Design an Intelligent Problem Solver in Mathematics based on Integrated-Knowledge Model

Hien D. Nguyen1,2, Linh Nguyen3, Nha P. Tran4, Van-Thanh Nguyen-Le5, Sang Vu2,6
1,2 University of Information Technology, Ho Chi Minh City, Vietnam
3 Vietnam National University, Ho Chi Minh City, Vietnam
3,4 University of Transport and Communications, Campus in Ho Chi Minh City, Vietnam
5 HCMC University of Technology and Education, Ho Chi Minh City, Vietnam
Email: {hiennd@uit.edu.vn, nphlinh2679@gmail.com, nhap_ph@utc.edu.vn, thanhnlv@hcmute.edu.vn, sangvu@uit.edu.vn}

Abstract—Knowledge engineering is one of AI technologies that play a principal role to design intelligent systems. Intelligent Problem Solver (IPS) in education is a useful intelligent system supporting e-learning in the fourth industrial revolution. Discrete Mathematics is one of important courses for the engineering curriculum at universities. In this paper, a knowledge model, which integrates the knowledge of operators and function intellectual, is proposed. This model is used to organize the knowledge base of an IPS in Discrete Mathematics. This system can automatically solve some common exercises with human-readable solutions, especially their reasoning is pedagogy and similar to the way of students' thinking. The experimental results show that the built system is emerging to evolve an intelligent system as a tutor how to solve problems in this course.

Keywords—Intelligent Problem Solver, knowledge engineering, automated reasoning, educational software.

I. INTRODUCTION

In the fourth industrial revolution, distance learning plays an important role to develop the quality of education [1]. E-learning is the way in which learning is imparted to students by using electronic media and information and communication technologies in education [2, 3]. The Intelligent Problem Solver (IPS) in education is an intelligent system supporting e-learning [4]. This system can give step-by-step solutions to inputted problems as teachers and students usually write them [5]. Users only declare hypothesis and goals of problems based on a simple language but strong enough for specifying problems. After specifying a problem, users can request the program to solve it automatically or to give instructions that help them to solve it themselves [6]. Some requirements for an IPS in education are [7, 8]:
- The program can automatically solve many kinds of basic and advanced problems.
- The input of problems is similar to the practice. Solutions of the program are pedagogy, step-by-step and readable, their reasoning simulates the thinking of students.
- This program can help the students to understand the lessons better as well as the methods for solving the exercises. The reasoning of a solution uses the knowledge of the learner about the course.

Discrete Mathematics (DiM) is an important course for technological students at universities. This is the basic knowledge about logic, relational algebra, combinatorial, and graph [9]. This course includes chapters about logic and Boolean algebra, which are the foundation of logical thinking. This course helps students to improve their skills in logical reasoning, problem-solving. The contents of these chapters are propositional logic, predicate logic, and logical operators, inference logical rules, Boolean expression and simplification, Karnaugh map. The IPS in DiM has to solve some common kinds of exercises of this course. Hence, the knowledge base of this system has to represent the logical operators with their properties, the rules of knowledge domain with deductive and equivalent rules [10, 11].

There is many software to support for solving logical problems. However, those are educational software, they cannot use in learning of DiM. VAMPIRE is an automatic theorem proving for first-order logic [12]. It can produce detailed proof. Nevertheless, its proofs are not natural, and they do not use the knowledge of DiM curriculum.

Isar Proof is a linear representation of natural deduction proofs in the style of Jaskowski [13]. This is a module for transforming automatic theorem provers proofs into readable proofs. It increases the intelligibility and robustness of the output. Because the purpose of this system does not tend the studying of DiM, its solutions are neither pedagogical nor appropriate for the students at universities.

Ontology is a useful method to organize the knowledge bases of intelligent systems [14, 15, 16], especially in educational systems [17]. The Computational Objects Knowledge Base (COKB) is an ontology based on an object-oriented approach [5, 17]. This method was applied to build some intelligent educational software, such as courses of Graph Theory [18] and Plane Geometry [19]. However, ontology COKB is very complex to deploy in the real-world. It is difficult to design the combining of practical multiple knowledge components.

