ACQUISITION OF PROCEDURAL KNOWLEDGE FROM DOMAIN EXPERTS

Peter Friedland

Computer Science Department
Stanford University
Stanford, Ca. 94305

ABSTRACT

Procedural knowledge forms an important part of expertise. This paper describes a system which allows domain experts to themselves enter procedural Knowledge into a knowledge base. The system, a stylized form of scientific English embodied within the Unit System for knowledge acquisition and representation, has been used successfully within the domain of molecular biology.

1. Introduction

Expert knowledge comes in two forms, declarative and procedural. Acquisition and representation of diverse forms of declarative knowledge seem reasonably well understood. However, a significant component of expertise takes a procedural form, ranging from low-level rules for data manipulaton to abstract strategies for problem solving. The problem of acquiring and representing procedural knowledge is still a major research endeavor. This paper will illustrate one solution to the problem being currently used extensively in the domain of molecular biology by the MOLGEN project at Stanford University". The primary goal of the MOLGEN proiect is to build knowledge based systems to provide expert assistance to the design of both synthetic and analytic experiments in molecular biology.

The paper will first describe the variety of procedural knowledge within the chosen domain. It will present arguments concerning why direct entry of procedural knowledge by domain experts is both desirable and necessary. The specific solution used by MOLGEN, a rule language of "genetic English" within the frame-based Unit System, will be detailed. Finally, a summary will be given of the experience of domain experts using the rule language over a period of two years and of the lessons applicable to many domains learned during that time. All examples in the paper are taken directly from the knowledge bases of various domain experts, specifically, those of Professor Laurence Kedes, Professor Douglas. Brutlag, and Dr Rene' Bach of Stanford University, and Professor John Sninsky of Albert Einstein College of Medicine.

'This work has been supported by the National Science Foundation, grant MCS7802777. Computer resources have been provided by the SUMEXAIM facility, NIH grant RR0078507. MOLGEN is a registered trademark of Stanford University

2. Types of Procedural Knowledge

There appear to be four major classes of procedural knowledge that are of importance in building expert systems within the domain of molecular biology. On the basis of extensive interviews of laboratory scientists, the classes are: data manipulation procedures, simulation procedures, selection heuristics, and experiment design strategies. These will be described and illustrated within this section.

Data manipulation procedures analyze existing information in the knowledge base in order to extend or correct the knowledge base. In large part they comprise rules which ensure completeness and consistency of the knowledge base As an example, take the case of describing a new DNA structure to a knowledge base There are dozens of items that might be known about the structure length, topology, nucleotide sequence, cutting maps, etc. Many of the properties can be calculated from one another; for example, both length and restriction cutting maps can be derived from nucleotide sequence Experts in the MOLGEN domain have described the procedures for making these inferences, and the procedures are used whenever new items are entered in the knowledge base.

Simulation procedures model the actions of various genetic operations such as denaturing a DNA molecule These contain the information needed to alter the representation of objects when the operator is hypothetically applied For example, one of the changes made to the representation of a DNA structure when a denaturing method is applied is to the property of "strandedness" Structures which are initially double stranded become single stranded.

Selection heuristics are used for choosing among competing alternative operations during the process of experiment design. A typical heuristic for choosing a single-strand specific cutting enzyme might be: "find all enzymes which selectively cut the specified substrate and then, from the ones which are available in your laboratory, pick the one with the largest turnover number."

Experiment design strategies are the knowledge of problem-solving methods. This knowledge can be at many levels of abstraction and domain-independence, from parameterized plans for doing sequencing experiments, to general strategies for detecting interesting features of molecules, to global strategies for experiment design, such as progressive refinement (embodied as a procedure).

3. Knowledge Acquisition Directly from Domain Experts

Given that it is desirable for a knowledge base to contain the types of knowledge discussed in the previous section, the problem remains of how best to acquire that knowledge. A time honored method has been for a computer scientist intermediary, the "knowledge engineer", to be instructed in the domain by experts, and then to translate that domain-specific knowledge into the particular representation system chosen. In contrast, the MOLGEN project has emphasized the approach of having the domain-experts themselves build the knowledge bases. The work of the computer scientists on the project has been to provide the appropriate representation tools for both the declarative and the procedural primitives of the domain.

