6

# Intelligent Problem Solvers in Education: Design Method and Applications

Nhon Van Do
*University of Information Technology*
*Vietnam*

## 1. Introduction

In this chapter we present the method for designing intelligent problem solvers (IPS), especially those in education. An IPS, which is an intelligent system, can consist of AI-components such as theorem provers, inference engines, search engines, learning programs, classification tools, statistical tools, question-answering systems, machine-translation systems, knowledge acquisition tools, etc (Sowa, John F. 2002). An IPS in education (IPSE) considered here must have suitable knowledge base used by the inference engine to solve problems in certain knowledge domain, and the system not only give human readable solutions but also present solutions as the way teachers and students usually write them. Knowledge representation methods used to design the knowledge base should be convenient for studying of users and for using by inference engine. Besides, problems need to be modeled so that we can design algorithms for solving problems automatically and propose a simple language for specifying them. The system can solve problems in general forms. Users only declare hypothesis and goal of problems base on a simple language but strong enough for specifying problems. The hypothesis can consist of objects, relations between objects or between attributes. It can also contain formulas, determination properties of some attributes or their values. The goal can be to compute an attribute, to determine an object, a relation or a formula. After specifying a problem, users can request the program to solve it automatically or to give instructions that help them to solve it themselves. The second function of the system is "Search for Knowledge". This function helps users to find out necessary knowledge quickly. They can search for concepts, definitions, properties, related theorems or formulas, and problem patterns. By the cross-reference systems of menus, users can easily get knowledge they need.

Knowledge representation has a very important role in designing the knowledge base and the inference engine of the system. There are many various models and methods for knowledge representation which have already been suggested and applied in many fields of science. Many popular methods for knowledge representation such as logic, frames, classes, semantic networks, conceptual graphs, rules of inference, and ontologies can be found in George F. Luger (2008), Stuart Russell & Peter Norvig (2010), or in Sowa, John F. (2000). These methods are very useful in many applications. However, they are not enough and not easy to use for constructing an IPSE in practice. Knowledge representation should be convenient for studying of users and for using by inference engine. Besides, problems need to be modeled so that we can design algorithms for solving problems automatically and propose a simple language for specifying them. Practical intelligent systems expect more powerful and useful models for knowledge representation. The *Computational Object Knowledge Base* model (COKB) presented in Nhon Van Do (2010) will be used to design the system. This model can be used to represent the total knowledge and to design the knowledge base component of systems. Next, computational networks (Com-Net) and networks of computational objects (CO-Net) in Nhon Van Do (2009) and Nhon Van Do (2010) can be used for modeling problems in knowledge domains. These models are tools for designing inference engine of systems.

We used COKB model, CO-Net and Com-Net in constructing some practical IPSE such as the program for studying and solving problems in plane geometry presented in Nhon Van Do (2000) and Nhon Do & Hoai P. Truong & Trong T. Tran (2010), the system that supports studying knowledge and solving of analytic geometry problems, the system for solving algebraic problems in Nhon Do & Hien Nguyen (2011), the program for solving problems in electricity, in inorganic chemistry, etc. The applications have been implemented by using programming tools and computer algebra systems such as C++, C#, JAVA, and MAPLE. They are very easy to use for students in studying knowledge, to solve automatically problems and give human readable solutions agree with those written by teachers and students.

The chapter will be organized as follows: In Section 2, the system architecture and the design process will be presented. In Section 3, models for knowledge representation are discussed. Designing the knowledge base and the inference engine of an IPSE will be presented in Section 4. Some applications will be introduced in section 5. Conclusions and future works are drawn in Section 6.

## 2. System architecture and the design process

The structure of an IPSE are considered here consists of the components such as knowledge base, inference engine, interface, explanation component, working memory and knowledge manager. In this setion, these components will be studied together with relationships between them; we will also study and discuss how an IPSE runs, and present a process to construct the system together with methods and techniques can be used in each phase of the process.

### 2.1 Components of the system

An IPSE is also a knowledge base system, which supports searching, querying and solving problems based on knowledge bases; it has the structure of an expert system. We can design the system which consists of following components:

- The knowledge base.
- The inference engine.
- The explanation component.
- The working memory.

- The knowledge manager.
- The interface.

Knowledge Bases contain the knowledge for solving some problems in a specific knowledge domain. It must be stored in the computer-readable form so that the inference engine can use it in the procedure of automated deductive reasoning to solve problems stated in general forms. They can contain concepts and objects, relations, operators and functions, facts and rules.

The Inference engine will use the knowledge stored in knowledge bases to solve problems, to search or to answer for the query. It is the "brain" that systems use to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions. It must identify problems and use suitable deductive strategies to find out right rules and facts for solving the problem. In an IPSE, the inference engine also have to produce solutions as human reading, thinking, and writing.

The working memory contains the data that is received from the user during operation of the system. Consequents from rules in the knowledge base may create new values in working memory, update old values, or remove existing values. It also stores data, facts and rules in the process of searching and deduction of the inference engine.

The explanation component supports to explain the phases, concepts in the process of solving the problem. It presents the method by which the system reaches a conclusion may not be obvious to a human user, and explains the reasoning process that lead to the final answer of the system.

The knowledge manager aims to support updating knowledge into knowledge base. It also supports to search the knowledge and test consistence of knowledge.

The user interface is the means of communication between a user and the system problem-solving processes. An effective interface has to be able to accept the queries, instructions or problems in a form that the user enters and translate them into working problems in the form for the rest of the system. It also has to be able to translate the answers, produced by the system, into a form that the user can understand. The interface component of the system is required to have a specification language for communication between the system and learners, between the system and instructors as well.

The figure 1 below shows the structure of the system.

The main process for problem solving: From the user, a problem in a form that the user enter is input into the system, and the problem written by specification language is created; then it is translated so that the system receives the working problem in the form for the inference engine, and this is placed in the working memory. After analyzing the problem, the inference engine generates a possible solution for the problem by doing some automated reasoning strategies such as forward chaining reasoning method, backward chaining reasoning method, reasoning with heuristics. Next, The first solution is analyzed and from this the inference engine produces a good solution for the interface component. Based on the good solution found, the answer solution in human-readable form will be created for output to the user.

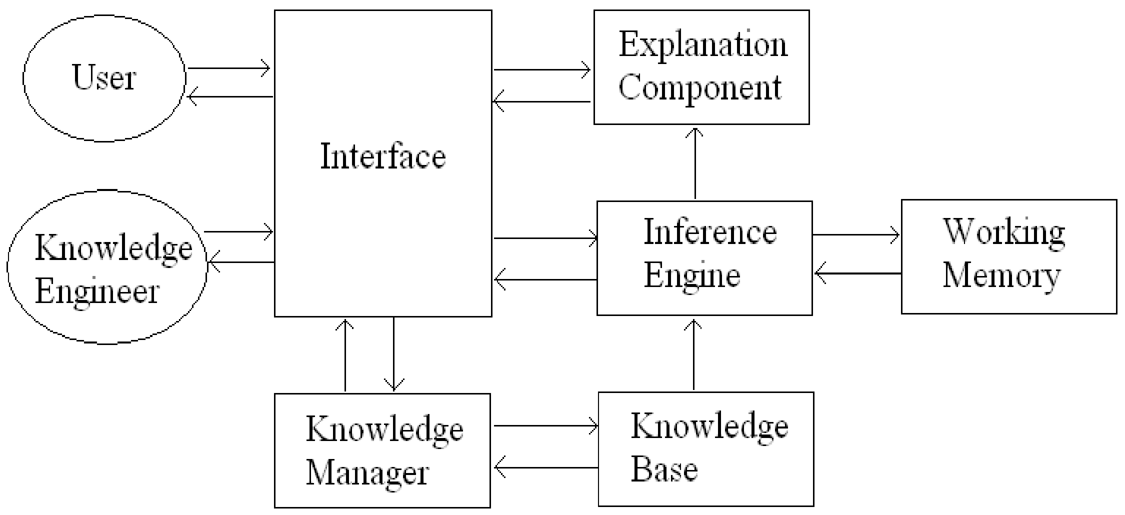

Fig. 1. Structure of a system

### 2.2 Design process

The process of analysis and design the components of the systems consists of the following stages.

