Artificial Intelligence Review **14:** 333–350, 2000.
Stephen J. Hegner et al. (eds.), Intelligent help systems for UNIX.

Strategies for Expressing Concise, Helpful Answers

DAVID N. CHIN
Department of Information and Computer Sciences, University of Hawaii, 1680 East West Rd., Honolulu, HI 96822, U.S.A. E-mail: chin@hawaii.edu

Abstract. An intelligent help system needs to take into account the user's knowledge when formulating answers. This allows the system to provide more concise answers, because it can avoid telling users things that they already know. Since these concise answers concentrate exclusively on pertinent new information, they are also easier to understand. Information about the user's knowledge also allows the system to take advantage of the user's prior knowledge in formulating explanations. The system can provide better answers by referring to the user's prior knowledge in the explanation (e.g., through use of similes). This process of refining answers is called *answer expression*. The process of answer expression has been implemented in the UCExpress component of UC (UNIX Consultant), a natural language system that helps the user solve problems in using the UNIX operating system. UCExpress separates answer expression into two phases: *pruning* and *formatting*. In the pruning phase, subconcepts of the answer are pruned by being marked as already known by the user (and hence do not need to be generated), or marked as candidates for generating anaphora or ellipsis (since they are part of the conversational context). In the formatting phase, UCExpress uses information about the user's prior domain knowledge to select among specialized *expository formats*, such as similes and examples, for expressing information to the user. These formats allow UCExpress to present different types of information to the user in a clear, concise manner. The result of UCExpress' answer expression process is an internal form that a tactical level generator can easily use to produce good English.

Keywords: answer expression, formatting, generation, planning, pruning, user modeling

1. Introduction

UC (UNIX Consultant) (Wilensky et al. 1984; Wilensky et al. 1988), is a natural language consultation program that answers the user's questions on how to use the UNIX[1] operating system. After understanding the user's query, UC produces a plan consisting of communicative acts. This plan must be further refined because the concepts that UC wants to communicate to the user are often not organized in easily understood formats and often complete to the point of verbosity. The process of refining communicative actions is called *answer expression* (Luria 1982). The subcomponent of UC that does answer expression is called *UCExpress*. This paper describes how UCEx-

press refines a communicative plan to produce a clear, concise answer for expression to the user.

To see why answer expression is necessary, consider the following example:

User: What is a directory?

A1: A directory is a file.

A2: A directory is a file that is used to contain files.

A3: A directory is a file. Only empty directories can be deleted. Directories cannot be edited. Directories contain files. Directories form a tree-like structure. Directories always contain themselves and their parents. A plan for listing a directory is to use the ls command.

The different replies are all correct, but contain differing amounts of information. The first answer does not give enough information to distinguish between files and directories. The second answer (what UC actually produces) provides just enough information in a concise form and is the best answer in this situation. The third answer is too verbose and overwhelms the user with information. An answer like A3 would be more appropriate in cases where the user had requested, "Tell me all you know about directories."

This example illustrates the problem of determining how much of the answer to express to the user. Another problem in answer expression is deciding what format to use to present the answer. Consider the following scenario:

User: How can I move a file to another machine?

A1: To move a file to another machine, type 'rcp' followed by one or more spaces or tabs followed by the name of the file to be moved followed by one or more spaces or tabs followed by the name of the machine followed by a colon followed by the new name of the file on the other machine followed by a carriage return followed by 'rm' followed by one or more spaces or tabs followed by the name of the file.

A2: Use rcp to copy the file to another machine and then use rm to delete it. For example, to move the file foo to the file foo1 on machine dali, type 'rcp foo dali:foo1'.

The first answer is correct and quite general, but it is so verbose that it is indecipherable. On the other hand, the second answer is succinct and gives the user information in an easily readable form, but it is considerably less general. In fact the second answer is somewhat inaccurate, since it applies only to copying a file named foo to a file named foo1. It is up to the reader to use analogous reasoning to apply this to other cases. Despite this lack of generality, the second answer form is clearly superior to the first. Note that for a program to format the answer in the second form requires additional computation to transform the general solution of A1 into an *example*.

A natural language system needs to incorporate knowledge about when and how to use special presentation formats like examples to more clearly convey information to the user.

These concerns about how much information to present to the user and about what format to use can be viewed as corresponding respectively to Grice's Maxims of Quantity and Quality (Grice 1975). Although such considerations can be considered part of generation, there are sufficient differences in both the necessary knowledge and the processing to separate such strategic concerns from the more tactical problems of generation such as agreement and word selection. These strategic problems are the domain of an expression mechanism such as UCExpress.

2. User modeling

Because answer expression is concerned with how much information to present to the user, an important consideration for answer expression is: what does the user already know? Given a model of the user's knowledge and the current conversational context, an answer expression mechanism like UCExpress can reduce the amount of information presented to the user by pruning information already known to the user. Also, the level of expertise of the user may predict which presentation strategies will be most effective. For example, consider the actual UC sessions shown in Figures 1 and 2.

