Pure and New Mathematics in AI 2024, 1(1), 9798.
https://doi.org/10.24294/pnmai9798

Article

Equivalent derivation machine implementation in advanced mathematics symbolic systems

Ping Zhu[1,2,*], Pohua Lv[1], Weiming Zou[3], Xuetao Jiang[1], Jin Shi[3], Yang Zhang[4], Yirong Ma[3]

[1] Beijing Broad Network & Information Company Limited, Beijing 101111, China
[2] Tellhow Institute of Smart City, Beijing 100176, China
[3] Beijing Tellhow Intelligent Engineering Company Limited, Beijing 100176, China
[4] Beijing Yizhuang Smart City Institute Group Company Limited, Beijing 100176, China
*** Corresponding author:** Ping Zhu, 1401626437@qq.com

CITATION

Zhu P, Lv P, Zou W, et al. Equivalent derivation machine implementation in advanced mathematics symbolic systems. Pure and New Mathematics in AI. 2024; 1(1): 9798. https://doi.org/10.24294/pnmai9798

ARTICLE INFO

Received: 22 October 2024
Accepted: 4 December 2024
Available online: 13 December 2024

Abstract: In order to give machines, the interpretable thinking ability of mathematicians, the automatic derivation engine for advanced mathematics symbolic systems was explored to develop, which could update machines from the shallow thinking ability, such as natural language understanding and elementary mathematical numerical computation, to deep thinking, such as equivalent derivation for symbolic systems. This article proposed the complex logic algorithm design and development method with the frameworks as the core components. Starting with problem-resolving examples, the initial idea, basic data structure, and programming features of this new method were introduced in detail. However, this article proposed the integrated development environment for this method, as well as the main scheduling algorithm, core process algorithm, workflow dynamic display algorithm, execution status monitoring algorithm, generalization processing method, etc. The new method could be applicable to intelligent system development tasks that needed to gradually accumulate instance experience and had practical significance for the complex logic algorithms development, visualization software design, reduction complexity for software test and maintenance, and software reliability improvement. This article used the application problem solved by partial differential equations as an example to explain this method from the whole process, such as lexical analysis, semantic analysis, symbolic system establishment, and equivalent derivation to result validation, demonstrating the new dynamism and potential for logic derivation-based classical artificial intelligence methods.

Keywords: logic derivation; semantic analysis; symbolic system; equivalent derivation; integrated development environment; complex logic algorithm; deep thinking

1. Introduction

In the past decade, research on automatic humanoid resolving of mathematics application problems (math word problems) mainly focused on the scope of elementary mathematics [1–3], which was the research topic based on natural language semantic understanding [4,5]. It could be systematically implemented by semantic understanding technologies, such as the limited local semantics extensions for the clauses with mathematics commonsense knowledge [6]. However, in terms of machine simulation of human mathematical abilities, the humanoid derivation of advanced mathematics symbolic systems, such as functions, calculus, and equations [7–9], also had great practical value. Research in this field could make machines think deeply about problem texts and even might have the potential to self-verify the mathematics theories. For artificial intelligence researchers who aimed to simulate the

human brain, this was a very attractive topic [10]. After 5 years of analyzing and resolving 1112 elementary mathematics application problems by machine, the authors established the computational resolving method system with limited local semantic extension. To enable the computer to have more comprehensive mathematics thinking abilities, the author explored the equivalent derivation of the mathematics symbolic system and the implementation mechanism for the computer. This work would mainly discuss the formal representation of problems and the equivalent derivation of symbolic systems, which had great significance for the comprehensive implementation of machine thinking.

The difference from the software toolkits such as MATLAB [11] and Scientific Notebook, which were widely used in scientific computing and had advanced mathematics symbolic computing capabilities, is that this article focused on the formal representation of semantics in advanced mathematics (including data element computing relationships) and their symbolic equivalent derivation, with the feature of process interpretability [12,13]. From the recent period perspective, this technology route was still suitable for application in the education field; from the long period perspective, it also enabled the possibility of interactive exploration and collaborative evolution between thinking machines and humans. It was precisely because it could achieve interpretability by intermediate derivation steps, which required a large amount of commonsense knowledge and domain knowledge, that this technology route had the potential for autonomous science exploration and mathematics verification. From machine text semantic understanding to interpretable automatic advanced mathematical problem resolving, this paper achieved the entire process by commonsense knowledge base at first time. In the future, with the support of axiomatic sets (commonsense knowledge and domain knowledge), it would be possible to achieve autonomous extension research for new mathematics theories by computer and interactively verify the results with human mathematicians. This was also the fundamental reason why interpretability was crucial for both machine thinking and artificial intelligence research. The strengths of this route included: The engine was implemented based on case analysis, which conformed to human thinking habits; gradual effect presentation of the engine made developers feel a sense of achievement in the implementation; the visual development platform could reduce the complexity of software debugging and maintenance; unified function scheduling operation provided the mechanism foundation for automatic system self-programming; the machine thinking process could be explained. The limitations included: It was a completely new technological route and method, with no integration of other mature modules, a huge workload, and slow progress; currently, it is only in the prototype development stage.

With the emergence of ChatGPT [14], the large language model technology had achieved significant success, becoming a hot topic and mainstream in the business and technology fields [15]. Despite the continuous efforts of many researchers, there were still many unsatisfactory aspects in numerical computation relationships, logic reasoning, commonsense reasoning, objective facts, and information updates. Many researchers were constantly striving to improve and enhance the large language model, with the main goal still focused on improving language understanding, logic reasoning, correctness, reliability, and security.

Research involving mathematical derivation and symbolic equivalent relationship verification was relatively rare. The current main target of the large language model was ordinary users, and the support and assistance provided for the highly specialized and knowledge-based scientific education field were still very limited. This article aimed to explore the evolution of autonomous semantic understanding, knowledge verification, conclusion generation, and human-computer interaction for thinking machines, targeting the field of scientific research. Taking the equivalent derivation in advanced mathematics symbolic systems as an example, the technology architecture for interpretable intelligent scientific research systems was designed to explore the implementation for scientific research machines. The main difference between this route and the popular machine learning methods was the interpretability of the derivation steps, rather than directly providing results.

2. Related work

This article implemented a humanoid thinking automatic processing platform using engineering methods, which included text semantic understanding, commonsense accumulation, mathematical knowledge citation, and equivalent derivation of mathematical symbols. Due to the complexity of language expression forms, the platform involved a wide range of data and pattern types, with complex language features and obvious feature sparsity, making it relatively difficult to accumulate feature samples. Currently, it is not suitable to use machine learning algorithms [16–18] that require a large number of feature samples, and machine learning processing of partial differential equations [19,20] also faced the same problem. Reinforcement learning [21,22], genetic algorithms [23], and deep learning algorithms [24–29] are all considered for future applications in this article. The interpretability principles were adopted in the technical route; language feature data required by these algorithms was precipitated in the platform. As there were many types of features, the quantity of samples for the single feature was relatively small, and the application loop had not yet been established. One of the main contributions of this article: Simplified traditional software engineering methods through visualization methods, enhanced the intuitiveness of system workflow, and reduced the complexity of system maintenance. Unlike classical visualization program design technologies [30–32], which were mainly aimed at the software development stage, the visualization method proposed in this article provided the new implementation approach for automatically understanding software logic changes [33] and planning thinking actions. The second main contribution of this article: Achieved interpretability of semantic understanding with clause vocabulary sequence patterns as the core idea, which was different from the semantic understanding technology routes based on large language models and traditional statistics [34–36]. The third main contribution of this article is the implementation of the interpretable derivation process for the symbol system of advanced mathematics with instances. The process of deriving mathematical knowledge [37–39] was different from the function black box processing method of advanced mathematics tools such as MATLAB and was more suitable for application in university teaching. Finally, it was worth pointing out that although the pre-training language model method [40] and the symbolic inference

enhancement of the thought chain technology for large language models [41] were mainstream directions, their main design ideas differed significantly from the methods proposed in this article.

