
\langle \text{summary}\rangle & \ (T \ldots \text{true predicates} \ldots) \\
\langle \text{NIL}\rangle & \ (\ldots \text{false predicates} \ldots) \\
\langle ? \rangle & \ (\ldots \text{error-producing predicates} \ldots)
\end{align*}
\]

where the \langle \text{summary}\rangle is T only if all are true, NIL only if some are false and none produce errors, and ? otherwise.

(FPOLL-SUMMARY predicates)

like FPOLL but returns only the "summary" portion, not the entire poll.

(FCHECK frame slot [values])

returns a poll of all constraints in the $REQUIRE facet of slot in frame applied to the values of the slot. Both local and inherited constraints are included. If optional values are supplied, they are checked against the constraints instead. Constraints are run in a Frame Environment with :FRAME, :SLOT and :FACET bound. Moreover, :VALUE and :VALUES are bound as described in FAPPLY-CONSTRAINTS.

Utility functions for predicates The treatment of predicates has been extended to include an explicit value for unknown, ?, as well as T or NIL.

(TRUE? x)

returns T only if x is neither NIL nor ?.

(FALSE? x)

returns T only if x is NIL.

(UNKNOWN? x)

returns T only if x is ?. The value of ? is ?.

8. ANNOTATING DATA IN FRAMES

Any data item can have several comments. Three labels are recognized by FRL:
IN:
The accompanying message is the name of the frame in which the data is stored. This comment is added automatically by FRL when the data is first accessed and by FHERITAGE.

Finherit:
The only recognizable message is CONTINUE. This tells Finherit to return data found further along the AKO chain appended to the data in the current frame.

Type:
Recognized by FNEED as the label for a message which is a type of $IF-NEEDED procedure. FNEED may selectively evaluate these procedures.

Comment Functions. These functions manipulate the comments of a datum object.

(FADD-COMMENT datum comment)
merges the comment specified by label and message into the datum. FADD-COMMENT returns the modified datum.

(FCOMMENT? datum label {message})
tests whether the datum has a comment matching the label and (optional) message. If so, it returns the comment. The comment matches if it includes message among its messages.

9. SAVING FRAMES IN A FILE

Saving the state of a frame in FRL is accomplished with either of the next two functions.

(FDUMP frames file)
outputs in file each frame in the list frames in DEFRAME form, ready to be read back in using the ordinary LISP reader.

(FSAVE frames file)
outputs in file each frame in the list frames in FASSERT form, ready to be read back in using the ordinary LISP reader.
Bibliography


Appendix A — The FRL LISP Environment

A.1 Interrupt Character Definitions.

^E
  Edit a function using LEDIT. It must previously have been read using CLOAD or FLOAD. See LEDIT documentation for further information. Actually, the value of ^EDITOR is the editor used by ^E.
  ^E(function) edits the function.
  ^E(<file names>) edits the file.
  ^E() re-edits the previous object.

^F
  ^F(frame) prints frame.
  ^F(frame slot) prints the slot of frame.
  ^F(frame slot facet ...) prints the structure accessed by the path frame, slot, ...
  ^F() reuses the previous argument.

^P
  ^P(function) prints function.
  ^P(atom indicator) prints the indicator property of atom.
  ^P() reuses the previous argument.

^^ Print a backtrace.

^\ Examine the stack; using (DEBUG).

^• Step through the next evaluation; (STEP t).
A.2 Control Characters in FRL

( o => non-standard LISP definition )

ACTION

~ STEP t
A record the TV screen in a file
B enter breakloop
C GC statistics OFF
D GC statistics ON
E edit a function
F print a frame
G quit to toplevel
H <backspace>
I <tab>
J <linefeed>
K redisplay input buffer; deletes a line during type-in to "_" prompter
L erase screen and redisplay input buffer
M <newline>, behaves like space
N delete word during type-in to "_" prompter
O unused
P print a function
Q enable file input
R enable file output
S disable terminal output until next READ
T unused
U undoes type-in during ">" prompted request, then reprompts.
V enable terminal output
W disable terminal output
X quit to errset
Y unused
Z quit to DDT
[<altmode>
] (DEBUG)
^ Print backtrace
_ unused

A.3 Syntax Table Definitions

The characters •, ~, !, and & are readmacros which read the next S-expression and respectively expand into (ATSIGN s-expression), (PERCENTSIGN s-expression), (EXCLAMATION s-expression), and (AMPERSAND s-expression).