The reasons for providing for knowledge entry by the domain experts are as follows. First, both accuracy and completeness of the knowledge base suffer when domain knowledge passes through the filter of a computer scientist, who is not an expert in the chosen domain. Much of the knowledge is complex and subtle, and its purpose may not be immediately apparent to the non-expert. Second, a large knowledge base may be built more quickly when non experts do not have to be intimately involved in describing each object and rule in the knowledge base. Finally, in building a knowledge base for a program to be used by other experts in the field, an element of trust, as embodied in the name of a known authority, is essential. Professor Kedes' description of cloning strategies is more likely to be used and trusted by other molecular biologists than is Peter Friedland's translation of Professor Kedes' strategies.

There are, of course, drawbacks and prerequisites to allowing the experts themselves to describe their domain in a knowledge base. Many computer scientists working in collaborative efforts with domain experts have noticed that relatively few experts are initially equipped with the ability to logically formalize the rules of their domain. This has been the experience of the MYCIN project [1], the Prospector project [2], and the author. Thus, the computer scientist has served a useful function by providing a logical organization to the domain experts' rules. Of course, scientists gradually become better formalizers as their experience with knowledge-based systems grows. All of the molecular biologist collaborators have expressed satisfaction that their ability to formalize in the laboratory has grown during their tenure on the project. However, it seems that the real trick is in making the domain expert comfortable in his new mode of expression, and that is the major topic of this paper.

4. A Method for Describing Procedural Knowledge--the Rule Language

The Unit System (described in [3], [4], and [5]) has provided a convenient representation system for the description of declarative information. Knowledge is organized as a hierarchial collection of frames or "units" with each unit containing many "slots", each holding an individual piece of knowledge. The Unit System has many user-oriented features that smooth the knowledge acquisition process for declarative knowledge. The capabilities of the Unit System have been extended to allow for the direct description by the experts of procedural knowledge.

Clearly, one way for domain experts to enter procedural knowledge would be for them to learn the underlying programming language of the knowledge base system, in this case INTERLISP. This would not be a very good solution: first, because of the large time investment required to teach domain experts to be proficient programmers, and second, because the underlying principle of making the knowledge transparent to any domain expert would be violated. Being able to access and understand the procedural strategies of the expert provider of such strategies is just as important to another expert user as being able to access and understand a list of any particular parametric property.

The solution we have chosen for allowing all types of procedural knowledge to be described is to provide a specially tailored language of "genetic English" as a simplified "programming language." This language allows domain experts to access and manipulate the various units and slots within their knowledge bases--the "variables" of their programming language. Much of the language is domain independent, as will be seen below, but its architecture allows parts to be specially tailored to the particular domain.

The idea of choosing a stylized or stereotypical language for procedural expertise has its roots in the MYCIN project [1]. There, a formalized language for writing production rules was used to enter expert knowledge about infectious disease diagnosis. This work was extended by the Teiresias system of Davis [6]. In parallel with the work described in this paper, both the GUIDON work of Clancey[7] in the representation of tutorial expertise and the CENTAUR work of Aikins[8] in the explicit encoding of the methods for control of inference have experimented with various forms of stylized procedural knowledge description within a domain specific knowledge base.

4.1. Overall Design

The language for procedural knowledge is implemented as the Rule Language within the Unit System. Interaction with the Rule Language is by the same basic mechanisms as is used for the different forms of declarative knowledge. The user "talks" to a special editor which provides facilities for entry and correcton of individual statements within the language. (It should be noted that the word "rule" in the context of the Rule Language is used to refer to a single statement in the language, rather than necessarily to a conventional condition-action production rule.)

Rules are parsed and interpreted in two discrete steps. At rule entry time, syntax is checked and rules are stored in an internal format of procedures and parameters. At interpretation time, binding for the parameters is found among the current context, and the action of the procedure is taken. This allows rules to be written about entities which don't yet exist in the knowledge base, and allows rules to be general to more than a single context.

Currently, rule statements may be one of the following forms:

1. an action-something which creates, deletes, or modifies a slot or unit, or which affects interpretation of the set of rule statements. Actions may be joined by the word "AND".

2. a conditional statement--of the form "IF boolean-expression THEN action ELSE action" where the ELSE part of the statement is ootional.

