Cogito, An Expert System to Give Installation Advice for UNIX 4.2BSD

A. Terry Bahill
and
*Pat Harris**
Systems and Industrial Engineering
University of Arizona
Tucson, AZ 85721

When you try to use a computer, your first effort inevitably fails. You then recall the sage advice, "If all else fails, read the instructions." So you decide to do this. But where do you start? For the tasks described in this paper the instructions were hundreds of pages long spread over a several manuals. So we wrote an expert system to help the human use this data.

In this paper we will discuss *Cogito*, an expert system that gives installation advice for bringing up the UNIX 4.2BSD operating system on a VAX computer [1]. A detailed reference manual is currently used for installation instructions. Task complexity limits this method. Cogito filters the information and presents only relevant advice about the user's computer system. Cogito remembers data the user has previously entered and uses this to customize its response. Cogito, written in M.1 and running on a personal computer, uses if-then production rules to encode the knowledge. Cogito's knowledge base has two components: classification knowledge and process knowledge. Classification knowledge transforms one kind of knowledge into another. Production rules are appropriate for Cogito's classification knowledge. Process knowledge directs the flow of information and the explanation for that knowledge. Cogito's process knowledge is difficult to encode with production rules because in describing the installation process the expert does not think in terms of rules.

Problem Statement

Installing a computer system is an evolutionary process consisting of a series of operations that transform a computer into a complex system capable of supporting many users and functions. There are two basic types of system installation: system building and device integration. System building is required when the computer hardware is delivered and involves installation of the operating system. Device integration is required when a new device is obtained and must be integrated with the rest of the system. This paper will only discuss system building.

The knowledge needed to do system installation is fact-intensive because hardware and software designers have already made many decisions regarding the system structure. Consequently, system installation dictates a long, interconnected series of steps to get the hardware and software to interact correctly. This process is especially complex with the UNIX operating system since it is designed to be as portable as possible, supporting many different types of computers and devices. For example, 4.2BSD UNIX supports three VAX models, three communication buses, 21 disk drive types, 11 tape drive types and 22 device types. Simple combinatorics yields the number of 45,000 minimal systems (one VAX cpu, one communication bus, one disk drive, one tape drive and one console terminal). Almost no one has a minimal system. However, our Systems and Industrial Engineering Department's VAX-UNIX system is close to minimal: it has one VAX cpu, two communication buses, one disk drive, one tape drive and two other devices. For this system there are approximately a half a million combinations.

*Present address Bell Communications Research, New Jersey.

Established Method

Information regarding system installation of UNIX 4.2BSD is contained in the *UNIX Systems Manager's Manual* [2], and the *UNIX Programmer's Manual Reference Guide* [3]. The *UNIX 4.2BSD Systems Manager's Manual* is an extensive reference document that explains in detail the making and installing of UNIX 4.2BSD. The *Programmer's Manual Reference Guide* provides detailed information about specific devices that may be attached to the computer. Unfortunately, these reference manuals are also forced to serve as the only tutorials available for making and installing the UNIX system. They are not well suited for this task for several reasons.

First, the manuals' structures do not provide a model of the problem for system installers to base their learning on. Additionally, the information is often written at a higher level than inexperienced system installers can understand and explanations are sometimes pages distant from the first example. Finally, the wide variety of possible VAX computers, disks, tape drives, controllers, printers, terminals and other hardware makes it impossible for a linear explanation, such as in a book, to distinguish all the relevant information for a particular user. Consequently, to install the operating system with this method, a system installer must painstakingly seek to understand and extract the information needed for his or her system from the vast amount of details for all possible combinations provided by the manuals.

Example Rules

We transformed this vast amount of information into if-then production rules for our expert system. The following shows examples of these rules.

This rule sets up the correspondence between the DEC disk name and the DEC-bus that it is attached to.

```
dt-hp-a:  if disk = 'RM03'
          or disk = 'RM05'
          or disk = 'RM80'
          or disk = 'RP06'
          or disk = 'RP07'
          then disk-bus = 'MASSBUS'.
```

This rule establishes the first level correspondence between DEC disk names and their UNIX counterparts.

```
dt-hp-b:  if disk = 'RM03'
          or disk = 'RM05'
          or disk = 'RM80'
          or disk = 'RP06'
          or disk = 'RP07'
          then standalone-disk-name = hp.
```

