Internal Use Only

TR-I-162

The FS-LF Manual

Version 1.2

FS-LF システム マニュアル

バージョン 1.2

John K. Myers

真龍主·星音

April 27, 1990

Abstract

This manual presents user documentation for the ATR Interpreting Telephony Research Laboratories FS-LF, the Feature Structure to Logical Form Translator. This package translates information represented in feature structure format into a logical form representation. Feature structures are translated directly, in a one-to-one format.

The FS-LF translator was developed to allow natural-language understanding programs that work with logical forms to be able to take input from the results of programs that work with feature structures.

The FS-LF translator version 1.2 is a special-purpose system that only works with Nadine-style feature structures, as defined by the Hasegawa RWS rewriting system. This system only translates information that is actually present, from one representation scheme to another; different efforts are required to add information that is not explicitly present in the feature structure (by doing anaphora resolution, etc.). The translation is performed using automatically constructed functions. The FS-LF system is thus similar to a compiler-compiler. This allows the translation to be extremely rapid, which permits use of the package in a run-time system.

© ATR Interpreting Telephony Research Laboratories
© ATR 自動翻訳電話研究所

THE FS-LF MANUAL

Version 1.2
Dec 27, 1989

John K. Myers

ATR Interpreting Telephony Research Laboratories
Sanpeidani, Inuidani, Seika-cho, Soraku-gun
Kyoto 619-02 Japan
Netmail: myers@atr-la.atr.co.jp

Abstract

This manual presents user documentation for the ATR Interpreting Telephony Research Laboratories FS-LF, the Feature Structure to Logical Form Translator. This package translates information represented in feature structure format into a logical form representation. Feature structures are translated directly, in a one-to-one format.

The FS-LF translator was developed to allow natural language understanding programs that work with logical forms to be able to take input from the results of programs that work with feature structures.

The FS-LF translator version 1.2 is a special-purpose system that only works with Nadine-style feature structures, as defined by the Hasegawa RWS rewriting system. This system only translates information that is actually present, from one representation scheme to another; different efforts are required to add information that is not explicitly there in the feature structure (by doing anaphora resolution, etc.). The translation is performed using automatically constructed functions. The FS-LF system is thus similar to a compiler-compiler. This allows the translation to be extremely rapid, which permits use of the package in a run-time system.

Copyright (c) 1989 ATR Interpreting Telephony Research Laboratories.

Contents

1 Introduction and Background

The aim of this work is to provide a translation facility that can translate, in a literal fashion, information that is contained in a feature structure format into the same information contained in a logical form format.

This is important because at ATR there is a parsing program that works with information in feature structure format, while there is also an understanding system that works with information in logical forms. Previously, the understanding system has been forced to use hand-generated logical forms as input, that approximate the output information from the parser. In order to allow the understanding system to take actual input from the results of the parser, it is important to translate the information into a format that the understanding system can work with.

It must be noted that the system integration of such an understanding system with such a parser actually comprises two issues. The first, and perhaps more important problem, is the *interpretation* of the *information* contained in the results of the parser into *information* that the understanding system can use. This may involve, for example, ellipsis and anaphora resolution. This problem is non-trivial. The second problem is the literal *translation* of the *format* of the results of the parser into a *format* that the understanding system can use. Information is simply rearranged; it is not reprocessed or used to create new information. The FS-LF system has been designed to perform this second task (format translation). The problem of reworking the information to fill in missing pieces must be handled by a more sophisticated system, for instance perhaps a rewriting system or a plan inference system.

2 Usage

The FS-LF system can be used either automatically or by people. It is designed to translate structures as fast as possible, so it can be used as part of an on-line system. There are basically only two main functions. First of all, the user specifies a number of rewriting rules, using `FS-to-LF-spec`. These tell the system how to rewrite a particular type of feature structure into a logical form. Then, the user tells the system to translate a feature structure, using `FS-to-LF`. The translated logical form is returned. This process will now be discussed in more detail.

The system assumes that there are different types of feature structures, and that each type should be translated in a different manner.

First, before sending a particular feature structure to the FS-LF system, a recognition program should rewrite the structure into a unique feature structure format that is (1) easily recognizable (i.e., has an identifiable type), (2) has all required information, and (3) does not vary topologically (the structure "shape" is always the same for that type of feature structure).

