COOL: A Chinese Object-Oriented Language for Knowledge Representation of Geometric Figures

Wing-Kwong Wong, Chao-Nan Chiu, and Hsi-Hsun Yang
Institute of Electronic and Information Engineering
National Yunlin University of Science and Technology
Touliu, Yunlin 640, Taiwan
wongwk@yuntech.edu.tw, {chaonan, jimmy}@cane.yuntech.edu.tw

Abstract

In order to provide a user-friendly interface for students to use a computational environment for learning programming in Chinese and learning elementary geometry, we have developed a Chinese Object-Oriented Language (COOL) for knowledge representation and a script language for describing geometric figures. COOL is used to define classes such as line segment and triangle and is similar in some aspects to other knowledge representation languages such as frames and semantic nets. A user can use the script language to draw geometric figures in a very simple, intuitive way. The script language is interpreted as Chinese Logo (CLogo) source code, whose execution will display the geometric figure in a CLogo drawing environment.

Keywords: Knowledge representation language, Chinese Logo, computer-assisted learning environment, geometry education, children education, Chinese programming

Introduction

In many countries children can learn programming as early as in elementary school [4, 11]. In Taiwan and other regions using Chinese, children are not so fortunate. They have few choices---they might learn Visual Basic or Logo [7]. While Visual Basic is quite complicated for children, Logo is a much better choice. However, most Logo implementations are in English or other non-Chinese languages. This poses a problem for children whose first language is Chinese because they have little or no English knowledge. Fortunately, now they can learn a fully Chinese Logo (CLogo) language [9] without worrying about the English obstacle.

In order to make CLogo more user-friendly, the learning environment provides on-line help. Such help includes on-line manual (http://plum.yuntech.edu.tw/CLogo/) and dynamic CLogo code generation from scripts input by user. This script has a simple syntax and is supported by a knowledge representation language called the Chinese Object-Oriented Language (COOL). COOL is developed in order to provide more interactive and dynamic help for children to learn CLogo programming and elementary geometry. COOL provides a simple script language for students to describe geometric figures. The script language will be interpreted by COOL as CLogo code. Hence the students can juxtapose the generated CLogo code with the original script. This should help them to learn CLogo with greater efficiency and with more fun. A better way in this direction is to provide a natural language interface so that textual description of geometric figures will be "understood" as a COOL script, which is interpreted as CLogo code. In this way, students need not learn COOL script. In fact, in a previous project [10], a system is developed to "understand" short sentences describing geometric objects as CLogo code without the use of COOL, which does not exist at that time. But the system uses some intermediate representation we call "semantic tree". This semantic tree is in fact very close to the script language of COOL and motivates us to design COOL in the very beginning.

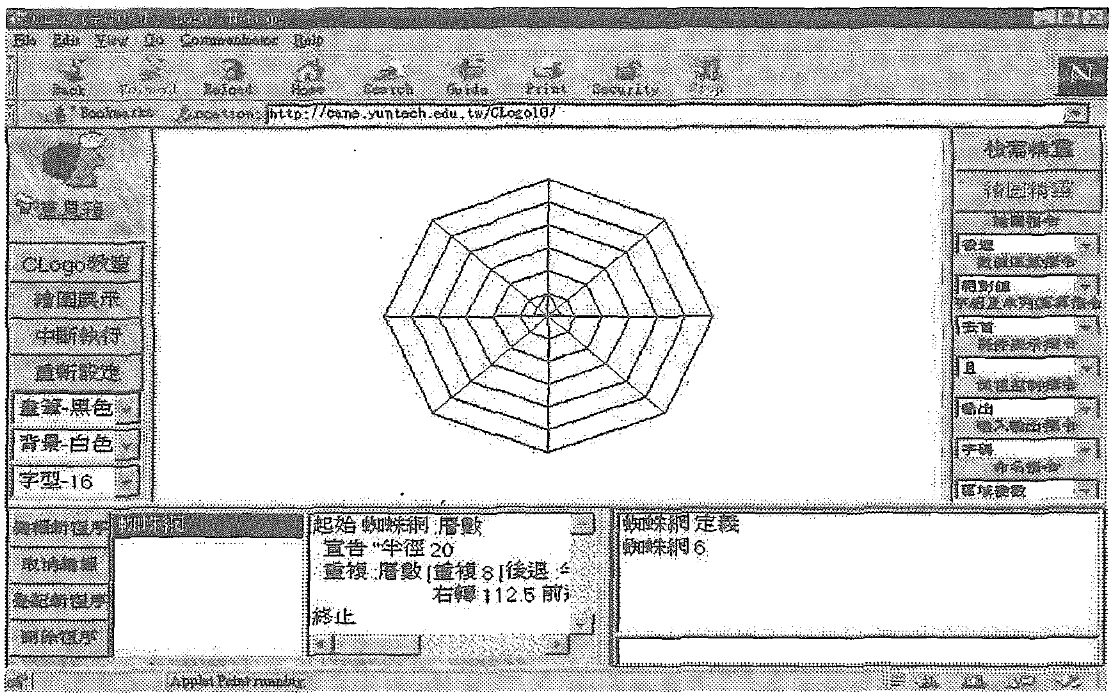
In the rest of this paper, we will present the CLogo learning environment, the design features of COOL and its script language, examples of how to use the script language, and how the script is interpreted as CLogo code. The final section will discuss the limitations of the current version of COOL, how it can be improved and in what areas it can be applied.