3. Implementation technologies

If the problem space was not completely determined or the samples could not be precisely enumerated (the samples scale, form, or features could not be accurately determined), higher requirements were required on the generalization ability of big data based intelligent algorithms. Generally, the scale of input data for intelligent algorithms based on big data was enormous, and the processing logic should be the condensed version of simplified and smaller computation models. There were two ways to generate this "condensed version" model by the following modes: (1) Up-down, which firstly determined the complete set of processing logic (macro architecture), and then gradually refined the branches (micro processing). For example, in deep learning of large language models, the hierarchy and scale of the neural network were first determined, and then the threshold and I/O link weights of each neuron were gradually trained and adjusted. (2) Down-up, referred to summarizing the general processing framework for each sample based on the processing workflows, gradually generalizing it to any valid input. For example, in this article, the way of humanoid machine resolving mathematics application problems, which analyzed and processed many problem examples, and summarized the general processing framework, was determined by the Down-up mode. The derivation rules of equations in advanced mathematics were generated in this way, and after generalization processing, they were applied to the machine automatic resolving process of general mathematics application problems. This article adopted the second mode and summarized the system technology levels as shown in the following **Figure 1**:

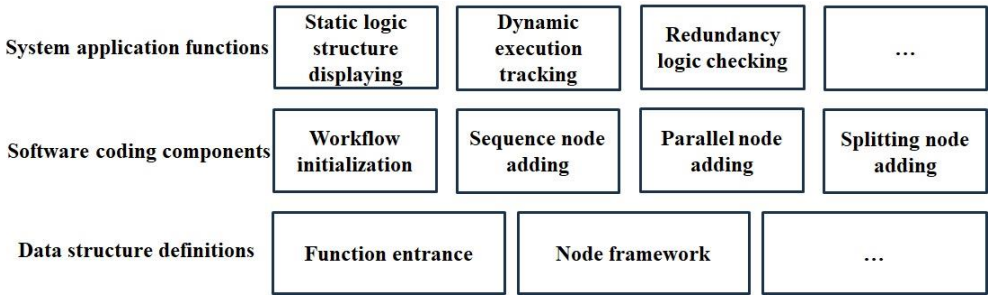

Figure 1. Technology levels.

The initial workflow of problem resolving could be the four step function units A, B, C, and D arranged in series in **Figure 2a**. In the subsequent instance process, there might be two types of workflow changes: (1) the addition of parallel selection function unit, for example, the addition of parallel selection function unit E for function unit B in **Figure 2b**. (2) The addition of function unit for serial selection function unit, such as the selection function unit F in **Figure 2c** (where F represented by a "real" branch with actual processing function and a "bypass" branch without any processing function). These were all based on the basic processing framework of samples, which was the same or similar, but only required local adjustments and

improvements. Based on this premise, the gradual refinement process could be implemented by Up-down mode through a standard pre-set data framework, similar to the training process for the large language model. The additions of branches and bypasses were all discrete gradual processes (refer to **Figure 2**).

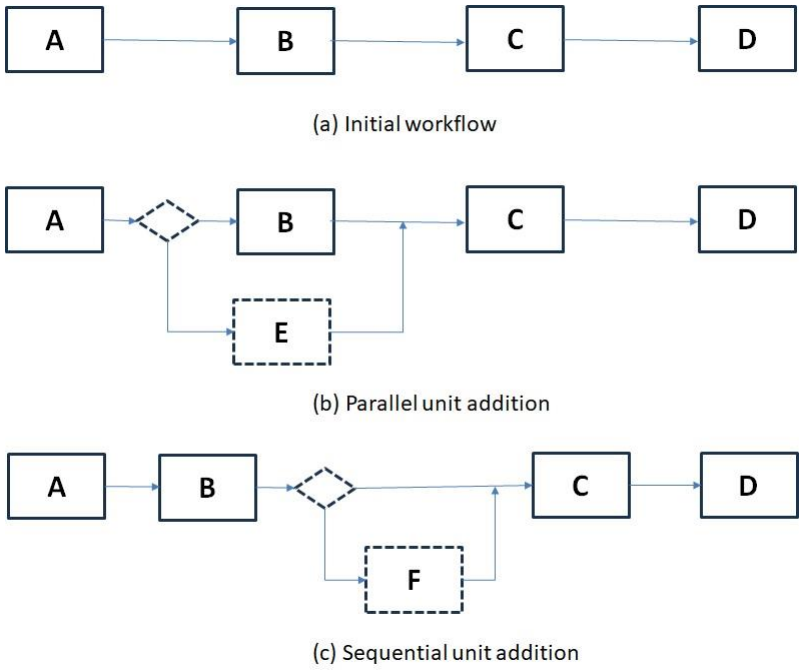

Figure 2. Gradual workflow refining processes.

If each function unit was represented by a general framework (data structure) that included the selection judgment component (which could have only one output branch, indicating absolute selection, i.e., no selection judgment and directly linking to subsequent function unit) and the function processing component (which could be empty, indicating direct linking before and after, without corresponding processing component), the whole software logic workflow could be nested and refined into a multi-dimension web that could be dynamically added with function units, which could become the large logic model. The only provided as many effective feature judgment and processing function units as possible was all that the developers needed to do, automatically/semi-automatically added to the machine decision-making system of the large logic model, forming the systematic judgment and decision generalization ability. The atom function unit was called the basic function unit, and the upper function unit was the combination of basic function units. The standardized function unit data structure laid the foundation for adaptive dynamic adjustment of the large logic model with active autonomous machine actions. For example, the function units could be represented by the framework using the following C-like language data structure (refer to Appendix).

Applying the above framework node data structure to the whole programming and debugging process of complex logic algorithms could form the framework nodes and logic links hierarchical web with recursive and loop structures. In the process of continuously refining this web system, in addition to the refining process of adding nodes at the same level mentioned above, there were also the forms of splitting and

refining operations, such as embedding sub-framework nodes into the basic framework nodes (without sub-frameworks). The processing principles were as follows: (1) If the embedding framework node located at the beginning of the splitting framework node, then copied the splitting framework nodes and linked the embedding framework node with the splitting framework back and forth, and used them as the sub-frameworks of the original splitting framework node; (2) If the embedded framework node located at the end of the splitting framework node, copied the splitting framework node and linked it back and forth with the embedding framework node as the sub-framework nodes of the original framework node; (3) If the embedding framework located in the middle of the source code of the splitting framework node, then took this location as the boundary to represent the splitting framework node into two new function units and initialized the two consecutive new framework nodes. Inserted the embedded framework node into the middle of these two nodes and linked them back and forth, becoming the sub-framework node of the original splitting framework node. (4) If the embedding framework node was the replacing framework node, the embedding framework node inherited the back-and-forth links of the replaced framework node, and then the addition action was performed according to principles (1) to (3) based on the location of the embedding framework node within the replaced framework node. For example, the Algorithm 1 Init_ splitting_chain() of adding embedding framework node I to splitting framework node G were listed as follows (In the initial state, framework nodes P, G, and K were arranged in order):

Algorithm 1 Init_ splitting_chain()

1: *struct STRUCT_GENERAL_FRAME_INTERFACE P, G, K, I, Lson1, Lson2;/// variables definitation;*
2: *Input: Framework nodes P, G, K, I, Lson1, Lson2;*
3: *Output: Framework net with the unified input entry pHead.*
4: *Algorithm description: Init_ splitting_chain()*
5: *① P.nID = 7; P.pPrev = NULL; P.pTNext = &G; P.pFNext = &G; P.pSon = NULL; P.pFathter = NULL; P.fun_execute_body = …; memset(&(P. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); …*
6: *② G.nID = 8; G.pPrev = &A; G.pTNext = &K; G.pFNext = &K; G.pSon = NULL; G.pFathter = NULL; G.fun_execute_body.App = G_content_fun; memset(&(G. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); …*
7: *③ K.nID = 9; K.pPrev = &G; K.pTNext = &…; K.pFNext = &…; K.pSon = NULL; K.pFathter = NULL; K.fun_execute_body = …; memset(&(K. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); …*
8: *④ pHead = &P;*

The function body of framework node G was shown in the following **Figure3**:

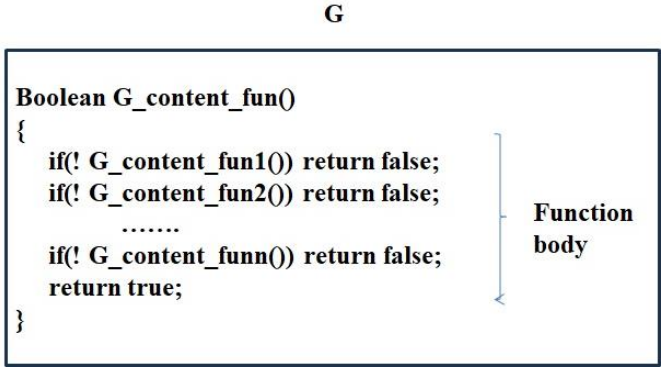

Figure 3. Function body of framework node G.