In session 1, UC gives an example of how to use the rm command whereas in session 2, UC does not give an example of how to use ls -i. This is because in the first session, the user is a novice, so UC cannot assume that the user knows the format of the rm command. However, in session 2, the user is an intermediate, so UC can assume that the user would know how to use ls -i. Also, in session 2, UC uses a simile to explain what ruptime does in terms of what uptime does. This simile is shorter and clearer than the full answer given by UC in session 1. However, this simile is only useful if the user already knows what uptime does. UC can assume this for the intermediate user of session 2, but cannot do so for the novice user of session 1. These examples show how a model of the user's knowledge can be used to improve the process of answer expression.

In UC, the KNOME (KNOwledge Model of Expertise) component models what the user knows about UNIX. More details can be found in Chin (1986, 1987, 1989), so this section will only give enough information so that the reader can understand how KNOME is used by UCExpress.

KNOME uses a *stereotype* approach (Rich 1979) where the characteristics of classes of users are organized under stereotypes. KNOME separates users into four levels of expertise (stereotypes): *novice*, *beginner*, *intermediate*,

```
# How can I delete a file?
Use rm.
For example, to delete the file named foo, type 'rm foo'.

# What does ruptime do?
ruptime is used to list the uptime of all machines on the network, list the number of all users
on them and list their load average.
```

Figure 1. UC session 1 with a novice user.

```
# How can I find out the inode of a file?
Use ls -i.

# What does ruptime do?
ruptime is like uptime, except ruptime is for all machines on the network.
```

Figure 2. UC session 2 with an intermediate user.

and *expert*. Individual users are classified as belonging to one of the above stereotype levels and inherit the characteristics of the stereotype. However, particular facts about the particular user override inheritance, so individual users differ from their stereotypes, which serve as reference points (Rosch 1978).

Besides stereotypes for users, KNOME also has stereotype levels for UNIX facts. This feature is termed a *double stereotype* system (Chin 1986; Chin 1989). Stereotype levels for UNIX facts include *simple*, *mundane*, *complex*, and *esoteric*. Examples of simple information are the rm, ls, and cat commands, the technical term "file," and the simple file command format (the name of the command followed by the name of the file to be operated upon). The mundane category includes the vi, diff and spell commands, the technical term "working directory," and the -l option of ls, while the complex category includes the grep, chmod, and tset commands, the term "inode," and the fact that write permission on the containing directory is a precondition for using the rm command for deleting a file. The esoteric category consists of information which is not in the mainstream usage of UNIX, but instead serves special needs. A good example is the spice program, that is useful only for people interested in semiconductor circuit simulations.

Thanks to the additional stereotype classification of UNIX information encoded in UC, it becomes extremely easy and space efficient to encode the

Table 1. Relation between user stereotypes and knowledge difficulty levels.

User stereotype	Knowledge difficulty level			
	Simple	Mundane	Complex	Esoteric
Expert	ALL	ALL	MOST	–
Intermediate	ALL	MOST	AFEW	–
Beginner	MOST	AFEW	NONE	–
Novice	AFEW	NONE	NONE	NONE

relation between user stereotypes and their knowledge of UNIX. The core of this knowledge is shown in Table I.

Table I indicates that the novice user in session 1 (see Figure 1) likely does not know the format for the rm command, which is a simple fact, and definitely does not know the uptime command, which is a mundane fact. On the other hand, the intermediate user in session 2 (see Figure 2) definitely knows the format for the ls -i command, which is a simple fact, and is likely to know the uptime command.

3. UCExpress

After other components of UC have identified a response to the user, this is passed to UCExpress, which decides how much of the response to present to the user and how to format it. The separation of this process of deciding how much of the answer to express from the process of figuring out the answer was first suggested by Luria (1982) who applied this distinction to a question answering system for story understanding. His system first found the causal chain that represented the answer, then used answer expression to decide how much of the causal chain to express to the user.

The response passed to UCExpress is in the form of a conceptual network in the KODIAK representation language (Wilensky 1987). UCExpress, operates on this input in two phases, *pruning* and *formatting*. During pruning, UCExpress prunes common knowledge from the answer using information about what the user knows based on the conversational context and a model of the user's knowledge. Next the answer is formatted using specialized expository formats for clarity and brevity. The final result is an augmented KODIAK conceptual network that is ready for direct generation into natural language using a tactical level generator such as KING (Jacobs 1986).

4. Pruning

When UCExpress is passed a set of concepts to communicate to the user, the first stage of processing prunes them by marking any extraneous concepts, so that later the generator will not generate them. The pruning is done by marking rather than actual modification of the conceptual network, since information about the node may be needed to generate appropriate anaphora for the pruned concept.

The guiding principle in pruning is to not tell the user anything that the user already knows. Currently UC models two classes of information that the user may already know. The first class of information is episodic knowledge from a model of the conversational context. The current conversational context is tracked by marking those concepts that have been communicated in the current session. The second class of information concerns the user's knowledge of UNIX related facts. Such user knowledge is modeled by KNOME. Thus any concept that is already present in the conversational context or that KNOME indicates is likely to be known to the user is marked and is not communicated to the user.