CLogo Learning Environment

The Logo programming language and visualization environment sets a landmark in computer-assisted learning since the seventies [7]. It offers an exploratory programming and visualization environment for students to learn programming [1, 2]. The main features of Logo include:

1. An imaginary turtle, upon moving, draws a visible polyline on its path in a canvas window.
2. A user can command the turtle to travel with primitive Logo commands such as moving forward, turning right, and repeating a set of commands.
3. New procedures can be defined and executed with different parameter values.
4. Procedures can be defined recursively.
5. The language offers a list data structure, similar to LISP.

Logo has attracted a lot of attention from researchers and has been controversial since its birth. While some researchers claim that it stimulates students to explore and learn about the microworld of turtle graphics [4], others rebut that unguided learning is inefficient and students waste a lot of time resulting in little cognitive development [3]. Moreover, some researchers claim that the learning in Logo programming can transfer to other areas of cognitive activity resulting in better problem solving and intellectual ability [11], while others use empirical experiments to show that no such transfer occurs [8]. We believe that with enough teacher guidance, students can explore the target concepts efficiently and that with carefully designed lessons, students can learn interesting and practical


File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://cane.yuntech.edu.tw/CLogo10/
CLogo教室
繪圖展示
中斷執行
重新設定
畫筆-黑色
背景-白色
字型-16
蜘蛛網
Applet Paint running

```
起始 蜘蛛網 :層數
 宣告 "半徑 20
 重複 :層數 [重複 8 [後退 :
            右轉 112.5 前
終止
```

```
蜘蛛網 定義
蜘蛛網 6
```

Figure 1: CLogo Learning Environment on WWW

mainstream math curriculum and other related subjects in a Logo environment.

In order to get rid of the English obstacle for Chinese children, we have designed and implemented a fully Chinese Logo (CLogo) [9]. CLogo is implemented in Java so that it is part of our homepage and can be accessed across the World Wide Web with the Netscape browser (Figure 1). Because Chinese string processing is not a standard feature of Java, the current version of CLogo does not show Chinese properly with Netscape browser Version 4.0.6 and above. Thus, users are required to use Netscape browser Version 4.0.5 when using CLogo on the Web.

Chinese Object-Oriented Language for Knowledge Representation

In the last few decades, many knowledge representational schemes have been proposed. Mylopoulos and Levesque [6] have classified these into four categories: logical representation schemes such as Prolog, procedural representation schemes such as rule-based production systems, network representational schemes such as semantic networks, and structured representation schemes such as frames and CLOS of LISP [5]. Our Chinese Object-Oriented Language falls into the fourth category. In COOL, a class definition uses the following template (the original Chinese definition of the triangle class is given in Appendix A):

[Class Name]
<Name>
[Parent Class]
<Name >
[Attributes]
<attribute> .. <attribute>
[Ranges of Values of Attributes]
<Attribute>:<Range> .. <Attribute>:<Range>
[Constraints]
<Equation/Inequality> .. <Equation/Inequality>
[Visualization Method Selection Preconditions]
<Preconditions for method>..<Preconditions for method>
[Visualization Methods]
<method> .. <method>

Inheritance is a characteristic of object-oriented programming languages. If class C inherits from class P, then class C gets all attributes, constraints, visualization methods, etc. from class P. In this way, P is called the parent class or superclass of C and C is called a child class or subclass of P. In COOL, each class inherits from only one parent class. This design of single inheritance is much simpler than that of multiple inheritance, where a class can have two or more parent classes. For example, the C++ programming language allows multiple inheritance. Inheritance provides an excellent way to reduce the amount of source code for defining classes, since the source code for defining P need not be repeated in the definitions of its subclasses. The saving can be quite significant because a medium-sized knowledge base might involve many classes, which can form a nice class hierarchy. For example, both classes of isosceles triangle and right angle triangle inherit from the triangle class; the triangle class and the line segment class inherit from the geometric object class. Since the discussion of the details of inheritance involves the concepts of "attributes", "constraints", and "visualization methods," we will delay examining an example of inheritance until after these concepts are introduced.