If the embedding location was between G_content_fun1 and G_content_fun2 inside the framework node G, the function G_content_fun2_n should be defined firstly as follows:

a) Embedding at starting location with Algorithm 2 splitting_chain_a()

Algorithm 2 splitting_chain_a()

Input: Framework nodes P, G, K, I, Lson1, Lson2;
Output: Framework net with the unified input entry pHead.
Algorithm description: splitting_chain_a()
① Init_ splitting_chain() /// Initialize the context parameters and function parameters of the framework node chain pHead.
② Lson1 = G; G. fun_execute_body = NULL; G. pSon = &I;
③ I.nID = 10; I. pPrev = G. pPrev; I.pTNext = &Lson1; I.pFNext = &Lson1; I.pFathter = &G;
④ Lson1.nID = 11; Lson1.pPrev = &I; Lson1.pTNext = G.pTNext; Lson1.pFNext = G. pFNext; I.pFathter = &G; STRUCT_FUNTION_ENTRANCE)); …

b) Embedding at ending location with Algorithm 3 splitting_chain_b()

Algorithm 3 splitting_chain_b()

Input: Framework nodes P, G, K, I, Lson1, Lson2;
Output: Framework net with the unified input entry pHead.
Algorithm description: splitting_chain_b()
① Init_ splitting_chain() /// Initialize the context parameters and function parameters of the framework node chain pHead.
② Lson1 = G; G. fun_execute_body = NULL; G. pSon = &Lson1;
③ Lson1.nID = 10; Lson1.pTNext = &I; Lson1.pFNext = &I; Lson1.pFathter = &G;
④ I.nID = 11; I. pPrev = &Lson1; I.pTNext = G.pTNext; I.pFNext = G.pFNext; I.pFathter = &G;

c) Embedding at middle location with algorithm *splitting_chain_c()*

If the embedding node I was contained in framework node G, firstly, I inherited the context links of G; Then, based on the location of I in the G function body, selected the principles (1) to (3) to perform the generation operations of G son nodes.

```
G_content_fun2_n
{
  If(!G_content_fun2()) return false;
  ……
  If(!G_content_funn()) return false;
  return true;
}
```

Then, executed following Algorithm 4 splitting_chain_c()*:*

Algorithm 4 splitting_chain_c()

Input: Framework nodes P, G, K, I, Lson1, Lson2;
Output: Framework net with the unified input entry pHead.
Algorithm description: splitting_chain_c()
① Init_ splitting_chain() /// Initialize the context parameters and function parameters of the framework node chain pHead.
② Lson1 = G; G. fun_execute_body = NULL; G. pSon = &Lson1;
③Lson1.nID = 10; Lson1.pTNext = &I; Lson1.pFNext = &I; Lson1.pFathter = &G; Lson1. fun_execute_body App = G_content_fun1;
④ I.nID = 11; I. = &Lson1; I.pTNext = &Lson2; I.pFNext = &Lson2; I.pFathter = &G;
⑤ Lson2 = G; Lson1.nID = 12; Lson2.pPrev = &I; I.pFathter = &G; Lson2. fun_execute_body App = G_content_fun2_n;

d) Replacement embedding

It was worth noting that in order to split the function body, all variables were stored in the form of global variables; Although only the processing operation order and workflow branch were illustrated here, loop and recursive embedding could be implemented by the combination of the above operation order and workflow branches, so would not be repeated here; Deleting framework node was the inverse process of adding nodes mentioned above, and would not be elaborated here due to space limitations.

4. Integrated development environment

Framework-oriented programming, the whole algorithm was a large decision tree composed of semantic framework nodes formed by the two-dimension basic logic condition table and function body. The core algorithm was the scheduling and execution process of framework nodes, and all semantic framework nodes worked together through the main scheduling algorithm and shared this core algorithm. Adapting to development requirements motivated the framework-oriented gradual updating system for complex intelligent algorithms, namely: (1) the explicit and clear logic hierarchy and source code modules, which were conducive to development and maintenance; (2) monitoring the execution status of each key function to improve the efficiency of system debugging; (3) The key function needed to have the ability to approximate and compare input data samples, in order to generalize the processing ability of the input. The system architecture of the integrated development environment was shown in the following **Figure 4**:

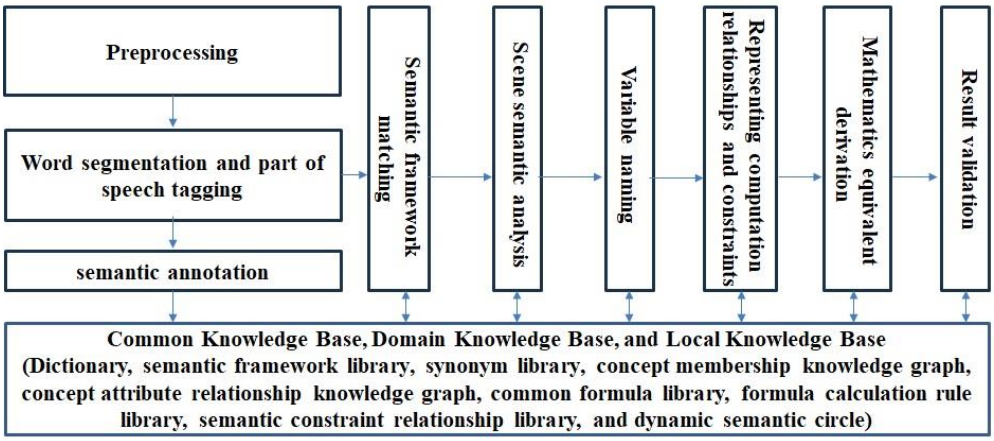

Figure 4. System architecture.

To simplify the illustration, this article took the mathematics symbol system equivalent derivation module as the example to introduce the implementation of the integrated development environment design requirements, including the main scheduling algorithm (refer to Algorithm 5), core processing algorithm (refer to Algorithm 6), workflow dynamic display algorithm (refer to Algorithm 7), execution status monitoring method, and generalization processing method.

a) Main scheduling algorithm

Algorithm 5 main_ scheduling_algorithm()

Input: global framework node web FUNWeb and header pointer pHead (global variable), global return variable bRet.
Output: logic derivation steps, intermediate results, and final answer.
Algorithm description: main_ scheduling_algorithm()
① FUNWeb ← initialize_framework_node_parameters(); /// Semantic initialization for current framework nodes;
② IF (pHead == NULL) RETURN TRUE; /// If the loop pointer was NULL, return success.
③ IF (pHead → fun_execute_body != NULL) bRet ← main_framework_node_algorithm(); /// Executed the function body of the current framework node recorded the return value to the Boolean variable bRet.
④ ELSE pHead = pHead → pSon;
GOTO ②;
///If current framework node had son node, jump to ②;
⑤ IF (bRet) pHead = pHead→ pTNext;
GOTO ②;
///If the current framework node was successfully executed, transferred the current framework node pointer along the "true" branch and jump to step ②;
⑥ ELSE pHead = pHead→ pFNext;
GOTO ②;
///If the current framework node was not successfully executed, transferred the current framework node pointer along the "false" branch and jump to step ②.

b) Core processing algorithm

Algorithm 6 core_processing_algorithm()

Input: Current frame pointer pHead (global variable).
Output: The current framework returned the global variable bRet.
Algorithm description: core_processing_algorithm()
①nHorLoopVar = 0; NVerLoopVar=0; bSucc=TRUE;///Initialize loop control variables.
②IF(pHead → sRefer_Table[nHorLoopVar][nVerLoopVar]. fun_execute_body == NULL) THEN
///If the entry in the current two-dimension basic logic condition table is NULL, the return value was determined based on the executed logic condition;
IF ((nVerLoopVar == 0)&&(nHorLoopVar==0)) THEN RETURN bSucc;
IF ((nVerLoopVar == 0)&&(nHorLoopVar!=0)) THEN RETURN bSucc;
ELSE IF (nVerLoopVar != 0) THEN RETURN bSucc;
ELSE nHorLoopVar++; goto ②;
③ IF (pHead → sRefer_Table [nHorLoopVar] [nVerLoopVar].bRet == pHead → sRefer_Table [nHorLoopVar] [nVerLoopVar]. fun_execute_body ()) THEN
bSucc = TRUE;
nVerLoopVar++;
GOTO ②;
④ELSE bSucc = FALSE; nHorLoopVar++; nVerLoopVar = 0; goto ②;
///If the entries in the two-dimension basic logic condition table were correctly executed, then continued to determine the next basic logic condition; Otherwise, recorded the failure flag and continued to determine the next logical condition.

c) Workflow dynamic display algorithm

Algorithm 7 dynamic_inhibition_list_display()

Input: global framework node web FUNWeb and the header pointer pHead (global variable)
Output: framework node web calling sequence FUNList;
Algorithm description: dynamic_inhibition_list_display()
① IF(pHead == NULL) return TRUE; /// If the current framework node pointer was NULL, return the successful execution flag;
② display_new_fun(pHead); /// Display information about the current framework node.
③ IF (pHead → pSon !=NULL) THEN
pHead = pHead → pSon;
GOTO ①;

Algorithm 7 (*Continued*)