The knowledge models of operators were studied and applied to build some IPS in DiM [18, 20]. Those systems can solve some problems about logic and retrieve the knowledge of this course through queries. Nonetheless, their knowledge bases had not yet organized naturally, and their proofs were still machinery.

In this study, a knowledge model is proposed to represent the knowledge of DiM, called Ops-Funs model. This model improves the model in [20, 21] by combining the knowledge of functions. It has a kernel as the knowledge of operators [20] integrating to the functional knowledge [22] via the integration of concepts and functions which were built based on the structure of concepts. The Ops-Funs model is adequate to represent the knowledge of DiM. Through the knowledge base as Ops-Funs model, some problems on the knowledge of logic are studied and design algorithms for solving them, such as the problems about simplifying the logical expression, proving the equivalence between two expressions and checking of logical reasoning. The IPS in DiM is constructed to support the learning of students for this course. This system can automatically solve some common exercises with human-
readable solutions, especially their reasoning is pedagogy and similar to the way of students’ thinking. This system is the potential to apply in practice and develop to tutoring system about how to solve an exercise.

The next section presents the knowledge model for Discrete Mathematics. Section III proposes some problems on this knowledge domain and designs algorithms for solving them. Section IV builds an Intelligent Problem Solver in DiM and shows experimental results of this system. The last section concludes the studies of this paper and gives some future works.

II. KNOWLEDGE REPRESENTATION FOR DISCRETE MATHEMATICS

This section presents a knowledge model for representing the knowledge domain of Discrete Mathematics. This knowledge domain has many domains [9, 23]. In this study, it is considered some domains about predicate logic, first-order logic, and Boolean algebra. This model is integrated between the knowledge model of operators [21] and the component of functions [24].

Definition 2.1 [21]: The knowledge model Operators, called Ops-model, consists of three components:

\[ \mathcal{X} = (C, \text{Ops}, \text{Rules}) \]

In which, \( C \) is a set of concepts, each concept in \( C \) is a class of objects with their behaviors to solve problems on them. \( R \) is a set of relations on the concepts. \( \text{Ops} \) is a set of operators; each operator is a binary mapping, we consider the properties of it are: commutative, associative, identity. \( \text{Rules} \) is a set of inference rules. A rule in this model is one of two forms: deductive rule or equation rule.

Definition 2.2: The knowledge model integrating between Ops-model and functions, called Ops-Funcs model, is a tube:

\[ \mathcal{X} = (C, \text{Ops}, \text{Rules}) + \text{Funcs} \]

In which, \( (C, \text{Ops}, \text{Rules}) \) is a knowledge model of operators as Ops-model, and \( \text{Funcs} \) is a set of functions of the knowledge domain.

A. \( C – \text{the set of concepts} \)

Each concept \( c \in C \) has the structure as follows:

\[ c = (\text{Attr}, \text{M_Funcs}, \text{Parent}) \]

where, \( \text{M_Funcs} \) is a set of functions. There are three important functions which are determined based on logical expressions: \( \text{getTruthValue()} \) - get the truth value of an expression, \( \text{getID()} \) - take the suffix string of that expression, and \( \text{getSubExpr()} \) - get the array of sub-expressions. \( \text{Parent} \) is a reference pointer to the expression, if \( c \) is the ROOT then \( c.\text{Parent} \) is NULL. \( \text{Attr} \) is the set of attributes, it includes:

\[ \text{Attr} = \{\text{id, truth_value, childs, operators}\} \]

where, \( \text{id} \) is character or suffix string represents expression.

\( \text{truth_value} \) is the truth value of constant or prime-expression.

\( \text{childs} \) are all sub-expressions.

\( \text{operators} \) is the set of operators of the expression.

B. \( \text{Ops – the set of operators} \)

\( \text{Ops} \) is the set of operators on the knowledge domain about Discrete mathematics in logic. Table 1 represents some operators of logic in this domain. Through those operators, there are definitions of logical expressions and their specifications.

