Prototyping an Expert System Shell with the Logic-Based Approach

Nittaya Kerdprasop, Kanjana Intharachatorn and Kittisak Kerdprasop

Data Engineering Research Unit, School of Computer Engineering,
Suranaree University of Technology, Thailand
nittaya@sut.ac.th

Abstract

Expert system is a computer program that is different from other conventional computer programs in that it incorporates specific knowledge, which can be human judgment, experience, and expertise, in order to provide knowledgeable advice to users. The main difference is caused from the structure of an expert system that contains the two loosely coupled parts: knowledge inference engine and knowledge base, instead of one tightly integrated structure. The loose coupling allows knowledge base contents to be dynamically added, removed, modified, or even completely changed to another subject area, whereas the inference engine remains intact and needs no modification. We refer to the knowledge inference engine as an expert system shell because it can be viewed as an outer layer program to infer knowledge advice from the inner knowledge base core. The development of expert system shell can be efficiently achieved through the support of logic-based language such as Prolog. In this paper, we propose and demonstrate a different scheme of expert system shell development using a constraint-based paradigm with the ECLiPSe constraint system. Comparisons of the two paradigms have been done in terms of computational time, memory usage, and lines of code. The experimental results reveal that the difference in lines of code of the two paradigms is insignificance, but the constraint-based paradigm uses less memory during execution and provides more concise form of knowledge representation.

Keywords: *Expert system shell, Logic programming, Prolog, ECLiPSe constraint system, Constraint logic programming*

1. Introduction

Expert system can be viewed as an intelligent system that acts as a human expert giving advice in some specific application domains. The field of expert system has emerged since the mid-1960s [15] as a successful implementation of artificial intelligence technology. The later coined term knowledge-based system has a close meaning to the expert system, but the main emphasis on decision support. The great success of pioneer medical expert systems such as MYCIN [21] and INTERNIST-1 [17] has attracted considerable attention from cross-discipline researchers including medical experts, computer scientists, engineers, decision analysts, and mathematicians to develop various applications of expert systems [6, 8, 11, 13, 16, 20, 23].

Despite its variety in application domains, most expert systems are composed of two main separate modules: a knowledge base and an inference engine. The knowledge base stores expertise in a particular domain. The inference engine deduces advice from the stored knowledge with reasoning mechanism, and also generates explanation to users regarding the inferred advice. The separation of knowledge base and inference engine allows users connecting a single inference engine to multiple knowledge bases [7, 9, 10].

Building an expert system shell with logic-based language such as Prolog [4] is not a difficult task because the language system provides knowledge representation formalism suitable for the rule-based expert system. We are, therefore, interested in developing the shell with a more advanced scheme of constraint logic programming. Our main focus is to compare efficiencies of both schemes.

Constraint programming has emerged four decades ago as a programming paradigm to solve constraint satisfaction and optimization problems [1, 5, 12]. It has been recently applied to solve biological [2], genomic [18], pattern mining [3], and algorithmic problems [19, 22, 24].

We present in this paper constraint-based and logic-based methods to implement the rule-based expert system shell. In Section 2, we briefly discuss background concepts of constraint logic programming. In Section 3, we explain the details of our implementation. Section 4 is experimentation and results. We then conclude the paper in Section 5.

2. Preliminaries on Constraint Programming

Constraint programming is a programming paradigm that is normally applied to solve combinatorial search problems such as flight scheduling, crew roster assigning, logistic planning, and many more of this kind. The main steps of constraint programming are:

(1) Users specify a problem by defining the variables together with their associated domains and constraints on these variables,

(2) The search procedure and constraint solver find solutions, which are values assigned to the specified variables such that all constraints are satisfied.

It is obvious from the program structure that constraint programming has been designed to solve constraint satisfaction problems that have been extensively studied in artificial intelligence. The efficiency of constraint programs is basically due to the constraint propagator feature in a constraint solver. The function of constraint propagator is to reduce the domains of variables by inferring from the existing constraints and then to prevent the search procedure from visiting parts of the search tree that do not contain any solution.

A constraint propagator takes as input a domain D from which a variable can be assigned its value, and a set of constraints C. The output of the propagator is a reduced domain D′. For instance, given that X, Y, Z are variables, the domains:

$$D(X) = \{a, c, d\},$$
$$D(Y) = \{a, b, c, d\},$$
$$D(Z) = \{c\},$$

and a set of constraints $C = \{ X=Y \wedge Y\neq Z \}$, the output of the constraint propagator are:

$$D'(X) = D'(Y) = \{a, d\}, \text{ and}$$
$$D'(Z) = \{c\}.$$

The repeated application of propagator can lead to increasingly stronger (that is, smaller) domains. The propagator continues until it reaches a fixed point in which the domains cannot be further reduced. At this stage, the search procedure (either global or heuristic-based) can efficiently start assigning possible value to each variable.

A toy example of map coloring [1] in Figure 1 illustrates the constrain-and-search strategy of constraint logic programming (CLP), as opposed to the generate-and-test of logic programming (LP) scheme. The problem is given four colors and four regions, the program has to provide coloring scheme such that two consecutive regions have different colors.

A constraint logic program is an extension of logic program by including constraints in the body of the clauses. Common structure of a constraint program is consisted of the part to define variables and constraints on variables and the part to search for a valid value on each variable. This is the style of constraint-and-search. The structure of constraint logic program is as follows:

```
solve(Variables) :-
        setup_constraints(Variables),
        search(Variables).
```

```
%% CLP style: Constrain-and-search
    :- lib(fd).        % use finite domain library
    map_color_CLP([A,B,C,D]) :-
                                        % firstly declare variables and their domains
            [A,B,C,D] :: [red, green, blue, yellow],
            alldifferent([A,B,C,D]),          % constrain variable values
            labeling([A,B,C,D]).              %  then search
```

```
%% LP style: Generate-and-test
    color(red).
    color(green).
    color(blue).
    color(yellow).

    map_color_LP([A,B,C,D]) :-
                                % generate solution
                color(A),
                color(B),
                color(C),
                color(D),
                                % then test for constraints
                 A \= B,
                 A \= C,
                 A \= D,
                 B \= C,
                 B \= D,
                 C \= D.
```

Figure 1. Constraint Logic Programming (top) versus Logic Programming Scheme (bottom)

At present, there are several constraint systems that provide functions to specify (or model) the problems and maintain the constraint consistency efficiently. They are called constraint programming systems if they are based on procedural languages. The systems are classified as constraint logic programming systems if they are based on logic programming languages. The main benefit of constraint logic programming scheme are two folds:

(1) the declarative style allows users to specify a problem itself, instead of specifying how to solve the problem, and

(2) a high level of knowledge representation facilitates the inclusion of new knowledge that is highly dynamic.