9: *///If the current framework node had son nodes, jump to ①;*
10: ④ IF (pHead → pTNext!=NULL) THEN
11: *pHead = pHead → pTNext;*
12: *GOTO ①;*
13: *///Transferred the current framework node pointer along the successful branch and jump to step ①;*
14: ⑤ IF (pHead → pFNext!=NULL) THEN
15: *pHead = pHead → pFNext;*
16: *GOTO ①;*
///Transferred the current framework node pointer along the failure branch and jump to step ①.

d) Execution status monitoring method

The execution status monitoring algorithm *execution_status_record_algorithm()* mainly monitored the execution status of the two-dimension basic logic condition table. It was worth noting that when the source program was modified, the original framework node logic condition result data should be guaranteed not to change, that was, the input and output data, and corresponding relationships were verified to be correct; However, the operation status of the two-dimension basic logic condition table might change, and it was necessary to delete or update the relevant status data of the framework nodes involved, and restarted the accumulation of operation status data. The execution status monitoring algorithm was based on the main scheduling algorithm *main_scheduling_algorithm()* and the core processing algorithm *core_processing_algorithm()*. When the system called each framework node, it recorded the relevant input and output information of the system when executing the two-dimension basic logic condition table. That was, when the algorithm executed step ③ in *core_processing_algorithm()*, it could add the statements for recording the input data and return information of each basic logic condition as following (refer to Algorithm 8):

Algorithm 8 execution_status_record_algorithm ()

1: *Input: Current frame pointer pHead (global variable).*
2: *Output: The current framework returned the global variable bRet.*
3: *Algorithm description: execution_status_record_algorithm ()*
4: *① nHorLoopVar = 0; NVerLoopVar=0; bSucc=TRUE;///Initialize loop control variables.*
5: *② IF(pHead → sRefer_Table[nHorLoopVar][nVerLoopVar]. fun_execute_body == NULL) THEN*
6: *///If the entry in the current two-dimension basic logic condition table is NULL, the return value was determined based on the executed logic condition;*
7: *IF ((nVerLoopVar == 0)&&(nHorLoopVar==0)) THEN RETURN bSucc;*
8: *IF ((nVerLoopVar == 0)&&(nHorLoopVar!=0)) THEN RETURN bSucc;*
9: *ELSE IF (nVerLoopVar != 0) THEN RETURN bSucc;*
10: *ELSE nHorLoopVar++;*
11: *goto ②;*
12: *③ pHead → Record_Input_Data(); /// Added execution steps for recording input data;*
13: *④ IF (pHead → sRefer_Table[nHorLoopVar] [nVerLoopVar].bRet == pHead → sRefer_Table[nHorLoopVar] [nVerLoopVar]. fun_execute_body ()) THEN*
14: *pHead → Record_Output_Data(); /// Recorded the return information of each basic logical condition (AND operation element);*
15: *bSucc = TRUE;*
16: *nVerLoopVar++;*
17: *goto ②;*
18: *⑤ pHead → Record_Output_Data(); /// Recorded the return information of each logic condition (OR operation element);*

Algorithm 8 (*Continued*)

19: *⑥bSucc = FALSE; nHorLoopVar++; nVerLoopVar = 0; goto ②; ///If the entries in the two-dimension basic logic condition table were correctly executed, then continued to determine the next basic logic condition; Otherwise, recorded the failure flag and continued to determine the next logical condition.*

e) Generalization processing capability method

The execution status monitoring algorithm *execution_status_record_algorithm()* in the previous section recorded the input/output data which became sample data correspondence of each basic logic condition (feature function), and the data could be used as the basis for improving the system's generalization processing ability for the inputs. Analyzed the features of sample data, defined the semantic distance between sample data or between sample data and input data based on these features, and determined whether to directly output the processing result corresponding to the sample data based on the semantic distance (calculated by the output of each feature function in the two-dimension basic logic condition table). In this way, the sample data could form the "anchor point" of the input space, and new inputs that approximated to the samples could directly output the samples results. The input approximation calculation based on semantic distance achieved the input processing ability coverage to the whole problem space.

5. Application example

5.1. Problem description

"An uncovered cuboid basin is made of the same thin sheet (C1), its volume is a constant value (C2). In order to minimize the use of the sheet as much as possible (C3), how much should be the length, width, and height of the basin (C4)?"

a) Semantic annotation

C1 ///Semantic framework identifier

Normal ///Semantic framework type

16 ///Quantity of the semantic framework entries

c An ///"c" represented a constant, and "An" represented the constant value.

t uncovered ///"t" represented other types of vocabulary.

*s cuboid||** ///"s" represented the noun.*

*s basin||** /// "**" represented that it could be replaced by other nouns.*

v is made of /// "v" represented the verb.

t the

t same

t thin

*s sheet||**

m Uncovered::Basin::Volume///"m" represented the intermediate data element variable, "Uncovered::Basin::Volume" was the local variable name.

m Uncovered::Basin::Surface~Area~

m Uncovered::Basin::Length

m Uncovered::Basin::Width

m Uncovered::Basin::Height

f uncovered /// "f" represented the focal features transmitted across semantic frameworks.

f basin

C2 ///Semantic framework identifier

Normal ///Semantic framework type

9 /// Quantity of the semantic framework entries

F uncovered,basin, /// "F" paired with "f", to receive the focal feature string transmitted, separated by symbols ",", with the same quantity of features transmitted by "f".

t its///"t" represented other types of vocabulary.

*s volume||**//"s" represented the noun, and "**" represented that it could be replaced by other nouns.*

v is ///"v" represented the verb.

c a

s constant

s value

- Uncovered::Basin::Volume //"-" referred to the variable mentioned earlier that contained the semantic feature "Uncovered::Basin::Volume". If it failed, the corresponding variable should be generated in the semantic scene mentioned earlier.

D Uncovered::Basin::Volume ///"D" represented the semantic feature "Uncovered::Basin::Volume" referred to the variable whose value was a constant.

C3///Semantic framework identifier

Normal///Semantic framework type

11 ///Quantity of semantic framework entries

F0 uncovered,basin, ///"F0" represented the transfer of focal features across semantic frameworks, not only with the same quantity of features defined by "f", but also with the same semantic string.

t In order to||To~ ///"t" represented other types of vocabulary.

v minimize ///"v" represented the verb.

t the

s use

t of

*s sheet||***

t as much as

t possible

Uncovered::Basin::Surface~Area~ ///"-" referred to the variable that contained the semantic feature "Uncovered::Basin::Surface~Area~" mentioned earlier. If it failed, the corresponding semantic variable should be generated in the semantic scene mentioned earlier.

m Uncovered::Basin::Surface~Area~::MINVALUE ///"m" represented the intermediate data element variable, where "Uncovered::Basin::Surface~Area~" referred to the local variable name, and "MINVALUE" represented the property constraint (minimum value) of the variable with the semantic "Uncovered::Basin::Surface~Area~".

C4///Semantic framework identifier

Normal///Semantic framework type

16///Number of semantic framework entries

F uncovered,basin, /// "F" paired with "f", to receive the focal feature string transmitted, separated by symbols ",", with the same quantity of features transmitted by "f".

n Uncovered::Basin::Length///"n" represented the data element variable, where matching words such as "how much" specifically referred to the question variable. "Uncovered::Basin::Length" was the local semantic variable name.

v should ///"v" represented the verb.

v be

t the ///"t" represented other types of vocabulary.

s length

y , ///"y" represented the punctuation.

s width

y ,

t and

s height

t of

t the

*s basin||** ///"s" represented the noun, and "**" represented it could be replaced by other nouns.*

w Uncovered::Basin::Width///"w" represented the question data element variable, "Uncovered::Basin::Width" was the local semantic variable name.

w Uncovered::Basin::Height

b) Data element variables

After preprocessing (format adjustment, unit base unification, symbol system unification, named entity recognition, etc.), the original text of the problem was subjected to sentence segmentation, word segmentation, and part of speech recognition. Then, the whole sentence, fragment, and clause were matched with the pre-annotated semantic framework base to extract the local knowledge, information, and data explicitly represented by the matched semantic framework. For automatic resolving of mathematical application problems, the local scene information (identification string) of the semantic framework was analyzed, and the global scene information of each analysis unit was formed through semantic inheritance and overloading. Next, filtered and merged with local semantic information (such as local variable names, calculation formulas, logic relationships, and semantic features) through semantic inheritance and overloading techniques, combined with overloading and supplementation of commonsense knowledge, and formed the global semantic set of analysis units, such as data element variables (refer to **Table 1**) and dynamic semantic circles.

Table 1. Data element variables.