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \neg p )</th>
<th>( p \lor q )</th>
<th>( p \land q )</th>
<th>( p \rightarrow q )</th>
<th>( p \leftrightarrow q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

* T: True F: False

Definition 2.3: Logical expression

1. If \( p \) has a value of \{0, 1\}, \( p \) is logical expression.

Then, it is called prime-expression.

2. If \( p \) is a logical expression, so is \( \neg p \)

3. If \( p, q \) are logical expressions, so is \( p \oplus q \) (with \( \oplus \in \{\land, \lor, \rightarrow, \leftrightarrow\} \)).

4. \( \neg p \) and \( p \oplus q \) are complex-expressions.

Definition 2.4: Let \( p \) and \( q \) be logical expressions. The tree \( T_p \), which represents the expression \( p \), is defined as follows:

1. If \( p \) is logical expression then \( T_p = p \)

2. If \( p = \neg q \) then \( T_p = \neg q \)

3. If \( p = q \oplus \ldots \oplus r \) then \( T_p = \ldots \oplus r \)

4. If \( p \) is sub-expression of \( q \), then \( T_p \) is sub-tree of \( T_q \)

Example 1: \( A = \neg p \land (p \lor (r \rightarrow q)) \land p \) is represented:

It can be shorted to:

Finally, this expression can re-form into the suffix:

\[ A = (\land, (\neg p), (\lor, p, (\rightarrow, r, q)), p) \]

However, this method has the disadvantage that the representation will group together expression with the same operator; therefore, the clause needs to be grouped with a pair of brackets “()” to create sub-expressions. For example, a clause \( p = a \lor b \land c \), should be grouped as \( p = a \lor (b \land c) \) or \( p = (a \lor b) \land c \). This also fits completely with natural language.

C. \( \text{Rules – the set of inference rules} \)

There are two kind of rules in \( \text{Rules} \)-set: equivalent rule and deductive rule.

\[ \text{Rules} = R_{\text{equivalent}} \cup R_{\text{deductive}} \]

- \( R_{\text{equivalent}} \) is the set of equivalent rules: each rule \( r \in R_{\text{equivalent}} \) has the form: \( r: \text{left}(r) = \text{right}(r) \)
where, left(r) and right(r) are logical expressions. Some examples of equivalent rules are as follows:

De Morgan: \( \neg(p \land q) \equiv \neg p \lor \neg q \)

Distribution: \( p \land (q \lor r) \equiv (p \land q) \lor (p \land r) \)

- \( R_{\text{deduce}} \) is the set of deduced rules: each rule \( r \in R_{\text{deduce}} \) has the form:
  \[ \begin{align*}
  r: & \ u(r) \Rightarrow v(r) \\
  \text{where,} & \ u(r) \text{ and } v(r) \text{ are sets of logical expressions} \\
  \text{representing the hypothesis and goal of rule } r, \text{ resp. Some} \\
  \text{examples of deduce rules are as follows:} \\
  \text{Disjunctive:} & \ \{ p \lor q, \neg p \} \Rightarrow \{ q \} \\
  \text{Modus Tollens:} & \ \{ p \Rightarrow q, \neg q \} \Rightarrow \{ \neg p \} \\
  \text{Hypothetical:} & \ \{ p \Rightarrow q, q \Rightarrow r \} \Rightarrow \{ p \Rightarrow r \} \\
  \end{align*} \]

D. **Funcs** – the set of functions

**Definition 2.5:** Some functions in **Funcs**-set:

1. **Checker**:\( p, r \): check the expression \( p \) can apply to the rule \( r \in \text{Rules} \).
2. **Replace**:\( p, q, s \): Replace the expression \( p \) by the expression \( q \) in the original expression \( s \).
3. **Replace**:\( A, B, p, q, s \): Replace a set of sub-expressions \( A \) by a set of sub-expressions \( B \) in the original expression \( s \).
4. **Copy**:\( p \): copy an expression.
5. **Length**:\( p \): return the length of an expression \( p \).
6. **Contain**:\( p, q \): check the expression \( p \) contains the expression \( q \).
7. **Not**:\( p \): the negative expression of \( p \).
8. **Simplify**:\( p \): return the expression \( g \) is the reducing expression of \( p \) and set of steps \( cd \) to simplify it.
9. **Equivalent**\( p, q \): Check the equivalent between two expressions.