**Stage 1:** Determine the knowledge domain and scope; then do collecting real knowledge consisting of data, concepts and objects, relations, operators and functions, facts and rules, etc. The knowledge can be classified according to some ways such as chapters, topics or subjects; and this classification help us to collect problems appropriately and easily. Problems are also classified by some methods such as frame-based problems, general forms of problems.

**Stage 2:** Knowledge representation or modeling for knowledge to obtain knowledge base model of the system. This is an important base for designing the knowledge base. Classes of problem are also modeled as well to obtain initial problem models.

The above stages can be done by using the COKB model, Com-Nets, CO-Nets, and their extensions. These models will be presented in section 3.

**Stage 3:** Establishing knowledge base organization for the system based on COKB model and its specification language. Knowledge base can be organized by structured text files. They include the files below.

- Files stores names of concepts, and structures of concepts.
- A file stores information of the Hasse diagram representing the component H of COKB model.
- Files store the specification of relations (the component R of COKB model).
- Files store the specification of operators (the component Ops of COKB model).
- Files store the specification of functions (the component Funcs of COKB model).
- A file stores the definition of kinds of facts.
- A file stores deductive rules.
- Files store certain objects and facts.

**Stage 4:** Modeling of problems and designing algorithms for automated reasoning. General problems can be represented by using Com-Nets, CO-Nets, and their extensions. The CO-Net problem model consists of three parts:

$$O = \{O_1, O_2, \ldots, O_n\},\ F = \{f_1, f_2, \ldots, f_m\},$$

$$Goal = [\ g_1, g_2, \ldots, g_m\ ].$$

In the above model the set O consists of n Com-objects, F is the set of facts given on the objects, and Goal is a list, which consists of goals.

The design of deductive reasoning algorithms for solving problems and the design of interface of the system can be developed by three steps for modeling:

**Step 1.** Classify problems such as problems as frames, problems of a determination or a proof of a fact, problems of finding objects or facts, etc…
**Step 2.** Classify facts and representing them based on the kinds of facts of COKB model.
**Step 3.** Modeling kinds of problems from classifying in step 1 and 2. From models of each kind, we can construct a general model for problems, which are given to the system for solving them.

The basic technique for designing deductive algorithms is the unification of facts. Based on the kinds of facts and their structures, there will be criteria for unification proposed. Then it produces algorithms to check the unification of two facts.

The next important work is doing research on strategies for deduction to solve problems on computer. The most difficult thing is modeling for experience, sensible reaction and intuitional human to find heuristic rules, which were able to imitate the human thinking for solving problems.

**Stage 5**: Creating a query language for the models. The language helps to design the communication between the system and users by words.

**Stage 6**: Designing the interface of the system and coding to produce the application. Intelligent applications for solving problems in education of mathematic, physic, chemistry have been implemented by using programming tools and computer algebra systems such as Visual Basic.NET or C#, SQL Server, Maple. They are very easy to use for students, to search, query and solve problems.

**Stage 7**: Testing, maintaining and developing the application. This stage is similar as in other computer systems.

The main models for knowledge representation used in the above process will be presented in the next section.

## 3. Knowledge representation models

In artificial intelligence science, models and methods for knowledge representation play an important role in designing knowledge base systems and expert systems, especially intelligent problem solvers. Nowadays there are many various knowledge models which have already been suggested and applied. In the books of Sowa (2002), George F. Luger

(2008), Michel Chein & Marie-Laure Mugnier (2009) and Frank van Harmelem & Vladimir & Bruce (2008) we have found popular methods for knowledge representation. They include predicate logic, semantic nets, frames, deductive rules, conceptual graphs. The above methods are very useful for designing intelligent systems, especially intelligent problem solvers. However, they are not suitable to represent knowledge in the domains of reality applications in many cases, especially the systems that can solve problems in practice based on knowledge bases. There have been new models proposed such as computational networks, networks of computational objects in Nhon Van Do (2009) and model for knowledge bases of computational objects (COKB) in Nhon Van Do (2010). The COKB model can be used to represent the total knowledge and to design the knowledge base component of practical intelligent systems. Networks of computational objects can be used for modeling problems in knowledge domains. These models are tools for designing inference engine of systems. The models have been used in designing some intelligent problem solvers in education (IPSE) such as the program for studying and solving problems in Plane Geometry in Nhon (2000), the program for solving problems about alternating current in physics. These applications are very easy to use for students in studying knowledge, to solve automatically problems and give human readable solutions agree with those written by teachers and students. In this section, the COKB model and computational networks, that are used for designing IPSE will be presented in details.

### 3.1 COKB model

The model for knowledge bases of computational objects (COKB) has been established from Object-Oriented approach to represent knowledge together with programming techniques for symbolic computation. There have been many results and tools for Object-Oriented methods, and some principles as well as techniques were presented in Mike (2005). This way also gives us a method to model problems and to design algorithms. The models are very useful for constructing components and the whole knowledge base of intelligent system in practice of knowledge domains.

#### 3.1.1 Computational objects

In many problems we usually meet many different kinds of objects. Each object has attributes and internal relations between them. They also have basic behaviors for solving problems on its attributes.

**Definition 3.1:** A computational object (or Com-object) has the following characteristics:

1. It has valued attributes. The set consists of all attributes of the object O will be denoted by **M(O)**.
2. There are internal computational relations between attributes of a Com-object O. These are manifested in the following features of the object:
   - Given a subset A of M(O). The object O can show us the attributes that can be determined from A.
   - The object O will give the value of an attribute.
   - It can also show the internal process of determining the attributes.

The structure computational objects can be modeled by (*Attrs, F, Facts, Rules*). *Attrs* is a set of attributes, *F* is a set of equations called computation relations, *Facts* is a set of properties or events of objects, and *Rules* is a set of deductive rules on facts. For example, knowledge about a triangle consists of elements (angles, edges, etc) together with formulas and some properties on them can be modeled as a class of C-objects whose sets are as follows:

*Attrs* = {A, B, C, a, b, c, R, S, p, ...} is the set of all attributes of a triangle,

*F* = {A+B+C = $\pi$; a/sin(A) = 2R; b/sin(B) = 2R; c/sin(C) = 2R; a/sin(A) = b/sin(B); ... },

*Facts* = {a+b>c; a+c>b; b+c>a ; …},

*Rules* = { {a>b} $\Leftrightarrow$ {A>B}; {b>c} $\Leftrightarrow$ {B>C}; {c>a} $\Leftrightarrow$ {C>A}; {a=b} $\Leftrightarrow$ {A=B};
{a^2= b^2+c^2} $\Rightarrow$ {A=pi/2}; {A=pi/2} $\Rightarrow$ {a^2 = b^2+c^2, b $\perp$ c}; ...}.

An object also has basic behaviors for solving problems on its attributes. Objects are equipped abilities to solve problems such as:

1. Determines the closure of a set of attributes.
2. Executes deduction and gives answers for questions about problems of the form: determine some attributes from some other attributes.
3. Executes computations
4. Suggests completing the hypothesis if needed.

*For example*, when a triangle object is requested to give a solution for problem {a, B, C} $\Rightarrow$ S, it will give a solution consists of three following steps:

Step 1: determine A, by A = $\pi$ -B-C;
Step 2: determine b, by b = a.sin(B)/sin(A);
Step 3: determine S, by S = a.b.sin(C)/2;

### 3.1.2 Components of COKB model

**Definition 3.2:** The model for knowledge bases of computational objects (COKB model) consists of six components:

(C, H, R, Ops, Funcs, Rules)

The meanings of the components are as follows:

- **C** is a set of concepts of computational objects. Each concept in C is a class of Com-objects.
- **H** is a set of hierarchy relation on the concepts.
- **R** is a set of relations on the concepts.
- **Ops** is a set of operators.
- **Funcs** is a set of functions.
- **Rules** is a set of rules.

There are relations represent specializations between concepts in the set **C**; **H** represents these special relations on **C**. This relation is an ordered relation on the set **C**, and **H** can be considered as the Hasse diagram for that relation.