4.1. *An example trace*

To see how pruning works in detail, consider the trace of a UC session shown in Figure 3. The trace shows UCExpress' processing of the question, "How can I print a file on the laser printer?" The answer given by UC is, "Use lpr -Plz," along with a reminder to file the printer output in the boxes. The actual KODIAK conceptual network that is passed to UCExpress, shown in Figure 4 is not nearly as succinct, because it contains all of the details of the command that are needed for planning.

If the KODIAK network passed to UCExpress were to be generated directly into English, it might look like the following:

> To print a file on the laser printer, use the lpr -Plz command. The command-format of the lpr -Plz command is "lpr" followed by concatenating "-P" with "lz" followed by the name of the file to be printed on the laser printer.

This literal paraphrase is harder to understand than UC's more concise answer. To see how UCExpress prunes the network to arrive at the actual answer, consider the division of the concepts into the following three subnetworks:

PLANFOR260: A plan for PRINT-EFFECT0 is
UNIX-LPR-Plz-COMMAND0

```
# How can I print a file on the laser printer?

The parser produces:
(ASK10 (listener10 = UC)
       (speaker10 = *USER*)
       (asked-for10 = (QUESTION10 (what-is10 = (ACTION14? (actor14 = *USER*))))))
(PRINT-ACTION0? (pr-effect0 = PRINT-EFFECT0?)
                (actor0-1 = *USER*)
                (cause0-0 = (ACTION14? ...)))
(HAS-PRINT-DEST0 (pr-dest0 = LASER-PRINTER0)
                 (pr-dest-obj0 = PRINT-EFFECT0?))
(HAS-PRINT-OBJECT1 (pr-object1 = FILE3?)
                   (pr-obj-obj1 = PRINT-EFFECT0?))

The goal analyzer produces:
((HAS-GOAL-ga0 (planner-ga0 = *USER*)
               (goal-ga0 = (KNOW-ga0? (knower-ga0 = *USER*)
                                      (fact-ga0 = (ACTION14? ...))))))

The planner is passed:
(PRINT-EFFECT0?)

The planner produces:
(PLANFOR260 (goals260 = PRINT-EFFECT0?)
            (plan260 = (UNIX-LPR-Plz-COMMAND0 (lpr-plz-file0 = FILE3?)
                                              (UNIX-LPR-Plz-COMMAND-effect0 =
                                               PRINT-EFFECT0?))))
(HAS-FILE-NAME18 (named-file18 = FILE3?)
                 (file-name18 = (lisp = nil)))
(LPR-Plz-HAS-FORMAT0
 (LPR-Plz-HAS-FORMAT-command0 = (UNIX-LPR-Plz-COMMAND0 ...))
 (LPR-Plz-HAS-FORMAT-format0 =
  (LPR-Plz-FORMAT1 (lpr-plz-file-arg1 = (file-name18 = aspectual-of (HAS-FILE-NAME18 ...)))
                   (LPR-Plz-FORMAT-step1 =
                    (SEQUENCE10 (step10 = lpr)
                                (next10 = (CONCAT00 (concat-step00 = -P)
                                                    (concat-next00 = lz))))))))
(HAS-COMMAND-NAME30 (HAS-COMMAND-NAME-named-obj30 = (UNIX-LPR-Plz-COMMAND0 ...))
                    (HAS-COMMAND-NAME-name30 = (SEQUENCE10 ...)))

Express:  now expressing the PLANFOR:
(PLANFOR260 ...)

Express:  not expressing the format of the command, UNIX-LPR-Plz-COMMAND0,
          since the user already knows it.

Express:  not expressing PRINT-EFFECT0?, since it is already in the context.

The generator is passed:
(TELL7 (listener7-0 = *USER*)
       (speaker7-0 = UC)
       (proposition7 = (PLANFOR260 ...))
       (effect7 = (STATE-CHANGE1 (final-state1 = (KNOW-ga0? ...)))))

The generator is passed:
(TELL8 (speaker8 = UC)
       (listener8 = *USER*)
       (proposition8 = (REMINDER10 ...)))

Use lpr -Plz.

Don't forget to file the printer output in the boxes.
```

Figure 3. UC session with an intermediate user showing trace of UCExpress.