Most constraint logic programming systems provide a large set of predefined constraints (such as 'alldifferent') and powerful search commands (such as 'labeling') to solve the combinatorial problems. The predefined constraints and exhaustive depth-first search procedure aim at solving a general class of constraint satisfaction problems. These facilities ease the development of knowledge intensive task such as expert system shell rapid prototyping.

3. The Development of Expert System Shell

3.1. Logic-based Implementation

The design of our rapid development of expert system shell has been graphically shown in Figure 2. The module to be used by the users is the 'menuask' module. Inference engine performs load and solve actions for loading the knowledge base into the working memory and solving the questions ask by the users, respectively. The working storage performs action related to reasoning and giving explanation to the users.

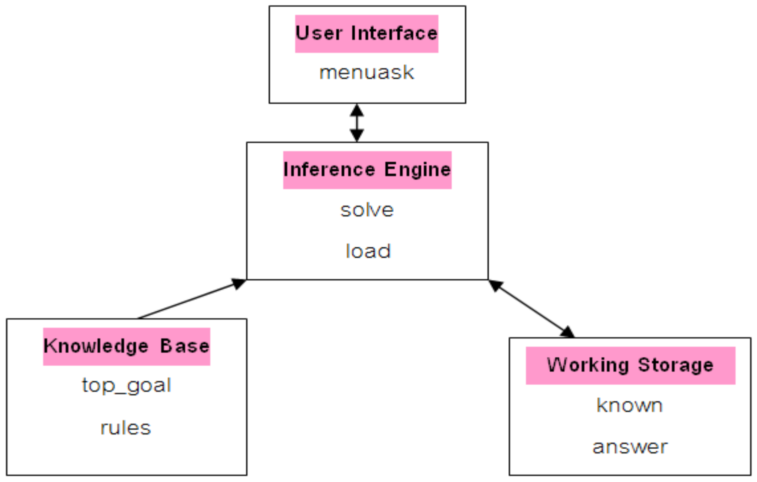
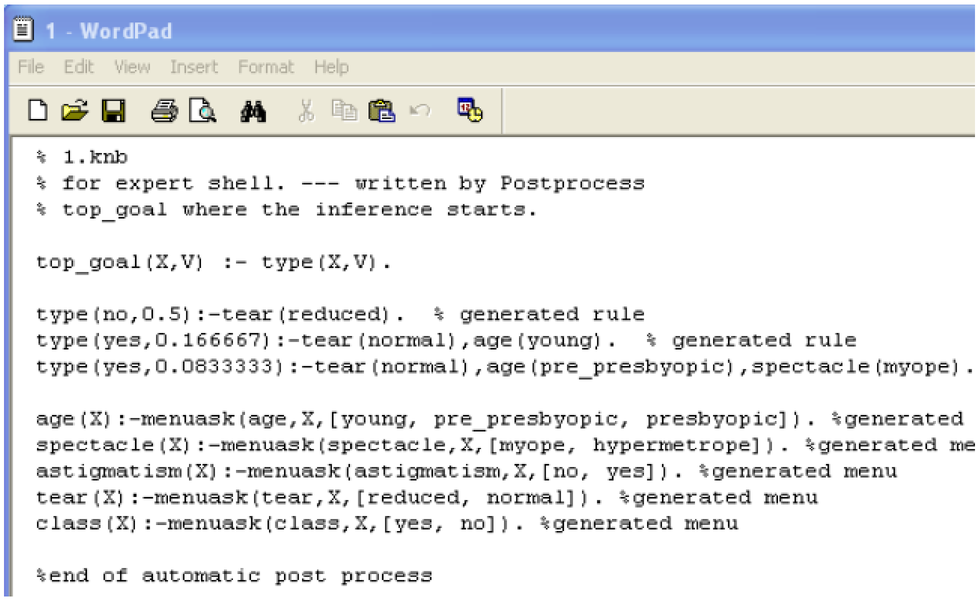


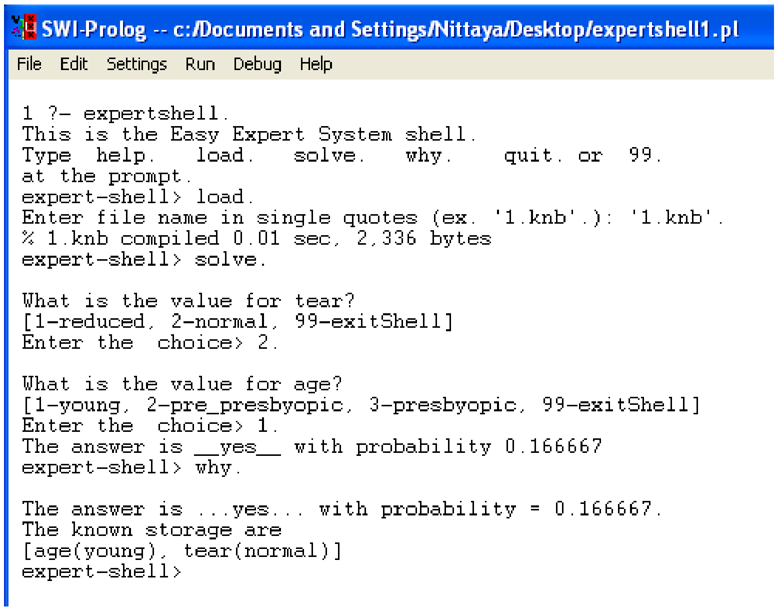
Figure 2. Structure of the Expert System Shell