**R** is a set of other relations on **C**, and in case a relation r is a binary relation it may have properties such as reflexivity, symmetry, etc. In plane geometry and analytic geometry, there are many such relations: relation "belongs to" of a point and a line, relation "central point" of a point and a line segment, relation "parallel" between two line segments, relation "perpendicular" between two line segments, the equality relation between triangles, etc.

The set **Ops** consists of operators on **C**. This component represents a part of knowledge about operations on the objects. Almost knowledge domains have a component consisting of operators. In analytic geometry there are vector operators such as addition, multiplication of a vector by a scalar, cross product, vector product; in linear algebra there are operations on matrices. The COKB model helps to organize this kind of knowledge in knowledge domains as a component in the knowledge base of intelligent systems.

The set **Funcs** consists of functions on Com-objects. Knowledge about functions is also a popular kind of knowledge in almost knowledge domains in practice, especially fields of natural sciences such as fields of mathematics, fields of physics. In analytic geometry we have the functions: distance between two points, distance from a point to a line or a plane, projection of a point or a line onto a plane, etc. The determinant of a square matrix is also a function on square matrices in linear algebra.

The set **Rules** represents for deductive rules. The set of rules is certain part of knowledge bases. The rules represent for statements, theorems, principles, formulas, and so forth. Almost rules can be written as the form "if <facts> then <facts>". In the structure of a deductive rule, <facts> is a set of facts with certain classification. Therefore, we use deductive rules in the COKB model. Facts must be classified so that the component **Rules** can be specified and processed in the inference engine of knowledge base system or intelligent systems.

### 3.1.3 Facts in COKB model

In the COKB model there are 11 kinds of facts accepted. These kinds of facts have been proposed from the researching on real requirements and problems in different domains of knowledge. The kinds of facts are as follows:

- **Fact of kind 1**: information about object kind. Some examples are ABC is a right triangle, ABCD is a parallelogram, matrix A is a square matrix.
- **Fact of kind 2**: a determination of an object or an attribute of an object. The following problem in analytic geometry gives some examples for facts of kind 2.

Problem: Given the points E and F, and the line (d). Suppose E, F, and (d) are determined. (P) is the plane satisfying the relations: E $\in$ (P), F $\in$ (P), and (d) // (P). Find the general equation of (P).

In this problem we have three facts of kind 3: (1) point E is determined or we have already known the coordinates of E, (2) point F is determined, (3) line (d) is determined or we have already known the equation of (d).

- **Fact of kind 3**: a determination of an object or an attribute of an object by a value or a constant expression. These are some examples in plane geometry and in analytic geometry: in the triangle ABC, suppose that the length of edge BC = 5; the plane (P) has the equation 2x + 3y – z + 6 = 0, and the point M has the coordinate (1, 2, 3).

- **Fact of kind 4**: equality on objects or attributes of objects. This kind of facts is also popular, and there are many problems related to it on the knowledge base. The following problem in plane geometry gives some examples for facts of kind 4.

Problem: Given the parallelogram ABCD. Suppose M and N are two points of segment AC such that AM = CN. Prove that two triangles ABM and CDN are equal.

In the problem we have to determine equality between two C-objects, a fact of kind 4.

- **Fact of kind 5**: a dependence of an object on other objects by a general equation. An example in geometry for this kind of fact is that w = 2*u + 3*v; here u, v and w are vectors.
- **Fact of kind 6**: a relation on objects or attributes of the objects. In almost problems there are facts of kind 6 such as the parallel of two lines, a line is perpendicular to a plane, a point belongs to a line segment.
- **Fact of kind 7**: a determination of a function.
- **Fact of kind 8**: a determination of a function by a value or a constant expression.
- **Fact of kind 9**: equality between an object and a function.
- **Fact of kind 10**: equality between a function and another function.
- **Fact of kind 11**: a dependence of a function on other functions or other objects by an equation.

The last five kinds of facts are related to knowledge about functions, the component **Funcs** in the COKB model. The problem below gives some examples for facts related to functions.

Problem: Let d be the line with the equation 3x + 4y - 12 = 0. P and Q are intersection points of d and the axes Ox, Oy.

a. Find the central point of PQ
b. Find the projection of O onto the line d.

For each line segment, there exists one and only one point which is the central point of that segment. Therefore, there is a function MIDPOINT(A, B) that outputs the central point M of the line segment AB. Part (a) of the above problem can be represented as to find the point I such that I = MIDPOINT(P,Q), a fact of kind 9. The projection can also be represented by the function PROJECTION(M, d) that outputs the projection point N of point M onto line d. Part (b) of the above problem can be represented as to find the point A such that A = PROJECTION(O,d), which is also a fact of kind 9.

Unification algorithms of facts were designed and used in different applications such as the system that supports studying knowledge and solving analytic geometry problems, the program for studying and solving problems in Plane Geometry, the knowledge system in linear algebra.

### 3.1.4 Specification language for COKB model

The language for the COKB model is constructed to specify knowledge bases with knowledge of the form COKB model. This language includes the following:

- A set of characters: letter, number, special letter.
- Vocabulary: keywords, names.
- Data types: basic types and structured types.
- Expressions and sentences.
- Statements.
- Syntax for specifying the components of COKB model.

The followings are some structures of definitions for expressions, Com-Objects, relations, facts, and functions.

**Definitions of expressions:**

```
expr            ::=     expr | rel-expr | logic-expr
expr            ::=     expr add-operator term | term
term            ::=     term mul-operator factor | factor
factor          ::=     – factor | element ^ factor | element
element         ::=     ( expr ) | name | number | function-call
rel-expr        ::=     expr rel-operator expr
logic-expr      ::=     logic-expr OR logic-term | logic-expr IMPLIES logic-term |
                        NOT logic-term |logic-term
logic-term      ::=     logic-term AND logic-primary |logic-primary
logic-primary   ::=     expr | rel-expr |function-call | quantify-expr |TRUE | FALSE
quantify-expr   ::=     FORALL(name <, name>*), logic-expr | EXISTS(name), logic-
                        expr
```

**Definitions of Com-object type:**

```
cobject-type    ::=     COBJECT name;

                        [isa]
                        [hasa]
                        [constructs]
                        [attributes]
                        [constraints]
                        [crelations]
                        [facts]
                        [rules]
                ENDCOBJECT;
```

**Definitions of computational relations:**

```
crelations      ::=     CRELATION:
                        crelation-def+
                        ENDCRELATION;

crelation-def   ::=     CR name;
                        MF: name <, name>*;
                        MFEXP: equation;
                        ENDCR;
equation        ::=     expr = expr
```

**Definitions of special relations:**

```
isa       ::=      ISA: name <, name>*;
hasa      ::=      HASA: [fact-def]
```

**Definitions of facts:**

```
facts            ::=     FACT: fact-def+
fact-def         ::=     object-type | attribute | name | equation | relation | expression
object-type      ::=     cobject-type (name) | cobject-type (name <, name>* )
relation         ::=     relation ( name <, name>+ )
```

**Definitions of relations based on facts:**

```
relation-def     ::=     RELATION name;
                         ARGUMENT: argument-def+
                         [facts]
                         ENDRELATION;
argument-def     ::=     name <, name>*: type;
```

**Definitions of functions – form 1:**

```
function-def     ::=     FUNCTION name;
                         ARGUMENT: argument-def+
                         RETURN: return-def;
                         [constraint]
                         [facts]
                         ENDFUNCTION;
return-def       ::=     name: type;
```

**Definitions of functions – form 2:**

```
function-def     ::=     FUNCTION name;
                         ARGUMENT: argument-def+
                         RETURN: return-def;
                         [constraint]
                         [variables]
                         [statements]
                         ENDFUNCTION;
statements       ::=     statement-def+
statement-def    ::=     assign-stmt | if-stmt | for-stmt
asign-stmt       ::=     name := expr;
if-stmt          ::=     IF logic-expr THEN statements+ ENDIF; |
                         IF logic-expr THEN statements+ ELSE statements+ ENDIF;
for-stmt         ::=     FOR name IN [range] DO statements+ ENDFOR;
```

## 3.2 Computational networks

In this section, we present the models computational networks with simple valued variables and networks of computational objects. They have been used to represent knowledge in many domains of knowledge. The methods and techniques for solving the problems on the networks will be useful tool for design intelligent systems, especially IPSE.