(FRL-READTABLE) selects this readable. (LISP-READTABLE) selects the standard LISP readable.
A.4 Global System Variables.

The following global variables are used by FRL: FASSERT, DEFRAME, FRAME, NEW-FRAMES, :USER, :FRAME, :SLOT, :FACET, :VALUE, :VALUES, PAGEPAUSE, VERSION, FGENSYM, REQUEST-PROMPTER, DEBUG, VERBOSE.

A.5 How big is FRL?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Program Space</td>
<td>26000 words</td>
</tr>
<tr>
<td>Lists</td>
<td>16324</td>
</tr>
<tr>
<td>Fixnum</td>
<td>5323</td>
</tr>
<tr>
<td>Symbol</td>
<td>1662</td>
</tr>
</tbody>
</table>
Appendix B -- Frames are built out of FLISTS

The foundation of FRL consists of a few LISP functions for manipulating flists. An flist is a recursive list structure defined as follows:

\[
\text{flist ::= ( indicator . bucket )} \\
\text{indicator ::= s-expression} \\
\text{bucket ::= ( item item ... ) [ A bucket can be NIL ]} \\
\text{item ::= flist}
\]

In FRL, flists are implemented as nested association lists. An embedded flist can be identified by specifying a path of indicators.

D.1 An Flist has two parts -- an INDICATOR and a BUCKET.

(FINDICATOR flist)

returns the indicator from flist.

(FBUCKET flist)

returns the bucket from flist.

(FINDICATORS flist)

returns a list of the indicators of the items in the bucket of flist.

(FINDICATORSI bucket)

returns a list of the indicators of the items in bucket.

D.2 Retrieving items from an Flist.

(FLISTGET flist ind1 ind2 ... indN)

returns the flist whose indicator matches indN, reached by first selecting the item in the bucket of flist whose indicator matches ind1 and then reapplying FLISTGET to this item (which is an flist itself) with remaining arguments ind2 ... indN. Thus, the indicators define a path leading deeper into the flists nested as items in flist. The analogy with LISP's GET function is not coincidental. NIL is returned if the path leads nowhere, i.e., either the embedding is less than N or no items at that level match indN.

D.3 Storing items in an Flist.
(FLISTPUT flist item ind1 ind2 ... indN)

adds item to the bucket pointed to (as in FLISTGET) by the indicator path ind ... indN. FLISTPUT then returns the modified flist. If the path formed by ind1 ... indN does not exist in flist, one is constructed. The order of FLISTPUT's arguments reflects its similarity to LISP's PUTPROP function, but with extra indicators specifying a complete path. If an item EQUAL to item already exists in the bucket, FLISTPUT does nothing; i.e., addition to an flist is a merging operation. Items in a bucket are always assumed to be unordered.

D.4 Deleting items from an Flist.

(FLISTDELETE flist ind1 ind2 ... indN)

deletes the entire item accessed in flist via the indicator path ind1 ... indN, i.e., it will have had indN as its indicator. FLISTDELETE returns the modified flist.

(FLISTCLEAR flist ind1 ind2 ... indN)

empties the bucket under indN, but leaves the indicator. FLISTCLEAR returns the modified flist.