The module 'top_goal' is the top module of the knowledge base that the inference engine will use in the process of searching for an appropriate recommendation. The first step of using the expert shell is to load the knowledge base into working memory. This is simply done through calling the module 'load'. The knowledge base file is then compiled. To start asking question, users have to invoke the system with the 'solve' command. Given the knowledge base of contact lens wearing recommendation as shown in Figure 3, the expert system processes the recommendation steps through a series of interactive statements as illustrated in Figure 4.



```
% 1.knb
% for expert shell. --- written by Postprocess
% top_goal where the inference starts.

top_goal(X,V) :- type(X,V).

type(no,0.5):-tear(reduced).  % generated rule
type(yes,0.166667):-tear(normal),age(young).  % generated rule
type(yes,0.0833333):-tear(normal),age(pre_presbyopic),spectacle(myope).

age(X):-menuask(age,X,[young, pre_presbyopic, presbyopic]). %generated
spectacle(X):-menuask(spectacle,X,[myope, hypermetrope]). %generated me
astigmatism(X):-menuask(astigmatism,X,[no, yes]). %generated menu
tear(X):-menuask(tear,X,[reduced, normal]). %generated menu
class(X):-menuask(class,X,[yes, no]). %generated menu

%end of automatic post process
```

Figure 3. Knowledge Base of Contact Lens Wearing


SWI-Prolog -- c:/Documents and Settings/Nittaya/Desktop/expertshell1.pl
File Edit Settings Run Debug Help

```
1 ?- expertshell.
This is the Easy Expert System shell.
Type  help.   load.   solve.   why.    quit. or  99.
at the prompt.
expert-shell> load.
Enter file name in single quotes (ex. '1.knb'.): '1.knb'.
% 1.knb compiled 0.01 sec, 2,336 bytes
expert-shell> solve.

What is the value for tear?
[1-reduced, 2-normal, 99-exitShell]
Enter the  choice> 2.

What is the value for age?
[1-young, 2-pre_presbyopic, 3-presbyopic, 99-exitShell]
Enter the  choice> 1.
The answer is __yes__ with probability 0.166667
expert-shell> why.

The answer is ...yes... with probability = 0.166667.
The known storage are
[age(young), tear(normal)]
expert-shell>
```

Figure 4. Expert System Shell Running with SWI-Prolog

Program coding for a logic-based expert system shell implementation is given as follows:

```
:-dynamic known/1, answer/2.
expertshell :-    greeting, repeat, write('expert-shell> '),
                  read(X), do(X),
                  (X == quit; X == 99),  writeln('>>>>Goodbye, see you later<<<<'), !.
greeting :-  write('This is the Easy Expert System shell.'), nl,
             native_help.
do(help) :- native_help, !.
do(load) :- load_kb, !.
do(solve) :- solve, !.
do(why) :-why, !.
do(quit).  do(99).
do(X) :-  write(X), write(' is not a legal command.'), nl, fail.
native_help :-   write('Type  help.   load.   solve.   why.    quit. or  99.'),nl,
                 write('at the prompt.'), nl.
load_kb :-    write('Enter file name in single quotes (ex. ''1.knb''.): '),
              read(F),
              reconsult(F).
solve :- retractall(known( _) ), retractall(answer(_,_)),
          top_goal(X,V),
          format('The answer is __~w__ with probability ~w',[X,V]),
          assert(answer(X,V)), nl.
solve :-  write('No answer found.'), nl.
menuask(Pred,Value,Menu) :- menuask(Pred,Menu),
                            atomic_list_concat([Pred,'(',Value,')'],X),
                            term_to_atom(T,X),known(T),!.
menuask(Pred,_) :- atomic_list_concat([Pred,'(','_',')'],X),
                   term_to_atom(T,X),known(T),!.     % not ask again
menuask(Attribute,Menu) :- nl, write('What is the value for '), write(Attribute),write('?'), nl,
                           addchoice(Menu,MenuRes), writeln(MenuRes),
                           write('Enter the  choice> '),  read(C),
                           member(C-V,MenuRes),
                           (C=99 -> abort ; true),
                           atomic_list_concat([Attribute,'(',V,')'],X),
                           term_to_atom(T,X),
                           asserta(known(T)).
why :- answer(A,V),
        format('~nThe answer is ...~w... with probability = ~w.~n',[A,V]),
        findall( X , known(X),Result),
        writeln('The known storage are'),
        writeln(Result).
addchoice(X,Res) :- length(X,Len),
                    numlist(1,Len,NumL),
                    map(NumL,X,Res).
map([],[],[99-exitShell]).
map([H|T],[X|TT],[H-X|T1]) :- map(T,TT,T1).

 % ====== End of Expert System Shell Program =========
```

3.2. Constraint-based Implementation

The design of constraint-based expert system shell (as shown in Figure 5) is slightly different from the logic-based method. Constraint features have been added in the user interface, inference engine, and knowledge base parts. The flowcharts of ‘solve’ and ‘why’, which are modules to solve the user’s query and then explain recommendation, respectively, are given in Figure 6.