These rules establish the UNIX-to-UNIX relationship between the standalone disk names and their controller designations. This demonstrates the nonobvious relationships that occur in UNIX-land that are very similar to the English language tradition of irregular verb forms; some forms fit the pattern, while most do not.

```
dtc-1:  if standalone-disk-name = hk
        then controller = hk.

dtc-2:  if standalone-disk-name = ra
        then controller = uda.

dtc-3:  if standalone-disk-name = rx
        then controller = fx.

dtc-4:  if standalone-disk-name = up
        then controller = sc.
```

These two rules show that non-DEC disks can be attached to either the UNIBUS or the MASSBUS, but the UNIX disk types are decidely different.

```
dt-26: if disk = 'AMPEX 330M'
       and disk-bus = 'UNIBUS'
       then disk-type = capricorn.
dt-34: if disk = 'AMPEX 300M'
       and disk-bus = 'MASSBUS'
       then disk-type = 9300.
```

User Satisfaction

Cogito is a better method for system installation than using the *UNIX Systems Manager's Manual*, in terms of overall user satisfaction, because the amount and relevancy of the information presented is significantly increased. For example, in the disk definition state the file */etc/fstab* must be created. Assume that the boot disk is a 'AMPEX 300M' connected to uba0 at drive 1. This implies that the disk address is 1. Also assume the user's name is 'Pat' and that he is ready to install *fstab*. The instructions given in the manual are generic and therefore the human must relate the general instructions to the specific application. Figure 1 is a copy of a the relevant section of the manual.

4.4.1. Initializing /etc/fstab

Change into the directory /etc and copy the appropriate file from:

```
fstab.rm03
fstab.rm05
fstab.rm80
fstab.ra60
fstab.ra80
fstab.ra81
fstab.rp06
fstab.rp07
fstab.rk07
fstab.up160m (160Mb up drives)
fstab.up300m (300Mb up drives)
fstab.hp400m (400Mb hp drives)
fstab.up (other up drives)
fstab.hp (other hp drives)
```

to the file /etc/fstab, i.e.:

```
# cd /etc
# cp fstab.xxx fstab
```

This will set up the initial information about the usage of disk partitions, which we see how to update more below.

Figure 1: Installation of *fstab* from [2].

Cogito's instructions to complete the same task are:

```
# cd /etc
# cp fstab.up300m junk
# vi junk
(Edit the file, Pat.)
    a. Add the line '/dev/up0b::sw::'.
    b. Give the global substitute command ':g/up0/s//up1/'.
    c. Save the new contents and quit the editor.)
# cat junk >> fstab
```

Cogito has remembered that half an hour ago the user said his name was Pat and his disk was an AMPEX 300M and has used this information to make its instructions specific. Cogito's instructions are personalized, relevant to the user's task, clear and complete!

This example illustrates that the manual's instructions are incomplete, and rely on previous knowledge and implicit knowledge: they are incomplete because the swap partition */dev/up0b*, must be added to the file; they rely on previous knowledge because the user must recall that the UNIX standalone disk name for a 'AMPEX 300M' is 'up'; and they rely on implicit knowledge because the user must somehow know that the disk address of the partitions needs to be changed from '0' to '1'.

To address the issue of whether Cogito is the best system for advising on UNIX system installation, qualifications must be made. As originally designed, Cogito has two user classes: a system builder and a system integrator. It's interesting to note that although the knowledge base is the same for both kinds of users, Cogito is inadequate for the system integrator user. All the appropriate information is given to the user but the instrument on which it is displayed is wrong. Transmission of the advice via a personal computer seems inconvenient when the VAX computer is running. Ideally, Cogito should have the capability to run on the VAX computer once it is up and running. This includes the ability to transfer information "learned" by Cogito in the system building stage into a database where it can be retrieved when configuring a new device. Unfortunately, this cannot be done because M.1 only runs on a personal computer.

Information Flow

Information flow is concerned with the data transfer from the user into Cogito and the corresponding transfer of advice from Cogito to the user. The information flow depends on the knowledge base structure. Since Cogito is based on the backward chaining inference engine M.1, both direction flows (into and out of Cogito) depend on a goal-oriented knowledge base structure. Ideally the knowledge base is a true reflection of how the expert thinks about the problem. In Cogito, the knowledge base was originally constructed without reference to a particular inference engine. For instance, an original rule near the end of the configuration state was:

```
if config done then
    # config NAME
    # cd ../NAME
    # make depend
    # make vmunix
```

As implemented in Cogito the control method of the inference engine imposed a backward chaining thought process to occur. Hence, using M.1, the implemented version of the same rule is:

```
if  display(' # config NAME ')
and display(' # cd ../NAME')
and display(' # make depend ')
and display(' # make vmunix ')
then config is done.
```

Looking at Cogito from the end user's viewpoint, the questions asked and the advice presented appear to be given in a logical, relevant and concise manner. However, the implemented knowledge base structure of Cogito suffers from an unnatural viewpoint, forced by the control method used in the inference engine.