This feature structure is passed to the FS-LF system. The system identifies the type of the feature structure, and looks up a translation method which corresponds to the FS identification type. The proper translation method is applied to the feature structure, resulting in a logical form. This should then be passed to a logical form understanding program. Thus, a logical form understanding program can be part of a run-time system

by using the FS-LF package to rapidly translate feature structures.

Feature structure types in version 1.2 are currently identified by the highest, most important slot value in the structure. Usually, this is the first "reln" value.

The translation method for each type of feature structure that the system can encounter should be specified ahead of time. Translation is basically a rewriting process from a feature structure into a logical form (instead of the more common feature structure-into-feature structure rewriting). A translation method is thus defined by specifying a rewriting rule consisting of a feature structure pattern and a resulting logical form pattern template to be filled in. The patterns can have constants and variables. In version 1.2, only one translation method is allowed for each type of feature structure.

3 How it works

A translation specification rewriting rule consists of an input feature structure pattern, and an output logical form pattern. When a specification is entered into the system, the system builds a special translation function for that type of feature structure. The name of the function is "`Trans-`*identification*`-to-LF`". (The feature structure type's identification is taken from the feature structure pattern, using `FS-LF-get-name-tag`; the identification is typically the first-level reln feature's slot value for the specified pattern.) The translation function, when it is called with a feature structure of the appropriate type, will return a list that is the desired logical form, with all of the logical form pattern's variables filled in.

The value for each variable in the instantiated logical form is obtained by calling one of a set of special mapping functions, named "`Map-`*identification-varname*". These functions are also compiled automatically by the system. They access a particular branch of a feature structure, and return its contents. The system creates these functions in the following manner. First, a variable in the logical form pattern is discovered. Variables in version 1.2 are marked by beginning with question marks (e.g., "?var"). Next, the corresponding location of the same variable in the feature structure pattern is discovered and recorded. The access path to this location is compiled into the corresponding `Map` function.

When the FS-to-LF system is asked to translate a feature structure, it first identifies the type of the feature structure. Next, it builds the name of the translation function from the identified type using string concatenation, and invokes that function. (Note that this indexing process requires constant time that should not change when a large base of translation rules is used. In other words, it is O(1), not O(log n).) The translation function is applied to the feature structure. The translation function calls each of its mapping functions on the feature structure, and assembles the results into the logical form, which is returned.

4 Data and Command Explanation

This section presents a description of the system's data and commands. First, the types of data used by the system are described. Next, the most commonly used commands are detailed. The commands are arranged in the order in which they are typically used. After this, other support commands that can be used are listed, and important system variables are also described.

4.1 System Data Types

There are five explicit kinds of data in the FS-LF system. These are:

Explicit Feature Structure An explicit feature structure is a written representation of a Hasegawa-style Nadine feature structure, used mostly by people as an input specification. An example is `[[reln greet][speaker GUEST]]`. It will be represented in this manual by the symbol "[FS]". An explicit feature structure is actually a series of unbound atoms in the input stream; the Lisp reader temporarily gets modified[2] to be able to accept these atoms as input.

Internal Feature Structure An internal feature structure is a variable containing a Lisp structure of type RWS::NODE, used mostly by programs. An example is the value `#S(RWS::NODE :VALUE ((RELN ...))...)`. It will be represented in this manual by the symbol "{FS}". Internal feature structures are usually verbose and inconvenient to read, and are therefore typically converted into explicit feature structure format by a pretty-printing program before being displayed to the user.

Unspecified Feature Structure An unspecified feature structure can be either an explicit [FS] or an internal {FS} feature structure. It will be represented by the symbol "FS". Unspecified feature structures are used only for input.

Logical Form A logical form is a variable (or a quoted form) containing a list of atoms or lists. An example is the expression `'((reln greet)(speaker guest)))` (note the beginning quote). It will be represented by the symbol "LF".

Logical Form Literal A logical form literal is an unquoted logical form expression (included in the system for convenience in input). An example is the literal expression `((reln greet)(speaker guest)))`. It will be represented by the symbol "LFq".

