TR-86-001

VIPS: A Visual Programming Synthesizer

K.Y.Cheng, C.C.Hsu, M.C.Lu, M.S.Hwu, and I.P.Lin

Institute of Information Science
Academia Sinica
and
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, ROC

ABSTRACT

In this paper, we propose a Visual Programming Synthesizer as a vehicle for nonprogrammers to develop their application programs. Here, the visual forms (V-Forms) dealt with contain text (in particular Chinese characters), graphs, bit-maps, animations, rules, and voices. Based on the V-Form model, a two-dimensional non-procedural interfacing language, consisting of a V-Form Definition Language (VDL) and a V-Form Manipulation Language (VML), is defined. VDL is used to define the logical and visual structure of V-Forms, while VML manipulates the contents of V-Form instances. After V-Form instances have been interactively sketched, a consistent internal structure and control flow for each application can be synthesized. V-Form instances that are text-only can be directly executed by VIPS interpreter, while V-Form instances other than text are treated as a series of procedure calls delivered to graphical and voice systems for execution. A complete CAI example is illustrated to demonstrate the features of the synthesizer.

1. INTRODUCTION

The importance for the study of Visual Language has been recognized in the United States and Japan in recent years [7,12,15,16,17]. The major issues of the study are to provide a programming environment for NP-professionals (Non-Programmers) to develop their own application programs visually. This kind of applications can be seen widely in the design of Office Information Systems [3,9,10,14,18] and others such as Computer Aided Instructions [1,8,13]. In these applications, the visual language is used to describe vision data -- a news, a menu, a screen layout, an engineering drawing, a typeset report, a font of type, and their abstract data type such as hierarchy, condition statements, and rule-based knowledge. In visual programming, they are represented as data objects in the shape of forms because people are familiar with forms.

In this paper, we propose a **VIsual Programming Synthesizer** (VIPS) as a vehicle for nonprogrammers to develop their own application programs. Here, the forms dealt with contain text (in particular, Chinese characters), static graphics (line drawing and bit-map), dynamic graphics (animation), rules, and voices. In order to distinguish them from office forms, we call them the V-Forms.

Just like other form models [15,16,17,18], our V-Form model consists of a V-Form type and a V-Form instance. A V-Form type describes the structure of a V-Form system, while a V-Form instance is obtained after filling values into the contents of the described V-Form type. The V-Form instance is a program that fits the concept of embedded procedures for computational and controlling requirements. Hence, facilities for conditional and unconditional control switches are also included in the V-Form model.

Based on the V-Form model, VIPS contains a two-dimensional non-procedural language as a user-friendly interfacing language to define and manipulate V-Forms on the screen. This interfacing language includes two parts: a **V-Form Definition Language** (VDL) and a **V-Form Manipulation Language** (VML). VDL is used to define the V-Form type (logical and visual structure of a V-Form) and V-Form instance. Once V-Form instances are obtained, VML can be invoked to display, insert, delete, update, group, and degroup these V-Form instances.

After processing VDL and VML, a complete internal form for each application can be synthesized. V-Form instances that are text-only can be directly executed by VIPS interpreter, while V-Form instances other than text are treated as a series of procedure calls delivered to graphic and voice systems for execution. In VIPS, rules are represented in predicate logic and properly arranged in a V-Form text that can easily interface to another system developed in the AI Laboratory of the Department of Computer Science in National Taiwan University. A rule in a V-Form text is translated into a Horn Clause and then delivered to a PROLOG interpreter [2,5,11] for execution.

In the following, we describe the language environment of VIPS in section 2, the V-Form model in section 3, the structure and syntax of V-Form Definition Language in section 4, and the V-Form Manipulation Language in section 5.

2. THE VISUAL LANGUAGE ENVIRONMENT

A conventional programming environment concerns how to provide programmers some convenient and useful tools to develop their application programs. In this environment, programming itself is a specialized arduous task requiring detailed textual instructions that must adhere to strict syntactical rules. As for non-programmers, using programming language to develop their own application programs is almost infeasible (if not impossible). On the other hand, a visual programming synthesizer concerns how to provide some kinds of tool for users who only need to use their specialized knowledge to describe the process of how to get the desired applications in a manner of **what-you-sketch-is-what-you-get.**

