Asian Journal of Information Technology 6 (4): 502-505, 2007

Implementation of an Intelligent Interpretative Software System

O.V. Viktorov and Afif Mghawish
P.O. Box 130, Amman 11733, Jordan

Abstract: An intelligent interpretative software system has been developed and tested. A hierarchical algorithm is used to implement indirect access to operations and data. The resulting system allows choice of suitable procedure of search with specific emphasis on heuristic algorithm variations.

Key words: Intelligence, decision making, logical, heuristic, search, operations

INTRODUCTION

A large amount of research on the most efficient techniques to store and retrieve data has been done and the associated problems now have satisfactory solutions. However, the problem of interpreting this large amount of information remains. A framework in which this problem may be attacked is given by the field of deductive databases. Deductive databases not only store explicit information in the manner of a relational database, but they also store rules that enable inferences based on the stored data to be made. Deductive systems, given via axioms and rules of inference, are a common conceptual tool in mathematical logic and computer science. They are used to specify many varieties of logics and logical theories as well as aspects of programming languages such as type systems or operational semantics (Lutz, 2003a, b, 2002; Lars and Peter, 1991; Robert, 1990).

BACKGROUND

By applying the inference rules deeply, logical expressions can be manipulated starting from their sub-expressions. This way, we can simulate analytic proofs in traditional deductive formalisms. Furthermore, we can also construct much shorter analytic proofs than in these other formalisms. However, deep applicability of inference rules causes much greater non-determinism in proof construction.

By redesigning the deep inference deductive systems, some redundant applications of the inference rules are prevented. By introducing a new technique which reduces non-determinism, it becomes possible to obtain a more immediate access to shorter proofs, without breaking certain proof theoretical properties such as cute limitation. Different implementations presented in this thesis allow performing experiments on the techniques that we developed and observe the performance improvements. Within a computation-as-proof-search perspective, we use deep inference deductive systems to develop a common proof-theoretic language to the 2 fields of planning and concurrency.

Open deductive system generalized logical structure: The block-diagram of generalized logic structure of open deductive system is shown in Fig. 1.

This structure is based on the following main principles:

- Program modules combine in blocks according to generality of tasks, which they solve.
- The hierarchical organization of modules in each block.
- Indirect access to internal disjunction of propositions representation, by means of the block which carries out a set of given operations (functions).

Let us discuss the basic functions of deductive system and relate them to the suggested logic structure.

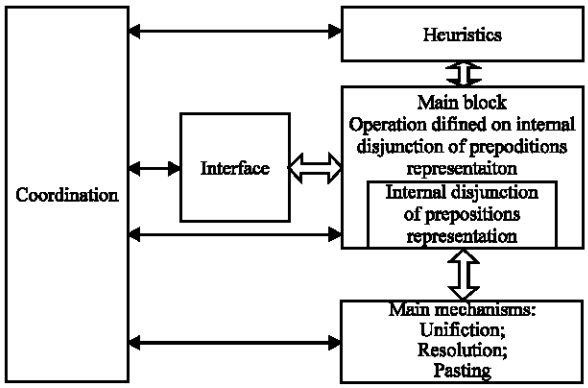

Fig. 1: Generalized logical structure of open deductive system

Corresponding Author: O.V. Viktorov, P.O. Box 130, Amman 11733, Jordan

```
struct node  /* internal clause representation */
{
char lit[PWRL];  /* pointer array */
struct node *branch (Lutz, 2003);
};
```

Fig. 2: Structure node

The main block of this system combines the internal disjunction of propositions representation and the operations set defined on it, hence it determines flexibility of all system. For satisfaction of various requirements, each disjunction of propositions set is represented as a linked list. Each unit of the list contains one disjunction of propositions and is defined by structure *node* (Fig. 2).

The unit consists of an array of pointers to the character string where each string contains one letter from given disjunction of propositions and also it contains the pointer to the following element of the list together with the pointer to the following allocated block of memory. The block of memory is dynamically allocated by system for each unit and the string linked to it. The basic operations set defined on internal disjunction of propositions representation is the following:

append - To add a new disjunction of propositions to the given disjunction of propositions set;

cpy_clause- To copy specified disjunction of propositions;

conc_clause- To concatenate given disjunction of propositions;

rmv_lit- To remove from given disjunction of propositions all predicates which are matched to the certain pattern;

rmv_dlit- To remove multiple copies of one predicate;

cmp_queue- To compare two lists and to return an attribute of comparison;

separate- To mark one of terms from specified predicate;

d_clsuse- To remove specified disjunction of propositions;

free_all- To release the allocated memory.

The block of the basic mechanisms carries out the following functions:

prove- To implement a strategy ; if it is proved that given disjunction of propositions subset is non-contradictive the function returns back *BOX* and in opposite case it returns *NULL*; (Fig. 3).

```
int  prove(n, p, sn, md)
struct n_node  **n,  **sn,  **md;
struct p_node  **p;
{
int rlsv( ) , rlsv_ord( ), cmp_queue( );
int j;
struct p_node  *ptr
struct n_node  *n_ptr
sruct n_node  **ptr1,  **ptr2,  **ptr3;
ptr = *p
while(TRUE)
        {
        ptr2 = md; ptr1 = sn
        while(*ptr1 != NULL)
                {
                 *ptr2 = NUL;
                 if(rslv_ord(p, 1, ptr1, ptr2) == BOX) return BOX;
                 ptr3 = ptr1; ptr1 = ptr2; ptr2 = ptr3
                }
        *p = *(p + 1) = *(p + 2);
        if(cmp_queue(*p, prt) == TRUE return NULL;
        ptr = *p; *sn = NULL;
        if(rslv(p, 0, n, sn) == BOX return BOX;
        }
}
```