Simply speaking, attributes are the properties of a class. For example, a point has attribute x-coordinate and attribute y-coordinate; a line segment has a length attribute and end-points attributes; a triangle has attributes of three sides, three vertices and three internal angles. In a frame system, an attribute is called a slot to be filled with a value. When an object is declared to belong to a class, the object would have some of its attributes values specified. For example, a specific point object A might have its x and y coordinates specified as 0 and 10 respectively; a line segment object AB might have its end-points specified as point objects A and B; a triangle ABC might have its sides

specified as line segments AB, BC, and CA.
When a user wants to draw a geometric figure, she might provide sufficient information. For example, she can specify the coordinates of three points of a triangle, which are sufficient for drawing the triangle. On the other hand, a user might not provide sufficient information. In this case, the system would generate at random some information needed for the drawing. For example, a user might just specify the lengths of two sides of a triangle. Then the system would generate at random the angle in degrees for the inclusive angle formed by the two sides. Hence, the range of the values of some attributes should be specified so that the values, if needed, can be generated at random within the range. A range is specified as the values within a lower bound and an upper bound, where the limiting values might be included or excluded. For example, an angle of a triangle might be limited to [30,120] (from 30 degrees to 120 degrees inclusively at both ends) even though the theoretic limit is (0,180) (from 0 to 180 degrees excluding 0 and 180) in order to restrict the generation of an "average-case" triangle.
Constraints of a class specify algebraic relations among the attributes of the class. These relations can usually be specified as equalities and inequalities. For example, the sum of the internal angles of a triangle must equal 180 degrees; the sum of two angles must be less than 180 degrees. Such information can be used in at least two ways. First, with this information one can check whether the user's inputs are consistent or not. If the user specifies a triangle with angles 30 degrees, 45 degrees, and 90 degrees, then the system can detect that the sum of angles does not equal 180 degrees and inform the user about this inconsistency. Second, the system can use the equality to derive the value of a non-instantiated attribute. For example, given the values of two angles of a triangle, the system can derive the value of the third angle. This inference capability is a very significant feature of COOL. In general, the ability to manipulate algebraic equations and inequalities provides a powerful inference engine for this type of knowledge representation language.
Another example will illustrate constraints involving boolean operators and how this is used to define a subclass of a class. We can define a right-angle triangle with the following simple COOL code:

```
[Class Name]
Right-angle triangle
[Parent Class]
Triangle
[Constraints]
Angle1=90 || Angle2=90 || Angle3=90
```

The right-angle triangle class inherits all properties and methods from the triangle class by declaring triangle as its superclass. In a right-angle triangle, one of the internal angles must be 90 degrees. This constraint can be specified using the boolean operator OR (symbol ||).

Visualization Methods of COOL

The language features of COOL introduced so far are declarative. This means the attributes, their ranges, and constraints can be specified in any order and would not affect the results produced by the system. However, the visualization methods of a class must be procedural rather than declarative. These methods specify the procedures of how to draw the objects of this class. For COOL, these methods are specified as CLogo procedures, which can be executed in a CLogo environment to draw the geometric figure objects. CLogo is chosen as the visualization language because it is simple, high-level and is a fully Chinese programming language. Also, COOL can work with CLogo for teaching children how to program in CLogo and explore elementary geometry concepts.
Consider a triangle given the length of its two sides and their inclusive angle. In secondary school math classes, students would know how to draw such a triangle. One would draw one side first, then using a protector to get the inclusive angle, draw another side, and finally draw the third side. This procedure is commonly known as the SAS (short for Side-Angle-Side) method. Similarly, in COOL, this procedure can be specified in the following pseudo CLogo code:

```
To SAS :s1 :a :s2
   Make "p3 position        //Remember the starting point
                            of the first side
                            // as the current position of the
                            turtle
   Forward length of :s1    // Draw line segment :s1 as
                            the first side of the triangle
   Right (180-:a)           // Prepare to form the
                            inclusive angle :a
   Forward length of :s2    // Draw line segment :s2 as
                            the second side
   MoveTo :p3               // Draw the third side of the
                            triangle
End
```