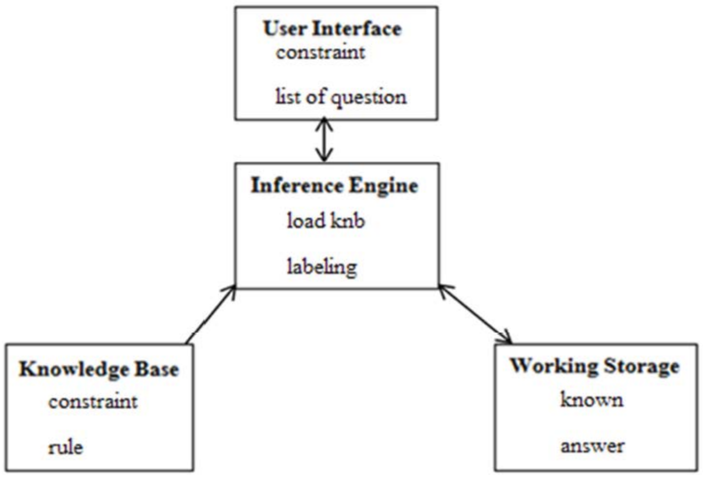


Figure 5. Structure of the Constraint-based Expert System Shell

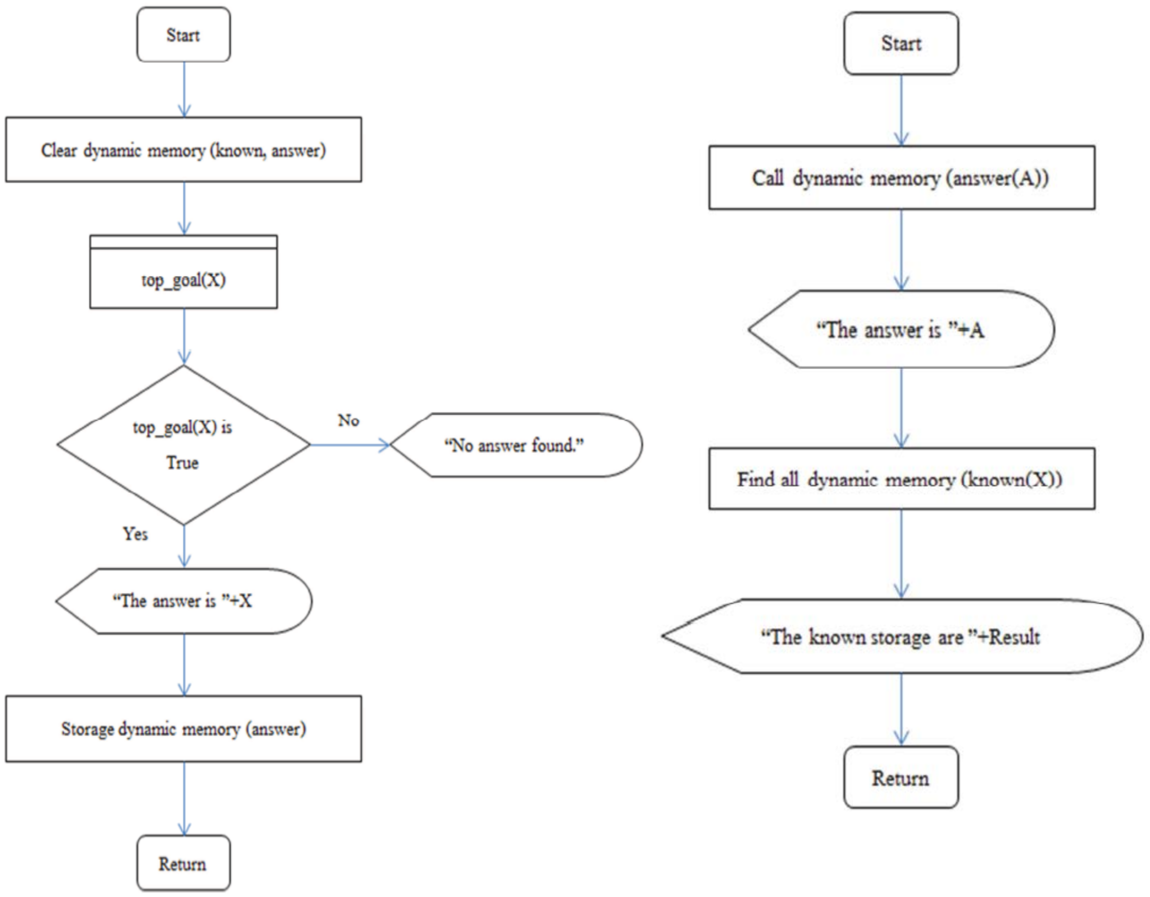
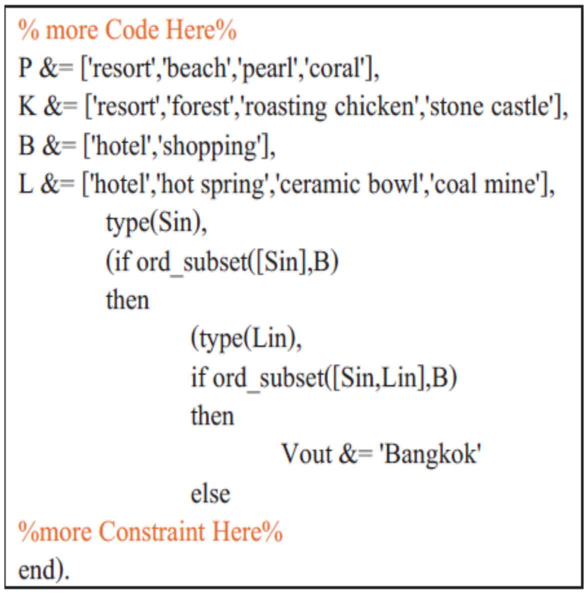

Start
Clear dynamic memory (known, answer)
top_goal(X)
top_goal(X) is True
No
"No answer found."
Yes
"The answer is "+X
Storage dynamic memory (answer)
Return
Start
Call dynamic memory (answer(A))
"The answer is "+A
Find all dynamic memory (known(X))
"The known storage are "+Result
Return

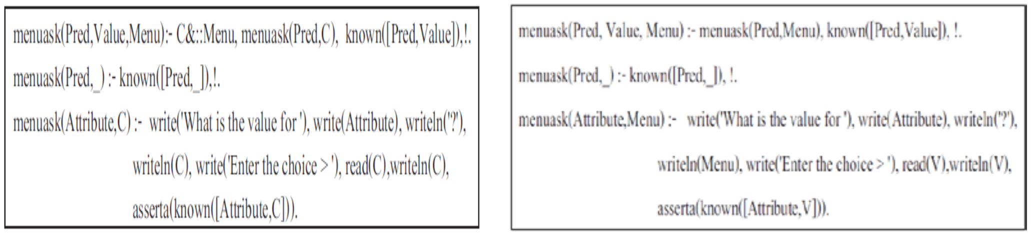
Figure 6. The 'solve' (left) and 'why' (right) Flowcharts