### 3.2.1 Computational networks with simple valued variables

In this part a simple model of computational networks will be presented together related problems and techniques for solving them. Although this model is not very complicated, but it is a very useful tool for designing many knowledge base systems in practice.

**Definition 3.3:** A *computational network (Com-Net) with simple valued variables* is a pair (M, F), in which M = $\{x_1, x_2, ..., x_n\}$ is a set of variables with simple values (or unstructured values), and F = $\{f_1, f_2, ..., f_m\}$ is a set of computational relations over the variables in the set M. Each computational relation $f \in F$ has the following form:

i. An equation over some variables in M, or
ii. Deductive rule $f : u(f) \rightarrow v(f)$, with $u(f) \subseteq M$, $v(f) \subseteq M$, and there are corresponding formulas to determine (or to compute) variables in v(f) from variables in u(f).We also define the set $M(f) = u(f) \cup v(f)$.

Remark: In many applications equations can be represented as deduction rules.

**Problems:** Given a computational net (M, F). The popular problem arising from reality applications is that to find a solution to determine a set $H \subseteq M$ from a set $G \subseteq M$. This problem is denoted by the symbol $H \rightarrow G$, H is the hypothesis and G is the goal of the problem. To solve the problem we have to answer two questions below:

Q1: Is the problem solvable based on the knowledge K = (M, F)?

Q2: How to obtain the goal G from the hypothesis H based on the knowledge K = (M, F) in case the problem is solvable?

**Definition 3.4:** Given a computational net K = (M, F).

i. For each $A \subseteq M$ and $f \in F$, denote $f(A) = A \cup M(f)$ be the set obtained from A by applying f. Let $S = [f_1, f_2, ..., f_k]$ be a list consisting relations in F, the notation $S(A) = f_k(f_{k-1}(\dots f_2(f_1(A)) \dots))$ is used to denote the set of variables obtained from A by applying relations in S.
ii. The list $S = [f_1, f_2, ..., f_k]$ is called a *solution* of the problem $H \rightarrow G$ if $S(H) \supseteq G$. Solution S is called a good solution if there is not a proper sublist S′ of S such that S′ is also a solution of the problem. The problem is *solvable* if there is a solution to solve it.

**Definition 3.5:** Given a computational net K = (M, F). Let A be a subset of M. It is easy to verify that there exists a unique set $\overline{A} \subseteq M$ such that the problem $A \rightarrow \overline{A}$ is solvable; the set $\overline{A}$ is called the closure of A.

The following are some algorithms and results that show methods and techniques for solving the above problems on computational nets.

**Theorem 3.1:** Given a computational net K = (M, F). The following statements are equivalent.

i. Problem $H \rightarrow G$ is solvable.
ii. $\overline{H} \supseteq G$.
iii. There exists a list of relations S such that $S(H) \supseteq G$.

**Algorithm 3.1:** Find a solution of the problem H→G.

```
Step 1: Solution ← empty;
Step 2: if  $G \subseteq H$  then        begin Solution_found ← true;  goto step 4; end
             else Solution_found ← false;
Step 3: Repeat
                  Hold ← H;
                  Select  $f \in F$;
                  while  not Solution_found  and (f found)  do begin
                          if  (applying f from H produces new facts)
                          then begin
                                  $H \leftarrow H \cup M(f)$;  Add f to Solution;
                                  end;
                          if  $G \subseteq H$  then
                                  Solution_found ← true;
                          Select new  $f \in F$;
                          end;        { while }
          Until Solution_found or (H = Hold);
Step 4: if  not Solution_found  then
                  There is no solution found;
             else
                  Solution is a solution of the problem;
```

**Algorithm 3.2:** Find a good solution from a solution S = [$f_1$, $f_2$, ..., $f_k$] of the problem H→G on computational net (M, F).

```
Step 1: NewS ← [];   V ← G;
Step 2: for  i := k  downto 1  do
            If  $v(f_k) \cap V \neq \emptyset$  the Begin
              Insert $f_k$ at the beginning of  NewS;
              $V \leftarrow (V - v(f_k)) \cup (u(f_k) - H)$;
              End
Step 3: NewS is a good solution.
```

On a computational net (M, F), in many cases the problem H→G has a solution S in which there are relations producing some redundancy variables. At those situations, we must determine necessary variables of each step in the problem solving process. The following theorem shows the way to analyze the solution to determine necessary variables to compute at each step.

**Theorem 3.2:** Given a computational net K = (M, F). Let [$f_1$, $f_2$, ..., $f_m$] be a good solution of the problem H→G. denote $A_0$ = H, $A_i$ = [$f_1$, $f_2$, ..., $f_i$](H), with i=1, ..., m. Then there exists a list [$B_0$, $B_1$, ..., $B_{m-1}$, $B_m$] satisfying the following conditions:

1. $B_m = G$,
2. $B_i \subseteq A_i$ , with i=0, 1, ..., m.
3. For i=1,...,m, [$f_i$] is a solution of the problem $B_{i-1} \rightarrow B_i$ but not to be a solution of the problem $B \rightarrow B_i$ , with B is any proper subset B of $B_{i-1}$.

### 3.2.2 Networks of computational objects

In many problems we usually meet many different kinds of objects. Each object has attributes and internal relations between them. Therefore, it is necessary to consider an extension of computational nets in which each variable is a computational object.

**Definition 3.6:** A computational object (or Com-object) has the following characteristics:

1. It has valued attributes. The set consists of all attributes of the object O will be denoted by M(O).
2. There are internal computational relations between attributes of a Com-object O. These are manifested in the following features of the object:
   - Given a subset A of M(O). The object O can show us the attributes that can be determined from A.
   - The object O will give the value of an attribute.
   - It can also show the internal process of determining the attributes.

Example 3.1: A triangle with some knowledge (formulas, theorems, etc ...) is an object. The attributes of a "triangle" object are 3 edges, 3 angles, etc. A "triangle" object can also answer some questions such as "Is there a solution for the problem that to compute the surface from one edge and two angles?".

**Definition 3.7:** A computational relation f between attributes or objects is called a *relation between the objects*. A network of Com-objects will consists of a set of Com-objects O = $\{O_1, O_2, ..., O_n\}$ and a set of computational relations F = $\{f_1, f_2, ... , f_m\}$. This network of Com-objects is denoted by (O, F).

On the network of Com-objects (O, F), we consider the problem that to determine (or compute) attributes in set G from given attributes in set H. The problem will be denoted by $H \to G$.

Example 3.2: In figure 2 below, suppose that AB = AC, the values of the angle A and the edge BC are given (hypothesis). ABDE and ACFG are squares. Compute EG.

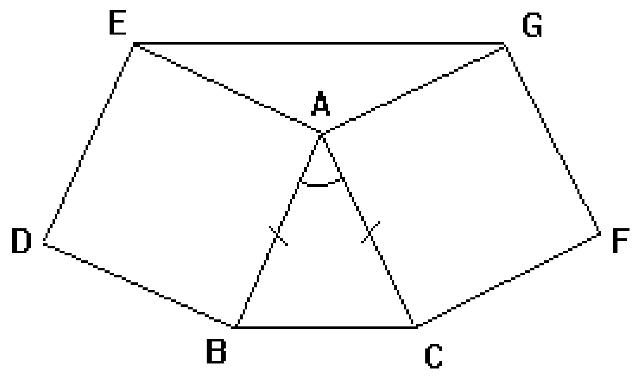

Fig. 2. A problem in geometry

The problem can be considered on the network of Com-objects (O, F) as follows:

O = {$O_1$: triangle ABC with AB = AC, $O_2$ : triangle AEG, $O_3$ : square ABDE, $O_4$ : square ACFG }, and F = {$f_1$, $f_2$, $f_3$, $f_4$, $f_5$} consists of the following relations
$f_1$ : $O_1.c = O_3.a$ {the edge c of triangle ABC = the edge of the square ABDE}
$f_2$ : $O_1.b = O_4.a$ {the edge b of triangle ABC = the edge of the square ACFG}
$f_3$ : $O_2.b = O_4.a$ {the edge b of triangle AEG = the edge of the square ACFG}
$f_4$ : $O_2.c = O_3.a$ {the edge c of triangle AEG = the edge of the square ABDE}
$f_5$ : $O_1.A + O_2.A = \pi$.

