An Example of a Special Purpose Input Language to LaTeX

Henry Baragar
Instantiated Software Inc., 20 Woodmount Crescent, Nepean, Ontario, K2E 5R1 Canada.
henry@instantiated.on.ca

Gail E. Harris
RES Policy Research Inc., 6th Floor, 100 Sparks Street, Ottawa, Ontario, K1P 5B7 Canada.
ak753@freenet.carleton.ca

Abstract

A special purpose language for documenting knowledge bases demonstrates how LaTeX can be augmented to add expressiveness for specific situations. The language, called TESLA, enables expert system analysts to mark up groups of rules into tables in a way which reflect the logical structure of the knowledge base. The TESLA style options generate LaTeX tables for use by expert system programmers and the equivalent English text typeset in a subsection for use by domain experts. This paper presents the syntax and implementation of this special purpose language. Despite the complex output requirements, the TeX implementation has proven to be very flexible and remarkably short.

Introduction

> *If I have seen further than other men,*
> *it is because I have stood on the shoulders of the giants.*
>
> —Isaac Newton

The logical treatment of documents is one of LaTeX's most important features. A benefit of this approach is that the source files for most LaTeX documents are usually almost as readable as the final output. As is true with any general purpose tool, there are cases that are not easily expressed in the input language of the tool. In this case, a special purpose language (or "little language"), as advocated by Jon Bentley (1990, page 83), can be of great benefit. A well-designed "little language" — in which the special case can be easily expressed — follows more closely the philosophy of LaTeX than does the contortion of LaTeX commands to achieve a desired result.

This paper presents a special purpose language for documenting knowledge bases which has a much more natural syntax than pure LaTeX for marking up the rules of a knowledge base. It has been used successfully to typeset the system documentation for the knowledge base portion of an application on a project where the documentation tool of choice for the rest of the system was Microsoft Word. We begin by describing problems with documenting knowledge bases. Then we present the "little language" that was designed specially for documenting knowledge bases, and show how it was implemented in TeX, yielding a special purpose input language to LaTeX. This is followed by some observations on the suitability and success of the solution. We conclude with a discussion of future directions and some recommendations for others wishing to implement special purpose languages, and in particular special purpose input languages to LaTeX.

The Challenge

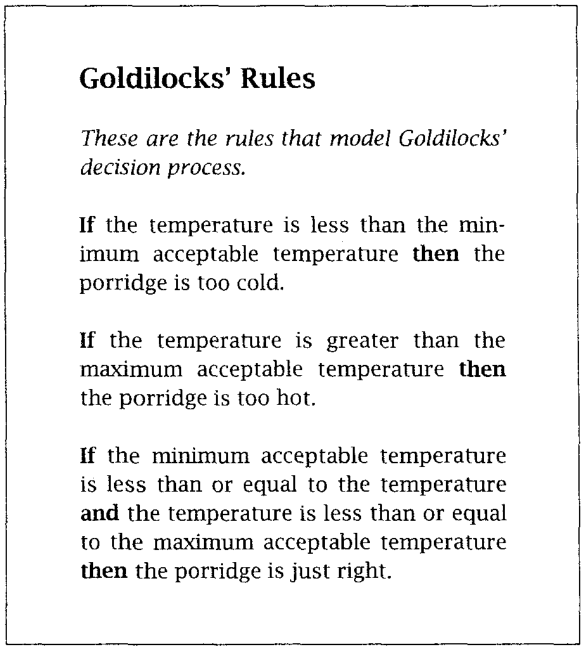
The problem of documenting knowledge bases was encountered on a project where an existing knowledge base with no external documentation had to be maintained and expanded. The first step in the project was to document, or reverse engineer, the knowledge base. This in itself is a challenge because expert system analysts are still struggling to find effective methods to document knowledge bases. Some methods, such as KADS[1], are too high level and do not document individual rules. Other lower level methods are usually tools tied to specific products — products not being used on this project. This project required a tool for documenting knowledge bases at the rule level, but not tied to a specific product.