Then, what is the visual language? Let **D** be the domain of what you sketch and y_i be a sketched object, $y_i \in$ **D**. Suppose y_i can be generated by an inferred grammar G_i [4], then we can define the visual language as:

$$L_V = \{ y_i \mid y_i \in \mathbf{D} \text{, and } y_i \in L^+(G_i) \}$$

where $L(G_i) = L^+(G_i) \cup L^-(G_i) = \{ \text{positive\_sample} \} \cup \{ \text{negative\_sample} \}$. The positive sample is an information sequence of $L(G_i)$ containing only codes from $L^+(G_i)$ which is a set of objects described by the user. On the other hand, the negative sample is the parasitical product of G_i and this is not included in the visual language. So, visual language L_V in nature is to get the desired applications from an inferred grammar G_i through interactive sketching.

There are some design methods for visual programming languages[7,12,15,16,17]. However, systematic approach to a theoretical sound methodology is still under developing. VIPS is interactive and application-oriented. It has the following features : (1) easy to use -- Various user friendly facilities such as icons, pointing devices, and menus are included. (2) Visual directed objects -- V-Forms are used as the fundamental objects of VIPS because they are more akin to the user's view point. (3) Non-conventional programming nature -- Users only need to describe the external representation of objects instead of writing a series of instruction codes. (4) For non-programmers -- Since application programs are generated automatically by VIPS, users can concentrate in expressing their knowledge to obtain a better presentation. (5) Portable -- V-Form system generated by VIPS is always consistent in its internal structure. To transport VIPS from one system to another is by means of porting an interpreter which interpretes the internal structure and contents of V-Forms.

3. THE V-FORM MODEL

VIPS allows users to open several windows on a screen. Each window is treated as a V-Form. The informant presentation data in a V-Form may include text, static graphics, dynamic graphics, rules, and voices. They are represented in **V-Forms.**

Let Π be a set of codes to be displayed, Δ be a set of alphanumerics, Ç be a set of Chinese characters, ß be a set of graphic codes, Ω be a set of drawing attributes, and $\Sigma = \Pi \cup \Delta \cup$ Ç $\cup$ ß $\cup \Omega$. Then, the content of a V-Form is a **regular expression** [6] over Σ.

A V-Form is a pair of a V-Form type **F** and a V-Form instance **I** which are defined below. A **V-Form type** consists of a scheme S and a template T_S for S.

[Definition 1] A **scheme** S is the **logical structure** of a V-Form. It can be recursively defined as:

```
<S>          ::= <M> | <A>
<A>          ::= <S> | <S>, <A>
<M>          ::= <Type>_<Identifier>
<Type>       ::= Text | Graph | Bit-map | Rule
                  Animation | Voice | VForm
```

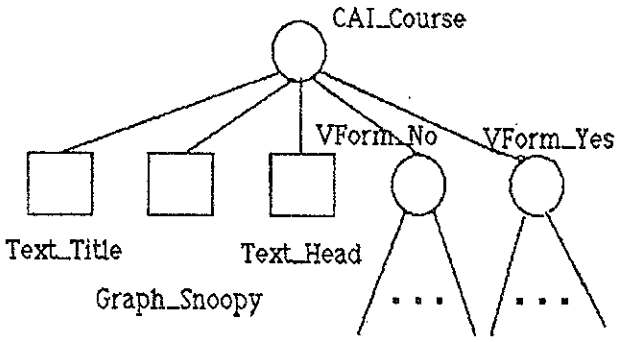


Fig. 3.2 Hierarchical structure for CAI_Course

[Definition 2] A **template** T_S is a **visual structure** of a V-Form that represents a two-dimensional display format and visual properties of a scheme S.

The template for the scheme of Fig. 3.1 is given in Fig. 3.3.

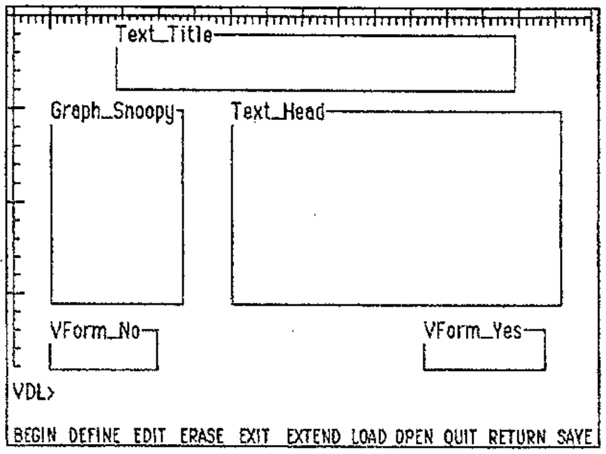

Text_Title
Graph_Snoopy
Text_Head
VForm_No
VForm_Yes
VDL>
BEGIN DEFINE EDIT ERASE EXIT EXTEND LOAD OPEN QUIT RETURN SAVE

