An expert system for drafting legal documents

by JAMES SPROWL

IIT/Chicago Kent College of Law

Chicago, Illinois

and

PERIYASAMY BALASUBRAMANIAN, TAIZOON CHINWALLA, MARTHA EVENS, and HENRIETTE KLAYANS

Illinois Institute of Technology

Chicago, Illinois

ABSTRACT

ABF is an expert system that assists attorneys in designing legal documents. The system starts by extracting from a library of legal forms a skeleton template that has embedded within it programming constructs such as conditionals and loops, references to other texts, and variables, which are later replaced by client-specific information in the course of a legal interview. Alternative passages are included or excluded dynamically as the interpreter encounters loops and conditionals. As the system analyzes the document, when it discovers that information is missing, it first looks in the client data file, then it tries to compute it, calling a subprogram if necessary. If all else fails, it generates an English question asking the user for the missing data. The user can stop the interpreter at any time, edit the draft, and reinitiate processing at any point. ABF has been implemented in PASCAL and runs on an IBM PC.
INTRODUCTION

For the past nine years, the American Bar Foundation (ABF) has been conducting research into automating some of the more routine aspects of practicing law, such as client interviewing and automated document assembly. The goal of this research has been the development of a computer system that can be set up by an attorney expert to assist other attorneys and their assistants in will, trust, and complaint drafting, tax return preparation, and other such tasks. A prototype system, called the ABF system, was developed on a large CDC computer at Northwestern University and was tested in the law student practice clinic at Illinois Institute of Technology's Chicago-Kent College of Law. The clinic staff attorneys developed a number of useful document libraries for generating wills, trusts, divorce petitions and decrees, guardianship petitions, and real estate closing agreements. The will and trust libraries developed by Robert Seibel (now at the University of Maine Law School) were the most successful. More than 500 wills were generated by the clinic for senior citizens living in Chicago. Later, the ABF system prototype was converted to an all-FORTRAN system by Professor Charles Saxon at Eastern Michigan University and was installed on an Amdahl computer at the University of Michigan, where it is used by students of law professor Layman Allen. The FORTRAN ABF system was also installed at the Newcastle (upon Tyne) Polytechnic School of Law, where it is used by students of law professor Michael Heather.

The recent emergence of powerful microcomputers has now made it possible to make this prototype system available to many law schools and law offices. After due consideration, we have decided to reimplement the prototype in PASCAL, using the UCSD P-System because of its well-known portability and widespread current availability in law schools and law offices. IBM and SAGE microcomputers have been used for the development work.

The microcomputer ABF prototype will be the first fully integrated prototype. It will include its own document library system and full-screen text editor and will be entirely self-sufficient.

In this paper we describe the new prototype and the ABF programming language it implements.

THE ABF LANGUAGE

Since attorneys are accustomed to working with form legal documents and statutes, the ABF system is designed to accept document descriptions that resemble closely the documents that can be found in an attorney's book of legal forms. The computational procedures that control document assembly are drafted in a language that causes these procedures to resemble statutes. The attorney is thus given the feeling that he or she is feeding form documents and statutes into a computer, which then writes its own client interviews and document drafts. But in reality the document and procedure drafting languages are simply subsets of a new general-purpose programming language, which we have named the ABF language.

Drafting Documents

In a typical legal document, variable information must be inserted in the text at many different points. For example, a typical divorce document might begin: "This matter was heard upon the verified petition of..." followed by the name of the petitioner, which naturally varies from client to client. When such a document is drafted in the ABF language, variable names are enclosed in square brackets and inserted at such points. Accordingly, an ABF model divorce document might begin as follows:

This matter was heard upon the verified petition of [the name of the petitioner] for dissolution of marriage....

When assembling this document for a specific client, the ABF system later scans the model document, finds this bracketed variable, and transforms the variable name into a question by putting "What is" in front of it and "?" after it. So the system generates the question:

What is the name of the petitioner?