Another method, called PPP (short for Point-Point-Point), to draw a triangle is one using the coordinates of the three vertices of the triangle. In CLogo code, PPP can be expressed as:

```
To PPP :p1 :p2 :p3
   PenUp                  // Turtle's future path becomes
                          invisible from this time
   MoveTo :p1             // Turtle moves to the first
                          vertex of the triangle
   PenDown                // Turtle's future path becomes
                          visible from this time
   MoveTo :p2             // Turtle moves to the second
                          vertex and draws the first side
   MoveTo :p3             // Draw the second side
   MoveTo :p1             // Draw the third side of the
                          triangle
End
```

Since there are usually several visualization methods, the system, when asking to draw an object of a geometric class, must pick one from the available methods. Each visualization method must specify its own preconditions. These preconditions specify which class attributes are needed as the method parameters and specify the relations among the attributes. These preconditions must be satisfied before the method can be adopted for drawing the object. For example, the preconditions of the visualization methods of a triangle object are specified as:

```
[Visualization Method Selection Preconditions]
SAS :s1 :a :s2
{   :s1 = line segment formed by point1 and point2 of
        angle :a
    :s2 = line segment formed by point2 and point3 of
        angle :a
}
```

PPP :point1 :point2 :point3
// :point1, :point2, :point3 must be point objects
{
}

For the visualization method SAS, the preconditions only check whether the given line segments :s1 and :s2 are the line segments forming the angle :a. The visualization method PPP requires that the coordinates of the vertices must be given.

Interpreting COOL Script

The previous section briefly describes the knowledge representation scheme of COOL. This section will explain how a script can be written to use the class knowledge defined in COOL. For example, the following script describes a triangle with two given sides and their inclusive angle:

Triangle ABC	*// declare ABC to be a triangle*
Line-segment AB.length = 100	*// Side AB is 100 pixels long*
Line-segment BC.length = 140	*// Side BC is 140 pixels long*
Angle ABC.measure = 40	*// Angle ABC is 40 degrees*
Triangle ABC.Draw	*// Draw triangle ABC*

Using the SAS visualization method, COOL interprets this script as the following CLogo source code:

Make "v3 position
Forward 100
Right (180-40)
Forward 140
MoveTo :v3

The previous example illustrates how a triangle can be drawn when sufficient conditions are given. The next sample script illustrates the case when insufficient conditions are given:

Triangle ABC
Angle ABC.measure = 60
Angle BCA.measure = 30
Triangle ABC.draw

When COOL tries to interpret this script, it will check the given information against the constraints defined in the triangle class. One of the constraints is:
Angle1.measure + Angle2.measure < 180

Suppose Angle1 is bound to Angle ABC and Angle2 is bound to Angle BCA. Then this constraint will be instantiated as:
Angle ABC.measure + Angle BCA.measure < 180

Obviously this constraint is satisfied. Consider another constraint in the triangle class:
Angle1.measure + Angle2.measure + Angle3.measure = 180

When COOL considers this constraint during the interpretation process for the above script, the constraint becomes:
60 + 30 + Angle CAB.measure = 180

Since there is only one unknown variable in the equation, COOL immediately derives the value of Angle CAB.measure as 180-100-20=90. Thus, the triangle object ABC has an inferred attribute (Angle CAB.measure) added to the already known information of triangle ABC. From this point on, COOL finds that no new information can be derived further so it checks the preconditions of all visualization methods in order to pick one method to draw the triangle. Unfortunately, it finds no visualization method whose preconditions can be satisfied. Instead, it finds a visualization method that requires three parameters, two of which are known information for triangle ABC. This method is the ASA method, short for Angle-Side-Angle. This method requires two angles and their common side to be known information for the method to be executed. For triangle ABC, all angles are known but no side is known. Hence COOL picks two known angles ABC and BCA and tries to generate the length of their common side BC. Since the length of any side of a triangle is limited to be (0,400]---400 pixels is the default length of a CLogo drawing window, COOL will generate an integer larger than 0 and less than or equal to 400, say 50, and assign it as the length of side BC. At this point, all preconditions for using the ASA method are satisfied and the corresponding CLogo code will be produced for drawing triangle ABC (Figure 2).