The challenge, to the expert system analyst, in documenting the rules of a knowledge base is in the need to present the documentation to two audiences. The first audience, the expert system programmer, uses the documentation to program the rules in the knowledge base. The second audience, the system owner or domain expert, uses the documentation to verify the correctness of the rules in the knowledge

[1] Although "KADS" was an acronym at one time (Knowledge Acquisition and Documentation System), it has changed and it is now considered a proper name in itself.

Goldilocks' Rules		
Conditions		Conclusion
These are the rules that model Goldilocks' decision process.		
T < min		too-cold
	T > max	too-hot
min ≤ T	T ≤ max	just-right

Figure 1: The tabular form for the expert system programmer.



Goldilocks' Rules

These are the rules that model Goldilocks' decision process.

If the temperature is less than the minimum acceptable temperature **then** the porridge is too cold.

If the temperature is greater than the maximum acceptable temperature **then** the porridge is too hot.

If the minimum acceptable temperature is less than or equal to the temperature **and** the temperature is less than or equal to the maximum acceptable temperature **then** the porridge is just right.

Figure 2: The English form for the domain expert.

base. The tabular presentation of Figure 1, preferred by the expert system programmer, is usually incomprehensible to the domain expert, who prefers English sentences and paragraphs, as in Figure 2. The challenge of accurately presenting both sets of documentation is often so great that the domain expert is often given inadequate summaries of the rules or is left to struggle with just the tabular representation of the rules. This often leads to a loss of confidence in the Expert System, as had happened on the project in question.

The challenge of presenting two sets of documentation would be considerably simplified if they could both be generated from the same source. This is not possible in Microsoft Word, the tool specified for documentation in this particular project. Considering the differences between the tabular form and the English language form illustrated in Figures 1 and 2, it was not even clear this would be possible in LaTeX. Nor was it clear that a LaTeX source file would be easily readable and maintainable. Thus, the challenge was to find a mechanism to document the rules of the knowledge base in a single source file, where the structure of the rules is visually apparent to the expert system analyst and where the documentation sets are appropriate to their intended audience.

The New Input Language

The best way to ensure that the structure of the rules is visually apparent in a source file documenting a knowledge base is to develop a new syntax for marking up rules that has a clean visual presentation. In this section, we present a special purpose language, or "little language" à la Jon Bentley (page 83), that has a syntax with the desired properties. We leave the details of implementing the language until the next section.

The syntax of the new input language — called TeX Expert System Language (TESLA)[2] — is very simple and has only five commands. These commands can be divided into three groups: definitions, groups of rules, and other commands. A clean visual presentation of the source file has been achieved by defining a syntactic structure for these commands which allows an ASCII text source file to be modeled after the layout of the tabular representation to be presented to the expert system programmer; this reflects the common backgrounds of the expert system analyst and the expert system programmer.

Definitions. A variable in a knowledge base is documented by giving an English language phrase that defines the variable. TESLA allows variable names from the knowledge base to be used directly in the TESLA source file. The knowledge base variable is left unchanged when it is presented to the expert system programmer, whereas it is mapped to the English Language phrase when it is presented to the domain expert. A TESLA definition, which has the following syntax:

```
[tvar␣|␣KB-var␣|␣English description␣]
```

is used to specify the mapping of the knowledge base variable to its English Language description. An example of a definition is:

```
[tvar␣|␣T␣|␣the␣temperature␣]
```

which defines the knowledge base variable T as the phrase "the temperature". That is, a reference to the knowledge base variable T in a TESLA rule is represented by the string "T" in the tabular form presented to the expert system programmer, whereas it is represented by the string "the temperature" in the English form presented to the domain expert.

[2] The language was developed for the Travel Expert System (TES) project, which also explains why all the commands begin with a "t".