[2]This function was written by Hasegawa-san.

4.2 Commonly Used Commands

4.2.1 Initialization Commands

(reset-FS-LF) Clears out the documentation hash-table. Allows new translation specifications to be defined. Note that in version 1.2, this function does *not* unintern the old translation functions. Thus, the old translation functions will still work, until they have been replaced by new ones. This feature may change in the future.

(FS-to-LF-spec FS LFq) Enters a specification for translating between a feature structure and a logical form into the system. The feature structure can be in explicit or internal format. The logical form should be a logical form literal (without a quote). The feature structure should have a unique identification. (The identification of a feature structure is its highest-level "reln" value, or other comparable tag.) All of the variables in the logical form should be found somewhere in the feature structure. However, the feature structure can have extra variables that are not included in the logical form. Variables start with question marks.

Because this function creates and then invokes the compiler on a series of mapping access functions, it is especially slow (about 4 seconds per call, depending on the size of the logical form pattern). It is expected that this function will be used off-line, before run-time of the actual translation task.

The feature structure pattern in version 1.2 currently only serves to determine the branch-path location of the logical form's variables. Thus, the recognition performed is very weak; the feature structure is allowed to have more structure than is shown in the pattern. In other words, matching does not have to be exact.[3]

Example:

```
(FS-to-LF-spec [[reln my-reln][value ?var]] (output form (?var)) )
```

4.2.2 Translation Commands

(FS-to-LF FS) Translates an FS into an LF. Returns the LF. The feature structure can be in explicit or internal format. The translation is performed by keying on the identification ("reln" value, etc.) of the feature structure. The translation method for this type of feature structure must have been previously specified by a FS-to-LF-spec command. Since this function tests to see whether the FS is in explicit or internal format, it should be used by people.

Example: `(FS-to-LF [[reln my-reln][value my-input]] )`

This will result in the form `(output form (my-input))`, given the previous specification.

[3]It can even have less entries, as long as all the variables' information is there.

4.3 Support Commands

4.3.1 Translation Commands

([FS]-to-LF FS) Another name for FS-to-LF. The FS may be in explicit format. Translates an FS into an LF. Returns the LF.

(Fast-FS-to-LF {FS}) Translates an FS into an LF. Returns the LF. The feature structure must be in internal format. Since this function does not test to see whether the FS needs to be converted into internal format, it should be used by other computer systems.

Example: `(Fast-FS-to-LF my-fs )`

({FS}-to-LF {FS}) Another name for Fast-FS-to-LF. Translates an FS into an LF. Returns the LF. The feature structure must be in internal format.

Example: `(FS-to-LF my-fs )`

4.3.2 Documentation and Debugging Commands

(get-trans-func name-or-FS) This function prints a report on the specified translation function. This is needed because the translation function is created and compiled internally, and no source code is otherwise available. The translation function is specified by the name (identifier) of the function, or equivalently by providing an example of the type of feature structure that the function translates (in explicit or internal format). The report in version 1.2 currently consists of a listing of the specification used to create the translation function, and a source listing of the defun used by the system to define the function.

Example: `(get-trans-func my-reln)` .

(get-mapping-func name-or-{FS} varname) This function prints a report on the specified mapping function. This is needed because the mapping function is created and compiled internally, and no source code is otherwise available. The mapping function is specified by the name (identifier) of the feature structure type that it performs a mapping for, or equivalently by providing an example of the type of feature structure that the function translates (in internal format). The variable is specified by name, without a quote. The report in version 1.2 consists of a source listing of the defun used by the system to define the function.

Example: `(get-mapping-func my-reln ?var)` .

(print-FS-LF-funcs) This function prints a table of the names of all the known translation functions. It is useful for seeing whether a function has been defined or not.

(print-FS-LF-mappings) This function prints a table of the names of all the known translation functions. It is useful for seeing whether a function has been defined or not.

4.3.3 System Generated Translation Functions

(TRANS—*ident*-TO-LF {FS}) This internal function builds a logical form that corresponds to the feature structure identified by ident. It implements the specified rewriting method for that type of feature structure. It gets created and compiled by the FS-LF system. The listing for this function can be found by using `(get-trans-func ident)`.