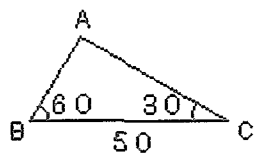


Figure 2. Triangle ABC drawn with the ASA method

Conclusion and Discussion

A potential application of COOL is to understand the natural language description of geometric figure input by students of elementary geometry and/or CLogo programming. We are developing another system to interpret texts describing geometric figures as COOL scripts, which can then be interpreted as CLogo code that can be executed to draw the figure. In a previous project, we achieve some success when interpreting simple sentences to be CLogo code without the use of COOL [10]. After that project, we believe it is natural and efficient for COOL to be an intermediate representation between text and CLogo.
At this stage, COOL is not as "cool" as we hope. A simple script such as the following is not interpreted properly by the current version of COOL

Triangle ABC
Triangle BCD
Triangle ABC.draw
Triangle BCD.draw

This script just describes two triangles sharing the same side BC. However, the current COOL system does not remember triangle ABC after its CLogo visualization code is generated. Hence, when COOL considers the triangle BCD, it has no idea that side BC is already drawn. In other words, side BC of triangle ABC and side BC of BCD will be generated as two different line segments. This of course is not correct and we are taking steps to improve COOL in this direction.
Another limitation of COOL is that it is not designed as a full-blown programming language such as C++. At this stage, COOL does not have methods other than those for

visualization and does not have any control constructs such as if-else, while-loop, etc. These are not needed for describing static geometric objects whose attributes will not change. In the future, COOL might provide these more advanced programming language features when applications concerning dynamic geometric objects are considered.

We can make the following short conclusion about COOL at this point. COOL is a Chinese object-oriented language for knowledge representation. It is used for describing knowledge of geometric objects. Its main features include inheritance, constraint checking, algebraic inference of new information, random generation of attribute values needed for visualization when insufficient information is given, and the specification of visualization methods using CLogo procedures. COOL will be used in a computational environment for students to learn CLogo programming and elementary geometry. Potential applications of COOL include natural language understanding of Chinese texts describing geometric figures, and computational environments for learning in domains involving geometry concepts or algebra concepts.

Acknowledgement

This research is supported by the National Science Council under contract NSC88-2511-S-224-002.

References

[1] Abelson, H. & Abelson, A. Logo for the Macintosh. Paradigm Software, Cambridge, MA. 1992.

[2] Friendly, M. Advanced LOGO: A language for learning. Lawrence Erlbaum, Hillsdale, NJ. 1988.

[3] Howell, R. D., Scott, P. B., and Diamond, J. The Effects of "Instant" Logo Computing Language on the Cognitive Development of Very Young Children. Educational Computing Research, Vol 3(2), pp.249-260. 1987.

[4] Lawler, R. W., Boulay, B. D., Hughes, M., Macleod, H.. Cognition and Computers: Studies in learning. Ellis Horwood, Chicester, England. 1986

[5] Luger, G. F. and Stubblefield, W. A.. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Benjamin/Cummings, Redwood City, CA. 1993

[6] Mylopoulos, J. and Levesque, H. J. An overview of knowledge representation languages. In Brodie, M. L., Mylopoulos, J. and Schmidt, J. W. On Conceptual Modelling. Springer-Verlag, New York. 1984.

[7] Papert, S.. Mindstorms. New York, Basic Books. 1980

[8] Simon, T.. Claims for LOGO---What should we believe and why? In J. Rutkowska and C. Crook. (Eds). Computers, Cognition and Development. John Wiley, pp.115-133. 1987

[9] Wong, W. K.. An environment for learning Chinese Logo programming. In Proceedings of Global Chinese Conference on Computer in Education, Hong Kong, pp.404-409. 1998

[10] Wong, W. K., Chan, T. W., Pai, S. T., Wang, Y. K., Chen, Y. S., Hsu, W. L.. Natural Language Educational Agents in a Networked Chinese Logo Learning Environment, Proceedings of ICCE, Vol. 1, AACE, Beijing, pp.220-227. 1998

[11] Hoyles, C. and Noss, R. (Eds). Learning Mathematics and Logo. MIT Press, Boston. 1992.