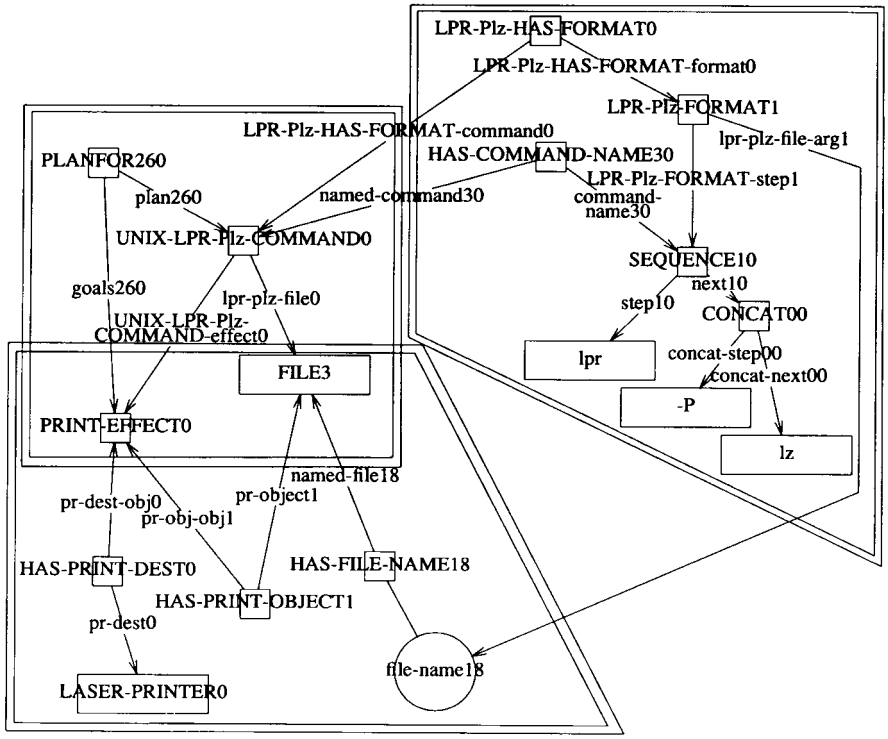

Figure 4. KODIAK representation of the lpr -Plz plan for printing.

PRINT-EFFECT0: Printing a file on the laser printer

LPR-Plz-HAS-FORMAT0: The command-format of the UNIX-LPR-Plz-COMMAND0 is "lpr -Plz <the name of the file to be printed>"

These three subnetworks are depicted in Figure 4 as regions enclosed in double lines. In traversing this network, UCExpress prunes LAS-PRINT-EFFECT0, because "printing a file on the laser printer" is already a part of the context (it is part of the user's question). Also, the command-format (LPR-Plz-HAS-FORMAT0) is pruned from UC's actual answer based on information from KNOME. In this case, KNOME was able to deduce that, since the user was not a novice, the user already knew the UNIX-LPR-Plz-FORMAT, which is an instance of the SIMPLE-FILE-FORMAT (the name of the command followed by the name of the file to be operated upon), which all non-novice users know. Finally what is left unpruned is the plan

part of PLANFOR260, UNIX-LPR-Plz-COMMAND0, which the generator translates as "Use lpr -Plz."

If the user was just a novice, then UC could not assume that the user already knew the command-format and instead would provide the following answer that includes an example of the lpr -Plz command-format:

> Use lpr -Plz.
> For example, to print the file foo on the laser printer, type 'lpr -Plz foo'.

Pruning is similar to the "msg-elmt" realization stage of McDonald's MUMBLE (McDonald 1984), which was used to generate pronouns when a concept had been previously mentioned by MUMBLE. However, since MUMBLE did not have access to a model of the user, it was not able to avoid expressing those concepts which a user model would indicate that the user already knows. Another approach is used by Appelt's KAMP system (Appelt 1985) in planning referring expressions. KAMP used mutual knowledge as a criterion for planning pronominal and anaphoric noun phrases. It would be very difficult to adapt such an approach to do pruning since KAMP does not deal with the uncertainty that is inherent in user models like KNOME that reason from stereotypes.

5. Formatting

After pruning, UCExpress enters the formatting phase, during which it tries to apply different *expository formats* to express concepts in a clearer manner. UCExpress' expository formats include *example*, *definition*, and *simile*. Each expository format is used to express different types of information. Formats are triggered by encountering particular concept types in the answer network. After triggering, the procedural component of the expository format is called to transform the concept into the corresponding format. The formats are not simple templates that can be filled in with readily available information. A fair amount of additional processing is needed to transform the information into the right format.

5.1. *Example format*

The example format is used in expressing general knowledge about complex (i.e., multi-step) procedures such as UNIX commands. In UC's representation of UNIX commands, every command has an associated command format. When expressing a command, UCExpress checks to see if it should also express the format of the command. If KNOME believes that the user already knows the format of the command, then there is no need to express the format.

Next, UCExpress checks to see if the format of the command is completely specified. If so, UCExpress collapses the command and format into a single statement as in the following UC dialog:

```
# How can I add general write protection to the file personal?
Type 'chmod o-w personal'.
```

An English rendition of the conceptual network passed to UCExpress for the above example might be something like:

> A plan for adding general write protection to the file personal is to use the chmod command with format 'chmod' followed by concatenating 'o' with '-' with 'w' followed by 'personal'.

Since the command is completely specified, the format of the command is combined with the command to form a shorter and more easily understood answer.

If the command is not completely specified, then UCExpress uses an example format to express the format of the command to the user. The key principle in producing examples is to be explicit. So, UCExpress first steps through a copy of the general procedure to transform any general information into specific instances. In cases where the under-specified part of the procedure has a limited range of options, an arbitrary member that is compatible with the rest of the procedure and with previous UCExpress choices is selected. Next, the new, completely specified copy of the format is combined with a copy of the command, much as in the above UC dialog. Finally the new plan is encapsulated in an example shell (which tells the generator to produce "For example,").