No	Location	Attribute	Name	Value
1	30	question variable	Uncovered::Basin::Length	unknown
2	38	middle variable	Uncovered::Basin::Volume	unknown
3	39	middle variable	Uncovered::Basin::Surface~Area~	unknown

Table 1. (*Continued*).

No	Location	Attribute	Name	Value
4	40	question variable	Uncovered::Basin::Width	unknown
5	41	question variable	Uncovered::Basin::Height	unknown
6	44	constant constraint variable	Uncovered::Basin::Volume	unknown
7	48	minimum value constraint variable	Uncovered::Basin::Surface~Area~	unknown

c) Dynamic semantic circle

The dynamic semantic circle (including explicit representation of commonsense knowledge) related to the above example included the following two formulas and their symbolic representations:

FL33///formula Identification ID

Cuboid, Uncovered, ///contextual features

Surface~Area~; Length; Width; Depth,Height,Thickness,Altitude,; /// data element variable table

Swimming~Pool~::Surface~Area~; Swimming~Pool~::Length; Swimming~Pool~::Width; Swimming~Pool~::Height /// formula variable table

Swimming~Pool~::Surface~Area~ = (Swimming~Pool~::Length × Swimming~Pool~::Width) + (Swimming~Pool~::Length × Swimming~Pool~::Height × 2) + (Swimming~Pool~::Width × Swimming~Pool~::Height × 2) ///formula semantic representation

Surface~Area~, Length, Width, Height,; S, x, y, z,; ///variable semantic symbol correspondence table

S = (1,1) × (3,+(1,1) x (1) y (1) z (0),+(1,2) x (1) y (0) z (1),+(1,2) x (0) y (1) z (1),)/1 /// symbolized representation of the formula

FL23///formula Identification ID

Cuboid, ///context features

Volume; Length; Width; Depth,Height,Thickness,Altitude,; /// data Element Variable Table

Cuboid~Volume~; Cuboid~Length~; Cuboid~Width~; Cuboid~Height~ /// formula variable table

Cuboid~Volume~ = Cuboid~Length~ × Cuboid~Width~ × Cuboid~Height~ ///formula semantic representation

Cuboid~Volume~, Cuboid~Length~, Cuboid~Width~, Cuboid~Height~,; V, x, y, z,; ///variable semantic symbol correspondence table

V = (1,1) × (1,+(1,1) x (1) y (1) z (1),)/1 ///symbolized representation of the formula

d) Thinking mechanism

Based on the information provided by the data element variable table and dynamic semantic circle, effective computing and derivations could be performed to resolve the problem. It should not only include general derivation, equation resolving, enumeration resolving, feature constraint derivation, and iterative exploring, but also include methods such as unequal relationship derivation and symbol system equivalent derivation.

5.2. Static logic structure

The development requirements for intelligent systems should become clear through the accumulation of experience in instance processing gradually. Clear static logic structure and dynamic workflow during program development were crucial for system upgrade and maintenance. The function unit framework adopted in this article was used to achieve explicit representation of software workflow between function nodes, which was the important component of software macro logic visualization. In static state (non-running state), the software logic structure could be represented by 10-tuple <ID, FR, SN, TY, BF, BT, INIT, BODY, RETN, NOTE>, where: ID represented the current function node identifier, FR represented the parent node, SN represented the son node, TY represented the current node type, BF represented the successor node when the current function returned "false", BT represented the successor node when the current function returned "true", INIT was the variable and function initialization function before the current function node was executed, BODY represented the function body of the current node, RETN represented the parameter conversion function after the function node was executed, and NOTE was the function description. For example, there could be the following function units (refer to **Table 2**):

Table 2. Function units.

ID	FR	SN	TY	BF	BT	INIT	BODY	RETN	NOTE
1	NULL	2	general	NULL	NULL	NULL	diff_symbol_deduce_module	NULL	Solving module of partial differential equations
2	1	NULL	general	3	3	NULL	diff_vars_init	NULL	Initialization of variables in partial differential equation systems
3	1	NULL	general	4	4	NULL	diff_formulas_init	NULL	Initialization of partial differential equation system
4	1	6	general	5	5	display_equations_init_0	display_equations	display_equations_retn_0	Print partial differential equation system
6	4	NULL	general	7	7	NULL	dispaly_equations_num	NULL	Print the number of equations
7	4	NULL	general	5	5	NULL	display_loop_multi_equations	NULL	Print equation sequentially
5	1	11	general	NULL	NULL	NULL	diff_formulas_resolve	NULL	Resolving systems of partial differential equations
…	…	…	…	…	…	…	…	…	…
17	13	20	recursive	18	19	NULL	resolving_equations	NULL	Resolving equations
18	5	NULL	general	NULL	NULL	NULL	diff_free_deduce_chain_false	NULL	Free the storage space of the partial differential equation solving process chain and return false
…	…	…	…	…	…	…	…	…	…
74	73	NULL	Loop(head)	83	75	polynomial_var_init	is_loop_each_equation_LPBN	NULL	Loop for each equatiom (head node)
75	73	NULL	general	80	76	NULL	is_nExponent_0	NULL	The polynomial in the equation numerator has no variables to be eliminated

Table 2. (*Continued*).

ID	FR	SN	TY	BF	BT	INIT	BODY	RETN	NOTE
76	73	NULL	general	78	77	NULL	is_equation_denominator_constants	NULL	The denominator in the equation is a constant
77	73	NULL	general	84	84	NULL	constants_multiapplication_constants	NULL	Multiplying constants in equation
78	73	NULL	general	79	79	NULL	polynomial_multiply_init_denominator	NULL	Polynomial multiplication denominator variable transformation initialize
79	73	NULL	general	84	84	NULL	polynomial_multiply	NULL	Polynomial multiplication denominator variable transforms
80	73	NULL	general	18	81	NULL	is_nExponent_1	NULL	The exponent of the variable to be eliminated in the numerator polynomial of the equation is 1
81	73	NULL	general	82	82	NULL	assign_eleminationvar_exponent_zero	NULL	The exponent of the variable to be eliminated in the equation is assigned 0
82	73	NULL	general	79	79	NULL	polynomial_multiply_init_numerator	NULL	Polynomial multiplication numerator variable transforms
83	73	NULL	general	18	71	NULL	result_return	NULL	Return of variable elimination function
84	73	NULL	Loop(tail)	74	74	NULL	loop_each_equation_step_LPTL	NULL	Loop tail node of the variable elimination function
…	…	…	…	…	…	…	…	…	…

The types of function unit nodes listed in **Table 2** included general, recursive, loop (head), and loop (tail). link types included parent, son, return false, and return true links. The execution process of the function units in the above example was shown in **Figures 5–7**:

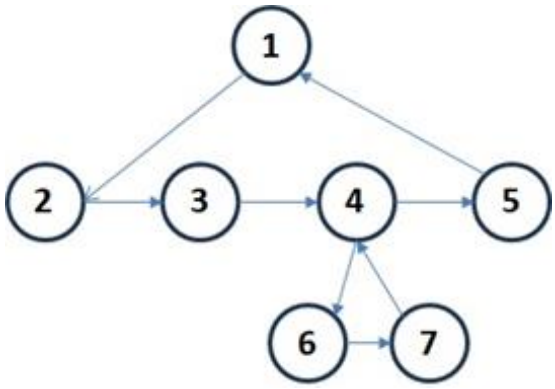

Figure 5. Son-function program structure.

Note: The untagged links represented the upper node returns the same value of "true" or "false" as the lower node (BT and BF links were the same). The original BODY program contents for function units with son nodes were not actually executed. For example, the execution sequence of **Figure 5** was: *1.INIT → 2.INIT → 2.BODY → 2.RETN → 3.INIT → 3.BODY → 3.RETN → 4.INIT → 6.INIT → 6.BODY → 6.RETN → 7.INIT → 7.BODY → 7.RETN → 4.RETN → 5.INIT → 5.BODY → 5.RETN → 1.RETN…*

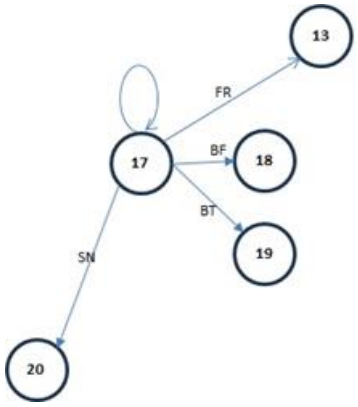

Figure 6. Recursive program structure.

Note: The original BODY program content of the recursive function unit with son nodes was not actually executed. For example, the node 17 in **Figure 6**, firstly executed its node initialization function 17.INIT, then recursively executed all its son nodes, subsequently executed the result variable transfer function 17.RETN of the recursive node, finally executed the subsequent workflow along 17.BT or 17.BF based on the return value "true" or "false".