Appendix A: Definition of the Triangle Class in COOL

```
[類別名稱]
三角形
[父類別]
幾何圖形
[屬性]
線段 1 = 線段(點 2,點 3) = 線段(點 3,點 2)
線段 2 = 線段(點 3,點 1) = 線段(點 1,點 3)
線段 3 = 線段(點 1,點 2) = 線段(點 2,點 1)
角 1 = 角(點 1,點 2,點 3) = 角(點 3,點 2,點 1)
角 2 = 角(點 2,點 3,點 1) = 角(點 1,點 3,點 2)
角 3 = 角(點 3,點 1,點 2) = 角(點 2,點 1,點 3)
點 1 = 點(1)
點 2 = 點(2)
點 3 = 點(3)
[隨機範圍]
角.角度:(0~180)
線段.長度:(0~400]
[限制條件]
角 1.角度+角 2.角度<180
角 2.角度+角 3.角度<180
角 3.角度+角 1.角度<180
角 1.角度+角 2.角度+角 3.角度=180
線段 1.長度+線段 2.長度>線段 3.長度
線段 2.長度+線段 3.長度>線段 1.長度
線段 3.長度+線段 1.長度>線段 2.長度
[繪圖條件]
@ 線角線 線段 1 線段 2 角 3
{     線段 1 = 線段(角 3.點 1,角 3.點 2)
      線段 2 = 線段(角 3.點 2,角 3.點 3)
}
@ 角線角 角 1 角 2 線段 3
{     線段 3 = 線段(角 1.點 2,角 2.點 2)
}
@ 點點點 點 1 點 2 點 3
{
}
[繪圖方法]
開始 線角線 {線段 1} {角 3} {線段 2}
         右轉 90
         記錄 "{角 3.點 1} 位置
         {線段 1}
         記錄 "位置 2 位置
         {角 3}
         {線段 2}
         記錄 "位置 3 位置
         移至 :位置 1
         寫文字 "{角 3.點 1} :{角 3.點 1}
         寫文字 "{角 3.點 2} :位置 2
         寫文字 "{角 3.點 3} :位置 3
結束
```

```
開始 角線角 {角 1} {線段 3} {角 2}
        記錄 "標記 []
        記錄 "角 3 0
        定方向 90
        記錄 "標記 置首 位置 :標記
        {線段 3}
        記錄 "標記 置首 位置 :標記
        {角 2}尋找 20 {角 1.角度} {角 2.角度}
        移至座標 首 尾 :標記 尾 尾 :標記
        定方向 90
結束
開始 點點點 {點 1} {點 2} {點 3}
        提筆
        {點 2}
        記錄 "點 2 位置
        {點 3}
        記錄 "點 3 位置
        {點 1}
        記錄 "點 1 位置
        落筆
        移至 :點 2
        寫文字 "2 位置
        移至 :點 3
        寫文字 "3 位置
        移至 :點 1
結束
[輔助程式]
角線角　角線角輔助程式.lgo
[結束]
```

Appendix B: File “角線角輔助程式.lgo”

```
起始 距離 :橫座標 2 :縱座標 2 :橫座標 1 :縱座標 1
    傳結果 (平方根 ((次方 (:橫座標 2 - :橫座標
      1) 2)+(次方 (:縱座標 2 - :縱座標 1) 2)))
終止

起始 頂角
     記錄 "暫時 :標記
     記錄 "高 距離 (首 首 :暫時) (尾 首 :暫時)
       (首 首 :暫時) (尾 尾 :暫時)
     記錄 "部份底 1 距離 (首 首 :暫時) (尾 尾 :
       暫時) (首 尾 :暫時) (尾 尾 :暫時)
     記錄 "部份底 2 距離 (首 尾 去尾 :暫時) (尾
       尾 去尾 :暫時) (首 首 :暫時) (尾 尾 :暫
       時)
     記錄 "部份角 1 (90 - 反正切(:高 / :部份底 1))
     假如否則 :部份底 2 = 0
         [ 記錄 "部份角 2 0]
         [ 記錄 "部份角 2 (90 - 反正切(:高 / :部
           份底 2))]
     記錄 "角 3 (:部份角 1 + :部份角 2)
     傳結果 :角 3
終止

起始 尋找 :邊長 2 :角 1 :角 2
     提筆
     假如 :邊長 2 < 1
         [ 記錄 "標記 置首 位置 :標記
     落筆
     移至座標 首 尾 去尾 :標記 尾 尾 去尾 :標記
     移至座標 首 首 :標記 尾 首 :標記
     停止]
     記錄 "標記 置首 位置 :標記
     記錄 "角 3 頂角
     假如否則 :角 3 > (180 - :角 1 - :角 2)
         [前進 :邊長 2
          尋找 :邊長 2 :角 1 :角 2]
         [後退 :邊長 2
          尋找 (:邊長 2 / 2) :角 1 :角 2]
終止
```

Appendix C: A Sample COOL script

```
三角形 ABC
角 BCA.角度 = 30
線段 BC.長度 = 50
角 ABC.角度 = 60
三角形 ABC.畫
```