**Definition 3.8:** Let (O, F) be a network of Com-objects, and M be a set of concerned attributes. Suppose A is a subset of M.

a. For each $f \in F$, denote f(A) is the union of the set A and the set consists of all attributes in M deduced from A by f. Similarly, for each Com-object $O_i \in O$, $O_i(A)$ is the union of the set A and the set consists of all attributes (in M) that the object $O_i$ can determine from attributes in A.
b. Suppose D = [$t_1$, $t_2$, ..., $t_m$] is a list of elements in $F \cup O$. Denote $A_0 = A$, $A_1 = t_1(A_0)$, . . ., $A_m = t_m(A_{m-1})$, and $D(A) = A_m$.

We have $A_0 \subseteq A_1 \subseteq . . . \subseteq A_m = D(A) \subseteq M$. Problem H→G is called *solvable* if there is a list $D \subseteq F \cup O$ such that $D(A) \supseteq B$. In this case, we say that D is a *solution* of the problem.

Technically the above theorems and algorithms can be developed to obtain the new ones for solving the problem H→G on network of Com-objects (O,F). They will be omitted here except the algorithm to find a solution of the problem. The worthy of note is that the objects may participate in solutions as computational relations.

**Algorithm 3.3:** Find a solution of the problem H→G on a network of Com-objects.

```
Step 1: Solution ← empty;
Step 2: if  $G \subseteq H$  then      begin  Solution_found ← true;  goto step 5;  end
            Else  Solution_found ← false;
Step 3: Repeat
                  Hold ← H;
                  Select  $f \in F$;
                  while  not Solution_found  and (f found)  do  begin
                        if  (applying f from H produces new facts)   then  begin
                              $H \leftarrow H \cup M(f)$;  Add f to Solution;
                              end;
                        if  $G \subseteq H$  then   Solution_found ← true;
                        Select new  $f \in F$;
                        end;      { while }
          Until   Solution_found  or  (H = Hold);
Step 4: if  not Solution_found  then  begin
                  Select  $O_i \in O$ such that  $O_i(H) \neq H$;
                  if  (the selection is successful) then  begin
                        $H \leftarrow O_i(H)$;  Add $O_i$ to Solution;
                        if  ($G \subseteq H$)  then  begin
                        Solution_found ← true;  goto  step 5;
                        end;
                else
                        goto  step 3;
                end;
        end;
Step 5: if  not Solution_found  then  There is no solution found;
        else  Solution is a solution of the problem;
```

Example 3.3: Consider the network (O, F) in example 3.2, and the problem H→G, where

$$H = \{O_1.a, O_1.A\}, \text{ and } G = \{O_2.a\}.$$

Here we have: $M(f_1) = \{ O_1.c , O_3.a \}$, $M(f_2) = \{ O_1.b , O_4.a \}$, $M(f_3) = \{ O_2.b , O_4.a \}$,

$$M(f_4) = \{ O_2.c , O_3.a \}, M(f_5) = \{ O_1.\alpha , O_2.\alpha \},$$

$$M = \{ O_1.a, O_1.b, O_1.c, O_1.A, O_2.b, O_2.c, O_2.A , O_2.a, O_3.a, O_4.a \}.$$

The above algorithms will produce the solution $D = \{ f_5, O_1, f_1, f_2, f_3, f_4, O_2\}$, and the process of extending the set of attributes as follows:

$$A_0 \xrightarrow{f_5} A_1 \xrightarrow{O_1} A_2 \xrightarrow{f_1} A_3 \xrightarrow{f_2} A_4 \xrightarrow{f_3} A_5 \xrightarrow{f_4} A_6 \xrightarrow{O_2} A_7$$

with $A_0 = A = \{O_1.a , O_1.A\}$,
$A_1 = \{O_1.a , O_1.A, O_2.A\}$,
$A_2 = \{ O_1.a , O_1.A, O_2.A, O_1.b, O_1.c \}$,
$A_3 = \{O_1.a , O_1.A, O_2.A, O_1.b, O_1.c, O_3.a\}$,
$A_4 = \{O_1.a , O_1.A, O_2.A, O_1.b, O_1.c, O_3.a, O_4.a\}$,
$A_5 = \{O_1.a , O_1.A, O_2.A, O_1.b, O_1.c, O_3.a, O_4.a, O_2.b\}$,
$A_6 = \{O_1.a , O_1.A, O_2.A, O_1.b, O_1.c, O_3.a, O_4.a, O_2.b, O_2.c\}$,
$A_7 = \{O_1.a , O_1.A, O_2.A, O_1.b, O_1.c, O_3.a, O_4.a, O_2.b, O_2.c, O_2.a\}$.

### 3.2.3 Extensions of computational networks

Computational Networks with simple valued variables and networks of computational objects can be used to represent knowledge in many domains of knowledge. The basic components of knowledge consist of a set of simple valued variables and a set of computational relations over the variables. However, there are domains of knowledge based on a set of elements, in which each element can be a simple valued variables or a function. For example, in the knowledge of alternating current the alternating current intensity i(t) and the alternating potential u(t) are functions. It requires considering some extensions of computational networks such as *extensive computational networks* and *extensive computational objects networks* that are defined below.

**Definition 3.9:** An *extensive computational network* is a structure (M,R) consisting of two following sets:

- $M = M_v \cup M_f$ is a set of attributes or elements, with simple valued or functional valued.$M_v = \{x_{v1}, x_{v2}, \ldots, x_{vk}\}$ is the set of simple valued variables. $M_f = \{x_{f1}, x_{f2}, \ldots x_{fm}\}$ is the set of functional valued elements.
- $R = R_{vv} \cup R_{fv} \cup R_{vf} \cup R_{fvf}$ is the set of deduction rules, and R is the union of four subsets of rules $R_{vv}$, $R_{fv}$, $R_{vf}$, $R_{fvf}$. Each rule r has the form r: $u(r) \rightarrow v(r)$, with u(r) is the hypotheses of r and v(r) is the conclusion of r. A rule is also one of the four cases below.

Case 1: $r \in R_{vv}$. For this case, $u(r) \subseteq M_v$ and $v(r) \subseteq M_v$.
Case 2: $r \in R_{fv}$. For this case, $u(r) \subseteq M_f$ and $v(r) \subseteq M_v$.
Case 3: $r \in R_{vf}$. For this case, $u(r) \subseteq M_v$ and $v(r) \subseteq M_f$.
Case 4: $r \in R_{fvf}$. For this case, $u(r) \subseteq M$, $u(r) \cap M_f \neq \emptyset$, $u(r) \cap M_v \neq \emptyset$, and $v(r) \subseteq M_f$.

Each rule in R has the corresponding computational relation in $F = F_{vv} \cup F_{fv} \cup F_{vf} \cup F_{fvf}$.

**Definition 3.10:** An *extensive computational Object* (ECom-Object) is an object O has structure including:

- A set of attributes $Attr(O) = M_v \cup M_f$, with $M_v$ is a set of simple valued variables; $M_f$ is a set of functional variables. Between the variables (or attributes) there are internal relations, that are deduction rules or the computational relations.
- The object O has behaviors of reasoning and computing on attributes of objects or facts such as: find the closure of a set $A \subset Attr(O)$; find a solution of problems which has the form $A \rightarrow B$, with $A \subseteq Attr(O)$ and $B \subseteq Attr(O)$; perform computations; consider determination of objects or facts.

**Definition 3.11:** An *extensive computational objects network* is a model **(O, M, F, T)** that has the components below.