Groups of rules. The rules of a knowledge base are usually documented as groups of related rules. For the expert system programmer this means that a group of related rules is presented as a table, whereas the group is presented as a subsection to the domain expert. A group of rules begins with:

[tgroup␣|␣*Group Name*␣|␣*n*␣]

where *Group Name* is a label for the group and *n* is the maximum number of *conditions*, excluding the *conclusion*, in the rules in this group. In the tabular form presented to the expert system programmer, *n* is one less than the number of columns in the table.

A group of rules ends with:

[tgroup␣|␣*Group Name*␣|␣e␣]

where *Group Name* should be the same as at the beginning of the group.

Occasionally, it may be desirable to visually separate subgroups of rules within a large group of rules. This is accomplished with the:

[tgroup␣|␣*Group Name*␣|␣-␣]

command, which inserts a horizontal line (`\hline`) into the table. Currently, it does nothing in the English form presented to the domain expert.

A rule in TeSLA has the following syntax:

[trule␣|␣*cond1*␣|␣*cond2*␣|␣...␣|␣*concl*␣]

where *cond1*, *cond2*, ..., are the *n* conditions of the rule and *concl* is the conclusion of the rule.

Each condition — as well as the conclusion — is a *relation* that has one of the following forms:

lhs␣*rel*␣*rhs*
rel␣*rhs*␣_
rhs␣_␣_

where *lhs* and *rhs* are TeSLA variables, and *rel* is a relation operator. In the tabular form, each condition and the conclusion is put in its own column. With suitable groupings of rules and arrangements of relations within columns, an expert system programmer can easily check that all possible combinations of relations have a known conclusion and that no two rules conflict with one another. The English form given to the domain expert, on the other hand, has every variable, relation, and implicit conjunction spelled out in full.

As an example, consider the rules already presented in Figure 1 and Figure 2, which would appear in the TeSLA source file as:

```
[tgroup | Goldilocks' | 3 ]
[trule | T < min  |            | too-cold _ _  ]
[trule | _        | T > max    | too-hot _ _   ]
[trule | min <= T | T <= max   | just-right _ _]
[tgroup | Goldilocks' | e ]
```

Note that this code fragment lacks the variable definitions and the command to add the extra descriptive text found in the figures. Also note that a quirk in the implementation requires that leading empty conditions must have a single "_" character, as in the `too-hot` rule above.

Other commands. There are two commands in TeSLA for adding annotations to the rules. The first, which has the following syntax:

[ttext␣|␣*text*␣]

provides a mechanism for adding arbitrary explanatory text into both the tabular and the English forms. The second, which has the following syntax:

[trem␣|␣*text*␣]

provides a mechanism for adding extra text, for remarks, only to the tabular form used by the expert system programmer. There has yet to be a requirement to add text to the English form used by the domain expert which is not also required by the expert system programmer.

Other syntax. There is little requirement for additional syntax in TeSLA. Syntax was added to TeSLA to treat all text between the "`;`" character and the end of a line as source file comments. The "%" character was rejected for introducing comments because percentages are used frequently in the knowledge base on this project. All other considerations for adding syntax have been rejected because of the extra effort that would be required to explain them.

The Implementation

Now that the syntax of TeSLA has been defined, the implementation details can be discussed. Two approaches to implementation were considered: either, build a preprocessor, or implement TeSLA directly in (La)TeX. At first, it seemed that the preprocessor approach would be easier to implement. This had the advantage that the output could be switched to Microsoft Word code if and when a definition of the file format for Microsoft Word could be found. However, good string manipulation tools, such as `perl` and awk, needed to implement the preprocessor were not readily available for the target environment (Microsoft DOS). Thus, the approach to implement TeSLA directly in (La)TeX was selected.

TeSLA is implemented in TeX as three style options. The first, `tesla.sty`, has the definitions of the TeSLA commands described above as well as all the other definitions common to both output forms. The other two files are `eng-form.sty`, which contains code specific to the English form, and `tab-form.sty` which contains code specific to the tabular form. This section begins by describing how to use these style files. Then it defines the implementation of TeSLA in detail as coded in the style files.