3. an iterative statement of the form "FOR iteration description boolean expression" or "FOR iteration-description action".

4. a label-simply a reference point indicated by the word "LABEL" followed by a single word.

Context of the rules is set at rule execution time. In a practical sense, context means a description of where the rule interpretation procedures are to look for the variables which were left unbound at parsing time. These variables may be the names of slots, units, columns in tables, or special reserved words which refer to dynamic properties during rule interpretation (see below). The rule interpretation system keeps track of units in the current context- these include the unit from which a rule is run (where the rule slot is stored), units implicitly mentioned in the rule statements (while iterating over a group of units, for example), and units which are explicitly added to the context by the user.

4.2. Actions

Actions may be classed as domain independent or specific to the domain of molecular biology. The following domain-independent actions are currently implemented: (for each action, a definition and an example or two will be given)

ADD-adds new elements to the end of a list. A00 CUTTING-SITE TO CUTTERS-LIST

ADD CONTEXT adds new units to the front of the context list. ADD CONTEXT RESTRICTION-ENZYMES

APPLY-causes interpretation of another rule slot (a way of "subroutining" or writing metarules). APPLY MAPPING-RULES TO SV40

CLEAR-"clears" a slot by restoring its value to its inherited value (or to NIL if the slot is first defined in the unit). CLEAR COOING TABLE

COMMENT a general printing verb. As with all verbs, its arguments are bound to slots or table columns if possible, otherwise taken as strings or integers. COMMENT A RESTRICTION MAP HAS BEEN CREATED. COMMENT STRUCTURE NAME HAS SITE-NUMBER SITE-NAME SITES

COMPACT -eliminates redundancies in lists, tables, and maps. COMPACT MULTI-CUTTERS

CREATE SLOT creates a new slot. CREATE SLOT CODING MAP IN PBR322 UNIT WITH DATATYPE MAP AND ROLE R

CREATE UNIT -creates a new unit. CREATE UNIT SV40 WITH PARENT DNA STRUCTURES

DELETE-CONTEXT-removes a unit from the current context. DELETE CONTEXT RESTRICTION-ENZYMES

DELETE UNIT deletes a unit. DELETE UNIT SV40

ENTER-adds a new row to a table. ENTER IN FRAGMENT TABLE ENZYME NAME LEFTEND RIGHTEND SIZE

EXIT-returns from interpreting the current rule slot.

FIND a general purpose pattern matcher (with a pattern language adapted from the UNIX pattern matcher). FIND AG(AT)' IN NUCLEOTIDE SEQUENCE INTO PATTERN-LOCATIONS

SET the basic verb for setting and changing the values of slots Slots can be set to constants, to the values of other slots, or to the values of constants and other slots modified by various unary or binary operators like PLUS, DIVIDED, INTERSECTION, or GET ELEMENT. SET ENZYME TO SAL1. SET TOTAL TURNS TO SUPERHELICITY DIVIDED BY WINDING NUMBER. SET RESTRICTION SITE TO GET ELEMENT SITENUMBER OF SITE LIST.

SET DATATYPE -sets the datatype of a list. SET DATATYPE OF CUTTING LOCATIONS LIST TO INTEGER

There are also several verbs specific to the molecular biology domain. These verbs, ENTERREGION, ENTERSITE, EXCISE, SET LENGTH, SET TOPOLOGY, INSERT, JOIN. LIGATE. SPLICE. TRANSCRIBE, and WRITE SEQUENCE, allow special manipulation of the MAP and SEQUENCE datatypes, used to store information relating to the base sequence of nucleic acid molecules.

4.3. Boolean Expressions

Boolean expressions consist of one or more relations, possibly modified by NOT, joined optionally by ANDs or ORs. Parentheses are allowed to clarify the expressions. Simple domain-independent binary relations equals, greater than, less than, list inclusion and the like are provided as well as a special relation for finding patterns in the MAP and SEQUENCE datatypes. Several reserved words are provided for boolean expressions, e.g. DEFINED to check whether a slot of a given name exists in the current context and FILLED to check whether a slot has been given a value in the current context.

