Expload

Bernie Simon

9 October 96

Usage

The task is run by entering the name of the task followed by three arguments. The first argument is the input load file, the second argument is the output load file, and the third argument is the name of the rules file. The input and output load file names may be the same, as long as you have `clobber` set to `yes`. For example,

```
expload input.lod output.lod rules.txt
```

The task converts each row in the input load file into one or more rows in the output load file. The output load file contains the same header information and columns as the input load file, with the addition of a new column, `expansion_id`. This new column contains the number of the row in the output table, starting at one. The expansion id is used as part of the unique key to join the file and row level instrument relations in the database.

Expansion are performed on the rows of the input table according to a set of rules. The rules are contained in the rules file, which is the third argument to the task. Each rule has two part, the target and the action.

Rules are applied in the following way. The task reads a row from the input load file. It then looks at the target part of each rule in the order they are placed in the rules file. The first rule whose target matches the input row is use to convert that row. Columns and values contained in the action part of the rule are used modify the input row to produce the output row. Because the action part of the rule may have several alternatives, the input row may be expanded into several output row, one for each alternative. After the new row is produced, the set of rules is searched again to see if any of the rules can be applied to the new row. This process continues until no further matches can be found, at which point the new rows are written to the output load file.

For example, suppose the rules file contained the following rule:

```
POLAR_ID = A && PASS_DIR = -1 => PASS_DIR = 0 ||
                                 PASS_DIR = 1 ||
                                 PASS_DIR = 2;
```

If a row has the values `A` in the `POLAR_ID` and `-1` in the `PASS_DIR` column, the input row will be expanded into three output rows, with the values `1`, `2`, and `3` in the `PASS_DIR` column.

This rule shows some of the syntax of a rules file. The target and action parts of a rule are separated by the symbol `=>` and the entire rule is terminated by

a semicolon. A rule may span several lines and the amount of whitespace used is optional. Comments may be placed on any line; they begin with a # and run to the end of the line. The different conditions in the target part of the rule are separated by the symbol &&. Each condition consists of a name and a value separated by an equals sign. The different results in the action part of the rule are separated by the symbol ||. Each result consists of a name and a value separated by an equals sign. If there is more than one name and value in the result, the different name-value pairs are separated by && symbols. An example of a rule with all these syntax elements is:

```
TARGET = ANY && OBSERVER = ANY =>            # Two conditions
  TARGET=M31 && OBSERVER = HUBBLE ||         # First result
  TARGET='OMEGA CENT' && OBSERVER = STRUVE ; # Second result
```

Notice that in the above example that an identifier containing a blank can be used if the identifier is enclosed in quotes. Double quotes could also have been used. Case is not significant in either the column name or the value. If a rule mention column names that are not in the load file being expanded, the rule is ignored.

Data Files

This task uses load files and rule files. The format of a load file is described in the documentation for mkload. The syntax for rules file was described informally above. The formal syntax in BNF format is:

```
file :==  rule | file rule

rule :==  phrase '=>' clause ';'

clause :==  phrase | clause '||' phrase

phrase :==  term | phrase '&&' term

term :== string '=' string

string :==  [-+_A-Za-z0-9]+ | '[^']*' | "[^"]*"
```

Algorithm

The main procedure for expload is `task`. It reads the input load file into memory, creates the output table with the same format as the input load file, calls `parse_rules` to convert the rules file into its internal form, and calls `use_rules` to expand the rows in the input load file according to these rules. Finally, it writes the output load file and frees the structures used to hold the tables internally.

`Parse_rules` converts the rules file into two tables: the target and action tables. The target table contains all the columns of the input load file, plus two new columns: `_first` and `_last`. The `_first` and `_last` columns contain the first and last row numbers of the rows in the action loadfile which correspond to the target loadfile. The values in the columns which match the input load file either contain a string to be matched for expansion to occur or the empty string if any value in that column is ok for matching. The action table contains the same columns as the input load file. The values in its columns contain the strings that replace values in the corresponding columns in the input load file. Since this is an expansion tool, in general there were be more than one row for each target row and each row in the input load file will be replaced by multiple rows in the output load file.

`Parse_rules` creates the target and action tables and then calls `yyparse` to fill them. The code that creates the target and action tables from the rule file is generated by flex (lex) and bison (yacc). As the rules file is parsed, the values in the target and action table are filled. Variables track the values used to fill the `_first` and (_last) columns in the target table. A flag variable, `missing`, is used to indicate whether a column name in the current rule was not found in the input load file. If so, the rows in the target and action table derived from that rule are discarded when the rule is finished parsing.

`Use_rules` creates a work table to hold partially expanded rows from the input table. The work table contains the same columns as the input load file, plus the additional columns `_target` and `_index`. The `_target` column contains the row number in the target table of the rule currently being expanded. The (_index) column contains the row number in the action table of the rule currently being expanded.

This function loops over all rows in the input table. First a row from the input table is copied into the last row of the work table. The function then enters a loop which does not exit until the work table is empty. `Find_rule` is called in a while loop to find the rule in the target table which matches the last row in the work table. If a match is found, `apply_rule` is called to overwrite the columns in the work table with the values in the action table. When `find_rule`

cannot find any more matches for the last row in the work table, the while loop is exited. Then the last row in the work table is copied into the output table and deleted from work table.

`Find_rule` returns the row number in the target table which matches the last row in the work table. If no match is found, it returns `ERR`. If there are no rows in the work file, the function takes an immediate exit and returns `ERR`. If the last row in the work table has already matched a row in the target table, indicated by a non-blank value in the `_target` column, that row number is returned. Otherwise, each row in the target table is searched for a match against the last row in the work table. If a match is found, the `_target` column in the work table is set to the row matched in the target table and the `_index` column in the work table is set to the value in the `_first` column in the target table.

`Apply_rule` compares the value in the `_index` column in ht last row of the work table to the value of `_last` in the corresponding row of the target table. If it is greater, the function deletes the last row of the work table and exits. If not, the function duplicates the last row in the work table with a new last row. The non-blank columns in the action table overwrite the corresponding column in the new last row of the work table. The `_target` column is left blank and the `_index` column is set to the value in the previous last row plus one. Thus the work table is maintained as a stack of partially expanded rows from the input table, with the `_target` column pointing to the row in the target column that matched that row and the `_index` column pointing to the row in the action table *after* the row which expanded it.

System Dependencies

Nothing in `expload` is system dependent, but since the task uses bison and flex to create the source files `yyparse.c` and `yylex.c`, it can only be compile from scratch on a Unix system.

Test Data

The files `f691434jy.ld` and `fosrules.txt` in the `test/data` subdirectory can be used to test the task. Run `expload` with the command line

```
expload f691434jy.ld tmp/output.ld fosrules,txt
```