The structure of TeSLA documents. TeSLA documents are composed of three main files and one or more rule files. The main files are usually named `main-eng.tex`, `main-tab.tex` and `main.tex`. The first two of these files, which are used to select the form, simply contain:

```
\documentstyle[tesla,form]{article}
\input{main}
```

where the *form* is either eng-form or tab-form, depending on whether it is in the file main-eng.tex or main-tab.tex, respectively. As can be seen from this code, these files input the file main.tex, which is the real TeSLA document.

To input a rule file into the main TeSLA file, the following command is used:

```
\inputrulefile{rule-file}
```

where *rule-file* is the name of a file containing TeSLA code. This command is defined as:

```
\newcommand{\inputrulefile}[1]{%
  \changecatcodes\input{#1}}
```

in tesla.sty.

The \changecatcodes command changes the categories of the digits, arithmetic and relation operators, and a few other characters to category 11, the same category as the alphabetic characters. This allows these characters to be used in TeSLA variable names and enables a broad range of variable names, including operators and numbers! The \changecatcodes command also changes the category of the ";" character to category 14 to make it the comment character, which is the "%" character in (LA)TeX. Finally, \changecatcodes changes the category of the "[" character to category 0 to make it an escape character, the same category as the "\" character in (LA)TeX. This allows the commands of TeSLA, such as [tvar, to be implemented directly as TeX commands.

The [tvar command. The [tvar command, like all the TeSLA commands, makes use of TeX's pattern-matching capability to implement TeSLA syntax. It is defined in tesla.sty as:

```
\gdef\tvar␣|␣#1␣|␣#2␣]{%
  \expandafter\gdef\csname #1\endcsname
    {\xform{#1}{#2}}}
```

There are two things to note in this definition:

1. The command has been *defined* as \tvar even though it is *used* as [tvar. The reason is that the category of the "[" character has not yet been set to category 0 (the same category as "\") as it will be when a rule file is read.
2. The spaces in the \gdef\tvar␣|␣#1␣|␣#2␣] are important. The implementation of TeSLA relies on the fact that TeX compresses all strings of white space to a single "␣", which means that the arguments to [tvar can be spaced out to improve the readability of the rule files. Although this may not be important for [tvar commands, it is important for [trule.

The [tvar command relies on the \xform command, which is defined differently in eng-form.sty than it is in tab-form.sty, to do its work. All of the TeSLA commands use the same mechanism, where a command beginning with "\x" is called to implement the real behavior, to achieve different behaviors depending on whether the English form or the tabular form is to be presented. The desired behavior for [tvar is that a TeSLA variable is defined as the definition supplied by the user in the English form and as itself in the tabular form. This is accomplished by defining \xform as:

```
\gdef\xform#1#2{#2}
```

in eng-form.sty, and as:

```
\gdef\xform#1#2{#1}
```

in tab-form.sty. Thus, for the TeSLA variable T from our Goldilocks example, we now have \T which expands to "the temperature" in the English form and to "T" in the tabular form.

The [tgroup command. The \tgroup command is really several commands and it looks at its second argument to determine which command to run. It is defined in tesla.sty as:

```
\gdef\tgroup␣|␣#1␣|␣#2␣]{%
\if 1#2 \xbegin{xone}{1}{2}{.466}{#1}\fi
\if 2#2 \xbegin{xtwo}{2}{3}{.300}{#1}\fi
\if 3#2 \xbegin{xthree}{3}{4}{.216}{#1}\fi
\if 4#2 \xbegin{xfour}{4}{5}{.166}{#1}\fi
\if 5#2 \xbegin{xfive}{5}{6}{.133}{#1}\fi
\if 6#2 \xbegin{xsix}{6}{7}{.1095}{#1}\fi
\if -#2 \xsep\fi
\if e#2 \xend\fi
}
```

where \xbegin, \xsep and \xend are defined differently, depending on the form.

The definition for \xbegin in eng-form.sty is to define [trule to be the command specified by the first argument and to introduce a new subsection using the fifth argument. It is implemented as:

```
\gdef\xbegin#1#2#3#4#5{%
  \gdef\trule{\csname #1\endcsname}
  \subsection*{#5 Rules}}
```

The second, third and fourth arguments to \xbegin are ignored in eng-form.sty.