- $O = \{O_1, O_2, \ldots, O_n\}$ is the set of extensive computational objects.
- M is a set of object attributes. We will use the following notations: $M_v(O_i)$ is the set of simple valued attributes of the object $O_i$, $M_f(O_i)$ is the set of functional attributes of $O_i$, $M(O_i) = M_v(O_i) \cup M_f(O_i)$, $M(O) = M(O_1) \cup M(O_2) \cup \ldots \cup M(O_n)$, and $M \subseteq M(O)$.
- $F = F(O)$ is the set of the computational relations on attributes in M and on objects in O.
- $T=\{t_1, t_2, \ldots, t_k\}$ is set of operators on objects.

On the structure (O,T), there are expressions of objects. Each expression of objects has its attributes as if it is an object.

## 4. Design of main components

The main components of an IPSE are considered here consists of the knowledge base, the inference engine. Design of these components will be discussed and presented in this section.

### 4.1 Design of knowledge base

The design process of the system presented in section 2 consists of seven stages. After stage 1 for collecting real knowledge, the knowledge base design includes stage 2 and stage 3. It includes the following tasks.

- The knowledge domain collected, which is denoted by K, will be represented or modeled base on the knowledge model COKB, Com-Net, CO-Net and their extensions or restrictions known. Some restrictions of COKB model were used to design knowledge bases are the model COKB lacking operator component (C, H, R, Funcs, Rules), the model COKB without function component (C, H, R, Ops, Rules), and the simple COKB sub-model (C, H, R, Rules). From Studying and analyzing the whole of knowledge in the domain, It is not difficult to determine known forms of knowledge presenting, together with relationships between them. In case that knowledge K has the known form such as COKB model, we can use this model for representing knowledge K directly. Otherwise, the knowledge K can be partitioned into knowledge sub-domains $K_i$ (i = 1, 2, …, n) with lower complexity and certain relationships, and each $K_i$ has the form known. The relationships between knowledge sub-domains must be clear so that we can integrate knowledge models of the knowledge sub-domains later.
- Each knowledge sub-domain $K_i$ will be modeled by using the above knowledge models, so a knowledge model $M(K_i)$ of knowledge $K_i$ will be established. The relationships on $\{K_i\}$ are also specified or represented. The models $\{M(K_i)\}$ together with their relationships are integrated to produce a model M(K) of the knowledge K. Then we obtain a knowledge model M(K) for the whole knowledge K of the application.
- Next, it is needed to construct a specification language for the knowledge base of the system. The COKB model and CO-Nets have their specification language used to specify knowledge bases of these form. A restriction or extension model of those models also have suitable specification language. Therefore, we can easily construct a specification language L(K) for the knowledge K. This language gives us to specify components of knowledge K in the organization of the knowledge base.
- From the model M(K) and the specification language L(K), the knowledge base organization of the system will be established. It consists of structured text files that stores the components concepts, hierarchical relation, relations, operators, functions, rules, facts and objects; and their specification. The knowledge base is stored by the system of files listed below,

- File CONCEPTS.txt stores names of concepts, and it has the following structure:
  **begin_concepts**
   <concept name 1>
   < concept name 2>
    ...
  **end_concepts**
  For each concept, we have the corresponding file with the file name <concept name>.txt, which contains the specification of this concept. This file has the following structure:
  **begin_concept**: <concept name>[based objects]
  specification of based objects
  **begin_variables**
  <attribute name> : <attribute type>;
  ...
  **end_variables**

**begin_constraints**
specification of constraints
**end_constraints**
**begin_properties**
specification of internal facts
**end_properties**
**begin_computation_relations**
**begin_relation**
specification of a relation
**end_relation**
...
**end_computation_relations**
**begin_rules**
**begin_rule**
kind_rule = "<kind of rule>";
**hypothesis_part:**
{ facts}
**goal_part:**
{ facts}
**end_rule**
...
**end_rules**
**end_concept**

- File HIERARCHY.txt stores information of the Hasse diagram representing the component H of COKB model. It has the following structure:
  **begin_Hierarchy**
  [<concept name 1>, <concept name 2>]
  ...
  **end_Hierarchy**
- Files RELATIONS.txt and RELATIONS_DEF.txt are structured text files that store the specification of relations. The structure of file RELATIONS.txt is as follows
  **begin_Relations**
  [<relation name>, <concept>, <concept>, ... ], {<property>, <property>, ...};
  [<relation name>, <concept>, <concept>, ... ], {<property>, <property>, ...};
  ...
  **end_Relations**
- Files OPERATORS.txt and OPERATORS_DEF.txt are structured text files that store the specification of operators (the component Ops of KBCO model). The structure of file OPERATORS.txt is as follows
  **begin_Operators**
  [<operator symbol>, <list of concepts>, <result concept>], {<property>, <property>, ...};
  [<operator symbol>, <list of concepts>, <result concept>], {<property>, <property>, ...};
  ...
  **end_Operators**

- Files FUNCTIONS.txt and FUNCTIONS_DEF.txt are structured text files that store the specification of functions (the component Funcs of KBCO model). The structure of file FUNCTIONS.txt is as follows
  **begin_Functions**
  <result concept>  <function name>(sequence of concepts);
  <result concept>  <function name>(sequence of concepts);
  ...
  **end_Functions**
- File FACT_KINDS.txt is the structured text file that stores the definition of kinds of facts. Its structure is as follows
  **begin_Factkinds**
  1, <fact structure>, <fact structure>, ...;
  2, <fact structure>, <fact structure>, ...;
  ...
  **end_Factkinds**
- File RULES.txt is the structured text file that stores deductive rules. Its structure is as follows
  **begin_Rules**
  **begin_rule**
  kind_rule = "<rule kind>";
  <object> : <concept>;
  ...
  **hypothesis_part:**
  <set of facts>
  **goal_part:**
  <set of facts>
  **end_rule**
  ...
  **end_Rules**

- Files OBJECTS.txt and FACTS.txt are the structured text files that store certain objects and facts. The structure of file OBJECTS.txt is as follows
  **begin_objects**
  <object name> : <concept>;
  <object name> : <concept>;
  ...
  **end_objects**

### 4.2 Design the inference engine

Design the inference engine is stage 4 of the design process. The inference engine design includes the following tasks:

- From the collection of problems obtained in stage 1 with an initial classification, we can determine classes of problems base on known models such as Com-Net, CO-Net, and their extensions. This task helps us to model classes of problems as frame-based problem models, or as Com-Nets and CO-Nets for general forms of problems. Techniques for modeling problems are presented in the stage 4 of the design process.

Problems modeled by using CO-Net has the form (O, F, Goal), in which O is a set of Com-objects, F is a set of facts on objects, and Goal is a set consisting of goals.

- The basic technique for designing deductive algorithms is the unification of facts. Based on the kinds of facts and their structures, there will be criteria for unification proposed. Then it produces algorithms to check the unification of two facts. For instance, when we have two facts *fact1* and *fact2* of kinds 1-6, the unification definition of them is as follows: fact1 and fact2 are unified they satisfy the following conditions
  1. *fact1* and *fact2* have the same kind k, and
  2. *fact1* = *fact2* if k = 1, 2, 6.
     [*fact1*[1], {*fact1*[2..nops(*fact1*)]}] = [*fact2*[1], {*fact2*[2..nops(*fact2*)]}] if k = 6 and the relation in *fact1* is symmetric.
     lhs(*fact1*) = lhs(*fact2*) and compute(rhs(*fact1*)) = compute(rhs(*fact2*)) if k =3.
     ( lhs(*fact1*) = lhs(*fact2*) and rhs(*fact1*) = rhs(*fact2*)) or
     ( lhs(*fact1*) = rhs(*fact2*) and rhs(*fact1*) = lhs(*fact2*)) if k = 4.
     evalb(simplify(expand(lhs(*fact1*)-rhs(*fact1*)- lhs(*fact2*)+rhs(*fact2*))) = 0) or
     evalb(simplify(expand(lhs(*fact1*)-rhs(*fact1*)+ lhs(*fact2*)-rhs(*fact2*))) = 0) if k = 5.
- To design the algorithms for reasoning methods to solve classes of problems, the forward chaining strategy can be used with artificial intelligent techniques such as deductive method with heuristics, deductive method with sample problems, deductive method based on organization of solving methods for classes of frame-based problems. To classes of frame-based problems, designing reasoning algorithms for solving them is not very difficult. To classes of general problems, the most difficult thing is modeling for experience, sensible reaction and intuitional human to find heuristic rules, which were able to imitate the human thinking for solving problems. We can use Com-Nets, CO-Nets, and their extensions to model problems; and use artificial intelligent techniques to design algorithms for automated reasoning. For instance, a reasoning algorithm for COKB model with sample problems can be briefly presented below.