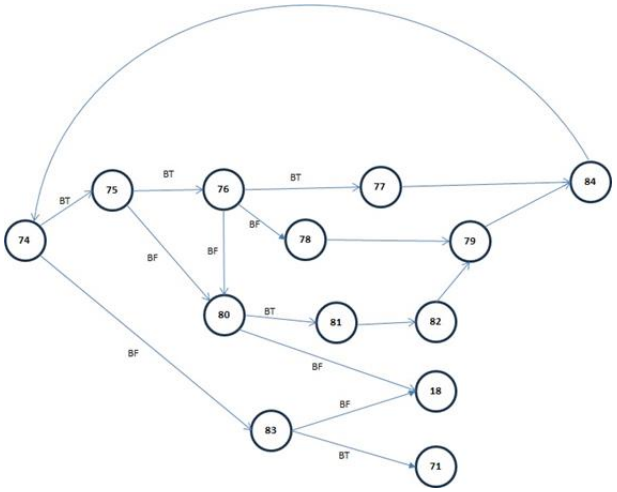

Figure 7. Loop program structure.

Note: Unlabeled link represented whether the previous node return "true" or "false" value, it should direct to the same subsequent node (BT and BF links are the same). Node 74 was the head node of the loop, and node 84 was the tail node of the loop.

5.3. Dynamic execution workflow

5.3.1. Son-nodes execution workflow

Table 3. Son-nodes execution workflow.

ID	TY	INIT	BODY	RETN
1	general	NULL	diff_symbol_deduce_module	NULL
2	general	NULL	display_equations_init_0	NULL
3	general	NULL	diff_formulas_init	NULL
4	general	display_equations_init_0	display_equations	display_equations_retn_0
6	general	NULL	dispaly_equations_num	NULL
7	general	NULL	display_loop_multi_equations	NULL
5	general	NULL	diff_formulas_resolve	NULL
…	…	…	…	…

As shown above, the son-nodes execution workflow (refer to **Table 3**) in **Figure 5** was: 1→2→3→4→6→7→5…

5.3.2. Recursive nodes execution workflow

Table 4. Recursive nodes execution workflow.

ID	TY	INIT	BODY	RETN
…	…	…	…	…
17	Recursive	NULL	resolving_equations	NULL
20	General	NULL	diff_resolving_equation_num_codition_1	NULL
…	…	…	…	…
17	Recursive	NULL	resolving_equations	NULL
20	General	NULL	diff_resolving_equation_num_codition_1	NULL
…	…	…	…	…
17	Recursive	NULL	resolving_equations	NULL
20	General	NULL	diff_resolving_equation_num_codition_1	NULL
…	…	…	…	…
17	Recursive	NULL	resolving_equations	NULL
20	General	NULL	diff_resolving_equation_num_codition_1	NULL
21	General	NULL	diff_resolving_equation_num_body_1	NULL
23	General	display_equations_init_n	display_equations	NULL
24	General	NULL	last_equal_exchange	NULL
58	General	NULL	is_diff_single_var_equation	NULL
59	General	NULL	single_var_equation_exchange	NULL
25	General	display_equations_init_n	display_equations	NULL
19	General	NULL	diff_free_deduce_chain_true	NULL
…	…	…	…	…

As shown above, the actual recursive nodes execution workflow (refer to **Table 4**) in **Figure 6** was... 17→20→... 17→20→... 17→20 ... 17→20→21→23→24→58→59→25→19... Among them, the first three sequence segments "17→20→…" were repeatedly executed three times, and the fourth time executed a different recursive process workflow from the previous three times.

5.3.3. Loop nodes execution workflow

Table 5. Loop nodes execution workflow.

ID	TY	INIT	BODY	RETN
…	…	…	…	…
74	Loop(head)	polynomial_var_init	is_loop_each_equation_LPBN	NULL
75	General	NULL	is_nExponent_0	NULL
76	General	NULL	is_equation_fenmu_constants	NULL
78	General	NULL	polynomial_chengfa_init_fenmu	NULL
79	General	NULL	polynomial_chengfa	NULL
84	Loop(tail)	NULL	loop_each_equation_step_LPTL	NULL

Table 5. (*Continued*).

ID	TY	INIT	BODY	RETN
74	Loop(head)	polynomial_var_init	is_loop_each_equation_LPBN	NULL
75	General	NULL	is_nExponent_0	NULL
76	General	NULL	is_equation_fenmu_constants	NULL
78	General	NULL	polynomial_chengfa_init_fenmu	NULL
79	General	NULL	polynomial_chengfa	NULL
84	Loop(tail)	NULL	loop_each_equation_step_LPTL	NULL
74	Loop(head)	polynomial_var_init	is_loop_each_equation_LPBN	NULL
75	General	NULL	is_nExponent_0	NULL
80	General	NULL	is_nExponent_1	NULL
81	General	NULL	assign_elemination_var_index_zero	NULL
82	General	NULL	polynomial_chengfa_init_fenzi	NULL
79	General	NULL	polynomial_chengfa	NULL
84	Loop(tail)	NULL	loop_each_equation_step_LPTL	NULL
74	Loop(head)	polynomial_var_init	is_loop_each_equation_LPBN	NULL
83	General	NULL	result_return	NULL
…	…	…	…	…

The loop usually quit from the "loop (head)" node, and the above loop nodes workflow (refer to **Table 5**) executed three iterations before leaving through node 83. Comparing the static logic structure and dynamic execution workflow of the program, it could be seen that the path 76→77→84 was not covered in this dynamic execution workflow (see **Figure 7**), which suggested that the system had the ability to detect the potential untested paths in the software.

5.4. Equivalent derivation

a) After semantic framework matching and global semantic analysis, it was determined that the calculation relationships contained in the original text, which included the formulas for uncovered basin volume and surface area. After unifying the variable symbol system, the following formulas could be obtained:

Uncovered::Basin::Volume a = 1.000000 × (1.0000000 x (1.000000) y (1.000000) z (1.000000)/1.000000)

Uncovered::Basin::Surface~Area~ S = 1.000000 × ((1.0000000 x (1.000000) y (1.000000) z (0.000000) + 2000000 x (1.000000) y (0.000000) z (1.000000) + 2000000 x (0.000000) y (1.000000) z (1.000000))/1.000000)

b) By introducing the Lagrange multiplier w, the equivalent combination of volume constant constraints was incorporated into the representation of the surface area computing function, resulting in the following constrained uncovered basin volume computing function:

Constrained::Uncovered::Basin::Surface~Area~ S' = 1.000000 × ((1.0000000 x (1.000000) y (1.000000) z (0.000000) w (0.000000) + 2.0000000 x (1.000000) y (0.000000) z (1.000000) w (0.000000) + 2.0000000 x (0.000000) y (1.000000) z (1.000000) w (0.000000) + 1.0000000 x (1.000000) y (1.000000) z (1.000000) w

(1.000000) − 1.0000000ax (0.000000) y (0.000000) z (0.000000) w (1.000000))/1.000000)

c) Derived the four variables of the constrained uncovered basin surface area computing function S', then could obtain a partial differential equation system consisting of four equations:

0 = 1.000000 × (1.000000 x (0.000000) y (1.000000) z (0.000000) w (0.000000) + 2.000000 x (0.000000) y (0.000000) z (1.000000) w (0.000000) + 1.000000 x (0.000000) y (1.000000) z (1.000000) w (1.000000))

0 = 1.000000 × (1.000000 x (1.000000) y (0.000000) z (0.000000) w (0.000000) + 2.000000 x (0.000000) y (0.000000) z (1.000000) w (0.000000) + 1.000000 x (1.000000) y (0.000000) z (1.000000) w (1.000000))

0 = 1.000000 × (2.000000 x (1.000000) y (0.000000) z (0.000000) w (0.000000) + 2.000000 x (0.000000) y (1.000000) z (0.000000) w (0.000000) + 1.000000 x (1.000000) y (1.000000) z (0.000000) w (1.000000))

0 = 1.000000 × (1.000000 x (1.000000) y (1.000000) z (1.000000) w (0.000000) − 1.000000ax (0.000000) y (0.000000) z (0.000000) w (0.000000))

d) After selecting the first elimination variable w and converting the representation, it could be represented by the following function:

w = 1.000000 × (1.000000 x (0.000000) y (1.000000) z (0.000000) w (0.000000) + 2.000000 x (0.000000) y (0.000000) z (1.000000) w (0.000000)) / −1.000000 x (0.000000) y (1.000000) z (1.000000) w (0.000000)

e) By substituting the elimination element, the equation system consisting of the following three equations could be obtained

0 = 1.000000 × (−2.000000 x (0.000000) y (1.000000) z (2.000000) w (0.000000) + 2.000000 x (1.000000) y (0.000000) z (2.000000) w (0.000000))