The definition for \xbegin in tab-form.sty is much longer (10 lines) and is not presented here. Like the definition in eng-form.sty, the definition of \xbegin in tab-form.sty defines [trule to be the command specified by the first argument. Then it introduces the code necessary to set up a tabular environment where the number of columns is specified by the third argument and where each column is typeset as a parbox whose width is specified by the fourth argument[3] multiplied by the text width

[3] The values of the fourth argument were determined by experimentation.

(\textwidth). The second argument always has a value of one less than the third argument and is required in order to avoid having to do arithmetic in TeX. It is used to specify the number of columns to be spanned by the "Conditions" heading. The fifth argument is used in the title of the table. Finally, the \xbegin makes some minor adjustments to the tabular environment to improve the visual presentation of the tables.

The \xsep command is defined to do nothing in eng-form.sty and to insert a horizontal line — using \hline — in tab-form.sty.

Similarly, the \xend command is defined as nothing in eng-form.sty, and as

```
\gdef\xend{\hline\end{tabular}}
```

in tab-form.sty.

The [trule command. As was seen in the discussion on [tgroup, the [trule command is defined in \xbegin to be one of \xone through \xsix. These commands are all very similar with \xtwo, for example, being defined in tesla.sty as:

```
\gdef\xtwo␣|␣#1|␣#2|␣#3]{%
  \if_#1␣\xpre␣\xone␣|␣#2|␣#3]%
  \else\xif{#1}␣\xand{#2}\xthen{#3}
  \fi
  }
```

Note the lack of spaces between each argument and the "|"or "]" character that follows the argument ("...␣#1|␣#2|␣#3]...") which preserves a space at the end of the argument and which will be used as a delimiter when the argument itself is parsed.

The \xtwo code says that if the first condition has been set to the character "_", then this is like a one-rule condition: do something specific to the form (\xpre) and call \xone. Note that \xthree will call \xtwo and \xfour will call \xthree, etc. The \xpre does nothing in the English form but is required in the tabular form to insert a "&" character to skip the first column. Otherwise, if the first argument is not the "_" character, then build up the rule using \xif, \xand and \xthen.

In the English form:

```
\xif{#1} \xand{#2}\xthen{#3}
```

produces the expected result of:

If *#1* **and** *#2* **then** *#3.*

except when *#2* is empty, in which case the *and-clause* is elided. This is implemented as:

```
\gdef\xif#1{\par {\bf If} \xrel #1}
\gdef\xand#1{\ifx#1\empty
             \else{\bf and} \xrel #1
             \fi
  }
\gdef\xthen#1{{\bf then} \xrel #1.}
```

in eng-form.sty.

In the tabular form:

```
\xif{#1} \xand{#2}\xthen{#3}
```

produces the expected result of putting the *if-clause* into the first column, the *and-clause* (if there is one) into the second column, and the *then-clause* into the third column. This is implemented in tab-form.sty as:

```
\gdef\xif#1{\RS\xrel #1}
\gdef\xand#1{& \RS\ifx#1\empty
                \else\xrel #1
                \fi
  }
\gdef\xthen#1{& \PBS\RS\xrel #1 \\}
\gdef\RS{\raggedright\sloppy\hspace{0pt}}
```

where \PBS is the \PreserveBackslash command as described in Goossens et al. (1994, page 108). The command \xrel is discussed below.

The \xrel command forms the heart of the TESLA style. It takes three arguments: *lhs*, *rel* and *rhs*, as described above in the description of the TESLA [trule command. Each argument must end in a space; this is why the spaces were left in by the \xone through \xsix commands. If the third argument, or the second and third arguments, is the character "_", then these arguments are elided. This is necessary to prevent extra space being inserted, particularly in the English form and especially just before a period. The \xrel command is implemented as:

```
\gdef\xrel␣#1␣#2␣#3␣{%
  \if_#3\if_#2\xvar{#1}%
        \else\xvar{#1}␣\xvar{#2}%
        \fi
  \else\xvar{#1}␣\xvar{#2}␣\xvar{#3}%
  \fi
  }
```

in tesla.sty. Once again, the spaces between \xvar commands are important, this time to put spaces between the text expansions of *lhs*, *rel* and *rhs* in the English form.