The name supplied by the user is then inserted into the document in place of the variable name enclosed in square brackets wherever that variable name appears. It is also placed in a client data file. In this manner a unique client data file is created for each client and may be used to control the assembly of other documents without having to ask the same questions over again.

The name of the petitioner appears in a number of different places within the divorce library. To save typing, the author of the document may define a short abbreviation for the variable name and use the abbreviation instead of the full variable name. If the document designer chooses the abbreviation "petname" for "the name of the petitioner," that fact can be communicated to the system by simply slipping it into the document like this:

This matter was heard upon the verified petition of [petname: the name of the petitioner] for...
Thereafter, the short form [petname] may be used. The system expands all such short names into full variable names automatically.

For logical (true-false) variables, the “What is” question is inappropriate. If the user capitalizes a helper verb in such a variable’s name, the system forms a question beginning with that helper verb. Thus, the variable name “stp fld: a stipulation HAS been filed” is converted into the question: “Has a stipulation been filed?”

Sometimes the document designer wishes to insert not just the value of a simple variable but an entire document. A document name is actually a variable name; the value associated with this new kind of variable name is the text of the document itself. To insert the text of one document into the middle of another document, the author simply inserts the name of the one document, enclosed in square brackets, into the other document.

The insertion of optional passages is controlled using an IF...END IF construction, and the insertion of repetitive passages is controlled using a REPEAT...END REPEAT construction. Both of these constructions are explained below.

Optional and Alternative Passages

The ABF language includes a full IF statement that permits optional and alternative passages to be selected. The full IF statement has the form

IF Boolean expression INSERT document 1
OTHERWISE
document 2
END IF

The Boolean expression is a logical proposition or expression that the processor evaluates to get a value of TRUE or FALSE. The following are examples of such propositions and expressions:

the client’s income IS GREATER THAN $10,000
the testator IS NOT married

If the proposition or expression is true, then Document 1 is processed and Document 2 is skipped. If it is false, then Document 1 is skipped. If the optional “OTHERWISE Document 2” part of the IF statement is omitted, Document 1 is processed if the Boolean expression is true and is skipped if the Boolean expression is false.

There is also an expanded IF statement that may be used to select one of several alternatives based upon the evaluation of several conditions:

IF Boolean expression 1 INSERT
document 1
OR IF Boolean expression 2 INSERT
document 2
OR IF Boolean expression 3 INSERT
document 3
END IF

A Boolean expression is made up of simple conditions connected by ANDs and ORs. The NOT operation is used within one or more of the simple conditions. There are two kinds of simple conditions: logical expressions and propositions. A logical expression is similar to the logical or relational expressions found in many well-known programming languages: two algebraic expressions of the same type separated by a relational operator such as GREATER THAN, EQUAL, or IS NOT GREATER THAN. The proposition is a construction not usually found in programming languages, so it requires more explanation. A proposition, like a logical expression, has a value of TRUE or FALSE, but its value is determined directly from a user response rather than from a calculation. For example, the proposition “the color of the sky IS blue” is evaluated by asking the question, “Is the color of the sky blue?” to which the user must respond either “yes” or “no.” A proposition may also be stated negatively, e.g., “the color of the sky IS NOT blue.” In this case, the system also asks “Is the color of the sky blue?” but now the value of the proposition is set TRUE for the “no” response rather than the “yes” response.

The proposition “the color of the sky IS blue” could be replaced by the logical expression “the color of the sky EQUALS <blue>.” This expression is evaluated by comparing the value of the variable “the color of the sky” to the string constant “blue.” If the variable is undefined, the system asks “What is the color of the sky?” to which the user may respond “blue” or “red” or any other legal value.

Repetitive passages—that is, passages that are to be duplicated and inserted into a document repeatedly—are bracketed by the commands REPEAT and END REPEAT. Within such passages, array variables are simply marked by a number-sign prefix to distinguish them from non-array variables, and the indexing of array variables is automatic. An embedded WHILE...UNTIL...EXIT statement controls termination of the repetitive insertion process, as in most standard programming languages. For example:

This contract covers the following states:

REPEAT

[the name of a state]
UNTIL that IS the last state
END REPEAT

This simple example, when processed, causes the questions to be asked repeatedly until the latter question is answered “no.” In this manner any number of state names may be added to the list.