The name for this function is determined by the FS-LF internal function `(make-translation-function-name 'ident)`.

(MAP—*ident-varname* {FS}) Each instance in this class of internal functions accesses a particular branch in a feature structure, and returns the contents of that branch. These functions are called by `TRANS-ident-TO-LF`. The type of feature structure that they should be applied to is identified by ident. These functions get created and compiled by the FS-LF system. The listing for one of these functions can be found by using `(get-mapping-func ident varname)`.

(where-is 'TRANS-ident-TO-LF) This Lisp function checks to see whether a given function has been interned (is present in the system) or not.

4.4 Significant Variables

FS-LF-specs-hash This hash-table holds the documentation for the translation functions. In version 1.2, the hash-key is the name of the function.

FS-LF-mappings-hash This hash-table holds the documentation for the mapping functions. In version 1.2, the hash-key is the name of the function.

5 Design philosophy

Because this system may be used as the logical form translator in an on-line system, it is important to make the translation itself be as fast as possible. (Since the translation specifications will be performed offline, it is alright for them to be not so fast.)

To accomplish this aim, a function-compiler was produced. The translation itself is performed by a special-purpose function that is dedicated to translating only a single specified kind of feature structure. Because it is a dedicated function, it is extremely fast. This function is built and compiled by the FS-LF system. Which special-purpose function to call is handled by a master switching function in `FS-to-LF`. In order to fill in the variables in the logical form, the translation function calls other special-purpose functions that access specific branch paths in the feature structure. These functions are also built and compiled by the system. The result is an extremely fast access-function-based system that basically uses no conditionals nor pattern-matching.

There is a small trade-off in space to achieve this speed. Because of this design, the system must create a function for every kind of feature structure it translates, and for every kind of variable in every kind of feature structure it accesses. However, since these routines are only a few lines long apiece, this should not be a significant penalty in memory space.

6 Features and Known Problems

The current version of the system, version 1.2, has some particular features. These include:

- **Returns {FS} value if FS is unexpectedly deep.** The translation routines in version 1.2 work by accessing and returning the value of a slot in a FS. If this slot unexpectedly contains a (deeper) {FS} instead of an atomic value, this {FS} is returned in the appropriate place in the middle of the LF. This feature is expected to be corrected in the next version.

- **FSs have only one method of translation.** Each type of FS has only one handler function that translates it into an LF. This serves to make translation simple and easy to understand. It is assumed that if one kind of FS can be translated into one of two different kinds of LFs, it will be classified by a preprocessor (such as a nonmonotonic rewriting system or a plan recognition system) which will reassign it an appropriate unique identification for translation. This design feature may be changed in the future if it turns out to be inconvenient.

- **The highest-level indicator in a feature structure pattern specification must be a constant.** (This is usually the value of the first `reln`.) This again is a result of identifying feature structures by their top level.

- **The significant contents of a FS must be known ahead of time.** One of the advantages of feature structures is that they can be arbitrarily expanded at run-time, by adding new (and perhaps previously unknown) feature slots and data. Version 1.2 of the FS-LF system allows unspecified expansion but does not take advantage of it: any extra data is simply not translated. The reasoning behind this is that

logical forms encode information based on the position of an item in the form; thus, the interpretation of each position in the form must be known and specified ahead of time in order for the logical form to be meaningful. Since logical forms can only represent known information classes, any unknown information should probably be discarded.

- **Reset does not unintern translation functions.** The old translation functions are still present after a reset. This feature may get corrected if there is sufficient demand.

- **Currently in version 1.2 no dotted lists are allowed in the output logical form.**

- **The FS-LF system is specialized for Hasegawa-style Nadine feature structures.** Other styles of feature structures, e.g. RETIF, are not handled in version 1.2.

7 Timing

An experiment was performed to test the speed of the FS-LF system, version 1.2. The feature structures representing the surface speech acts of utterances from conversation 1 were processed. The system processed 20 surface speech act feature structures, using 15 FS-to-LF-spec rules. During five runs, the system took an average of 0.07 seconds to process all of the utterances, for an average of about 0.0035 seconds per feature structure translation. The logical forms were all three levels of parenthises deep, and used an average of 7 variables apiece. (Translation time is basically a linear function of the number of variables in the resulting logical form.) The experiment was performed on a 3620 Symbolics Lisp Machine.