Finally, the \xvar command simply expands the TESLA variable passed to it from \xrel. If the variable is undefined, then the variable name is used, typeset in italics. It is implemented as:

```
\gdef\xvar#1{\expandafter
   \ifx\csname #1\endcsname\relax{\it #1}%
   \else \csname #1\endcsname
   \fi
 }
```

in tesla.sty. Note that the different representations for the different forms have already been encoded in the variable by the [tvar command.

The [ttext command. The [ttext command is defined in tesla.sty as:

```
\gdef\ttext␣|␣#1␣]{\xtext{#1}}
```

where \xtext is defined as \par{\em #1} in the English form, and as

```
\gdef\xtext#1{%
 \multicolumn{\numcols}{|p{.9\textwidth}|}
   {\em #1}\\\hline
 }
```

in tabular form. Note that `\numcols` was defined by the `[tgroup` command.

The `[trem` command. The definition of the `[trem` command is analogous to the definition of `[ttext`. However, `\xrem` is defined as nothing in the English form and as `\xtext{\sc #1}` in the tabular form.

Pre-defined variables. Since the relationship and arithmetic operators are treated like normal TESLA variables, it is trivial to predefine many of the these operators in `tesla.sty`. For example,

```
\gdef\<{\xform{$<$}{is less than}}
```

predefines the < relation.

Observations

We hope that our paper has shown the TeX implementation of the TESLA language is elegant and remarkably compact. The `tesla.sty` file is only 102 lines (of undocumented TeX code), the `tab-form.sty` is 32 lines and the `eng-form.sty` is 18 lines. The implementation in TeX was less difficult than anticipated. It is also shorter than the anticipated preprocessor solution, yet is at least as robust and flexible. It also has the benefit of handling the inclusion, with some restrictions, of LaTeX code into the rules.

Comparison with "pure" LaTeX. The improvement in the visual presentation of the source code of TESLA compared with "pure" LaTeX is striking. Consider our Goldilocks example as it might be written in LaTeX:

```
\begin{tgroup}{Goldilocks'}
\trule{\T\LT\min}{          }{\tooCold}
\trule{          }{\T\GT\max}{\tooHot}
\trule{\min\LE\T}{\T\LE\max}{\justRight}
\end{tgroup}
```

where the content of the document is obscured by too many "\", "{" and "}" characters.

The dictionary. At one point in the project, there was a great rush to produce a dictionary of the knowledge base variables. It was a simple matter to search the source files for all lines with `[tvar`, sort this list, and process it with a simple style that implements the `[tvar` command as an item in a description list — all in less than half an hour. This activity revealed several duplicate variable definitions that might not otherwise have been caught, and forms a counterpart to the implementation of `[tvar` which typesets, in italics, undefined variables as their variable name. This reinforces the advantages of separating the logical structure of a document from the details of typesetting.

Conclusions

> *I have stood on the shoulders of*
> *Jon Bentley and Donald Knuth.*
>
> —Henry Baragar

The TESLA language has met its original goals. The structure of the rules is visually apparent in a TESLA source file and it has been used successfully for a knowledge base with more than 270 rules using over 250 variables. The two forms of output have been well received by their intended audiences. Surprisingly, some of the expert system programmers have found the English form has helped them to understand the context of the rules that they were reading in the tabular form, a context that is sometimes lost in the brevity of using only variable names.

Spurred by the success of the implementation of TESLA, we would like to enhance the functionality of the language. First, we would like to expand the `[tgroup` command to express relationships between the tables, which then could be graphed and included in the documentation. Second, we would like to enhance the `[trule` command to generate code for a particular Expert System shell, which would significantly reduce the consistency problems between the documentation and the code. This capability could be extended to multiple Expert System shells.