To see the algorithm in more detail, consider the UC dialog shown in Figures 5 and 6.

The conceptual answer that is passed to UCExpress in the dialog can be paraphrased in English as:

> A plan for changing the read permission of a file is to use the chmod command with format 'chmod' followed by concatenating <the protection-user-type> with <the protection-value-type> with 'r' followed by <the name of the file to be changed>.

In stepping through the above format, <the protection-user-type> is under-specified. In order to give an example, a particular value is needed, so UCExpress arbitrarily chooses a value from the list of possible fillers (user, group, other, or all). The same is done for <the protection-value-type>. In the case of 'r', this is already a fully specified value for protection-access-type,

```
# How can I change the read permission of a file?

The parser produces:
(ASK10 (listener10 = UC)
       (speaker10 = *USER*)
       (asked-for10 = (QUESTION10 (what-is10 = (ACTION14? (actor14 = *USER*))))))
(CHANGE-PROT-FILE-ACTION0? (ch-prot-effect0 = (CHANGE-PROT-FILE-EFFECT0?
                                               (change-prot0 = FILE-PROTECTION1)
                                               (change-file0 = FILE3?)))
                           (actor0-1 = *USER*)
                           (cause0-0 = (ACTION14? ...)))
(HAS-FILE-PROTECTION2 (prot-file2 = FILE3?)
                      (file-prot2 = FILE-PROTECTION1))
(HAS-ACCESS-TYPE1 (access-protection-type1 = READ-PROT)
                  (prot-type-arg1 = FILE-PROTECTION1))

The goal analyzer produces:
((HAS-GOAL-ga0 (planner-ga0 = *USER*)
               (goal-ga0 = (KNOW-ga0? (knower-ga0 = *USER*)
                                      (fact-ga0 = (ACTION14? ...))))))

The planner is passed:
((CHANGE-PROT-FILE-EFFECT0? ...))
```

Figure 5. First half of UC session showing an answer that contains an example.

so UCExpress maintains the selection. However, with <the name of the file to be changed>, there is no list of possible fillers. Instead, UCExpress calls a special procedure for selecting names. This naming procedure chooses names for files starting with 'foo' and continuing in each session with 'foo1', 'foo2', etc. Other types of names are selected in order from lists of those name types (e.g., machine names are chosen from a list of local machine names). By selecting the names in order, name conflicts (e.g., two different files with the same name) can be avoided.

Another consideration in creating examples is that new names must be introduced before their use. Thus 'foo' should be introduced as a file before it appears in 'chmod g+r foo'. This is done implicitly by passing the entire PLANFOR as the example, so that the generator will produce 'to add group read permission to the file named foo' as well as the actual plan.

5.2. *Definition format*

The definition format is used to express definitions of terminology. The UC-define procedure first collects the information that will be expressed in the definition. Collecting the right amount of information involves satisfying the Gricean Maxim of Quantity (Grice 1975). The usual procedure is to collect the information that the term has some semantic category, and then add the primary usage of the term. In rare cases where the node does not have a

```
The planner produces:
(PLANFOR330 (goals330 = (CHANGE-PROT-FILE-EFFECT0? ...))
            (plan330 = (UNIX-CHMOD-COMMAND0 (chmod-file0 = FILE3?)
                                            (chmod-protection0 = FILE-PROTECTION1)
                                            (UNIX-CHMOD-COMMAND-effect0 =
                                             (CHANGE-PROT-FILE-EFFECT0? ...)))))
(HAS-FILE-NAME19 (named-file19 = FILE3?)
                 (file-name19 = (lisp = nil)))
(HAS-PROT-VALUE1 (prot-type-arg1-1 = FILE-PROTECTION1)
                 (value-protection-type1 = (lisp = nil)))
(HAS-USER-TYPE1 (prot-type-arg1-0 = FILE-PROTECTION1)
                (user-protection-type1 = (lisp = nil)))
(CHMOD-HAS-FORMAT0 (CHMOD-HAS-FORMAT-command0 = (UNIX-CHMOD-COMMAND0 ...))
                   (CHMOD-HAS-FORMAT-format0 = (CHMOD-FORMAT0 (CHMOD-FORMAT-step0 = chmod)
                                                              (CHMOD-FORMAT-args0 = ...))))
(HAS-COMMAND-NAME80 (HAS-COMMAND-NAME-named-obj80 = (UNIX-CHMOD-COMMAND0 ...))
                    (HAS-COMMAND-NAME-name80 = chmod))

Express:  now expressing the PLANFOR:
(PLANFOR330 ...)

Express:  creating an example for the incomplete plan, CHMOD-FORMAT0

Express:  choosing a name, foo, for an example file.

Express:  selecting USER-PROT -- print name, u, to fill in a parameter of the example.

Express:  selecting ADD-STATUS -- print name, +, to fill in a parameter of the example.

Express:  created the example(s):
((TELL7 (speaker7-0 = UC)
        (listener7-0 = *USER*)
        (proposition7 =
         (EXAMPLE0 (example0 = (PLANFOR330-0
                                (goals330-0 = (CHANGE-PROT-FILE-EFFECT0-0?
                                               (change-prot0-0 = FILE-PROTECTION1-0)
                                               (change-file0-0 = FILE3-0?)))
                                (plan330-0 = (TYPE-ACTION0
                                              (speaker0-4 = *USER*)
                                              (type-string0 =
                                               (CHMOD-FORMAT0-0
                                                (CHMOD-FORMAT-step0-0 = chmod)
                                                (CHMOD-FORMAT-args0-0 =
                                                 (CHMOD-TWO-ARG-SEQ0-0
                                                  (chmod-file-arg0-0 = ... foo)
                                                  (CHMOD-TWO-ARG-SEQ-step0-0 =
                                                   (PROT-ARG-SEQ0-0
                                                    (user-bit0-0 = ... u)
                                                    (PROT-ARG-SEQ-concat-next0-0 =
                                                     (ARG-SEQ0-0 (value-bit0-0 = ... +)
                                                                 (access-bit0-0 =
                                                                  r)))))))))))))

Express:  not expressing CHANGE-PROT-FILE-EFFECT0?, since it is already in the context.

The generator is passed:
(TELL6 (effect6 = (STATE-CHANGE1 (final-state1 = (KNOW-ga0? ...))))
       (listener6-0 = *USER*)
       (speaker6-0 = UC)
       (proposition6 = (PLANFOR330 ...)))

The generator is passed:
(TELL7 ...)

Use chmod.

For example, to add group read permission to the file named foo, type 'chmod g+r foo'.
```