III. **ALGORITHMS FOR SOLVING PROBLEMS ON THE KNOWLEDGE DOMAIN ABOUT DISCRETE MATHEMATICS**

In the knowledge domain about Discrete Mathematics collected from [25, 26], there are three kinds of problems:

(1) Simplify the logical expression.
(2) Prove the equivalence between two expressions.
(3) Checking of logical reasoning.

Based on this knowledge domain \( \mathcal{K} = (C, \text{Ops}, \text{Rules}) \) + **Funcs** organized as Ops-Funcs model, this section presents the models of those problems and algorithms for solving them.

A. **Simplify the logical expression**

**Definition 3.1** [27]: Let \( g \) and \( h \) be logical expressions. A relation “simpler than” \( (<<) \) is a binary relation such that:

\( g << h \) if and only if \( \text{Length}(g) \leq \text{Length}(h) \)

The relation “simpler than” \( (<<) \) has properties are: reflexive, transitive.

**Definition 3.2:** Let \( g \) be logical expressions. The expression \( f \) is called the simplest equivalent expression of \( g \), denoted \( f = \text{simplify}(g) \), if and only if:

\[ f = g \text{ and } \forall \text{expression } h, h = g \Rightarrow f << h \]

**Algorithm 3.1:** Simplification the logical expression.

**Input:** the logical expression \( g \).

**Output:** \( f \) - the simplest equivalent expression of \( g, f = \text{simplify}(g) \).

Trans - the solution steps of transformation.

The idea for solving the problem: Sort equivalent rules in \( R_{\text{equivalent}} \). The algorithm tries to reduce \( g \) by applying rule \( r \in R_{\text{equivalent}} \). Then, the expression \( g \) is transformed to \( g' \) with \( g' << g \). After that, using equivalent rules to transform all sub-expression of \( g' \) to get other states \( g'' \). If \( g' << g'' \), then the simpler expression of \( g' \) is \( g'' \), otherwise, the simpler expression of \( g \) is \( g'' \). We continue to reduce on the simpler expression of \( g' \). All transformation steps are stored in list \( Trans \).

**Step 1:** Collect all sub-expressions of the logical expression \( p \).

\[ S_p = \{ s \mid \text{Contains}(s, g) = \text{"True"} \} \]

**Step 2:** Find the simplest equivalent expression.

\[ f' := g; \]

**Trans** := \[
\]

For each sub-expression \( s \) in \( S_p \), do

Search rule \( r \in \text{Rules} \) can be applied on \( s \) through

**Checker:** \( f, r \) function.

Apply \( r \) to transform \( s \) to \( s' \).

\[ p' := \text{Replace}(s, s', r); \]

if \( p' << f \) then

\[ f := p'; \]

Add \( r \) into \( Trans \).

**Step 3:**

Return \( f \) is the simplest equivalent expression of \( g \) and \( Trans \) is the list of transformation steps.

B. **Prove the equivalence between two expressions**

The problem about proving the equivalence between two expressions means the finding of transformation steps between them.

Let \( p, q \) be logical expressions. Prove: \( p = q \) means the finding of the list of transformation steps \( P_{T} \) which can be applied to transform between \( p \) and \( q \).