Much more complicated examples are possible, since REPEAT...END REPEAT passages may be nested to any desired depth.
Drafting Computational Procedures

There are times when it is possible to calculate the value of a variable from the values of other variables. The document designer may then decide to write an ABF procedure to perform this calculation. In our divorce example, the system can calculate the personal pronoun *he or she* for the respondent once the user has supplied the personal pronoun *he or she* for the petitioner, since the petitioner and respondent are of opposite sex. The procedure to calculate the value of `respron`: the personal pronoun of the respondent might look like this:

```plaintext
IF petpron: the personal pronoun of the petitioner
    EQUALS <she>
    LET respron: the personal pronoun of the respondent
        = <he>
    OTHERWISE
    LET respron = <she>
```

The syntax for procedures was deliberately chosen to make it possible to write procedures that resemble statutes very closely.

**INTERNAL OPERATIONS OF THE ABF SYSTEM**

To simplify the task of building a complex information-gathering and document assembly system, the ABF system permits one to begin by simply drafting form documents that define the system output. By extracting variable names from these documents and converting them into questions, the ABF system is able to ask for the data it needs to assemble the documents. Whenever the system asks for data, the system designer may alter the way the question is asked or supply the system with a procedure containing instructions on how the data are to be computed from other data values. In this manner, the system is actually redesigned from the top down while it is running.

The articulation of the main components of the ABF system necessary to give the user this freedom can be seen in Figure 1. The user of the ABF system starts out looking at the command screen provided by the command screen manager. The command screen is split three ways. The top of the screen contains a list of the commands available in the current context. A window of text may appear next. Below the text is a snapshot of the top of a historical command list containing the names of the most recently executed commands. Prompts for new commands and questions formulated by the system appear at the bottom of the screen, where the user types in new commands and the answers to questions.

When the user is ready for a client interview, he/she signals the system by a command such as "PROCESS draft will OF John Smith." In response, the Librarian (the ABF file handler) locates the model document called "will"; and if necessary, the Compressor is called to put the document into compressed internal form. When the Compressor finds a new variable name, it inserts this name into the Variable Name Table and the System Identifier Table and replaces it in the document by a number indicating its offset in the System Identifier Table. Much of the document is just straight text—

boilerplate text, as lawyers call it. This text is inserted in string form into the Boilerplate Table and is represented in the compressed document by a pair of integers indicating the table offsets of the first and the last characters. Operators are replaced by the special operator tokens encoding operator type and precedence. The compressed version of the document is typically much shorter than the raw version. It contains only operator tokens, system identifier table offsets, and boilerplate offsets. This compressed document serves as input to the Interpreter.

When the Interpreter processes a document, it must execute the operators in turn; but before it can execute an operator, it needs to know the values of the arguments. It calls the Seeker to find these values, as will be explained below.

Once the Interpreter has finished with a document, it calls the Decompressor to reassemble it into text form. If the Decompressor finds that some part of the document could not be finished because of missing data, it calls upon a Decompiler to pick up the pieces and put them together.

When the draft document is in satisfactory shape, the user calls the Formatter to put it into final form. The user can then display or print this final form or even edit it further.

The user can also edit an existing document or create a new one. The system sets up the document for editing on the full screen and calls the Screen Editor. This same Screen Editor is also used on a partial screen whenever the user enters input, whether it is a command or the answer to a question.

The ABF System recognizes four elementary data types: numbers, dollar amounts, text strings, and logical values. Array variables are also permitted. The system also recognizes three complex data structures: documents, procedures, and replacement questions.