(FLISTREPLACE flist item ind1 ind2 ... indN)

Item replaces all existing items in the bucket accessed in flist via the indicator path ind1 ... indN. It is equivalent to an FLISTCLEAR followed by an FPUTLIST. FLISTREPLACE returns the modified flist.
### Appendix C -- Index to FRL Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(AKO? f1 f2)</td>
<td>6</td>
</tr>
<tr>
<td>(DEFFRAME name slot1 slot2 ... slotN)</td>
<td>5</td>
</tr>
<tr>
<td>(FADD-COMMENT datum comment)</td>
<td>17</td>
</tr>
<tr>
<td>(FALSE? x)</td>
<td>16</td>
</tr>
<tr>
<td>(FAPPLY-CONSTRAINTS constraints values)</td>
<td>15</td>
</tr>
<tr>
<td>(FASSERT name slot1 slot2 ... slotN)</td>
<td>4</td>
</tr>
<tr>
<td>(FBUCKET (list))</td>
<td>22</td>
</tr>
<tr>
<td>(FCHECK frame slot {values})</td>
<td>16</td>
</tr>
<tr>
<td>(FCHILDREN frame slot)</td>
<td>6</td>
</tr>
<tr>
<td>(FCOMMENT? datum label {message})</td>
<td>17</td>
</tr>
<tr>
<td>(FCOPY (frame))</td>
<td>4</td>
</tr>
<tr>
<td>(FCREATE (name))</td>
<td>5</td>
</tr>
<tr>
<td>(FDADDELEMENT frame slot {facet} datum label message)</td>
<td>8</td>
</tr>
<tr>
<td>(FDESTRESS frame slot)</td>
<td>6</td>
</tr>
<tr>
<td>(FDESTROY (frame))</td>
<td>5</td>
</tr>
<tr>
<td>(FDUMP frames file)</td>
<td>17</td>
</tr>
<tr>
<td>(FERASE (frame))</td>
<td>5</td>
</tr>
<tr>
<td>(FEVAL s-expression frame slot {facet})</td>
<td>9</td>
</tr>
<tr>
<td>(FGENERATE (name))</td>
<td>11</td>
</tr>
<tr>
<td>(FGENERATE frame slot {facet})</td>
<td>12</td>
</tr>
<tr>
<td>(FHERITAGE frame slot {facet} datum label)</td>
<td>7</td>
</tr>
<tr>
<td>(FHERITAGESLOTS (frame))</td>
<td>7</td>
</tr>
<tr>
<td>(FINDICATOR (list))</td>
<td>22</td>
</tr>
<tr>
<td>(FINDICATORS (list))</td>
<td>22</td>
</tr>
<tr>
<td>(FINDICATORS1 (bucket))</td>
<td>22</td>
</tr>
<tr>
<td>(FINHERIT frame slot {facet})</td>
<td>10</td>
</tr>
<tr>
<td>(FINHERIT1 (frame slot {facet})</td>
<td>11</td>
</tr>
<tr>
<td>(FINHERIT2 (frame slot {facet})</td>
<td>11</td>
</tr>
<tr>
<td>(FININSTANTIATE frame {name})</td>
<td>7</td>
</tr>
<tr>
<td>(FLINK? slot f1 f2)</td>
<td>6</td>
</tr>
<tr>
<td>(FLISTCLEAR (list) ind1 ind2 ... indN)</td>
<td>23</td>
</tr>
<tr>
<td>(FLISTDELETE (list) ind1 ind2 ... indN)</td>
<td>23</td>
</tr>
<tr>
<td>(FLISTGET (list) ind1 ind2 ... indN)</td>
<td>22</td>
</tr>
<tr>
<td>(FLISTPUT (list) item ind1 ind2 ... indN)</td>
<td>23</td>
</tr>
<tr>
<td>(FLISTREPLACE (list) item ind1 ind2 ... indN)</td>
<td>23</td>
</tr>
<tr>
<td>(FNAME (frame))</td>
<td>4</td>
</tr>
<tr>
<td>(FNAME? (frame))</td>
<td>4</td>
</tr>
<tr>
<td>(FNEED frame slot {types})</td>
<td>15</td>
</tr>
<tr>
<td>(FPOLL predicates)</td>
<td>16</td>
</tr>
<tr>
<td>(FPOLL-SUMMARY predicates)</td>
<td>16</td>
</tr>
<tr>
<td>(FPUT frame slot {facet} datum label message)</td>
<td>8</td>
</tr>
<tr>
<td>(FPUT-STRUCTURE (frame)</td>
<td>8</td>
</tr>
<tr>
<td>(FPUT-STRUCTURE frame slot-structure)</td>
<td>8</td>
</tr>
</tbody>
</table>
(FPUT-STRUCTURE frame slot facet-structure) 8
(FPUT-STRUCTURE frame slot facet datum-structure) 8
(FPUT-STRUCTURE frame slot facet datum comment) 8
(FRAME frame) 4
(FRAME? frame) 4
(FRAME+ frame) 4
(FRED frame) 29
(FREMOVE frame slot} facet{ datum} label{ message}) 8
(FREPLACE frame slot} facet{ datum} label{ message}) 8
(FRESET) 5
(FRINGE frame slot) 5
(FRUN s-expression frame} slot{ facet}) 14
(FSAVE frames file) 17
(FSLOTS frame) 4
(FTREE frame slot) 6
(GENERIC? frame) 7
(INDIVIDUAL? frame) 7
(TRUE? x) 16
(UNKNOWN? x) 16
### Appendix D -- Table of Data Retrieval Functions