**Input:** \( p, q \) be logical expressions.

**Output:** \( P_{T} \) - list of transformation steps from \( p \) to \( q \) (or from \( q \) to \( p \))

**Algorithm 3.2:**

**Step 1:**

Let \( f_0 := \text{simplify}(p) \) and \( T_p := \text{simplify}(p) \).

\[ f_0 := \text{simplify}(p) \quad \text{and} \quad T_p := \text{simplify}(p). \]

* \( \text{simplify}(s) \) returns transformation steps to compute \( \text{simplify}(s) \) by Algorithm 3.1

**Step 2:**

\[ P_{T} := []; \]

if \( (f_0 = q) \) then \( P_{T} := T_p \)

elif \( (f_0 = p) \) then \( P_{T} := T_q \)

elif \( (f_0 = f_0) \) then \( P_{T} := T_p \cup T_q \)

**Step 3:**

if \( (P_{T} \text{ is not empty}) \) then

Return \( P_{T} \) as a solution of the problem
else the problem cannot be solved.

C. **Checking of logical reasoning**

The problem about checking of logical reasoning is the searching of deductive rules to infer the goal from the hypothesis of the problem. The model of this problem is:

\[ A \Rightarrow B \]

where \( A, B \) are sets of logical expressions, \( A \) is the set of hypothesis, \( B \) is the set of goals.

**Definition 3.3** [27]: Let knowledge domain \( \mathcal{K} = (C, \text{Ops}, \text{Rules}) \) + **Funcs** as Ops-Funcs model, rule \( r \in \text{Rules} \) and \( A \) is a set of facts.

a/ if \( r \in \text{Rule}_{\text{deductive}} \) r has the form: \( u(r) \Rightarrow v(r) \)

\( r \) can be applied on \( A \) if \( u(r) \subseteq A \)

Let \( r(A) := A \cup v(r) \)

b/ if \( r \in \text{Rule}_{\text{equivalent}} \) r has the form left(r) = right(r)
The architecture of an IPS was presented in [5]. The idea for solving this problem is using forward chaining strategy to deduce from A to B. For each expression p in A, search a rule r can apply to p, it will generate p'. Then, add p' to A. This reasoning is stopped until nothing can be deduced from A, or the goal B has been found from the A.

**Step 1:**  
\[ Known := A; \]
\[ Sol := []; \]
**Step 2:**  
\[ flag := true; \]
\[ while (flag \text{ true}) \]
\[ flag := false; \]
\[ for \ r \ in \ Rules \]
\[ if (r \text{ can be applied to } Known) \]
\[ flag := true; \]
\[ Add r \text{ into } Sol. \]
\[ Update Known by r(Known) \]
\[ if (B \subseteq Known) \text{ Goto Step 3; \]} \]
\[ fi; \]
\[ od; \# \text{ for} \]
\[ od; \# \text{ while} \]
**Step 3:**  
\[ if (B \subseteq Known) \]
\[ The reasoning A \Rightarrow B \text{ is true and } Sol \text{ is a solution.} \]
\[ else \]
\[ The reasoning A \Rightarrow B \text{ is false} \]

**IV. DESIGN AN INTELLIGENT PROBLEM SOLVER FOR DISCRETE MATHEMATICS**

Discrete Mathematics is an important course of the curriculum about technology at universities. An Intelligent Problem Solver (IPS) in this course helps students to learn it better, especially for studying by themselves. This section presents the designing of an IPS in DiM and experimental results of this system. This system is used to support the learning of Vietnamese students in this course.

A. The architecture of the IPS in Discrete Mathematics

The architecture of an IPS was presented in [5]. This architecture is as Fig. 1.

The main process for IPS: The system analyzes the inputed problem which was specified by the specification language through the user interface. The inference engine determines its hypothesis and goals, then they will be recorded into working memory. After that, the knowledge base is used to search objects, facts and rules which can be applied to this problem. Besides, the system does some reasoning strategies and using reasoning rules to solve the problem. When the solution is found, it is outputted in human-readable form for the user via the interface. More than that, the reasoning of the solution simulates the solving method of a student in the real-world.
Then, we have:

- The heuristic rule for evaluating the priority of inference rules: Based on the hypothesis and goal of the inputted problem, this rule will arrange reasoning rules to apply priority.

- The heuristic rules to select a list of transformation rules: In the practice, when a logical expression is expanded, there are some other inference rules will be applied to reduce the new expression. These heuristic rules help to construct some common lists of transformation rules used together.