**Finding Values for Variables**

When the Seeker is asked by the Interpreter to find the value of a variable, it first checks to see if the variable has already been defined. If so, the value of the variable is retrieved from the client data file. If the variable has not been
defined, the Seeker checks to see whether a procedure exists that can be executed to compute the value of the variable. If such a procedure is found, the Seeker calls the Interpreter recursively to execute the procedure. If no procedure is found, the Seeker next looks for a replacement question defined by the document designer. If it finds one, it calls upon the Interpreter to assemble the replacement question and then prompts the user with that question. Otherwise, the Seeker must build-form a default question by appending "What is" or "Is it true that" to the variable's name (or by shifting a capitalized helper verb to the start of the variable's name) and then prompt the user with the question so formed.

The user has three different options at this point:

1. Answer the question.
2. Refuse to answer the question by typing "!!"
3. Replace the question with a new replacement question.
4. Replace the question with a new procedure that computes the variable.

If the user answers the question, the system accepts the answer and replaces all occurrences of that variable with the value supplied as long as the data type is correct. The ABF system decides the type of a variable, not from a formal declaration, but by remembering the type of the value associated with a variable the first time a document is processed.

If the user refuses to answer a question, the Seeker marks the variable as never-to-be-defined. This causes the system to leave the bracketed variable name in the document and to refrain from bothering the user with questions about it again during the processing of the draft.

The user may also supply a Replacement Question to the system. The Replacement Question is stored in the system library, and any subsequent reference to this same variable when it is undefined involves the Replacement Question. Replacement Questions may contain bracketed variables and optional passages.

If the user feels that the necessary information can be computed from facts already known to the system, then he or she may decide to supply the system with a procedure to compute its value. (The code given above to calculate the personal allowance is a trivial example of such a procedure.) Unlike procedures in conventional programming languages, an ABF procedure is not given a name. Instead, it is referenced by the names of the variables it computes.

From the user's point of view, there is no program. The user sees only a library containing a collection of documents, procedures, replacement questions, and client data files arranged by the system in neat document form to suit the user's convenience. The statute-like procedures appear to govern the automated assembly of the legal documents.

DESIGNING A SYSTEM TO OPERATE WITHOUT CRUCIAL ITEMS OF DATA

One cannot design an automated law office system to anticipate all possible client circumstances. Not only is the range of possible client circumstances entirely open-ended, but a system that even attempts to anticipate all possible circumstances produces an unbearably long interview. Such a system will frequently ask questions that are irrelevant or inappropriate to the needs of any particular client. Sometimes questions cannot be answered because answers are simply not available. It is essential that legal practice systems be capable of generating usable documents, even when data are not supplied to the system (either because the data are not available or because the questions asked are irrelevant or inappropriate).

After much thought and discussion, we decided to design the system so that an exclamation mark typed in answer to any question signals to the system that the user does not wish to answer the question. In response, the system sets the corresponding variable into a special never-to-be-defined state. The system then proceeds to execute procedures and assemble documents as best it can without the values of the variables the user has elected not to supply.

The system proceeds as follows: If the variable is one that is simply inserted into the text of the document at various points, the system leaves the bracketed name of the variable in the document text and does not replace it with a value. The finished document is thus partly finished—it still contains bracketed variables corresponding to the unanswered questions. These may be edited out manually, or the document may be reprocessed at a later time.

If the variable is one that appears in the preamble of an IF...END IF optional passage, the system normally cannot determine whether to insert or exclude the passage. Accordingly, the optional passage is simply left in the document preceded by the IF command and followed by the END IF command. Insofar as the user refuses to answer a question essential to determining how many copies of such a passage are to be inserted into a document.

From the viewpoint of the computer scientist, the system effectively decompiles all passages that cannot be processed because the user refuses to supply the necessary answers to questions. The decompiled versions of documents and procedures may be simplified in comparison to the originals to the extent that data were available to enable mathematical and logical expressions to be partially evaluated and simplified. For example, by supplying some answers and withholding others, one can cause the system to simplify a complex set of tax code provisions into a much simpler set of provisions that illustrate what legal effect the answers withheld will have upon a particular client. Thus, the expression

\[
\text{LET tblinc: the taxable income = ginc: the gross income}
\]
\[
- \text{adj: the adjustments to gross income} - \text{ded: the deductions from gross income}
\]