**Definition 4.1**: Given knowledge domain K = (C, H, R, Ops, Funcs, Rules), knowledge sub-domain of knowledge K is knowledge domain which was represented by COKB model, it consists of components as follows

$$K_p = (C_p, H_p, R_p, Ops_p, Funcs_p, Rules_p)$$

where, $C_p \subset C$, $H_p \subset H$, $R_p \subset R$, $Ops_p \subset Ops$, $Funcs_p \subset Funcs$, $Rules_p \subset Rules$.

Knowledge domain $K_p$ is a restriction of knowledge K.

**Definition 4.2**: Given knowledge sub-domain $K_p$, Sample Problem (SP) is a problem which was represented by networks of Com-Objects on knowledge $K_p$, it consists of three components ($O_p$, $F_p$, $Goal_p$); $O_p$ and $F_p$ contain objects and facts were specificated on knowledge $K_p$.

**Definition 4.3**: A model of Computational Object Knowledge Base with Sample Problems (COKB-SP) consists of 7 components: (C, H, R, Ops, Funcs, Rules, Sample); in which, (C, H, R, Ops, Funcs, Rules) is knowledge domain which presented by COKB model, the Sample component is a set of Sample Problems of this knowledge domain.

**Algorithm 4.1**: To find a solution of problem P modelled by (O,F,Goal) on knowledge K of the form COKB-SP.

**Step 1.** Record the elements in hypothesis part and goal part.
**Step 2.** Find *the Sample Problem* can be applied.
**Step 3.** Check goal G. If G is obtained then goto step 8.
**Step 4.** Using heuristic rules to select a rule for producing new facts or new objects.
**Step 5.** If selection in step 3 fails then search for any rule which can be used to deduce new facts or new objects.
**Step 6.** If there is a rule found in step 3 or in step 4 then record the information about the rule, new facts in Solution, and new situation (previous objects and facts together with new facts and new objects), and goto step 2.
**Step 7.** Else {search for a rule fails} Conclusion: Solution not found, and stop.
**Step 8.** Reduce the solution found by excluding redundant rules and information in the solution.

**Algorithm 4.2**: To find sample problems.

Given problem P = (O, F, Goal) on on knowledge K of the form COKB-SP. The Sample Problem can be applied on P has been found by the following procedure:

```
Step 1: $H \leftarrow O \cup F$
Step 2: SP ← Sample
        Sample_found← false
Step 3: Repeat
        Select S in SP
  if facts of  H can be applied in (S.Op and S.Fp)  then
          begin
            if kind of S.Goalp = kind of Goal then
                Sample_found  ← true
           Else  if S.Goalp $\subseteq$ H then
                Sample_found  ← true
          end
        SP ← (SP – S)
    Until SP = {} or Sample_found
Step 3: if Sample_found then
           S is a sample problem of the problem;
        else
           There is no sample problem found;
```

This algorithm simulates a part of human mind when to find SP that relate to practical problem. Thereby, the inference of system has been more quickly and effectively. Moreover, the solution of problem is natural and precise.

## 5. Applications

The design method for IPS and IPSE presented in previous sections have been used to produce many applycations such as program for studying and solving problems in Plane Geometry, program for studying and solving problems in analytic geometry, program for solving problems about alternating current in physics, program for solving problems in inorganic chemistry, program for solving algebraic problems, etc. In this section, we introduce some applications and examples about solutions of problems produced by computer programs.

- The system that supports studying knowledge and solving analytic geometry problems. The system consists of three components: the interface, the knowledge base, the knowledge processing modules or the inference engine. The program has menus for users searching knowledge they need and they can access knowledge base. Besides, there are windows for inputting problems. Users are supported a simple language for specifying problems. There are also windows in which the program shows solutions of problems and figures.
- The program for studying and solving problems in plane geometry. It can solve problems in general forms. Users only declare hypothesis and goal of problems base on a simple language but strong enough for specifying problems. The hypothesis can consist of objects, relations between objects or between attributes. It can also contain formulas, determination properties of some attributes or their values. The goal can be to compute an attribute, to determine an object, a relation or a formula. After specifying a problem, users can request the program to solve it automatically or to give instructions that help them to solve it themselves. The program also gives a human readable solution, which is easy to read and agree with the way of thinking and writing by students and teachers. The second function of the program is "Search for Knowledge". This function helps users to find out necessary knowledge quickly. They can search for concepts, definitions, properties, related theorems or formulas, and problem patterns.

Examples below illustrate the functions of a system for solving problems of analytic geometry, a system for solving problems in plane geometry, and a system for solving algebraic problems. The systems were implemented using C#, JAVA and MAPLE. Each example presents the problem in natural language, specifies the problem in specification language to input into the system, and a solution produced from the system.

**Example 5.1**: Let d be the line with the equation 3x + 4y - 12 = 0. P and Q are intersection points of d and the axes Ox, Oy.

a. Find the midpoint of PQ
b. Find the projection of O on the line d.

Specification of the problem:

Objects = {[d,line], [P,point], [Q,point]}.
Hypothesis = { d.f = (3*x+4*y-12 = 0), Ox.f = (y = 0), O = [0, 0], P = INTERSECT(Ox, d), Q = INTERSECT (Oy, d), H = PROJECTION(O, d), Oy . f = (x = 0) }.
Goal = { MIDPOINT(P, Q), H }.

Solution found by the system:

Step 1: {d.f = (3*x+4*y-12 = 0), Ox.f = (y = 0), Oy.f = (x = 0)} $\rightarrow$ {d.f, Ox.f , Oy.f }.

Step 2: {Ox.f, Oy.f, d.f} → {Ox, Oy, d}.
Step 3: {P = INTERSECT(Ox,d), d, Ox} →{P = [4, 0]}.
Step 4: {d, Oy, Q = INTERSECT(Oy,d)} → {Q = [0, 3]}.
Step 5: {P = [4, 0], Q = [0, 3]} → {P, Q}.
Step 6: {P, Q} → {MIDPOINT(P,Q) = [2, 3/2]}.
Step 7: {d, H = PROJECTION(O,d), O} →{ H = [36/25, 48/25]}.
Step 8: {H = [36/25, 48/25]} → {H}.

**Example 5.2**: Given two points P(2, 5) and Q(5,1). Suppose d is a line that contains the point P, and the distance between Q and d is 3. Find the equation of line d.

Specification of the problem:

Objects = {[P, point], [Q, point], [d, line]}.
Hypothesis = {DISTANCE(Q, d) = 3, P = [2, 5], Q = [5, 1], ["BELONG", P, d]}.
Goal = [d.f ].

Solution found by the system:

Step 1: {P = [2, 5]} → {P}.
Step 2: {DISTANCE(Q, d) = 3} → {DISTANCE(Q, d)}.
Step 3: {d, P} → {2d[1]+5d[2]+d[3] = 0}.
Step 4: {DISTANCE(Q, d) = 3}

$$\rightarrow \Rightarrow \frac{|5d[1]+d[2]+d[3]|}{\sqrt{d[1]^2 + d[2]^2}} = 3 .$$

Step 5: $\left\{ d[1]=1,\ 2d[1]+5d[2]+d[3]=0, \frac{|5d[1]+d[2]+d[3]|}{\sqrt{d[1]^2 + d[2]^2}} = 3 \right\}$

$$\Rightarrow \left\{ d.f=(x+\frac{24}{7}y-\frac{134}{7}=0), d.f=(x-2=0) \right\}$$

Step 6: $\left\{ d.f=(x+\frac{24}{7}y-\frac{134}{7}=0), d.f=(x-2=0) \right\}$

$$\Rightarrow \{\ d.f\ \}$$

**Example 5.3**: Given the parallelogram ABCD. Suppose M and N are two points of segment AC such that AM = CN. Prove that two triangles ABM and CDN are equal.