The knowledge base content in our constraint-based implementation has been changed from the rule-based format 'IF-THEN' to the constraint clauses, as shown in Figure 7. This simple knowledge base contains preferences of tourists. The 'menuask' module is also modified to include domain constraint. A comparative implementation of this module is demonstrated in Figure 8. Example of program running using the tourist attraction site knowledge base can be illustrated in Figure 9.



```
% more Code Here%
P &= ['resort','beach','pearl','coral'],
K &= ['resort','forest','roasting chicken','stone castle'],
B &= ['hotel','shopping'],
L &= ['hotel','hot spring','ceramic bowl','coal mine'],
        type(Sin),
        (if ord_subset([Sin],B)
        then
                (type(Lin),
                if ord_subset([Sin,Lin],B)
                then
                        Vout &= 'Bangkok'
                else
%more Constraint Here%
end).
```

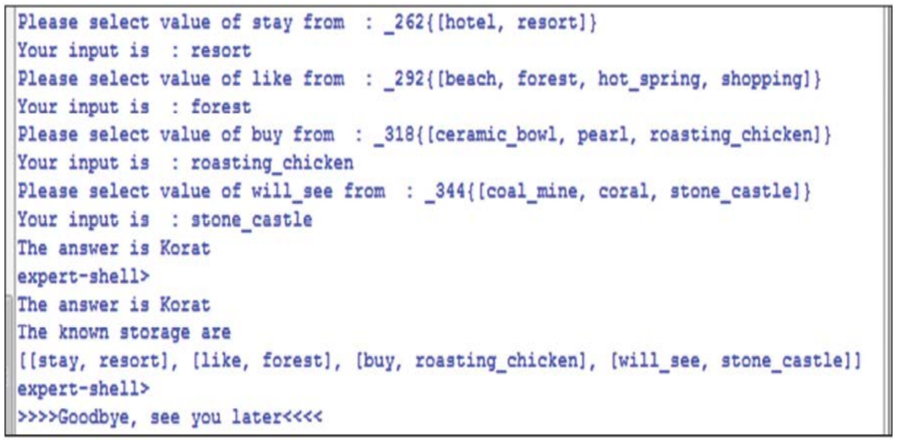
Figure 7. Knowledge base of the Constraint Expert System Shell



```
menuask(Pred,Value,Menu):- C&::Menu, menuask(Pred,C),  known([Pred,Value]),!.
menuask(Pred,_) :- known([Pred,_]),!.
menuask(Attribute,C) :-  write('What is the value for '), write(Attribute), writeln('?'),
                writeln(C), write('Enter the choice > '), read(C),writeln(C),
                asserta(known([Attribute,C])).
```

```
menuask(Pred, Value, Menu) :- menuask(Pred,Menu), known([Pred,Value]), !.
menuask(Pred,_) :- known([Pred,_]), !.
menuask(Attribute,Menu) :-   write('What is the value for '), write(Attribute), writeln('?'),
                writeln(Menu), write('Enter the choice > '), read(V),writeln(V),
                asserta(known([Attribute,V])).
```

Figure 8. The 'menuask' Module in CLP (left) and LP (right)



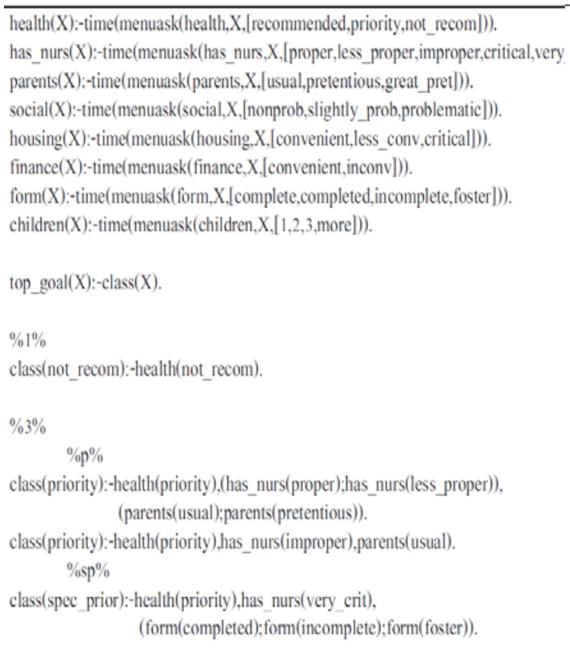
```
Please select value of stay from  : _262{[hotel, resort]}
Your input is  : resort
Please select value of like from  : _292{[beach, forest, hot_spring, shopping]}
Your input is  : forest
Please select value of buy from  : _318{[ceramic_bowl, pearl, roasting_chicken]}
Your input is  : roasting_chicken
Please select value of will_see from  : _344{[coal_mine, coral, stone_castle]}
Your input is  : stone_castle
The answer is Korat
expert-shell>
The answer is Korat
The known storage are
[[stay, resort], [like, forest], [buy, roasting_chicken], [will_see, stone_castle]]
expert-shell>
>>>>Goodbye, see you later<<<<
```