Some instances of data retrieval are common enough to justify a unique name. A popular collection follows in tabular form, grouped according to the type of inheritance used to retrieve the data. The general form for the following functions is:

\[
\langle \text{function}\rangle\ (\text{frame}\ slot\ facet)\ .
\]

<table>
<thead>
<tr>
<th>Retrieval function</th>
<th>count</th>
<th>comment</th>
<th>evaluate</th>
<th>indirect</th>
<th>inherit</th>
<th>(returns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>*fdatum-only</td>
<td>ONE</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NONE</td>
<td>indicator</td>
</tr>
<tr>
<td>*fdata-only</td>
<td>ALL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NONE</td>
<td>list of indicators</td>
</tr>
<tr>
<td>*fdatum</td>
<td>ONE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NONE</td>
<td>bucket</td>
</tr>
<tr>
<td>*fdata</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NONE</td>
<td>list of buckets</td>
</tr>
<tr>
<td>fdatum-only</td>
<td>ONE</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>indicator</td>
</tr>
<tr>
<td>fdata-only</td>
<td>ALL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>list of indicators</td>
</tr>
<tr>
<td>fdatum</td>
<td>ONE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>bucket</td>
</tr>
<tr>
<td>fdata</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>list of buckets</td>
</tr>
<tr>
<td>fheritage</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>HERITAGE</td>
<td>list of buckets</td>
</tr>
</tbody>
</table>

The general form of the following functions is:

\[
\langle \text{function}\rangle\ (\text{frame}\ slot)\ .
\]

<table>
<thead>
<tr>
<th>Retrieval function</th>
<th>count</th>
<th>comment</th>
<th>evaluate</th>
<th>indirect</th>
<th>inherit</th>
<th>(returns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>*fvalue-only</td>
<td>ONE</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>none</td>
<td>indicator</td>
</tr>
<tr>
<td>*fvalues-only</td>
<td>ALL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>none</td>
<td>list of indicators</td>
</tr>
<tr>
<td>*fvalue</td>
<td>ONE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>none</td>
<td>bucket</td>
</tr>
<tr>
<td>*fvalues</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>none</td>
<td>list of buckets</td>
</tr>
<tr>
<td>fvalue-only</td>
<td>ONE</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>indicator</td>
</tr>
<tr>
<td>fvalues-only</td>
<td>ALL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>list of indicators</td>
</tr>
<tr>
<td>fvalue</td>
<td>ONE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>bucket</td>
</tr>
<tr>
<td>fvalues</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>list of buckets</td>
</tr>
<tr>
<td>fvalue-only1</td>
<td>ONE</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>1</td>
<td>indicator</td>
</tr>
<tr>
<td>fvalues-only1</td>
<td>ALL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>1</td>
<td>list of indicators</td>
</tr>
<tr>
<td>fvalue1</td>
<td>ONE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>1</td>
<td>bucket</td>
</tr>
<tr>
<td>fvalues1</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>1</td>
<td>list of buckets</td>
</tr>
<tr>
<td>fvalue-only2</td>
<td>ONE</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>2</td>
<td>indicator</td>
</tr>
<tr>
<td>fvalues-only2</td>
<td>ALL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>2</td>
<td>list of indicators</td>
</tr>
<tr>
<td>fvalue2</td>
<td>ONE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>2</td>
<td>bucket</td>
</tr>
<tr>
<td>fvalues2</td>
<td>ALL</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>2</td>
<td>list of buckets</td>
</tr>
</tbody>
</table>