The 20 input feature structure utterances and their resulting output logical forms are presented in Appendix A. The 15 FS-to-LF-spec rewriting rules used to specify the translations are presented in Appendix B.

8 Future Extensions

Some of the system's features should be modified. One example is the method of translating non-atomic variable values, i.e. when a desired feature slot does not contain an atom but contains a deeper feature structure. Some acceptable convention will have to be determined for translating arbitrary feature structures into useful logical forms, for instance a paired list or a dotted-pair assoc list.

Other extensions will depend upon what is needed and/or useful. Perhaps one extension that will eventually be required is a logical form to feature structure (LF-FS) inverse translator.

A third extension might be to concentrate on making the system run faster. For instance, if the legal function check is removed, the system will probably run in less than half of the current time required. The indexing process can also be improved.

9 Conclusion

This manual has presented the theory and practice of the use of "FS-LF", the ATR Feature Structure to Logical Form Translator. With the functions and examples described here, the user can specify translation rules, and can translate feature structures into logical forms. Translation is performed by directly rewriting information extracted from the feature structure into logical-form format. This is done in a rapid manner using dedicated functions. The resulting system can be used at run-time as a translation system to enable a logical-form-based understanding program to use the output of a feature-structure-based semantic parser.

A Examples

```
**   FS-LF DEMO   **
**Translating utterances from number 0 to 19.**


(FS-to-LF

 [[RELN GREETING-OPEN]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP1]
           [PREDICATE MOSHIMOSHI]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(GREETING-OPEN SP1 SP2 ?CLUE$1 ?TOPIC$1 (PRED&CASES (PREDICATE MOSHIMOSHI) (AGENT SP1)))



(FS-to-LF

 [[RELN CONFIRM-VALUE-A]
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER KOKUSAIKAIGI_JIMUKYOKU]
           [OBJECT SP2]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(CONFIRM-VALUE-A SP1 SP2 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE DESU) (OBJECT SP2) (IDENTIFIER KOKUSAIKAIGI_JIMUKYOKU)))



(FS-to-LF

 [[RELN AFFIRMATIVE]
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER ?ID$1]
           [OBJECT SP2]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:
```

```
(AFFIRMATIVE SP2 SP1 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE DESU) (OBJECT SP2) (IDENTIFIER ?ID$1)))


(FS-to-LF

 [[RELN ASK-TOPIC]
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER ?ID$1]
           [OBJECT ?OBJECT$1]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(ASK-TOPIC SP2 SP1 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE DESU) (OBJECT ?OBJECT$1) (IDENTIFIER ?ID$1)))


(FS-to-LF

 [[RELN INTRODUCE-OBJECT]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP2]
           [OBJECT KAIGI-MOUSHIKOMI]
