Title:
IFOG: Inductive Functional Programming for Geometric Processing

Authors:
Masaji Tanaka, tanaka@mech.ous.ac.jp, Okayama University of Science
Yuki Takamiya, takamiya@mech.ous.ac.jp, Okayama University of Science
Naoki Tsubota, tsubota@mech.ous.ac.jp, Okayama University of Science
Kenzo Iwama, whatisiwama@yahoo.co.jp, EngiCom Corporation

Keywords:
IFOG, Inductive Programming, Functional Programming, Geometric Processing, Property, Combinatorial Search

DOI: 10.14733/cadconfP.2015.285-288

Introduction:
Since decades, especially in CAD and CG, to solve various kinds of problems and/or to develop automatic systems, not only geometric modeling techniques but also combinatorial searches of geometric elements such as line segments have been applied extensively. For example, to develop the automatic system that converts 2D drawings into 3D models, a great many automatic recognitions of complex geometric elements such as primitives and features are required for their programming. Generally, it is troublesome and time consuming to program the combinatorial searches for programmers because of the following reason. Firstly, they are basically algorithmic. Also, their formalization would be difficult because there are few mathematical bases in them. If a programmer wants to detect each parallelogram from a 2D drawing drawn in a 2D CAD, firstly he/she might search four straight lines, and then calculate their relationships. As the result, his/her programs for the detection would consist of too many procedures by using conventional programming techniques. On the other hand, in object-oriented programming (OOP), various kinds of classes can be defined, and programs almost consist of passing messages among objects as instances of the classes, e.g. [1]. So the class of straight lines can be made in the programming. However, though this class can be defined in detail, too many passing messages among straight lines would be required for the programming of this problem. In this paper, a new programming technique called IFOG (Inductive Functional prOgramming for Geometric processing) is proposed. IFOG enables to realize easier programing for programmers, especially for beginners, than conventional programming techniques for geometric processing such as this problem.

Main Idea:
Suppose that a little child has already known how to draw a straight line in a paper by using a pen. A teacher could teach him/her which line is longer than another line inductively by indicating plural examples of two straight lines drawn in papers. Generally, how to measure the length of a straight line and its real value are knowledge for humans. Also, the value is a property of the line. Continuously, when he/she learns about advanced geometry, the relationships among straight lines and the properties of them would be understood as knowledge step by step. The philosophical basis of IFOG is this learning process. In IFOG, the knowledge is expressed as program functions and the properties of geometric elements such as straight lines.

We have developed IFOG system that could be a semi-automatic programming framework for geometric processing. In the system, each of the functions could be automatically generalized inductively from plural examples as instances that are input by user(s). Also, the properties of each geometric element could be automatically generated from the functions step by step and generalized

as a class inductively. Suppose users make a great many functions and properties of geometric elements in the system. When these functions and properties are applied to a drawing, it would become an intelligent drawing. If a user wants to make a program that requires complex combinatorial searches of geometric elements in the drawing, his/her programming in the system would become easier and simpler than conventional programming. In the past, the authors attempted to program various kinds of combinatorial searches to restore partial omissions of 2D mechanical drawings by using properties of geometric elements, e.g. [4-5]. However, it was difficult to formalize a great many properties of geometric elements. IFOG would solve this issue by generating the properties inductively.

An example of programming in IFOG system is indicated as follows. Fig. 1 illustrates Example 1 that is a line drawing imaging a sketch of a cuboid. Example 1 is drawn in a 2D CAD and consists of seven points (P_1, P_2, .. , P_7) and nine lines (L_1, L_2, .. , L_9). In the system, initially the class of lines is filed as Class_Line.txt as in the left side of Fig. 2. In the class, two properties are defined as a number and terminals. Here, "di" (i=1,2,3) means default. When Example 1 is input to the system, firstly nine instances of lines are automatically made from the class and they are filed as Line.txt as in the right side of Fig. 2. Here, suppose a user wants to make a function to get length of a line. This function is named Get_length_of_line(). The left side of Fig. 3 illustrates the procedures to get length of L1. They are written by the user. Also, he/she can write the procedures to get length of L9 as in the right side of Fig. 3. When these two procedures in Fig. 3 are compared, it is found that each of two different numbers can be changed into a variable. For example, 'L1' and 'L9' can be expressed as 'Lv1' where 'v1' is a variable. In the same way, eight variables (v1, v2, … v8) can be made as in Fig. 4. This figure expresses the function Get_length_of_line(). Here, the two procedures can be two instances of the function, and their generalization is automatically processed in the system. This generalization technique is based on [2-3]. In this function, Memo_Line_property(Lv1, Length, v8) is a sub-function to add property 'Length' to the properties of L_{v1}. So, "Length: 51" and "Length: 36" are added to the properties of L_1 and L_9 respectively as in the left side of Fig. 5. Moreover, "Length: d4" is automatically added to the class of lines in the system as in the middle of Fig. 5, and then each length of the other lines is automatically added to Line.txt as in the right side of Fig. 5 in the system.

In IFOG system, classes and their instances are stored as text files in a PC. Generally in OOP, the definition of classes is strictly formalized and it is difficult to change the contents of them because instances are made from their classes. On the contrary, in IFOG system, reading and writing of the properties of classes and their instances is easy and flexible for programmers because classes can be made from instances inductively and they are stored as text files.