Some general naming rules have been observed. "-ONLY" signifies that the comments have been stripped off. "*" denotes no inheritance. Singular and plural forms distinguish functions which return a list of all data items from those that expect to find a
single datum.

The following predicates return T only if data exists to be retrieved; i.e., if the corresponding retrieval function (see Table) would return non-nil. The predicate forms, however, do not return useable data.

(*FDATUM? frame slot facet)
(FDATUM? frame slot facet)
(FHERITAGE? frame slot facet)
(*FVALUE? frame slot)
(FVALUE? frame slot)
(FVALUE1? frame slot)
(FVALUE2? frame slot)
Appendix E -- The FRL Trace Function

**FTRACE** is FRL's tracer for frame actions. It's syntax parallels LISP's TRACE except that a predefined set of actions are traced rather than functions and a more limited set of options are available. Traceable actions are IF-ADDED, IF-REMOVED, IF-NEEDED, CREATE, DESTROY, and INSTANTIATE. Options are COND, BREAK, ENTRY and EXIT. For example,

(FTRACE IF-ADDED)

causes trace information to be printed out before and after any IF-ADDED method is executed.

Additional information can be specified using the ENTRY and EXIT options. The COND option controls whether anything at all is printed; BREAK breaks. For example,

(FTRACE (IF-ADDED COND (NOT (MEMO :SLOT '(AKO INSTANCE)))
    BREAK (EQ :SLOT 'FOO)
    ENTRY ( (INDIVIDUAL? :FRAME) ) ))

prints the usual stuff about if-added methods run on any slots other than AKO and INSTANCE, breaks if an if-added method is run for the FOO slot of a frame, and prints whether or not the frame is an Individual along with the entry information.

(FTRACE) returns a list of actions currently being traced.

(FUNTRACE) stops tracing entirely.

(FUNTRACE action1 action2 ...) stops tracing selectively.
Appendix F -- The FRL Frame Editor

Frames can be modified in place using FRL’s Frame Editor, FRED. Changes to values in a frame have the same side-effects as FASSERT and FERASE.

(FRED frame)

displays frame on the top of the screen while recognizing a few editing commands at the bottom. (FRED) edits the previously edited frame. A cursor (.), when present, appears just before the sub-structure being edited.

W displays the whole frame. Ordinarily, the frame is compacted to fit the screen.
J moves the cursor to the beginning of the frame, its name.
F moves the cursor to the first item in the structure.
L moves the cursor to the last item.
N moves the cursor to the next item.
P moves the cursor to the previous item.
U moves up within structures.
D moves down.
G reads a name and goes to that slot.
S prints all the slots in the frame.
H prints the heritage of the current level of the frame.
K kills the next item after the cursor. It is bound to the atom "".
I inserts a new sub-structure. Prefix your response with "%%" to return the value of the expression.
. reads a name and binds the current value of "" to it.
V prints the values of the current slot.
C runs a check on the requirements in the current slot.
E reads and evaluates a single form.
Q quits.
? prints FRED’s documentation.