0 = 1.000000 × (−2.000000 x (0.000000) y (2.000000) z (1.000000) w (0.000000) + 1.000000 x (1.000000) y (2.000000) z (0.000000) w (0.000000))

0 = 1.000000 × (1.000000 x (1.000000) y (1.000000) z (1.000000) w (0.000000) − 1.000000ax (0.000000) y (0.000000) z (0.000000) w (0.000000))

f) After selecting the second elimination variable x and converting the representation, it can be represented by the following function:

x = 1.000000 × (1.000000 x (0.000000) y (1.000000) z (0.000000) w (0.000000))

g) By substituting the elimination element, we could obtain the equation system consisting of the following two equations

0 = 1.000000 × (−2.000000 x (0.000000) y (2.000000) z (1.000000) w (0.000000) + 1.000000 x (0.000000) y (3.000000) z (0.000000) w (0.000000))

0 = 1.000000 × (1.000000 x (0.000000) y (2.000000) z (1.000000) w (0.000000) − 1.000000ax (0.000000) y (0.000000) z (0.000000) w (0.000000))

h) After selecting the third elimination variable z and converting the representation, it could be represented by the following function:

z = 1.000000 × (0.500000 x (0.000000) y (1.000000) z (0.000000) w (0.000000))

i) By substituting the elimination element, the following equation could be obtained:

0 = 1.000000 ×(0.500000 x (0.000000) y (3.000000) z (0.000000) w (0.000000) − 1.000000ax (0.000000) y (0.000000) z (0.000000) w (0.000000))

j) Resolving the cubic equation yielded the variable y as:

y = 1.000000 × (cbrt(2.000000a) x (0.000000) y (0.000000) z (0.000000) w (0.000000))

k) By reversing the previous representation of the elimination variable function, the variables z and x could be resolved as follows:

z = 1.000000 × (0.500000cbrt(2.000000a) x (0.000000) y (0.000000) z (0.000000) w (0.000000))

x = 1.000000 × (cbrt(2.000000a) x (0.000000) y (0.000000) z (0.000000) w (0.000000))

Among them, cbrt() was the cube root function identifier.

6. Conclusion

The equivalent derivation in symbol systems was an important part of realizing machine self-scientific exploration and knowledge discovery, involving semantic understanding of problem scenes, machine representation conversion of mathematic symbols, large-scale language pattern matching, development and maintenance of complex logic algorithms, and core key technologies for machine thinking modes implementation. It required resolving practical engineering problems such as extracting commonsense knowledge from massive information, resolving semantic conflicts, global semantic accumulation and condensation, formal representation and self-improvement of the symbol systems, hypothesis proposing, and verifying. This article proposed the engineering implementation method based on framework chain visualization programming to solve the non-linear increasing problem complexity in software caused by progressive processing logic accumulation. Using advanced mathematical application problems as examples, the implementation process of equivalent transformation for partial differential equations was introduced, which had important implications for the implementation of equivalent derivation engines in mathematical symbolic systems. In the future, on the one hand, we would comprehensively implement the machine representation of the mathematics symbol systems and the derivation rules of equivalence relationships; on the other hand, we would continue to explore the self-generation of symbol systems, discover new laws through machine implementation, and try to implement deep machine thinking.

Author contributions: Conceptualization, PZ; methodology, PZ; software, PZ; validation, YM; formal analysis, WZ; investigation, XJ; resources, PL; data curation, JS; project administration, PL; supervision, WZ; writing—original draft, PZ; writing—review and editing, YZ. All authors have read and agreed to the published version of the manuscript.

Conflict of interest: The authors declare no conflict of interest.

References

1. Newcomb A, Kalita J. Explaining Math Word Problem Solvers. In: Proceedings of the 2022 6th International Conference on Natural Language Processing and Information Retrieval; 2022.

2. Qin J, Huang Z, Zeng Y, et al. An Introspective Data Augmentation Method for Training Math Word Problem Solvers. In: IEEE/ACM Transactions on Audio, Speech, and Language Processing; 2024.
3. Paliwal P. Adversarial Analysis and Methods for Math Word Problems. In: Proceedings of the International Conference on Computing, Machine Learning and Data Science. 2024.
4. Zhu P, Lv P, Shi J, et al. Design and implementation of text understanding system based on semantic tagging Instances. In: Proceedings of the 4th International Conference on Artificial Intelligence in Electronics Engineering (AIEE '23); 2023.
5. Patel M, Dogan FI, Zeng Z, et al. Semantic scene understanding for human-robot interaction. In: Proceedings of the Companion of the ACM/IEEE International Conference on Human-Robot Interaction (HRI '23); 2023.
6. Nguyen TP, Razniewski S, Varde A, and Weikum G. Extracting cultural commonsense knowledge at scale. In: Proceedings of the ACM Web Conference (WWW '23); 2023.
7. Iravanian S, Gowda S, Rackauckas C. Hybrid Symbolic-Numeric and Numerically-Assisted Symbolic Integration. In: Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation; 2024.
8. Taihei Oki and Yujin Song. Structural preprocessing method for nonlinear differential-algebraic equations using linear symbolic matrices. In: Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation (ISSAC '24); 2024.
9. Kaltofen EL. Encounters in Symbolic Computation: Ideas for the Ages. In: Proceedings of the 2024 International Symposium on Symbolic and Algebraic Computation; 2024.
10. Gewaltig MO. Towards Simulating the Human Brain. In: Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation; 2017.
11. Meilong C, Xiehua Y, Shaozi L, et al. Design of Graphic Interactive Experimental Platform Based on MATLAB. In: Proceedings of the 2022 12th International Conference on Information Technology in Medicine and Education (ITME); 2022.
12. Valina L, Teixeira B, Reis A, et al. Explainable Artificial Intelligence for Deep Synthetic Data Generation Models. In: Proceedings of the 2024 IEEE Conference on Artificial Intelligence (CAI); 2024.
13. Hickling T, Zenati A, Aouf N, et al. Explainability in Deep Reinforcement Learning: A Review into Current Methods and Applications. ACM Computing Surveys. 2023; 56(5): 1-35. doi: 10.1145/3623377.
14. Ahmed YA, Sharo A. On the education effect of CHATGPT: Is AI CHATGPT to dominate education career profession? In: Proceedings of the 2023 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS); 2023.
15. Arulmohan S, Meurs MJ, Mosser S. Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In: Proceedings of the 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C); 2023.
16. Franceschelli G, Musolesi M. Creativity and Machine Learning: A Survey. ACM Computing Surveys. 2024; 56(11): 1-41. doi: 10.1145/3664595
17. Subagdja B, Shanthoshigaa D, Wang Z, et al. Machine Learning for Refining Knowledge Graphs: A Survey. ACM Computing Surveys. 2024; 56(6): 1-38. doi: 10.1145/3640313
18. Seyyedi A, Bohlouli M, Oskoee SN. Machine Learning and Physics: A Survey of Integrated Models. ACM Computing Surveys. 2023; 56(5): 1-33. doi: 10.1145/3611383
19. Brunton SL, Kutz JN. Promising directions of machine learning for partial differential equations. Nature Computational Science. 2024; 4(7): 483-494. doi: 10.1038/s43588-024-00643-2
20. Duan X, Wang X, Zhao P, et al. DeepLogic: Joint Learning of Neural Perception and Logical Reasoning. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2022: 1-14. doi: 10.1109/tpami.2022.3191093
21. Vamplew P, Foale C, Hayes CF, et al. Utility-based reinforcement learning: Unifying single-objective and multi-objective reinforcement learning. In: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS '24); 2024.
22. Flageat M, Lim B, Cully A. Evolutionary Reinforcement Learning. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion; 2024.
23. Harada T, Alba E. Parallel Genetic Algorithms. ACM Computing Surveys. 2020; 53(4): 1-39. doi: 10.1145/3400031
24. Zamanzadeh Darban Z, Webb GI, Pan S, et al. Deep Learning for Time Series Anomaly Detection: A Survey. ACM Computing Surveys. 2024; 57(1): 1-42. doi: 10.1145/3691338