Might produce the interview

What is the gross income?
$10,000

What is the adjustments to gross income?
$3,750

What is the deductions from gross income?
!
Since the user did not answer the final question, the expression could not be fully evaluated but was simplified to:

\[
\text{LET } \text{txblinc}: \text{ the taxable income } = \$6,250 - \text{ded: the deductions from gross income}
\]

Some very interesting and not fully explored problems arise when one attempts to execute programs with less than a complete set of data in this manner. Of particular interest is the case where a passage in a document that could not be fully processed contains a command to alter a variable that has already been assigned a value. For example, consider the following document:

```
(text)
[the name of the contractor]
(more text)
IF a second contract with a different contractor is desired
INSERT
(text)
LET the name of the contractor = the new contractor's name
(text)
[the name of the contractor]
(more text)
END IF
```

When this document is processed, the following interview might be generated:

**What is the name of the contractor?**
George L. Burroughs

Is a second contract with a different contractor desired?  
!

Here, the user does not yet know whether a second contract is desired, so the user refuses to answer the question. Accordingly, the language IF...INSERT...END IF is left in the document. But the system must scan this text and discover that a new value will be assigned to the variable "the name of the contractor" if this optional passage is ever selected. The system must therefore set the variable "the name of the contractor" into an undefined state and refrain from inserting its value into the remainder of the document.

The above example illustrates why the system must scan all document portions, procedures, and new questions that cannot be fully processed because the user has refused to answer one or more questions. The scanning must cover every possible algorithmic path. For example, if an IF...INSERT...OTHERWISE INSERT...END IF clause cannot be processed, the system must scan both the IF part and the OTH-

ERWISE part, searching for commands that redefine variables. Had the same clause been processed fully, the system would have processed only one of these two parts, discarding the other. Any defined variable that is redefined must be set into a never-to-be-defined state to avoid the possibility of the wrong value being inserted into a document or used in a computation.

This whole field of processing without a complete set of data, decompiling, and scanning unprocessed portions for commands that redefine variables is a field that needs to be much more fully explored by the computer science community to ensure that this kind of processing is given a sound theoretical basis.

**SUMMARY**

The conversion of the ABF System to run in PASCAL in the UCSD P-System on an IBM Personal Computer has involved redesign of the language to take advantage of full-screen editing and recursion. The result integrates word processing and artificial-intelligence techniques to provide a novel system for writing legal documents. ABF forms English questions when it cannot find values for variables. It provides a rich collection of conditional operators to control the insertion of alternative passages in the text, and it enables repetitive passages to be created without explicit subscripts for the variables. The ABF System is unique in its ability to function without crucial items of data, decompiling expressions where they cannot be fully processed. It is also unique in its ability to generate a client data file that may be used to control the assembly of other documents. The System encourages both top-down and bottom-up design by causing any portion of a system to be fully operative without dummy subroutines and by permitting new procedures and questions to be defined at run time. And by generating questions directly from variable names, the system gives the programmer positive incentives to provide long, meaningful variable names.

**REFERENCES**


From the collection of the Computer History Museum (www.computerhistory.org)
Computer communications

Neal Laurance, Track Chair

The Computer Communications track, NCC '84, is composed of six sessions covering a range of communications issues from satellite links to downloading programmable controllers in manufacturing plants. In addition to the sessions specifically on communications, the track includes a session on the use of computers in manufacturing. Besides this track, one can find elements of computer communications developments in the tracks entitled The Automated Office, Information Processing Management, Personal Computers, Educational and Societal Issues, Software, and Database Management. Indeed, as distributed processing becomes the norm, the problems and promises of computer communication systems become an issue that pervades the whole field of computing today.

The past year has been one of enormous advances. In the area of local area networks, the IEEE 802 committee has completed work on two standards, and it now seems possible to connect many different vendors' products to the same local area network. But at the same time the divestiture of AT&T from the regional Bell Operating Companies has opened up whole new areas of communications possibilities. The question now is whether the advent of digital telephones and computer communication technology will make local area networks obsolete before they really get started.