Figure 9. Running Result of Constraint Expert System Shell

4. Experimentation and Results

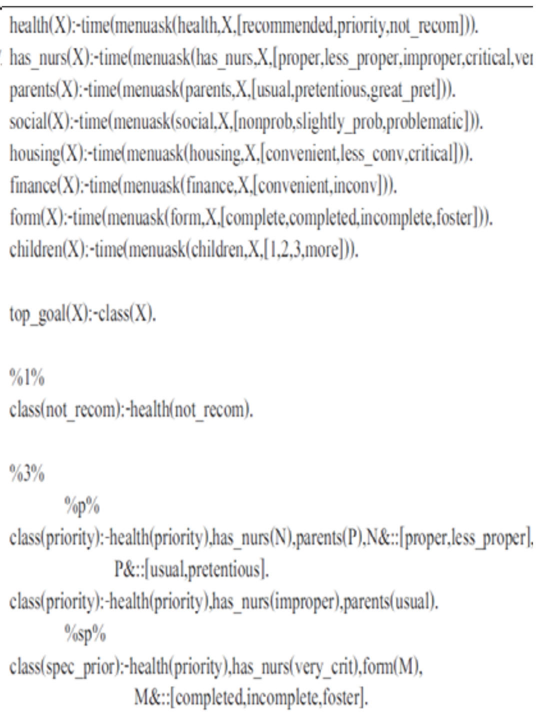
4.1. Experimental Setting

On comparing performances of logic-based and constraint-based expert system shell implementation, both schemes are coded and executed in the same environment using ECLiPSe constraint system (www.eclipseclp.org). We then test the programs with the knowledge base obtained from the nursery data of the UCI repository (http://archive. ics.uci.edu/ml/ datasets/Nursery). The nursery database contains ranking information (not recommmend, recommend, very recommend, priority, spec_prior) from the applications applied to nursery school. The ranking outcomes are considered from several attributes: occupation of parents, child's nursery, family structure, financial standings, social and health conditions of the family. Expert systems in LP (using Prolog) and CLP (using ECLiPSe) schemes of this specific domain can be shown (some part) in Figures 10 and 11, respectively.



```
health(X):-time(menuask(health,X,[recommended,priority,not_recom])).
has_nurs(X):-time(menuask(has_nurs,X,[proper,less_proper,improper,critical,very
parents(X):-time(menuask(parents,X,[usual,pretentious,great_pret])).
social(X):-time(menuask(social,X,[nonprob,slightly_prob,problematic])).
housing(X):-time(menuask(housing,X,[convenient,less_conv,critical])).
finance(X):-time(menuask(finance,X,[convenient,inconv])).
form(X):-time(menuask(form,X,[complete,completed,incomplete,foster])).
children(X):-time(menuask(children,X,[1,2,3,more])).

top_goal(X):-class(X).

%1%
class(not_recom):-health(not_recom).

%3%
        %p%
class(priority):-health(priority),(has_nurs(proper);has_nurs(less_proper)),
                  (parents(usual);parents(pretentious)).
class(priority):-health(priority),has_nurs(improper),parents(usual).
        %sp%
class(spec_prior):-health(priority),has_nurs(very_crit),
                     (form(completed);form(incomplete);form(foster)).
```

Figure 10. Logic-based Expert System for Nursery Problem



```
health(X):-time(menuask(health,X,[recommended,priority,not_recom])).
has_nurs(X):-time(menuask(has_nurs,X,[proper,less_proper,improper,critical,ve
parents(X):-time(menuask(parents,X,[usual,pretentious,great_pret])).
social(X):-time(menuask(social,X,[nonprob,slightly_prob,problematic])).
housing(X):-time(menuask(housing,X,[convenient,less_conv,critical])).
finance(X):-time(menuask(finance,X,[convenient,inconv])).
form(X):-time(menuask(form,X,[complete,completed,incomplete,foster])).
children(X):-time(menuask(children,X,[1,2,3,more])).

top_goal(X):-class(X).

%1%
class(not_recom):-health(not_recom).

%3%
        %p%
class(priority):-health(priority),has_nurs(N),parents(P),N&::[proper,less_proper],
                 P&::[usual,pretentious].
class(priority):-health(priority),has_nurs(improper),parents(usual).
        %sp%
class(spec_prior):-health(priority),has_nurs(very_crit),form(M),
                     M&::[completed,incomplete,foster].
```

Figure 11. Constraint-based Expert System for Nursery Problem

4.2. Results

To test computational time of the expert system shell on the nursery knowledge base, we apply the time function from the utility library of ECLiPSe system by including a directive command ":- lib(util)" as a first line of both the logic-based and constraint-based programs. Timing function can then be performed through the built-in time predicate, "time(top_goal(X))".

Program performances of logic-based and constraint-based implementation are compared in terms of computational time (or percentage CPU usage observable from the resource monitoring system software) to deduce solution from the knowledge base (Figure 12), running time that starts from accepting user's query until showing recommendation (Figure 13), file sizes of the knowledge base and the expert system shell (Figure 14), and lines of code of the knowledge base parts and the expert system shell part (Figure 15).

Running time comparisons, presented in Figure 13, are the results from a series of experimentation performed with the call on predicates expertshell, top_goal, and menuask. These predicates are implemented in both the logic-based and constraint-based paradigms, and the expert system shell of both paradigms can consult knowledge base that either in the form of Prolog language or CLP language. We thus design the three kinds of experimentation:

(1) PL-PL means to process the logic-based expert system shell with the logic-based knowledge,

(2) CLP-CLP means to process the constraint-based expert system shell with the constraint-based knowledge, and

(3) CLP-PL means to process the constraint-based expert system shell with the Prolog-based knowledge.

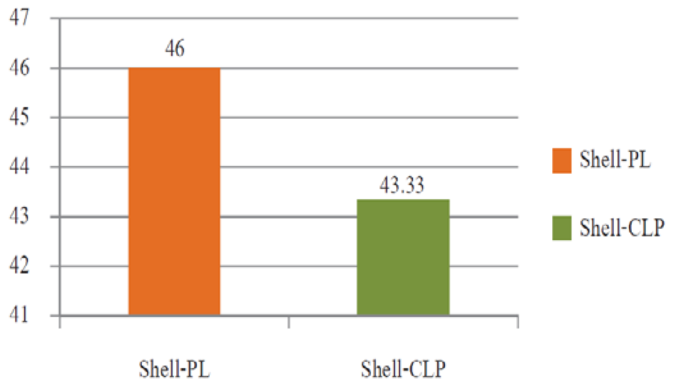


Figure 12. Program Performance Comparison in Terms of Computational Time (% CPU usage, averaged from three experiments) during Knowledge base Consulting and Knowledge Inferring of the Expert System Shell