           [PREDICATE OSHIERU]
           [RECIPIENT SP1]]]
  [CLUE ?CLUE]
  [HR ?HEARER]
  [SP ?SPEAKER]
  [TOPIC KAIGI-MOUSHIKOMI]]
  )
    ...gives:

(INTRODUCE-OBJECT ?SPEAKER ?HEARER ?CLUE KAIGI-MOUSHIKOMI
 (PRED&CASES (PREDICATE OSHIERU) (AGENT SP2) (RECIPIENT SP1) (OBJECT KAIGI-MOUSHIKOMI)))


(FS-to-LF

 [[RELN ASK-ACTION]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP1]
           [OBJECT ?OBJ$1]
           [PREDICATE ?PRED$1]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
```

```
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(ASK-ACTION SP1 SP2 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE ?PRED$1) (AGENT SP1) (OBJECT ?OBJ$1)))



(FS-to-LF

 [[RELN DIRECTION]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP1]
           [MEAN FORM]
           [OBJECT ?OBJ$1]
           [PREDICATE TETSUDUKIWOSURU]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(DIRECTION SP2 SP1 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE TETSUDUKIWOSURU) (AGENT SP1) (OBJECT ?OBJ$1) (MEAN FORM)))



(FS-to-LF

 [[RELN ASK-TRUTH-N]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP1]
           [OBJECT FORM]
           [PREDICATE MOTSU]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(ASK-TRUTH-N SP2 SP1 ?CLUE$1 ?TOPIC$1 (PRED&CASES (PREDICATE MOTSU) (AGENT SP1) (OBJECT FORM)))



(FS-to-LF

 [[RELN NEGATIVE-TRUTH]
  [ACTION [[RELN PRED&CASES]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
```

```
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(NEGATIVE-TRUTH SP1 SP2 ?CLUE$1 ?TOPIC$1 (PRED&CASES (PREDICATE DESU)))


(FS-to-LF

 [[RELN CONFIRM]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP2]
           [MANNER ?MANNER$1-10]
           [OBJECT ?OBJ$1-10]
           [PREDICATE WAKARU]
           [RECIPIENT ?RECIP1-10]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(CONFIRM SP2 SP1 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE WAKARU) (AGENT SP2) (RECIPIENT ?RECIP1-10) (OBJECT ?OBJ$1-10)
  (MANNER ?MANNER$1-10)))


(FS-to-LF

 [[RELN WILL-DO-ACTION-A]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP2]
           [MANNER ?MANNER$1-11]
           [OBJECT FORM]
           [PREDICATE OKURU]
           [RECIPIENT SP1]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(WILL-DO-ACTION-A SP2 SP1 ?CLUE$1 ?TOPIC$1
 (PRED&CASES (PREDICATE OKURU) (AGENT SP2) (RECIPIENT SP1) (OBJECT FORM)
  (MANNER ?MANNER$1-11)))


(FS-to-LF

 [[RELN ASK-VALUE]
```

```
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER ?ID$]
           [OBJECT ADDRESS]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$]
  [HR SP1]
  [SP SP2]
  [TOPIC ADDRESS]]
  )
    ...gives:

(ASK-VALUE SP2 SP1 ?CLUE$ ADDRESS
 (PRED&CASES (PREDICATE DESU) (OBJECT ADDRESS) (IDENTIFIER ?ID$)))



(FS-to-LF

 [[RELN ASK-VALUE]
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER $ID]
           [OBJECT NAME]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$]
  [HR SP1]
  [SP SP2]
  [TOPIC NAME]]
  )
    ...gives:

(ASK-VALUE SP2 SP1 ?CLUE$ NAME (PRED&CASES (PREDICATE DESU) (OBJECT NAME) (IDENTIFIER $ID)))



(FS-to-LF

 [[RELN INFORM-VALUE]