Figure 6. Second half of UC session showing an answer that contains an example.

usage, some other property of the node is chosen. For example, a definition of a directory would include the information:

1. directories are files
2. directories are used to contain files

After such information is collected, it must be transformed into a definition format. This involves creating instances of both the term and its category and then combining the two pieces of information into one coherent statement. The latter task requires an attachment inversion where the distinguishing information is reattached to the term's category rather than to the term itself. For example, consider creating a definition for the term "directory," which has the category "file" (in UNIX, a directory is simply a special type of file). The information that distinguishes directories from other types of files is that directories are used to contain files (as opposed to documents, movies, spreadsheets, etc.). To create the definition, the distinguishing information is reattached from the term directory to its category, file to create the following definition.

User: What is a directory?

UC: A directory is a file that is used to contain files.

This attachment inversion is not specific to English but seems to be a general universal linguistic phenomenon in the expression of definitions. Here are some other examples of the definition format:

User: What is a file?

UC: A file is a container that is used to contain text, code, or files.

User: What is a container?

UC: A container is an object that is used to contain objects.

User: What is rm?

UC: Rm is a command that is used to delete files.

User: What is a search path?

UC: A search path is a list of directories that is used by the csh to search for programs to execute.

5.3. *Simile format*

The simile format is used by UCExpress to provide explanations of what a command does in terms of other commands already known to the user. This format is invoked when UCExpress attempts to explain a command that has a sibling or a parent in the command hierarchy that the user already knows (as modeled in KNOME). An example is explaining what ruptime does in terms of the command uptime. A trace of UC's processing is shown in Figure 7.

```
# What does ruptime do?

The parser produces:
(ASK10 (listener10 = UC)
       (speaker10 = *USER*)
       (asked-for10 = (QUESTION10 (what-is10 = STATE13?))))
(HAS-EFFECT21? (effect-of-command21 = STATE13?)
               (command-of-effect21 = UNIX-RUPTIME-COMMAND0))

The goal analyzer produces:
((HAS-GOAL-ga0 (planner-ga0 = *USER*)
               (goal-ga0 = (KNOW-ga0? (knower-ga0 = *USER*)
                                      (fact-ga0 = STATE13?)))))

UCEgo: trying to find effects for UNIX-RUPTIME-COMMAND0
       the effects are:
((HAS-EFFECT6-0 (command-of-effect6-0 = (UNIX-RUPTIME-COMMAND0 ...))
                (effect-of-command6-0 = (LIST-ACTION3-0 (list-loc3-0 = TERMINAL1-0)
                                                        (list-objs3-0 = UP-TIME1-0))))
 (HAS-EFFECT7-0 (command-of-effect7-0 = (UNIX-RUPTIME-COMMAND0 ...))
                (effect-of-command7-0 = (LIST-ACTION4-0 (list-loc4-0 = TERMINAL1-0)
                                                        (list-objs4-0 = NUMBER1-0))))
 (HAS-EFFECT8-0 (command-of-effect8-0 = (UNIX-RUPTIME-COMMAND0 ...))
                (effect-of-command8-0 = (LIST-ACTION5-0 (list-loc5-0 = TERMINAL1-0)
                                                        (list-objs5-0 = LOAD-AVERAGE1-0)))))

UCExpress: Found a related command, so creating a comparison
between UNIX-RUPTIME-COMMAND2 and UNIX-UPTIME-COMMAND0

Express:  not expressing UNIX-RUPTIME-COMMAND0, since it is already in the context.

The generator is passed:
(TELL5 (effect5 = (STATE-CHANGE1 (final-state1 = (KNOW-ga0? ...))))
       (listener5-0 = *USER*)
       (speaker5-0 = UC)
       (proposition5 =
        (HAS-EFFECT24 (command-of-effect24 = (UNIX-RUPTIME-COMMAND0 ...))
                      (effect-of-command24 =
                       (AND0 (step0-0 = (LIST-ACTION3-0 (list-loc3-0 = TERMINAL1-0)
                                                        (list-objs3-0 = UP-TIME1-0)))
                             (next0-0 = (AND1 (step1-0 = (LIST-ACTION4-0
                                                          (list-loc4-0 = TERMINAL1-0)
                                                          (list-objs4-0 = NUMBER1-0)))
                                              (next1-0 = (LIST-ACTION5-0
                                                          (list-loc5-0 = TERMINAL1-0)
                                                          (list-objs5-0 =
                                                           LOAD-AVERAGE1-0)))))))))

ruptime is like uptime, except ruptime is for all machines on the network.
```