The two “Multi-Vendor Networks” sessions on Tuesday morning deal with the results of some pilot programs on an implementation of the draft ISO protocols for Levels 3 and 4 of the OSI reference model. During the past year the National Bureau of Standards (NBS) has been conducting workshops on this implementation with participants from both the computer industry and communications users. The workshops have resulted in two pilot implementations, each involving several computer vendors. The first pilot program, carried out at the NBS in Gaithersburg, uses for its communication medium an IEEE 802.3 CSMA/CD coaxial cable. It has the office environment as a general background assumption. The second pilot program was led by General Motors Corporation as part of their Manufacturing Automation Project (MAP). It uses the same ISO transport layer for its demonstration, but it is based on the IEEE 802.4 token bus system. It assumes a manufacturing plant floor as an environment and includes as one of its elements communications to and from programmable controllers. Each pilot program has about six active computer equipment vendors as participants, and NBS and GM are demonstrating the multivendor method in the conference exhibit area. The two sessions consist of panel discussions by participants in each pilot, outlining the progress in this area to date and anticipating the future developments in an ISO communication network.

The session titled “Videotex” brings together a panel of experts to discuss the latest developments in this very interesting technology. Last year we heard reports of highly successful field trials of videotex and teletext in limited geographical areas. This was to be the year of the first commercial offerings in what promises to be the next mass communication market. What have been the experiences in this area? Is videotex on track with its promise of two-way video communication in the nation’s homes, or have the effects of the recent economic recession delayed the timetable?

Despite the successful efforts of IEEE 802 committee to bring a degree of standardization to the local area network arena, the number of different offerings in the local area network field continues to grow. In part this growth is fueled by the ubiquitous personal computer and its need to communicate at a cost commensurate with the low cost of the computer itself. The session entitled “Update on Local Area Networks” will bring together a panel to discuss the latest developments in this area and help chart a path to the future.
Of special interest will be the outlook for the yet-to-be­
announced IBM entry into the local area network field. And
how will AT&T’s participation in this market change the
course of future developments?

Of special interest these days is the linking of manufac­
turing plant floor information to the corporate data structure.
Traditionally, manufacturing plants used computers for their
operations independently from the computer systems used in
corporate reporting and analysis. That situation is slowly
changing, with the result that office-level systems are finding
their way into manufacturing environments. The “Computer
Integrated Automation” session focuses on several examples
of the use of personal computers in manufacturing plants and
explores the effect of automation and computer systems on
manufacturing productivity.

“Computer Systems and Devices” brings together four pa­
pers on communication. One paper explores the concept of a
work station tied to remote computer facilities via satellite
links, as might be used in the travel industry. A second dis­
cusses a very low-cost implementation of a shared medium
network, using only RS 232 hardware and twisted pair. The
fundamental problem of relating the design of a telecommuni­
cations system to a corporate business strategy is discussed in
the third. The last treats a different kind of communication:
conveying to the driver of an automobile information about
the route to be followed to his/her destination.

The next session in this track assembles a panel to discuss
one of the fascinating events of the past year, the divestiture
of AT&T. Far from being a smooth transition, the repercus­
sions of the breakup are still being felt. Understandably, news
accounts have focused on effects on the general public. The
effects on business communications have been at least as sig­
nificant, and the biggest effect of all may prove to be the
changes in data communication. Certainly, at this point, the
communications market does not look the way the experts
predicted as recently as 12 months ago. What are the con­
tinued long-term effects likely to be? Will the imposition of
usage fees signal a large scale move to digital PBXs? What
effect will that have on the development of local area net­
works? This session will attempt to answer these and other
questions.

The final session of this track, “Integrated Networks,” as­
sembles a panel of experts to discuss the issues and potential
benefits of tying diverse networks together. Communications
gateways now make possible the connection of local area
networks, wide area networks, and data processing networks
into an integrated communications system. These gateways
offer exciting possibilities for information processing and
transmission.