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER OOSAKASHI_KITAKU_CHAYAMACHI_23]
           [OBJECT ADDRESS]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
  [TOPIC ADDRESS]]
  )
    ...gives:

(INFORM-VALUE SP1 SP2 ?CLUE$1 ADDRESS
 (PRED&CASES (PREDICATE DESU) (OBJECT ADDRESS) (IDENTIFIER OOSAKASHI_KITAKU_CHAYAMACHI_23)))



(FS-to-LF
```

```
 [[RELN INFORM-VALUE]
  [ACTION [[RELN PRED&CASES]
           [IDENTIFIER SUZUKI_MAYUMI]
           [OBJECT NAME]
           [PREDICATE DESU]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
  [TOPIC NAME]]
  )
    ...gives:

(INFORM-VALUE SP1 SP2 ?CLUE$1 NAME
 (PRED&CASES (PREDICATE DESU) (OBJECT NAME) (IDENTIFIER SUZUKI_MAYUMI)))



(FS-to-LF

 [[RELN CONFIRM]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP2]
           [MANNER ?MANNER$1]
           [OBJECT ?OBJECT$1]
           [PREDICATE WAKARU]
           [RECIPIENT ?RECIP$1]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC NAME]]
  )
    ...gives:

(CONFIRM SP2 SP1 ?CLUE$1 NAME
 (PRED&CASES (PREDICATE WAKARU) (AGENT SP2) (RECIPIENT ?RECIP$1) (OBJECT ?OBJECT$1)
  (MANNER ?MANNER$1)))



(FS-to-LF

 [[RELN CONFIRM]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP2]
           [MANNER ?MANNER$1]
           [OBJECT ?OBJECT$1]
           [PREDICATE WAKARU]
           [RECIPIENT ?RECIP$1]]]
  [CLUE ?CLUE$1]
  [HR SP1]
  [SP SP2]
  [TOPIC ADDRESS]]
  )
    ...gives:
```

```
(CONFIRM SP2 SP1 ?CLUE$1 ADDRESS
 (PRED&CASES (PREDICATE WAKARU) (AGENT SP2) (RECIPIENT ?RECIP$1) (OBJECT ?OBJECT$1)
  (MANNER ?MANNER$1)))



(FS-to-LF

 [[RELN WILL-DO-ACTION-A]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP2]
           [MANNER ?MANNER$1-16]
           [OBJECT FORM]
           [PREDICATE OKURU]
           [RECIPIENT SP1]]]
  [CLUE ?CLUE$]
  [HR SP1]
  [SP SP2]
  [TOPIC ?TOPIC$]]
  )
    ...gives:

(WILL-DO-ACTION-A SP2 SP1 ?CLUE$ ?TOPIC$
 (PRED&CASES (PREDICATE OKURU) (AGENT SP2) (RECIPIENT SP1) (OBJECT FORM) (MANNER ?MANNER$1-16))
 )



(FS-to-LF

 [[RELN ACCEPT-OFFER]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP1]
           [PREDICATE NEGAU]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
  [TOPIC ?TOPIC$1]]
  )
    ...gives:

(ACCEPT-OFFER SP1 SP2 ?CLUE$1 ?TOPIC$1 (PRED&CASES (PREDICATE NEGAU) (AGNT SP1)))



(FS-to-LF

 [[RELN GREETING-CLOSE-UNIT]
  [ACTION [[RELN PRED&CASES]
           [AGENT SP1]
           [PREDICATE SHITSUREISURU]]]
  [CLUE ?CLUE$1]
  [HR SP2]
  [SP SP1]
  [TOPIC ?TOPIC$1]]
```

```
  )
    ...gives:

(GREETING-CLOSE-UNIT SP1 SP2 ?CLUE$1 ?TOPIC$1 (PRED&CASES (PREDICATE SHITSUREISURU) (AGNT SP1))
 )



Time taken to translate 20 utterances using Fast-FS-TO-LF, version 1.2:
Evaluation of (LOOP FOR UTTERANCE-NUM FROM 0 TO ...) took 0.064971 seconds of elapsed time inclu
  0.002 seconds processing sequence breaks,
  0.003 seconds in the storage system (including 0.000 seconds waiting for pages):
    0.000 seconds processing 11 page faults including 0 fetches,
    0.003 seconds in creating and destroying pages, and
    0.000 seconds in miscellaneous storage system tasks.
341 list, 230 structure words consed in WORKING-STORAGE-AREA.


<end of demo>
```