Figure 7. UC session showing the simile format.

The processing involves comparing the effects of the two commands and noting where they differ. In the above example, the effects of uptime are to list the uptime of the user's machine, list the number of all users on it, and list its load average. The effects of ruptime are similar except it is for all machines on the user's network. The comparison algorithm does a network comparison of the effects of the two commands. A collection of differences is

generated, and the cost of expressing these differences (measured in number of concepts) is compared with the cost of simply stating the effects of the command. If expressing the differences is more costly, then the simile format is not used. On the other hand, if expressing the differences is less costly, then the differences are combined into a shell of the form "<CommandA> is like <CommandB>, except [<CommandA> also ...] [and] [<CommandA> does not ...] [and] ..."

6. Conclusion

6.1. *A comparison*

McKeown's TEXT system (1985) is perhaps the closest in spirit to UCExpress. TEXT provided definitions using an *identification schema*. This is similar to UCExpress' definition format except TEXT did not worry about how much information to convey. TEXT was designed to produce only paragraph length descriptions, hence it was not overly concerned with how much information to provide. The definition format requires more knowledge about the domain in order to select the most relevant information for a short description. TEXT also used a *compare and contrast schema* to answer questions about the differences between objects in a database. This is similar to UCExpress' simile format except that the compare and contrast schema was not used for giving descriptions of an object in terms of another that the user already knew. Since TEXT did not have a complete model of the user, it was unable to determine if the user already knew another object that could be contrasted with the requested object. This lack of a user model was also evident in the fact that TEXT did not provide anything similar to the pruning phase of UCExpress. Pruning is probably more relevant in a conversational context such as UC as contrasted with a paragraph generation context such as TEXT. On the other hand, TEXT was able to keep track of the conversational focus much better than UC. Focus does not seem to be quite as essential for a system like UC that gives brief answers.

Other related research includes work on using examples for explanation and for argument in a legal domain (Rissland 1983; Rissland et al. 1984). The difference between those examples and the examples created by UCExpress is that Rissland's examples are pre-formed and stored in a database of examples whereas UCExpress creates examples interactively, taking into account user provided parameters. Rissland's HELP system dealt only with help about particular subjects or commands rather than arbitrary English questions like UC, thus HELP did not have to deal with questions such as how to print on a particular printer. Also by using pre-stored text, HELP was not concerned

with the problem of transforming knowledge useful for internal computation in a planner to a format usable by a generator.

The TAILOR system (Paris 1989) used an idea of user expertise similar to KNOME's to tailor explanations to the user's level of expertise. TAILOR concentrated on higher level strategies for explanation than UCExpress. For example, TAILOR used notions of the user's level of expertise to choose among process-oriented or parts-oriented description strategies in building up a paragraph. TAILOR could also mix the two types of strategies within a paragraph to explain different aspects of a system. Such considerations are more important when generating longer explanations as in TAILOR, than when generating brief explanations as in UCExpress.

6.2. *Summary*

UC separates the realization of speech acts into two processes: deciding how to express the speech act in UCExpress, and deciding which phrases and words to use in UC's tactical level generator. Through this separation, the pragmatic knowledge needed by expression is separated from the grammatical knowledge needed by generation. UCExpress makes decisions on pragmatic grounds such as the conversational context, the user's knowledge, and the ease of understanding of various expository formats. These decisions serve to constrain the generator's choice of words and grammatical constructions.

Of course, it is sometimes impossible to realize all pragmatic constraints. For example, UCExpress may specify that a pronoun should be used to refer to some concept since this concept is part of the conversational context, but this may not be realizable in a particular language because using a pronoun in that case may interfere with a previous pronoun (in another language with stronger typed pronouns, there may not be any interference). In such cases, the generator needs to be able to relax the constraints. By passing the generator all of the conceptual network along with additional pragmatic markings on the network UCExpress allows the generator to relax constraints as needed. This way, the generator has access to any information needed to relax the constraints added by UCExpress.