Some sample expressions are:

```
IF RESTRICTION-MAP IS FILLED
IF FRAGSIZE < DISTANCE
IF 3SPLICE IS NOT EQUAL TO 4SPLICE
IF CUTTING-LIST INCLUDES ENZYME
```

4.4. Iterative Statements

Iterative statements allow a user to iterate over groups of units, rows of a table, or elements of a list. If the iteration is to be over a group of units, then the parent of the desired group (with optional additional restrictions on the values of slots within the units) is specified. Context is automatically updated during each step of the iteration. During iteration over a table, individual entries in the table may be retrieved or altered by reference to the column names. During iteration over a list, the words ELEMENT, NEXT ELEMENT, and PREVIOUS ELEMENT allow the obvious reference. For example:

```
FOR ALL DNA-STRUCTURES
FOR ALL PHAGES WITH LENGTH > 500 AND
    STRANDEDNESS EQUAL DOUBLE
FOR EACH ROW IN RESTRICTION-ENZYMES TABLE
FOR EACH ELEMENT OF CUTTERS LIST
```

4.5. Facilities for Making Rule-Writing Easier

As the power of the Rule Language has increased over the last several years, so too have the facilities for "programming" in it. Users can make free use of common synonyms and "skip" words to make the rules more readable Rules can be entered directly by means of a rule editor which allows insertion, deletion, and replacement of rule statements. Users can also access their favorite text editor (TV, EMACS, etc.) to write and modify rule slots. As was previously stated, variable binding does not occur until rule interpretation time, so that the slots upon which a given set of rules operates do not have to exist when the rules are written.

4.6. An Example Rule Procedure

The following example shows a set of rules recently written by Professor Brutlag. The purpose of the set of rules is to create a table of fragment sizes when a piece of DNA is cut by several restriction enzymes. The rules test to see if sufficient knowledge has been entered into the knowledge base to create the table, and manipulate information that may exist in several possible forms.

The point of the example is not that the Rule Language allows programs to be written which otherwise could not be written, but that it allows non programmer domain experts to themselves describe procedures and experiment with manipulating and formatting the knowledge of their domain. It is certainly true that in a real sense the domain experts are becoming programmers in using the rule language, but they are using a language which was designed in response to their requests for a formalized subset of "genetic English." It is also a language which experience has shown is readable by non-programmers almost immediately and writeable (using the available interactive assistance) with only a few minutes basic introduction.

It will be clear that the sample procedure is neither sophisticated nor particularly "clean" in programming style. In fact, some readers may note with alarm the use of go to statements and labels within this example. As was stated above, the design of the language has come primarily from the requests of users. They currently find this type of syntax to be more understandable than the stylistically preferable syntax of structured programming, but it is hoped that a means of improving the control mechanisms of the language can be found. The essential point is that Professor Brutlag was able to adquately embody the data manipulation procedure he desired after only a very basic introduction to the Rule Language.