Specification of the problem:

Objects = {[A, POINT], [B, POINT], [C, POINT], [D, POINT], [M, POINT], [N, POINT],
[O1, PARALLELOGRAM[A, B, C, D], [O2, TRIANGLE[A, B, M]],
[O3, TRIANGLE [C, D, N]]}.
Hypothesis = { [« BELONG », M, SEGMENT[A, C]],

[« BELONG », N, SEGMENT[A, C]], SEGMENT[A, M] = SEGMENT[C, N] }.
Goal = { O2 = O3}.

Solution found by the system:

Step 1: Hypothesis
→ { O2.SEGMENT[A, M] = O3. SEGMENT[C, N],
O2.SEGMENT[A, B] = O1. SEGMENT[A, B],
O3.SEGMENT[C, D] = O1.SEGMENT[C, D]}.
Step 2: Produce new objects related to O2, O3, and O1
→ {[O4, TRIANGLE[A, B, C]], [O5, TRIANGLE[C, D, A]]}.
Step 3: {[O1, PARALLELOGRAM[A, B, C, D]}
→{O4 = O5, SEGMENT[A, B] = SEGMENT[C, D]}.
Step 4: { O2.SEGMENT[A, B] = O1.SEGMENT[A, B],
O3.SEGMENT[C, D] = O1.SEGMENT[C, D],
SEGMENT[A, B] = SEGMENT[C, D]}
→{O2.SEGMENT[A, B] = O3.SEGMENT[C, D]}.
Step 5: {[« BELONG », M, SEGMENT[A, C]]}
→ {O4.angle_A = O2.angle_A}.
Step 6: {[« BELONG », N, SEGMENT[A, C]]}
→ { O5.angle_A = O3.angle_A }.
Step 7: {O4 = O5 }
→ {O4.angle_A = O5.angle_A}.
Step 8: { O4.angle_A = O2.angle_A ,
O5.angle_A = O3.angle_A ,
O4.angle_A = O5.angle_A }
→{ O2.angle_A = O3.angle_A}.
Step 9: { O2.SEGMENT[A, M] = O3. SEGMENT[C, N],
O2.SEGMENT[A, B] = O3.SEGMENT[C, D],
O2.angle_A = O3.angle_A }
→ {O2 = O3}.

**Example 5.4**: Let the equation, with m is a parameter, and x is a variable:

$$\left(m^2 - 4\right)x + 2 = m$$

Solve this equation by m.

Solution found by the system:

Solve the equation:

$$\left(m^2 - 4\right)x + 2 = m$$

$$\left(m^2 - 4\right)x = -2 + m$$

The coefficient of x has a set of roots:

$$\{-2 \quad , \quad 2\}$$

+ if parameter $m = -2$ , then:

"This equation has no root"

+ if parameter $m = 2$ , then:

"This equation has set of roots is the set of real numbers "

+ if parameter $m \notin \{-2 , 2\}$ , then:

$$x = \frac{-2 + m}{m^2 - 4} = \frac{1}{m + 2}$$

## 6. Conclusions and future work

In this chapter, we proposed a method for designing intelligent problem solvers (IPS), especially those in education (IPSE). These systems have suitable knowledge base used by the inference engine to solve problems in certain knowledge domain, they not only give human readable solutions of problems but also present solutions as the way instructors and learners usually write them. Knowledge bases contain concepts of computational objects (Com-Objects), relations, operators, functions, facts and rules. The *Computational Object Knowledge Base* model (COKB) and its specification language can be used for knowledge modeling, for designing and implementing knowledge bases. The COKB model has been established from Object-Oriented approach for knowledge representation together with programming techniques for symbolic computation. The design of inference engine requires to model problems and to design reasoning algorithms with heuristics and sample problems. Computational networks (Com-Net) and networks of computational objects (CO-Net) can be used effectively for modeling problems and construction of reasoning algorithms in practical knowledge domains. These models are tools for designing inference engine of systems.

The proposed design method has been used to produce applycations in many fields such as mathematics, physics and chemistry. They support studying knowledge and solving problems automatically based on knowledge bases. Users only declare hypothesis and goal of problems base on a simple language but strong enough for specifying problems. The programs produce a human readable solution, which is easy to read and agree with the way of thinking and writing by students and instructors.

Designing an intelligent problem solver in education is a very challenging task, as domain knowledge and human thinking are very complicated and abstract. There are domains of knowledge with functional attributes such as knowledge of alternating current in physics. This motivates another extensions of COKB model, Com-Net and CO-Net, and develops design techniques. They will accept simple valued variables and also functional variables.

On the network of computational objects, operators will be considered. Such future works our models more powerful for representing knowledge in practice. Besides, To have a user-interface using natural language we have to develop methods for translating problems in natural language to specification language, and vice versa.

## 7. References

Chitta Baral (2003). Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, ISBN 0 521 81802 8.

Fatos X.; Leonard B. & Petraq J. P. (2010). Complex intellgent systems and their applications. *Springer Science+Business Media, LLC*. ISBN 978-1-4419-1635-8

Frank van Harmelem, Vladimir & Bruce (2008). Handbook of Knowledge Representation. *Elsevier*, ISBN: 978-0-444-52211-5.

George F. Luger (2008). Artificial Intelligence: Structures And Strategies For Complex Problem Solving. Addison Wesley Longman

Johns M. Tim (2008). Artificial Intelligence – A System Approach. *Infinity Science Press LLC*, ISBN: 978-0-9778582-3-1

Michel Chein & Marie-Laure Mugnier (2009). Graph-based Knowledge representation: Computational foundations of Conceptual Graphs. *Springer-Verlag London Limited*. ISBN: 978-1-84800-285-2.

Mike O'Docherty (2005). Object-oriented analysis and design: understanding system development with UML 2.0. *John Wiley & Sons Ltd*, ISBN-13 978-0-470-09240-8.

Nhon Do & Hien Nguyen (2011). A Reasoning Method on Computational Network and Its Applications. *Proceedings of the International MultiConference of Engineers and Computer Scientists* 2011 (ICAIA'11), pp. 137-141

Nhon Do & Hoai P. Truong & Trong T. Tran (2010). An Approach for Translating Mathematics Problems in Natural Language to Specification Language COKB of Intelligent Education Software. *Proceedings of 2010 International Conference on Artificial Intelligence and Education*, Hangzhou, China. 10-2010

Nhon Van Do (2010). Model for Knowledge Bases of Computational Objects. *International Journal of Computer Science Issues*, Vol. 7, Issue 3, No 8, pp. 11-20

Nhon Van Do (2009). Computational Networks for Knowledge Representation. *Proceedings of World Academy of Science, Engineering and Technology*, Volume 56, pp. 266-270

Nhon Do (2008). An ontology for knowledge representation And Applications*. Proceedings of World Academy of Science, Engineering and Technology*, Volumn 32, pp. 23-31, ISSN 2070-3740

Nhon Van Do (2000). A Program for studying and Solving problems in Plane Geometry. *Proceedings of International on Artificial Intelligence 2000*, Las Vegas, USA, 2000, pp. 1441-1447

Sowa, John F. (2002). Architectures for Intelligent Systems. *IBM Systems Journal*, vol. 41, no.3, pp. 331-349

Sowa, John F. (2000). Knowledge Representation: Logical, Philosophical and Computational Foundations. *Brooks/Cole Thomson Learning*, ISBN 0 534-94965-7

Stuart Russell & Peter Norvig (2010). Artificial Intelligence – A modern approach (Third edition). *Prentice Hall*, by Pearson Education, Inc.

ISBN 978-953-51-0054-6
Hard cover, 366 pages
**Publisher** InTech
**Published online** 02, March, 2012
**Published in print edition** March, 2012