Fig. 3: Function *prove ()* for positive hyperresolution

rslv- Generates resolvent set (the letters contained the most significant predicate symbols are resolved); if it is proved that given disjunction of propositions subset is non-contradictive the function returns back *BOX* and in opposite case it returns *NULL*;

rlsv_ord- Generates ordered resolvent set (the letters contained the most significant predicate symbols are resolved); if it is proved that given disjunction of propositions subset is non-contradictive the function returns back *BOX* and in opposite case it returns *NULL*;

extract- Resolve 2 disjunction of propositions and checks the empty disjunction of propositions or the predicate-answer (returns back *BOX* if the result of check is positive and otherwise it returns *NULL*;

unify- Search most the general unifier of two given letters (returns *TRUE* if the unifier is found, otherwise returns *NULL)*;

compose- Carries out a composition of an unifier with set single-element substitution (returns *TRUE* if operation is executed successfully, otherwise it returns *NULL)*;

substn_all- Carries out replacement of arguments of all predicates in given disjunction of propositions according to most general unifier;

substn_p- Carries out replacement of the given predicate according to most general unifier;

substn_t- Carries out the certain substitution in the given term.

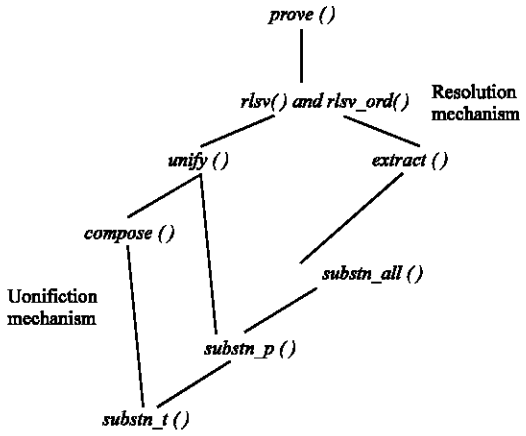

Fig. 4: Functional level and links in main mechanism block

The functional level hierarchy of the basic mechanisms block is shown in Fig 4. At the highest level there is a function *prove ( ).*

The only *prove( )* function depends strongly on the chosen strategy. Hence, experiments show that strategy change does not cause big overhead. Other functions can be grouped depending on mechanisms they employs, so any changes in mechanism, does not affect functions of other group. Function dependence on the certain mechanism decreases with reduction of its level that is direct consequence of the hierarchical approach.

The interface block provides information interchange with external devices. Depending on given requirements, it is possible to increase interactive opportunities of system. If necessary, some its parts can be described, using logic programming languages, for example, Prolog or LISP.

The openness of system allows to investigate and to compare various heuristics. The structure of system allows finding suitable heuristic procedures easily.

At last, the coordination block includes internal language sources that control computation process.

Example: Let us present an example that shows how the simplest deductive system works. In one room there is the monkey, a chair and a banana attached to the ceiling. The monkey can move in the room, transfer a chair and clime up it. The monkey would like to get a banana very much, but she can satisfy her wish, if she a chair in the certain place (Fig. 5). What strategy of the monkey's behavior should be?

The following predicates are in use:

P (X, Y, Z, S): In state S the monkey is in point X, a banana - in point Y and a chair is in point Z;

Input sequence:
~ P(XYZS), P(ZYZw(XZS))
~ P(XYZS), P(YYYc(XZS))
~ P(bbbS), R(u(S))
P(abcs)
~ R(S), Z(S)
Resolvents:
P(cbcw(acs))
P(bbbc(cbw(acs)))
R(u)c)cbw(acs)))
Z(u(c(cbw(acs)))

Fig. 5: Monkey-banana deductive system

R (S): in state S the monkey can get a banana;

Z (S): it is an answer.

Functions: W (Y, Z, S): generates the state that comes after state S when the monkey has moved from point Y to point Z.

c (Y, Z, S): Generates the state that comes after the state S and the monkey has moved from point Y to point Z, having takes the chair;

u (S): Generates state that comes after state S when the monkey climes up the chair.

First three disjunctions of propositions describe rules of monkey behavior in here world (room), the fourth disjunction of propositions is a description of an initial situation and the fifth corresponds to the presented question. The last disjunction of propositions contains the answer for it.

RESULTS

Authors have developed the base variant of deductive system that is characterized by a high degree of openness:

- A new module can be added easily;
- Mmodule replacement does not cause essential change of all system.

The openness of system allows to investigate and to compare various heuristics. The structure of system allows finding suitable heuristic procedures easily. The most suitable strategy can be chosen depending on presented system requirements and a tentative efficiency estimation of various strategies.

CONCLUSION

Application of the suggested principles of system design will allow time reducing of a logic conclusion and to make deductive systems more flexible in the respect to strategy changes and module replacement, so it provides their efficient practical usage.

REFERENCES

Lars Hallnas and Peter Schroeder-Heister, 1991. A Proof-theoretic Approach to Logic Programming, J. Logic and Computation, 1: 635-660.

Lutz Straßburger, 2002. A Local System for Linear Logic, Logic for Programming, Artificial Intelligence and Reasoning, LPAR (Matthias Baaz and Andrei Voronkov, Eds.) Lecture Notes in Artificial Intelligence, 2514: 388-402.

Lutz Straßburger, 2003. Linear Logic and Noncommutativity in the Calculus of Structures, PhD. Thesis, Technische Universit at Dresden.

Lutz Straßburger, 2003. MELL in the Calculus of Structures, Theoretical Computer Sci., 309: 213-285.

Robert Harper, 1990. Systems of Polymorphic Type Assignment in LF. Technical Report CMU-CS-90-144, Carnegie Mellon University, Pittsburgh, Pennsylvania.