The set of rules shown here are stored in a slot named FRAGMENT-RULES in a unit named RESTRICTION RULES in Professor Brutlag's knowledge base, DNALAB. All of the rule statements are shown in upper case; some of the statements have been annotated in italics.

```
SET FRAGMENT-UNIT TO INPUT ENTER THE
DNA-STRUCTURE FOR WHICH YOU WANT FRAGMENT
DATA AND ADD-CONTEXT FRAGMENT-UNIT
```
This rule gets from the user the name of a structure-containing unit and adds that unit to the current rule interpretation context

```
IF RESTRICTION-MAP IS FILLED THEN GO TO LABEL L1

APPLY SUBMAP

IF RESTRICTION-MAP IS NOT FILLED THEN STOP
```
The preceding three rule statements check to see if a restriction map has already been made for the given DNA structure. If it has not, then another set of rules is called to make the map.

```
LABEL L1

CLEAR FRAGMENT

COMMENT
ENTER RESTRICTION ENZYMES TO CLEAVE DNA STRUCTURE
(ONE PER LINE END WITH CR)

LABEL L2

SET ENZYME TO INPUT :

IF ENZYME IS NOT FILLED THEN GO TO L3

IF RESTRICTION MAP CONTAINS ENZYME THEN SET
LOC-LIST TO LOCATIONS AND APPLY SFRAG1 OTHERWISE
COMMENT FRAGMENT-UNIT CONTAINS NO ENZYME SITES

GO TO LABEL L2
```
The preceding rule statements get the names of the restriction enzymes for which a fragment table is desired. For each enzyme, the rules check to see if the structure's restriction map contains that enzyme. If it does then another rule, SFRAG1, which adds a row to the fragment table for each cutting site of the enzyme, is called. SFRAG1 is:

```
        FOR EACH ELEMENT IN LOC-LIST ENTER IN
        FRAGMENT ELEMENT ENZYME ELEMENT
```

```
LABEL L3

SET FRAGMENT TO SORT FRAGMENT BY SIZE
```
Sort the table so that the values of the column called SIZE are in increasing numerical order.

```
CLEAR LOC-LIST AND CLEAR SITE-LIST

IF TOPOLOGY IS LINEAR THEN ADD END TO SITE-LIST
AND ADD 0 TO LOC-LIST
```
If the DNA structure has been defined as linear instead of circular, then take care of the special case of the ends.

```
FOR EACH ROW IN FRAGMENT TABLE ADD SITE1 TO
SITE-LIST AND ADD LOCATION1 TO LOC-LIST

SET NMAX TO LENGTH-OF FRAGMENT TABLE

IF TOPOLOGY IS LINEAR THEN ADD END TO SITE-LIST
AND ADD LENGTH TO LOC-LIST AND
SET NMAX TO NMAX + 1
```

```
IF TOPOLOGY IS CIRCULAR THEN SET SCR-N TO
GET-ELEMENT 1 OF LOC-LIST AND SET SAVE TO
SCR-N + LENGTH AND SET SCR TO GET-ELEMENT
1 OF SITE-LIST AND ADD SCR TO SITE-LIST AND
ADD SAVE TO LOC-LIST
```
If the DNA structure is circular, then handle the case of a fragment that might wrap around the ends.

```
CLEAR FRAGMENT TABLE

SET N TO 0

FOR EACH ELEMENT IN SITE-LIST SET N TO N + 1 AND
SET SCR TO ELEMENT AND SET SCR2 TO NEXT-ELEMENT
AND APPLY SFRAG2
```
This rule iterates through the list of all of the cutting sites of the chosen enzymes and calls another rule, SFRAG2, which calculates the size of all of the fragments.

```
SET FRAGMENT TO SORT FRAGMENT BY SIZE

COMMENT YOUR FRAGMENTS ARE LISTED BY LENGTH
IN FRAGMENT TABLE
```

4.7. A Sample Execution of Fragment-Rules

```
EDIT : edit fragment-rules

Rule Editor
RuleEd: interpret

ENTER THE DNA-STRUCTURE FOR WHICH YOU WANT
FRAGMENT DATA:    sv40

ENTER RESTRICTION ENZYMES TO BE ENTERED IN MAP
(ONE PER LINE END WITH <CR>)
:    hind2
:    hind3
:

YOUR FRAGMENTS ARE LISTED BY LENGTH
IN FRAGMENT TABLE

RuleEd: done

EDIT : print fragment

FRAGMENT:

SIZE      SITE1     LOCATION1  SITE2     LOCATION2

20        HIND2     502        HIND2     522
29        HIND2     473        HIND2     502
215       HIND3     1494       HIND3     1709
240       HIND2     2060       HIND2     2300
259       HIND3     3477       HIND2     3736
267       HIND2     3736       HIND3     4003
351       HIND3     1709       HIND2     2060
369       HIND2     2300       HIND2     2669
447       HIND3     1047       HIND3     1494
525       HIND2     522        HIND3     1047
544       HIND3     5172       HIND2     473
808       HIND2     2669       HIND3     3477
1169      HIND3     4003       HIND3     5172

EDIT : done
(Leaving EDIT of unit REST-RULES)
```

5. Experience with the Rule Language

The Rule Language has been evolving over the period of about two years. Approximately a dozen molecular biologists are actively using it to describe procedural knowledge Of that effort, over half falls into the creation of data manipulation rules, simply because the embodiment of those rules in an easily manipulable knowledge base is of direct utility to the users in their individual laboratories The domain experts actively share and criticize each other s procedures