B Specifications Used in the Examples

```
;;;  -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-

;;; NAME: FS-LF-SPECS-EX1
;;; VERSION: 1.0
;;; WHAT IT DOES: Provides an example of the FS-LF translation specifications.
;;;
;;; HISTORY
;;; -------
;;; Dec 21 '89 John Myers


(reset-FS-LF)

;---------------------------------------------------------------------

(fs-to-lf-spec

[[reln greeting-open]
 [sp   ?s1]
 [hr   ?s2]
 [clue   ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?s1]]]]


 (greeting-open ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?s1)))
)
;---------------------------------------------------------------------
(fs-to-lf-spec

[[reln confirm-value-a]
 [sp   ?s1]
 [hr    ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate  ?pred]
 [object      ?obj]
 [identifier   ?ID]]]]

 (confirm-value-a ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (object ?obj) (identifier ?ID)))
)
;---------------------------------------------------------------------
(fs-to-lf-spec
  [[reln affirmative]
 [sp    ?s1]
 [hr   ?s2]
 [clue       ?clue]
```

```
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [object       ?obj]
 [identifier    ?ID]]]]

 (affirmative ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (object ?obj) (identifier ?ID)))

)

;-------------------------------------------------------------
(fs-to-lf-spec

[[reln ask-topic]
 [sp    ?s1]
 [hr   ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [object       ?obj]
 [identifier    ?ID]]]]

 (ask-topic ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (object ?obj) (identifier ?ID)))

)
;-------------------------------------------------------------------------------
(fs-to-lf-spec

[[reln introduce-object]
 [sp   ?s1]
 [hr    ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]
 [recipient   ?recp]
 [object       ?obj]]]]

 (introduce-object ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?agnt)(recipient ?recp)(object ?obj)))

)
;-------------------------------------------------------------------------------
(fs-to-lf-spec

[[reln ask-action]
 [sp   ?s1]
 [hr    ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
```

```
 [predicate   ?pred]
 [agent       ?agnt]
 [object      ?obj]]]]

 (ask-action ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?agnt)(object ?obj)))

)
;-------------------------------------------------------------------------------
(fs-to-lf-spec

[[reln direction]
 [sp    ?s1]
 [hr   ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]
 [object      ?obj]
 [mean        ?meaning]]]]

 (direction ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?agnt)(object ?obj)(mean ?meaning)))

)
;-------------------------------------------------------------------------
(fs-to-lf-spec

  [[reln ask-truth-n] ;NOTE: ASK-TRUTH in dialog; none in plan-set.
 [sp   ?s1]
 [hr   ?s2]
 [clue    ?clue]
 [topic   ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]
 [object     ?obj]]]]

 (ask-truth-n ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?agnt)(object ?obj)))

)
;-------------------------------------------------------------------------
(fs-to-lf-spec
[[reln negative-truth]
 [sp    ?s1]
 [hr   ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]]]]

 (negative-truth ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred)))
```

```
)
;-------------------------------------------------------------------------
(fs-to-lf-spec

[[reln confirm]
 [sp    ?s1]
 [hr    ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]
 [recipient  ?recp]
 [object  ?obj]
 [manner     ?manner]]]]

 (confirm ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?agnt)(recipient ?recp)(object ?obj)(manner ?manner)))

)

;-------------------------------------------------------------------------
(fs-to-lf-spec

[[reln will-do-action-a]
 [sp   ?s1]
 [hr ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]
 [recipient   ?recp]
 [object  ?obj]
 [manner     ?manner]]]]

 (will-do-action-a ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred) (agent ?agnt)(recipient ?recp)(object ?obj)(manner ?manner)))

)
;-------------------------------------------------------------------------
(fs-to-lf-spec

[[reln ask-value]
 [sp   ?s1]
 [hr    ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [object       ?obj]
 [identifier    ?ID]]]]

 (ask-value ?s1 ?s2 ?clue ?topic
```

```
     (pred&cases (predicate ?pred)(object ?obj)(identifier ?ID)))

)

;-------------------------------------------------------------------
(fs-to-lf-spec

[[reln inform-value]
 [sp        ?s1]
 [hr        ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [object       ?obj]
 [identifier    ?ID]]]]

 (inform-value ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred)(object ?obj)(identifier ?ID)))

)
;-------------------------------------------------------------------
(fs-to-lf-spec

[[reln accept-offer]
 [sp    ?s1]
 [hr   ?s2]
 [clue       ?clue]
 [topic     ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]]]]

 (accept-offer ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred)(agnt ?agnt)))
)
;-------------------------------------------------------------------
(fs-to-lf-spec
   [[reln GREETING-CLOSE-UNIT]
 [sp   ?s1]
 [hr   ?s2]
 [clue  ?clue]
 [topic ?topic]
 [action [[reln   pred&cases]
 [predicate   ?pred]
 [agent       ?agnt]]]]

 (greeting-close-unit ?s1 ?s2 ?clue ?topic
     (pred&cases (predicate ?pred)(agnt ?agnt)))

)
```

C List of Commands and Variables

(reset-FS-LF)

(FS-to-LF-spec FS LFq)

(FS-to-LF FS)

([FS]-to-LF FS)

(Fast-FS-to-LF {FS})

({FS}-to-LF {FS})

(get-trans-func name-or-FS)

(get-mapping-func name-or-{FS} varname)

(print-FS-LF-funcs)

(print-FS-LF-mappings)

(TRANS—*ident*-TO-LF {FS})

(MAP—*ident-varname* {FS})

FS-LF-specs-hash

FS-LF-mappings-hash