In IFOG system, the detection of a parallelogram in Example 1 could be formalized as a function as follows. Suppose 'Connecting line numbers at Pd' is added as two properties of the class of lines. Also, 'Parallel line numbers' is added as a property of the class of lines. For example, the properties of L_1 can be expressed as follows. { No.: L1; Terminal No.: P1, P2; Length: 51; Connecting line No. at P1: L4; Connecting line No. at P2: L2, L3; Parallel line No.: L5, L7; }. An instance of the function named Get_parallelogram() can be described as follows.

Get_parallelogram()

Procedure:

(L1, L5) = Get_two_lines_whose_lengths_are_the_same(); yes = Two_lines_are_parallel(L1, L5);

(L3, L4) = Get_line_whose_terminals_are_in(L1, L5);

yes = Lengths_of_two_lines_are_the_same(L3, L4); yes = Two_lines_are_parallel(L3, L4);

return (L1, L5, L3, L4);

This instance consists of four sub-functions. Each of their programming could be simple and easy by using properties of lines. In addition, the class of parallelograms would be made in the instance. Furthermore, the detection of a sketch of a cuboid in Example 1 could be formalized easily as a function in IFOG system.

Conclusions:

In this paper, a new programming technique called IFOG and its system are proposed. In IFOG system, functions and the properties of geometric elements are automatically generalized from instances that

are user input. As the result, IFOG could realize easier programming for geometric processing than conventional programming techniques such as OOP. An example of programming in IFOG system is indicated by using Example 1. When more curved lines and 3D geometric elements are handled in IFOG system, the text files expressing their properties would become large rapidly and inductively. It is an important issue for our IFOG system. However, basically reading and writing of the files are automatically executed by computers. Therefore, although it is important to make robuster format of the files, we estimate that the increase of the amount of information of the files does not become a serious problem.

References:

[1] Budd, T.: A Little Smalltalk, Addison-Wesley Publishing, 1987.

[2] Fujiwara, M.; Iwama, K.: A program that acquires how to execute sentences, WSEAS Transactions on Computers, 8(8), 2009, 1348-1357.

[3] Iwama, K.: A robotic program that acquires concepts and begins introspection, NueroQuantology, 4(4), 2006, 321-328.

[4] Tanaka, M.; Kaneeda, T.; Yamahira, T.; Iwama, K.: A Method to Restore Partial Omissions in 2D Drawings, Computer-Aided Design & Applications, 3(1-4), 2006, 341-347. http://dx.doi.org/10.1080/16864360.2006.10738472

[5] Tanaka, M.; Kaneeda, T.; Sasae, D.; Fukagawa, J.; Yokoi, R.: The Learning System to Restore Operations of Isolated Line Segments in 2D Drawings, Computer-Aided Design & Applications, 5(1-4), 2008, 354-362. http://dx.doi.org/10.3722/cadaps.2008.354-362

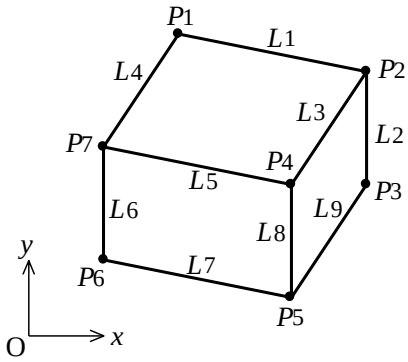

Fig. 1: Example 1.

```
No.: Ld1;
Terminals: Pd2, Pd3;
```

Fig. 2: Initial Class_Line.txt and Line.txt.

```
No.: L1;
Terminals: P1, P2;
No.: L2;
Terminals: P2, P3;
.....
No.: L9;
Terminals: P3, P5;
```

```
Get_length_of_line(L1)
procedure:
(P1, P2) = Get_terminal_numbers_of(L1);
(40, 80) = Get_position_of(P1);
(90, 70) = Get_position_of(P2);
51 = ( (40-90)^2 + (80-70)^2 )^0.5;
Memo_Line_property(L1, Length, 51);
return 51;
```

```
Get_length_of_line(L9)
procedure:
(P3, P5) = Get_terminal_numbers_of(L9);
(90, 40) = Get_position_of(P3);
(70, 10) = Get_position_of(P5);
36 = ( (90-70)^2 + (40-10)^2 )^0.5;
Memo_Line_property(L9, Length, 36);
return 36;
```

Fig. 3: Two procedures to get length of a line.

```
Get_length_of_line(Lv1)
procedure:
(Pv2, Pv3) = Get_terminal_numbers_of(Lv1);
(v4, v5) = Get_position_of(Pv2);
(v6, v7) = Get_position_of(Pv3);
v8 = ( (v4-v6)^2 + (v5-v7)^2 )^0.5;
Memo_Line_property(Lv1, Length, v8);
return v8;
```

Fig. 4: The Generalization of Fig. 3 into a function.

```
No.: L1;
Terminals: P1, P2;
Length: 51;
No.: L2;
Terminals: P2, P3;
.....
No.: L8;
Terminals: P3, P5;
No.: L9;
Terminals: P3, P5;
Length: 36;
```

```
No.: Ld1;
Terminal No.: Pd2, Pd3;
Length: d4;
```

```
No.: L1;
Terminals: P1, P2;
Length: 51;
No.: L2;
Terminals: P2, P3;
Length: 30;
.....
No.: L8;
Terminals: P3, P5;
Length: 30;
No.: L9;
Terminals: P3, P5;
Length: 36;
```

Fig. 5: Update of Class_line.txt and Line.txt.