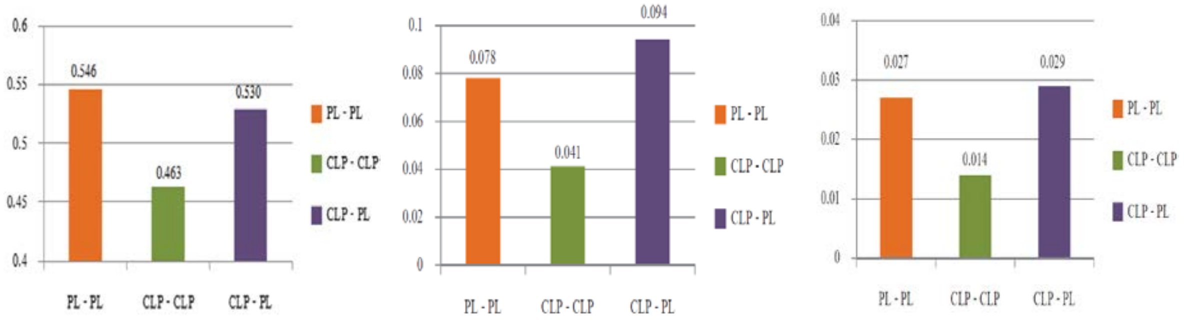


Figure 13. Time Comparison including Interactive User Interfacing Time (in seconds) observed from the calling of Predicates expertshell (left), top_goal (middle), and menuask (right)

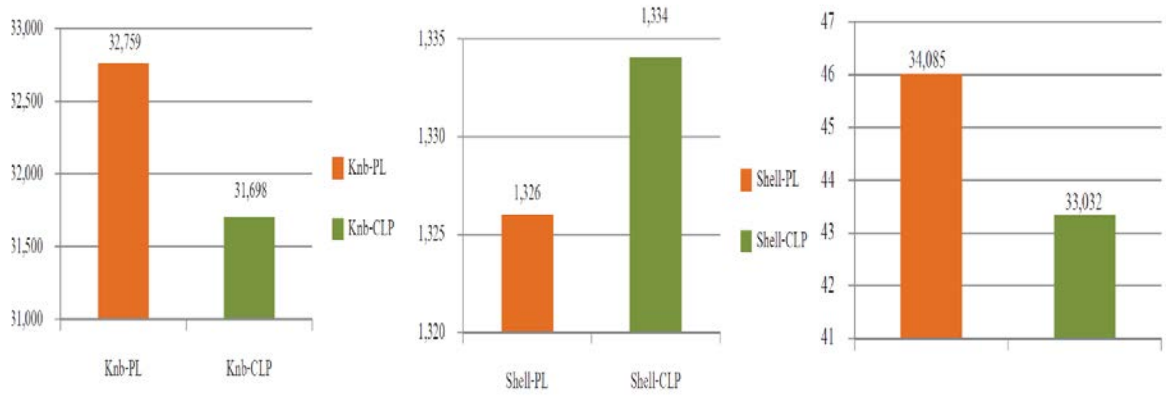


Figure 14. File Size Comparison (in bytes) of LP versus CLP on the Knowledge Base Part (left), the Expert System Shell Part (middle), and the Integrated Part of Knowledge Base and Inference Engine (right)

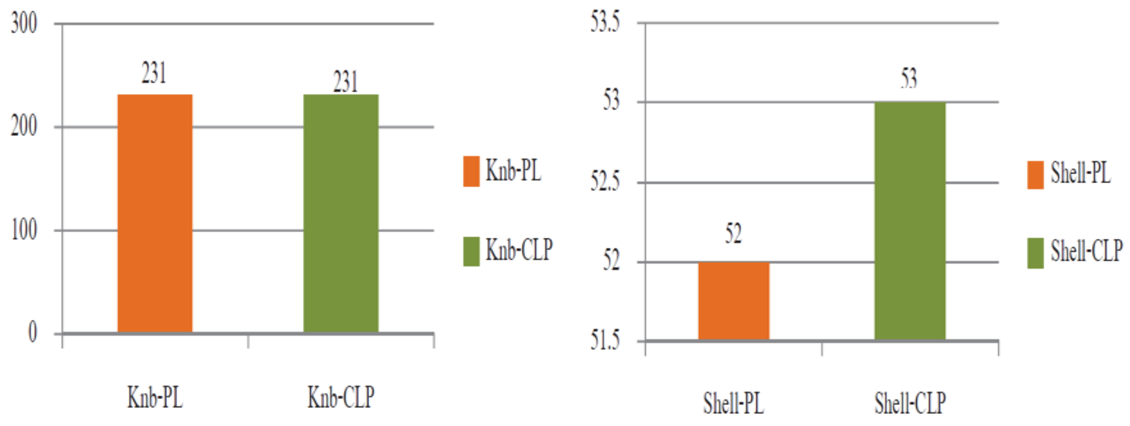


Figure 15. Line-of-code Comparison (in unit: number of lines) of LP versus CLP on the Knowledge Base Part (left) and the Expert System Shell Part (right)

From the experiments, we can observe that the program implemented with constraint-based paradigm consumes less percentage of CPU resource. The running time of CLP program on the CLP knowledge base is faster than the logic-based paradigm using Prolog language. The constraint-based formalism of encoding knowledge is also shorter than the pure logic-based formalism. But to compare the expert system shell part, we observe that the CLP coding is longer with a little bit bigger file size than the Prolog coding.

5. Conclusion

We present in this paper a comparative study of rapid prototyping the expert system shell using logic and constraint based programming paradigms. The main purpose is to observe program coding difficulty, running and memory usage behavior. The insight understandability is expected to be fundamental knowledge for designing constraint solver that is more appropriate for the expert system shell development.

Knowledge base contents are conventionally constructed by knowledge engineers who are not an expert of the specific domains. With constrain-based method, the knowledge elicitation task is expected to be less error prone. The constraint paradigm is also planned to be used as automatic knowledge extraction scheme to learn specific knowledge from stored experiences and expertise.

Acknowledgements

This research was supported by the SUT Research and Development Fund, Suranaree University of Technology.