This example of a special purpose input language to LaTeX illustrates the utility of application-specific mark-up languages and the suitability of using TeX for the implementation. We hope this example will encourage others to consider creating "little languages" in TeX in those cases where the logical structure of their documents is lost in the typesetting commands in their source files. We certainly have found the benefits have been extraordinary and the difficulties surprisingly minor.

Acknowledgements

We would like to thank Christina Thiele who was the one to finally convince us to write this paper and was kind enough to preview it for us. Also, we would like to thank Christine Detig who reviewed the paper and provided helpful comments.

Bibliography

Bentley, Jon, *More Programming Pearls: Confessions of a Coder*, Reading Mass.: Addison-Wesley, 1990.

Goossens, Michel, Frank Mittelbach, and Alexander Samarin, *The LaTeX Companion*, Reading Mass.: Addison-Wesley, 1994.

Knuth, Donald, *The TeXbook*, Reading Mass.: Addison-Wesley, 1989.

Lamport, Leslie, *LaTeX: A Document Preparation System*, Reading Mass.: Addison-Wesley, 1986.

Appendix

The `tesla.sty` file.

```
\setlength{\parskip}{\baselineskip}
\setlength{\parindent}{0pt}

\gdef\empty{}

\gdef\tvar | #1 | #2 ]{%
    \expandafter\gdef\csname #1\endcsname{\xform{#1}{#2}}%
    }
\gdef\tgroup | #1 | #2 ]{%
    \if 1#2 \xbegin{xone}{1}{2}{.466}{#1}\fi
    \if 2#2 \xbegin{xtwo}{2}{3}{.300}{#1}\fi
    \if 3#2 \xbegin{xthree}{3}{4}{.216}{#1}\fi
    \if 4#2 \xbegin{xfour}{4}{5}{.166}{#1}\fi
    \if 5#2 \xbegin{xfive}{5}{6}{.133}{#1}\fi
    \if 6#2 \xbegin{xsix}{6}{7}{.1095}{#1}\fi
    \if -#2 \xsep\fi
    \if e#2 \xend\fi
    }
\gdef\ttext | #1 ]{\xtext{#1}}
\gdef\trem | #1 ]{\xrem{#1}}

\gdef\xone | #1| #2]{%
    \xif{#1} \xthen{#2}
    }
\gdef\xtwo | #1| #2| #3]{%
    \if_#1 \xpre \xone | #2| #3]%
    \else\xif{#1} \xand{#2}\xthen{#3}
    \fi
    }
\gdef\xthree | #1| #2| #3| #4]{%
    \if_#1 \xpre \xtwo | #2| #3| #4]%
    \else\xif{#1} \xand{#2}\xand{#3}\xthen{#4}
    \fi
    }
\gdef\xfour | #1| #2| #3| #4| #5]{%
    \if_#1 \xpre \xthree | #2| #3| #4| #5]%
    \else\xif{#1} \xand{#2}\xand{#3}\xand{#4}\xthen{#5}
    \fi
    }
\gdef\xfive | #1| #2| #3| #4| #5| #6]{%
    \if_#1 \xpre \xfour | #2| #3| #4| #5| #6]%
    \else\xif{#1} \xand{#2}\xand{#3}\xand{#4}\xand{#5}\xthen{#6}
    \fi
    }
\gdef\xsix | #1| #2| #3| #4| #5| #6| #7]{%
    \if_#1 \xpre \xfive | #2| #3| #4| #5| #6| #7]%
    \else\xif{#1} \xand{#2}\xand{#3}\xand{#4}\xand{#5}%
        \xand{#6}\xthen{#7}
    \fi
    }

\gdef\xrel #1 #2 #3 {%
   \if_#3\if_#2\xvar{#1}\else\xvar{#1} \xvar{#2}\fi
   \else\xvar{#1} \xvar{#2} \xvar{#3}%
   \fi
```

```
    }
\gdef\xvar#1{%
    \expandafter
        \ifx\csname #1\endcsname\relax{\it #1}%
        \else \csname #1\endcsname%
        \fi
    }

\newcommand{\inputrulefile}[1]{%
    \changecatcodes
    \input{#1}
    }

\newcommand{\changecatcodes}{
    \catcode`\+=11          \catcode`\0=11
    \catcode`\-=11          \catcode`\1=11
    \catcode`\*=11          \catcode`\2=11
    \catcode`\(=11          \catcode`\3=11
    \catcode`\)=11          \catcode`\4=11
    \catcode`\<=11          \catcode`\5=11
    \catcode`\!=11          \catcode`\6=11
    \catcode`\>=11          \catcode`\7=11
    \catcode`\:=11          \catcode`\8=11
    \catcode`\==11          \catcode`\9=11
    \catcode`\_=11
    \catcode`\[=0
    \catcode`\;=14
    }

{
\changecatcodes
\gdef\<{\xform{$<$}{is less than}}
\gdef\>{\xform{$>$}{is greater than}}
\gdef\!={\xform{$\neq$}{is not equal to}}
\gdef\=={\xform{$=$}{is equal to}}
\gdef\<={\xform{$\leq$}{is less than or equal to}}
\gdef\>={\xform{$\geq$}{is greater than or equal to}}
\gdef\:={\xform{$\leftarrow$}{is assigned}}
\gdef\+{\xform{$+$}{added to}}
\gdef\+={\xform{$+$$=$}{is incremented by}}
\gdef\decrement{\xform{$-$$=$}{is decremented by}}
\gdef\minus{\xform{$-$}{less}}
\gdef\*{\xform{$*$}{multiplied by}}
\gdef\memberOf{\xform{$\in$}{is one of}}
\gdef\notMemberOf{\xform{$\not\in$}{is not one of}}
\gdef\isnot{\xform{$\neg$}{not}}
}
```