Those heuristic rules simulate the reasoning of human mind when solving a problem in the DiM course. Therefore, combining the knowledge base, the reasoning strategy of the built system is consistent with the learned knowledge and suitable to support the learning of students.

Example 2: Prove the equivalence between two logical expressions: \( x \rightarrow (y \rightarrow z) = y \rightarrow (x \rightarrow z) \)

The solution of the system for this problem is as Fig. 3:

![Fig. 3: The solution of the designed system for Example 2 (Vietnamese).](image)

The meaning of Fig. 3:

Proof:

Split the expression into two parts:
- left = \( x \rightarrow (y \rightarrow z) \)
- right = \( y \rightarrow (x \rightarrow z) \)

Then, we have:

left = \( x \rightarrow (y \rightarrow z) \) (rule of implication)

\[\begin{align*}
\equiv x \rightarrow (-y \lor z) & \quad (\text{rule of implication}) \\
\equiv \neg x \lor (-y \lor z) & \quad (\text{rule of implication}) \\
\equiv \neg x \lor \neg y \lor z & \quad (\text{rule of combination}) \\
\equiv \neg y \lor (-x \lor z) & \quad (\text{symmetric}) \\
\equiv y \rightarrow (-x \lor z) & \quad (\text{rule of implication}) \\
\equiv y \rightarrow (x \rightarrow z) & \quad (\text{rule of implication})
\end{align*}\]

Thus, we have left = right (q.e.d)

The exercises in Kind 4 can be solved by applying algorithm for solving problems in Kind 2 and Kind 3. Table 2 and Fig. 4 summary experimental results on four kinds of collected exercises.

<table>
<thead>
<tr>
<th>Kind</th>
<th>Number of exercises</th>
<th>Success</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>11</td>
<td>69%</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>15</td>
<td>68%</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>16</td>
<td>76%</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>10</td>
<td>83%</td>
</tr>
<tr>
<td>Total</td>
<td>71</td>
<td>52</td>
<td>73%</td>
</tr>
</tbody>
</table>

The designed IPS in DiM can solve common kinds of exercises the curriculum at university. The solutions of the system are step-by-step, reasoning like the solving method of human. Although the proportion of this system is 73%, it tends to tutor how to solve a problem for students. The proposed method is emerging to design intelligent tutoring system for giving some tips to solve problems as a practical tutor [28, 29].

V. CONCLUSION AND FUTURE WORK

This study proposed a knowledge model integrating between the knowledge of operators and the intellectual component of functions, called Ops-Funcs model. This integrating model is used to organize the knowledge base of Discrete Mathematics. Based on this knowledge model, some problems on this knowledge domain have been proposed and the algorithms for solve them were also designed. Those problems are simplifying the logical expression, proving the equivalence between two expressions and checking of logical reasoning.

Using the designed knowledge base and algorithms, an IPS in DiM is constructed to support the learning of students for this course at universities. The reasoning of the system is combined the heuristic rules which simulate the way of human thinking for solve exercises. Thus, this system can automatically solve some common exercises with human-readable solutions, especially their reasoning is pedagogy and similar to the way of students’ thinking. Besides, the system is constructed based on criteria of an IPS in education, so the system can meet some requirements in education.

In the future, some methods for tutoring how to solve problems will be studied [30]. The IPS in DiM will work as an instructor to guide students how to solve an exercise. Besides, the system will be studied to upgrade the ability for solving some problems about first-order logic to determine the least generalization [31, 32]. In the real-world, the knowledge domain includes many sub-domains in different forms, such as the knowledge of operators [21, 27] and relations [33, 34]; hence, the studying of the method for integrating many sub-domains will be able to apply more effectively in many practical aspects [35, 36]. Moreover, as
the results in [37], the click-and-drag test is better than the conventional static test for academic achievement of university students; thus, the combination of the IPS with an evaluation testing system through multiple choices test [38, 39] and click-and-drag test will give an educational system to support the e-learning completely.

REFERENCES