All of the major users of the Rule Language (and indeed of the entire Unit System) chose to learn to use the system by direct interaction rather than by first reading instructions They make frequent use of the built-in help facilities the online rule language manual stored within all knowledge bases They make frequent requests for new rule verbs and new boolean relations The rule language began, in tact as only the SET verb along with a simple IF ... THEN ELSE structure and the boolean relations = . > and < All of the additional structure in the language was created because of requests from molecular biologist users In a very real sense the users created their own programming language

The work of the computer scientists involved in creating the Rule Language has been in taking the expert definitions of primitive actions and relations and implementing those definitions in the parsing and interpretation functions of the Rule Language The computer scientists therefore, have had only to learn relatively small chunks of molecular biology

A measure of success of this method for describing procedural knowledge may be seen in the enthusiasm shown by the many MOLGEN collaborators in building complex knowledge bases Indeed, these groups are using the Unit System and the Rule Language in large part because they get the immediate payoff of sophisticated manipulation of diverse types of information that have become awkward to handle in the laboratory Furthermore many of the groups are advancing from simple data manipulation procedures to more complex experiment design strategies (these of use for DNA sequencing and cloning, for example)

6. Conclusions

This section will discuss some of the general issues that have arisen during the course of the MOLGEN work in the acquisition and representation of procedural knowledge Perhaps the most generally useful lesson »s that system builders should not try to design interactive knowledge acquisition programs without the direct consultation of users Features that seem trivial to computer scientists are realty vital to non computer scientist domain experts Features that seem especially clever to computer scientists may hardly be used by the domain experts The bas»c system design should be flexible enough to allow easy changes This work has shown that such a philosophy can lead to the acceptance of a knowledge representation system by busy domain experts

The distinctions between the Rule Language and any general-purpose programming language are becoming fuzzier as the language evolves. Since this evolution is guided totally by demands of its users, the speculation may be made that domain experts really knew how to program all along but just didn't have a convenient language. If they continue to be successful in their desire to adequately describe many different types of procedural knowledge while remaining ignorant of internal representation details, then perhaps the simple structure of the Rule Language may serve as a guide for many different real world domains. Like the vocabulary of domain specific declarative knowledge, the "verbs" of the Rule Language tend to be strongly stylized. This helps to avoid the need for complex natural language processing.

The point was made in [4] that simple experiment design strategies seem to serve as the driving force for at least a major fraction of experiment designs. Complexity arises in the semantic richness of the objects and operators of the domain. The Rule Language is syntactically simple, while, potentially, semantically rich. The implementation allows semantic augmentation by adding new verbs and relations in a simple, modular manner. Indeed, the Unit System and rule language are beginning to be used in a variety of other scientific domains. It is still early to make many conclusions about the nature of expert rules and their collection into strategies, but an experimental laboratory for collecting empirical data has been created.

ACKNOWLEDGMENTS

The author would like to thank his many collaborators in the MOLGEN group for their contributions to this work. In particular, Russell Greiner and Yumi Iwasaki helped develop parts of the Rule Language. Professors Laurence Kedes and Douglas Brutlag have shown unflagging enthusiasm for the knowledge acquisition work discussed in this paper. Special thanks to James Bennett for extensive comments on earlier drafts of the paper.

References

1. Shortliffe, E. H., *MYCIN: Computer Based Medical Consultations.* American Elsevier, 1076.

2. Hart, P. E. and Duda, R. O., "PROSPECTOR - A Computer Based Consultation System for Mineral Exploration," Tech. report 155, SRI, 1977.

3. Stefik, M. J., "An Examination of a Frame-Structured Representation System," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence,* UCAI, 1979, pp. 845-852.

4. Friedland, P., "Knowledge-Based Experiment Design in Molecular Genetics," Computer Science Department Report CS-79 771, Stanford, October 1979.

5 Smith, R. G. and Friedland, P., "A User's Guide to the Unit System," Heuristic Programming Project Memo HPP-80-28, Stanford, December 1980.

6. Davis, R., "Applications of Meta Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases," Computer Science Department Report CS-76-552, Stanford, July 1976.

7. Clancey, W. J., "Tutoring Rules for Guiding a Case Method Dialogue," *International Journal of Man-Machine Studies,* Vol. 11, 1979, pp. 25-49.

8 Aikins, J. S., "Prototypes and Production Rules: A Knowledge Representation for Computer Consultations," Computer Science Department Report CS-80-814, Stanford, August 1980.