The eng-form.sty file.

```
\gdef\xform #1#2{#2}

\gdef\xbegin#1#2#3#4#5{
    \gdef\trule{\csname #1\endcsname}
    \subsection*{#5 Rules}
    }
\gdef\xsep{}
\gdef\xend{}
```

```
\gdef\xpre{}

\gdef\xif#1{\par {\bf If} \xrel #1}
\gdef\xand#1{\ifx#1\empty\else{\bf and} \xrel #1 \fi }
\gdef\xthen#1{{\bf then} \xrel #1.}

\gdef\xtext#1{\par{\em #1}}
\gdef\xrem#1{}
```

The `tab-form.sty` **file.**

```
\gdef\xform#1#2{#1}

\gdef\xbegin#1#2#3#4#5{
    \gdef\trule{\csname #1\endcsname}
    \gdef\numcols{#3}
    \par\begin{tabular}{*{#3}{|p{#4\textwidth}}|}
    \hline
    \multicolumn{#3}{|c|}{\rule{0pt}{2.8ex}\large\bf #5 Rules}\\
    \hline
    \multicolumn{#2}{|c|}{\rule{0pt}{2.8ex}\large Conditions}
        & \large Conclu\-sion \\
    \hline\hline
    }
\gdef\xsep{\hline}
\gdef\xend{\hline\end{tabular}}

\gdef\xpre{&}

\gdef\xif#1{\RS\xrel #1}
\gdef\xand#1{& \RS\ifx#1\empty\else\xrel #1\fi}
\gdef\xthen#1{& \PBS\RS\xrel #1 \\}

\gdef\xtext#1{\multicolumn{\numcols}{|p{.9\textwidth}|}{\em #1}\\
              \hline}
\gdef\xrem#1{\xtext{\sc #1}}

%see LaTeX Companion, page 132 (for \hspace{0pt})
\gdef\RS{\raggedright\sloppy\hspace{0pt}}

%see LaTeX Companion, page 108
\gdef\PreserveBackslash#1{\let\temp=\\#1\let\\=\temp}
\let\PBS=\PreserveBackslash
```