References

[1] K. R. Apt and M. Wallace, “Constraint Logic Programming using ECLiPSe”, Cambridge University Press, **(2007)**.
[2] M. Bavarian and V. Dahl, “Constrained based methods for biological sequence analysis”, Journal of Universal Computer Science, vol. 12, no. 11, 2006, **(2006)**, pp. 1500-1520.
[3] S. Bistarelli and F. Bonchi, “Soft constraint based pattern mining”, Data and Knowledge Engineering, vol. 62, **(2007)**, pp. 118-137.
[4] I. Bratko, “Prolog Programming for Artificial Intelligence”, 3rd ed., Pearson, **(2001)**.
[5] L. De Raedt, T. Guns and S. Nijssen, “Constraint programming for itemset mining”, Proceedings of KDD Conference, **(2008)**, pp. 204-212.
[6] A. Fallahi and S. Jafari, “An expert system for detection of breast cancer using data preprocessing and Bayesian network”, International Journal of Advanced Science and Technology, vol. 34, **(2011)**, pp. 65-70.
[7] E. A. Feigenbaum, “Expert systems: principles and practice”, The Encyclopedia of Computer Science and Engineering, Stanford: Knowledge Systems Laboratory, **(1992)**.
[8] E. Flior, T. Anaya, C. Moody, M. Beheshti, H. Jianchao and K. Kowalski, “A knowledge-based system implementation of intrusion detection rules”, Proceedings of the 7th International Conference on Information Technology: New Generations, **(2010)**, pp. 738-742.
[9] J. C. Giarratans and G. D. Riley, “Expert systems: Principles and Programming”, fourth edition, Canada: Thomason Learning, **(2005)**.
[10] R. Girratano, “Expert Systems”, Principles and Programming, PWS, **(1998)**.
[11] L.-C. Huang, S.-S. Tseng and Y.-S. Chu, “Building a wafer fab lot scheduling knowledge-based system”, WSEAS Transactions on Information Science and Applications, vol. 3, no. 10, pp. 1994-2001, **(2006)**.
[12] N. Kerdprasop and K. Kerdprasop, “Frequent pattern discovery with constraint logic programming”, International Journal of Mathematical Models and Methods in Applied Sciences, vol. 5, no. 8, **(2011)**, pp. 1345-1353.
[13] J. Lee, B. Song, T. Kim, D. Seo and S. Bae, “A design and implementation of u-health diagnosis system using expert system and neural network”, International Journal of Future Generation Communication and Networking, vol. 1, no. 1, **(2008)**, pp. 83-90.
[14] C. Leon, F. Biscarri, I. Monedero, J. I. Guerrero, J. Biscarri and R. Millan, “Integrated expert system applied to the analysis of non-technical losses in power utilities”, Expert Systems with Applications, vol. 38, **(2011)**, pp. 10274-10285.
[15] S.-H. Liao, “Expert system methodologies and applications-a decade review from 1995 to 2004”, Expert Systems with Applications, vol. 28, **(2005)**, pp. 93-103.
[16] R. M. Mateo, J. Lee and B. D. Gerardo, “Healthcare expert system based on group cooperation model”, International Journal of Software Engineering and Its Applications, vol. 2, no.1, **(2008)**, pp. 105-116.
[17] R. A. Miller, H. E. Pople and J. D. Myers, “INTERNIST-1, An experimental computer-based diagnostic consultant for general internal medicine”, New England Journal of Medicine, vol. 307, no. 8, **(1982)**, pp. 468-476.
[18] C. Rigotti, I. Mitasiunaite, J. Besson, L. Meyniel, J. F. Boulicaut and O. Gandrillon, “Using a solver over the string pattern domain to analyze gene promoter sequences”, S. Dzeroski (ed.), Inductive Databases and Constraint-Based Data Mining, Springer-Verlag, **(2010)**, pp. 407-423.
[19] M. Seda, “Solving resource-constrained project scheduling problem as a sequence of multi-knapsack problems”, WSEAS Transactions on Information Science and Applications, vol. 3, no. 10, **(2006)**, pp. 1785-1791.
[20] W. B. Schwartz, “Medicine and the computer: The promise and problems of changes”, New England Journal of Medicine, vol. 283, **(1970)**, pp. 1257-1264.

[21] E. H. Shortliffe, “Computer-Based Medical Consultations: MYCIN”, Elsevier, **(1976)**.
[22] G. Y.-C. Wong, K. Y. Wong and K. H. Yeung, “Solving 3-SAT using constraint programming and fail detection”, WSEAS Transactions on Computers, vol. 4, no. 2, **(2005)**, pp. 148-153.
[23] S. Yang, H. Kuma and P. Park, “An expert system for wide area surveillance based on ontology”, International Journal of Database Theory and Application, vol. 5, no. 2, **(2012)**, pp. 1-16.
[24] L. Zhao, T. Gu and J. Qian, “A goal-independent constraint-based correct partial answers semantics”, WSEAS Transactions on Computers, vol. 6, no. 6, **(2007)**, pp. 947-952.

Authors

Nittaya Kerdprasop is an associate professor at the School of Computer Engineering, Suranaree University of Technology, Thailand. She received her bachelor degree in Radiation Techniques from Mahidol University, Thailand, in 1985, master degree in Computer Science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in Computer Science from Nova Southeastern University, U.S.A, in 1999. Her research of interest includes Knowledge Discovery in Databases, Artificial Intelligence, Logic Programming, and Intelligent Databases.

Kanjana Intharachatorn is a computer engineer and research assistant with the Data Engineering Research Unit. She received her bachelor degree in Computer Engineering from Suranaree University of Technology (SUT), Thailand, in 2009, and master degree in Computer Engineering from SUT in 2012. Her current research includes Data Mining, Constraint Logic Programming, and Artificial Intelligence.

Kittisak Kerdprasop is an associate professor and chair of the School of Computer Engineering, Suranaree University of Technology, Thailand. He received his bachelor degree in Mathematics from Srinakarinwirot University, Thailand, in 1986, master degree in Computer Science from the Prince of Songkla University, Thailand, in 1991 and doctoral degree in Computer Science from Nova Southeastern University, U.S.A., in 1999. His current research includes Data mining, Artificial Intelligence, Functional and Logic Programming Languages, Computational Statistics.