Choice of Expert System Shell

We had two conveniently available expert systems shells to choose from: M.1, primarily a back chainer, and OPS5, primarily a forward chainer. Although the problem domain seems to be data driven, which would suggest a forward chainer, we found that either shell worked. The knowledge base was just a little longer using the back chainer. The decision to use M.1 was primarily based on the differences in rule implementations. Cogito's rules are English-like phrases, whereas, with OPS5, the rules are reminiscent of LISP code, the implementation language. Experts not familiar with LISP have trouble reading and understanding the content of the rules. It is important that the knowledge engineer be able to verify with the expert that the intent of the rule is the same as the coded rule. This is especially crucial if the expert system is giving incorrect advice.

Testing

Testing this expert system was difficult. It was impossible to present it with every possible combination of inputs and evaluate its outputs. The best test we devised was to let the intended users use it in many hypothetical circumstances. If the knowledge base was incomplete, then, in some situations the advice given to the users should be incorrect. Cogito was tested by the Systems Administrator of the Department of Systems and Industrial Engineering, a Professor of Systems and Industrial Engineering, and the Systems Administrator for the Department of Computer Science. They all found Cogito's advice to be complete and correct.

One of our colleagues suggested that the system be tested by 12 random graduate students and that the results be subjected to statistical analysis. We felt this would be unfair, because the system was designed to be used by people with a good knowledge of computer hardware and UNIX software and we only knew of four such people on the University of Arizona campus (the builder of the system and the three testers).

In an effort to bolster our testing we asked several non-expert computer users to try the system. Their evaluations tended to emphasize the difficulties they had using it or making sense of its queries or output, and the extent to which they could "fool" the system with plausible (but nonsensical) inputs. They did not fool this system. So, we did our best to test Cogito, but we were not able to prove that it would always give the correct advice. Testing seems to be a problem with most expert systems.

The best way to test this system would have been to use it to bring up a a brand new system. Unfortunately no such system was available. The next best test would have been to shut down an existing system, and rebuild it using Cogito. Unfortunately no one wants to let you shut down their operational system. Recently, however, due to inadequate glue on the heads of our RA81 disk, we had to rebuild our system from the distribution tapes. Cogito helped us. We found a few omissions in Cogito's advice, but no mistakes. It was a big help. We completed the task in about 12 hours.

This experience has reinforced our belief that all expert systems are inadequately tested. There are no quantitative procedures for testing expert systems. Most tests merely involve running a few case studies; they do not exhaust all possibilities. For example, we are confident that Cogito works well for a small VAX 750 system but we cannot be sure that it will work as well for a 730 or a 780.

Appropriateness of Expert System Technology

How can one identify a task that is appropriate for an expert system? First, there must be a human expert who performs that task better than most other people. Second, the task must be one to which the human expert can explain the solution in words, not one that requires the expert to draw a picture to explain what to do. Third, can the problem be solved routinely in a 20 minute, or even a one hour, telephone conversation with the expert? If so, the problem is a good candidate for a personal computer based expert system. If a human would take two days to solve the problem, it is far too complicated for an expert system; if the human gives the answer in two seconds, it is too simple.

Given these criteria, giving advice for bringing up UNIX on a VAX computer was inappropriate for a personal computer based expert system. Bringing up UNIX cannot be done in a one hour conversation with an expert. We think it would take an expert one to two days to do the task. (It took us three months to do it the first time!) The 700 rules of this expert system filled up two floppy disks. We succeeded in making an expert system that worked, but it was hard work. We think a more powerful tool (such as KEE, S.1, ART, or Knowledge Craft) would have been more appropriate.

For further information about Cogito, phone Professor Bahill at (602) 621-6561.

Acknowledgement

We thank Phil Kaslo and Bill Ganoe for testing our expert system.

References

[1] P. N. Harris, *COGITO: An expert system that gives advice for making and installing UNIX 4.2BSD on VAX-11 series computers.* University of Arizona: master's thesis, 1986.

[2] *UNIX Systems Manager's Manual.* El Cerrito: USENIX Association, 1985.

[3] *UNIX Programmer's Manual Reference Guide.* El Cerrito: USENIX Association, 1985.