7. Recent developments

An abbreviated version of this paper was previously presented at the 1988 AAAI Conference (Chin 88). Answer expression is currently being extended for multi-media output in the MC (Maintenance Consultant) system. A

description of of the multi-media input and user modeling aspects of MC can be found in (Chin et al. 94).

Acknowledgements

The work described in this paper was done at the University of California, Berkeley as part of my Ph.D. thesis. I wish to thank Robert Wilensky who supervised this work. I also wish to thank the members of BAIR (Berkeley Artificial Intelligence Research) who have contributed to the UC project. This research was sponsored in part by the Defense Advanced Research Projects Agency (DoD), ARPA order No. 4871, monitored by Space and Naval Warfare Systems Command Command under contract N00039-84-C-0089, by the Office of Naval Research under contract N00014-80-C-0732, by the National Science Foundation under grant MCS79-06543, and by the Office of Naval Research under contract N00014-97-1-0578.

Note

[1] UNIX is a trademark of X/Open, Inc.

References

Appelt, D. E. (1985). *Planning English Sentences*. Cambridge: Cambridge University Press.

Chin, D. N. (1986). User Modeling in UC, the UNIX Consultant. In Proceedings of *The CHI-86 Conference*, 24–28. Boston, MA: Association for Computing Machinery.

Chin, D. N. (1987). *Intelligent Agents as a Basis for Natural Language Interfaces*. Ph.D. diss., Computer Science Division, University of California, Berkeley, CA. Also available as UCB/CSD 88/396, Computer Science Division, University of California, Berkeley, CA.

Chin, D. N. (1988). Exploiting User Expertise in Answer Expression. In Proceedings of *The Seventh National Conference on Artificial Intelligence*, 756–760. Saint Paul, MN: AAAI Press.

Chin, D. N. (1989). KNOME: Modeling What the User Knows in UC. In Kobsa, A. & Wahlster, W. (eds.) *User Models in Dialog Systems*, 74–107. Berlin: Springer-Verlag.

Chin, D. N., Inaba, M., Pareek, H., Nemoto, K., Wasson, M. & Miyamoto, I. (1994). Multi-Dimensional User Models for Multi-media I/O in the Maintenance Consultant. In Proceedings of *Fourth International Conference on User Modeling*, 139–144. Hyannis, MA: User Modeling, Inc.

Grice, H. P. (1975). Logic and Conversation. In Cole, P. & Morgan, J. L. (eds.), *Studies in Syntax* **III**, 41–58. New York: Seminar Press.

Jacobs, P. S. (1986). *A Knowledge-Based Approach to Language Production*. Ph.D. diss., University of California, Berkeley, CA. Also available as UCB/CSD 86/254, Computer Science Division, University of California, Berkeley, CA.

Luria, M. (1982). Dividing up the Question Answering Process. In Proceedings of *The Second National Conference on Artificial Intelligence*, 71–74. Pittsburgh, PA: AAAI Press.

McDonald, D. D. (1984). Natural Language Generation as a Computational Problem: an Introduction. In Brady, M. & Berwick, R. C. (eds.), *Computational Models of Discourse*, 209–265. Cambridge, MA: MIT Press.

McKeown, K. R. (1985). Discourse Strategies for Generating Natural-Language Text. *Artificial Intelligence* **27**: 1–41.

Paris, C. L. (1989). Tailoring Object Descriptions to a User's Level of Expertise. In Kobsa, A. & Wahlster, W. (eds.) *User Models in Dialog Systems*, 200–232. Berlin: Springer-Verlag.

Rich, E. (1979). User Modeling via Stereotypes. *Cognitive Science* **3**: 329–354.

Rissland, E. L. (1983). Examples in Legal Reasoning: Legal Hypotheticals. In Proceedings of *The Eighth International Joint Conference on Artificial Intelligence* **1**, 90–93. Karlsruhe, Germany: Morgan Kaufmann Publishers.

Rissland, E. L., Valcarce, E. M. & Ashley, K. D. (1984). Explaining and Arguing with Examples. In Proceedings of *The Fourth National Conference on Artificial Intelligence*, 288–294. Austin, TX: AAAI Press.

Rosch, E. (1978). Principles of Categorization. In Rosch, E. & Lloyd, B. B. (eds.) *Cognition and Categorization*. Hillsdale, NJ: Lawrence Erlbaum.

Wilensky, R. (1987). *Some Problems and Proposals for Knowledge Representation*. UCB/CSD 87/351, Computer Science Division, University of California, Berkeley, CA.

Wilensky, R., Arens, Y. & Chin, D. N. (1984). Talking to UNIX in English: An Overview of UC. *Communications of the ACM* **27**(6): 574–593.

Wilensky, R., Chin, D. N., Luria, M., Martin, J., Mayfield, J. & Wu, D. (1988). The Berkeley UNIX Consultant Project. *Computational Linguistics* **14**(4): 35–84.