25. Wan Y, Bi Z, He Y, et al. Deep Learning for Code Intelligence: Survey, Benchmark and Toolkit. ACM Computing Surveys. 2024; 56(12): 1-41. doi: 10.1145/3664597
26. Wijesekara PADSN, Wang YK. A Mathematical Epidemiological Model (SEQIJRDS) to Recommend Public Health Interventions Related to COVID-19 in Sri Lanka. COVID. 2022; 2(6): 793-826. doi: 10.3390/covid2060059
27. Wijesekara PADSN. Deep 3D Dynamic Object Detection towards Successful and Safe Navigation for Full Autonomous Driving. The Open Transportation Journal. 2022; 16(1). doi: 10.2174/18744478-v16-e2208191
28. Nguyen K, Proença H, Alonso-Fernandez F. Deep Learning for Iris Recognition: A Survey. ACM Computing Surveys. 2024; 56(9): 1-35. doi: 10.1145/3651306
29. Spannaus A, Hanson HA, Tourassi G, et al. Topological Interpretability for Deep Learning. In: Proceedings of the Platform for Advanced Scientific Computing Conference; 2024.
30. Kletsko E, van Rozen R. Advanced Game Engine Wizardry for Visual Programming Environments. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments; 2024.
31. Homer M. Reclaiming the Unexplored in Hybrid Visual Programming. In: Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software; 2024.
32. Homer M. In-Line Compositional Visual Programming. In: Proceedings of the 8th International Conference on the Art, Science, and Engineering of Programming; 2024.
33. Glock J. Aiding Developer Understanding of Software Changes via Symbolic Execution-based Semantic Differencing. In: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings; 2024.
34. Krestel R, Aras H, Andersson L, et al. In: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, Proceedings of the 5th Workshop on Patent Text Mining and Semantic Technologies (PatentSemTech2024); 2024.
35. Huang S, Luan Z. Semantic-Aware Log Understanding and Analysis. In: Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing; 2024.
36. Fang J, Wang W, Luo T, et al. Progressive Multimodal Pivot Learning: Towards Semantic Discordance Understanding as Humans. In: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management; 2024.
37. D'Aquin M, Bunoiu R, Cirstea H, et al. Combining representation formalisms for reasoning upon mathematical knowledge. In: Proceedings of the 12th Knowledge Capture Conference 2023; 2023.
38. Liu J, Huang Z, Ma Z, et al. Guiding Mathematical Reasoning via Mastering Commonsense Formula Knowledge. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining; 2023.
39. Greiner-Petter A, Schubotz M, Müller F, et al. Discovering Mathematical Objects of Interest—A Study of Mathematical Notations. In: Proceedings of The Web Conference 2020; 2020.
40. Zhao WX, Zhou K, Gong Z, et al. JiuZhang: A Chinese Pre-trained Language Model for Mathematical Problem Understanding. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining; 2022.
41. Xu J, Fei H, Pan L, et al. Faithful logical reasoning via symbolic chain-of-thought. Available online: https://arxiv.org/pdf/2405.18357.pdf (accessed on 1 May 2024).

Appendix

(a) Data structure definitions

struct STRUCT_FUNTION_ENTRANCE

{

*Boolean (*App)();*

///Function pointer, the function parameters were the ID of the basic logic condition table and the pointers array pointing to the I/O data pool. The logic meanings of the function parameters were initialized in the basic logic condition table.

Char str_annotation [MAX_ATTRIBUTES_LIST_LEN]; /// The text string describing what the current function could do.

Boolean bRet;///Function returned the Boolean value.

Boolean bSEL;///Function used the Boolean value as the flag, where "true" indicated that this function is determined as the basic logic condition, and "false" indicated that this function was ignored.

};

struct STRUCT_GENERAL_FRAME_INTERFACE

{

Long nID;/// Current framework node identifier ID;

Boolean bInit=false;/// "false" for the first execution of the loop node, "true" for all others.

Int nType;///Divided into recursive nodes, loop beginning nodes, loop ending nodes, general nodes, etc;

*struct STRUCT_GENERAL_FRAME_INTERFACE * pPrev;///Pointed to the previous framework node.*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pTNext;///Pointed to the next framework node when the function returned "true".*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pFNext;///Pointed to the next framework node when the function returned "fslse".*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pSon;///Pointed to the first son framework node.*

*struct STRUCT_GENERAL_FRAME_INTERFACE * pFather;///Pointed to the parent framework node.*

*struct STRUCT_FUNTION_ENTRANCE * fun_init;/// Initialized the logic branch variables.*

*struct STRUCT-FUNTION-INTRANCE * fun_execute_body;///Logic branch function body.*

*struct STRUCT-FUNTION-INTRANCE * fun_rtn;///Logic branch function body returned values passing function.*

*struct STRUCT_FUNTION_ENTRANCE *sRefer_Table [MAX_OR_CONDITIONS_NUM] [MAX_AND_FUNCTION_NUM];*

///Two-dimension basic logic condition table

*Boolean (*is_result_ validated) ();*

///Function pointer for result validation.

*Boolean (*Record_Input_Data) ();*

///Function pointer for input data recording function.

*Boolean (*Record_Output_Data) ();*

///Function pointer for recording output results.

*Boolean (*Record_Output_Format_Return) ();*

///Function pointer for unified output format function.

};

(b) Variables definitions

Each framework node could be represented by the struct *STRUCT_GENERAL_FRAME_INTERFACE*, with the following node variable declaration:

struct STRUCT_GENERAL_FRAME_INTERFACE A, B, C, D, E, F;

*struct STRUCT_GENERAL_FRAME_INTERFACE * pHead;*

The initialization algorithm *Init_chain_a()* of the framework-oriented programming in **Figure 5** was represented as follows (the *sRefer_Table* pointer was *NULL*, indicating the unconditional execution of the function pointed by *fun_execute_body*):

(c) Algorithm A1 Init_chain_a()

Algorithm A1 Init_chain_a()

Input: Framework nodes A, B, C, D, E, F;
Output: Framework net with the unified input entry pHead.
Algorithm description: Init_chain_a()
*①A.nID = 1; A.pPrev = NULL; A.pTNext = &B; A.pFNext = &B; A.pSon = NULL; A.pFathter = NULL; A.fun_execute_body = …; memset(&(A. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); … /// Initialize the context parameters and function parameters of the framework node A.*
*②B.nID = 2; B.pPrev = &A; B.pTNext = &C; B.pFNext = &C; B.pSon = NULL; B.pFathter = NULL; B.fun_execute_body = …; memset(&(B. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); …/// Initialize the context parameters and function parameters of the framework node B.*
*③C.nID = 3; C.pPrev = &B; C.pTNext = &D; C.pFNext = &D; C.pSon = NULL; C.pFathter = NULL; C.fun_execute_body = …; memset(&(C. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); … /// Initialize the context parameters and function parameters of the framework node C.*
*④D.nID = 4; D.pPrev = &C; D.pTNext = NULL; D.pFNext = NULL; D.pSon = NULL; D.pFathter = NULL; D.fun_execute_body = …; memset(&(D. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE)); … /// Initialize the context parameters and function parameters of the framework node D.*
⑤pHead = &A;

(d) Algorithm add_chain_b()

The Algorithm A2 add_chain_b() of the framework-oriented programming process in **Figure 6** could be listed as follows:

Algorithm A2 add_chain_b()

Input: Framework nodes A, B, C, D, E;
Output: Framework net with the unified input entry pHead.
Algorithm description: add_chain_b()
①Init_chain_a() /// Initialize the context parameters and function parameters of the framework node chain pHead.
②E. sRefer_Table = B_E_sRefer;///"B_E_sRefer" was the prerequisite table for the function framework node B and E.
③B. pTNext =&C; B. pFNext =&E;///After framework node B was executed, turned to framework node C when returned true, turned to framework node E when returned false.
*④E.nID =5; E.pPrev=&B; E.pTNext=&C; E.pFNext=&C; E.pSon=NULL; E.pFathter=NULL; E.fun_execute_body=…; memset(&(E. sRefer_Table [0][0]),0, MAX_OR_CONDITIONS_NUM * MAX_AND_FUNCTION_NUM * sizeof(struct STRUCT_FUNTION_ENTRANCE));… /// After adding the new framework node E, it unconditionally switched to framework node C (similar to the "else" statement of conditional statements).*

(e) Algorithm add_chain_c()

The Algorithm A3 add_chain_c() of the framework-oriented programming process in **Figure 7** could be listed as follows:

Algorithm A3 add_chain_b()

Input: Framework nodes A, B, C, D, F;
Output: Framework net with the unified input entry pHead.
Algorithm description: add_chain_b()
①Init_chain_a() /// Initialize the context parameters and function parameters of the framework node chain pHead.

Algorithm A3 (*Continued*)

5: *②B. PTNext = &F; B. pFNext = &F;///After framework node B was executed, turned to framework node E.*

6: *③F. NID = 6; F.pPrev = &B; F.pTNext = &C; F.pFNext = &C; F.pSon = NULL; F.pFooter = NULL; F. fun_execute_body = ...; F. sRefer_Table = B_F_sRefer;///"B_F_sRefer" was the prerequisite table for the framework node F; If the default prerequisite was "false", the function body fun_execute_body should not be executed and should directly switch to the framework node C.*