Fig. 3.3 A template for the CAI_Course

A V-Form F_j is said to be a **sub-VForm** of F_i if and only if F_i contains F_j. If F_j is a sub-VForm of F_i and is not equal to F_j, then F_j is said to be a **proper sub-VForm** of F_i.

Sub-VForms are also V-Froms. They may contain any number of other V-Forms. The proper sub-VForms surrounded only by the V-Form F_i are called the **maximum proper sub-VForms** of F_i. Two V-Forms F_j, F_k, which are the maximum proper sub-VForms of F_i, are called **brothers**.

In Fig. 3.3, the maximum sub-VForms of the V-Form: CAI_Course are all the V-Forms shown and they are all brothers, where Text_Title, Graph_Snoopy, and Text_Head are atoms, but VForm_No and VForm_Yes are not.

[Definition 3] A **V-Form instance** for a V-Form type F is defined as a mapping which assigns a value to each atom of F. The value of a V-Form may have the following types: text, graph, bit-map, animation, and rule.

One of the V-Form instance of Fig. 3.3 is shown in Fig. 3.1.

As stated above, a V-Form type can be filled with different contents to obtain different kinds of V-Form instances. Therefore, a V-Form type is just like a **language**, from which a user can write many **programs**, which, in this model, are V-Form instances.

4. THE V-FORM DEFINITION LANGUAGE

As mentioned in section 3, a V-Form consists of a V-Form type and a V-Form instance. Hence, V-Form Definition Language (VDL) is divided into two phases, namely, a **skeleton phase** and an **editing phase** to define V-Form type and V-Form instance respectively.

4.1 Skeleton Phase

Once a user has prepared his/her own well-written sheets in V-Forms, he/she can use skeleton phase to define the V-Form type. In this phase, a user can directly manipulate the screen visually. The basic commands provided in skeleton phase are summarized below (Table 1).

Table 1. Basic commands in VDL Skeleton Phase

```
<VDL>              ::= <BEGIN> [<COMMANDS>] <END>
```

```
<COMMANDS>     ::= <COMMAND> [<COMMANDS>]
<COMMAND>      ::= <DEFINE> | <EDIT> | <ERASE> | <EXTEND>
                 | <LOAD> | <OPEN> | <RETURN> | <SAVE>
<BEGIN>        ::= BEGIN <IDENTIFIER>
<DEFINE>       ::= DEFINE <IDENTIFIER> AS <TYPES>
<TYPES>        ::= TEXT | GRAPH | BIT-MAP | ANIMATION
                 | RULE | VOICE | VFORM
<EDIT>         ::= EDIT <IDENTIFIER>
<ERASE>        ::= ERASE <IDENTIFIER>
<EXTEND>       ::= EXTEND <IDENTIFIER>
<LOAD>         ::= LOAD <FILENAME>
<OPEN>         ::= OPEN <TYPES> AT <POSITION> WITH
                   <SIZE> [<OPTIONS>]
<POSITION>     ::= <INTEGER> <INTEGER>
<SIZE>         ::= <INTEGER> <INTEGER>
<OPTIONS>      ::= AS <IDENTIFIER>
<RETURN>       ::= RETURN
<SAVE>         ::= SAVE <FILENAME>
<END>          ::= EXIT | QUIT
```

For convenience, all interactive sequences are illustrated in line-mode commands. But actually in VIPS, a mouse is used as a menu-driven device.

For the V-Form type given in Fig. 3.3, the interactive sequence is as follows:

```
VDL> BEGIN CAI_Course
VDL> OPEN TEXT AT 14 1 WITH 55 3 AS Title
VDL> OPEN GRAPH AT 5 5 WITH 18 11 AS Snoopy
VDL> OPEN TEXT AT 30 5 WITH 45 11 AS Head
VDL> OPEN VFORM AT 5 17 WITH 14 2 AS No
VDL> OPEN VFORM AT 57 17 WITH 14 2 AS Yes
VDL>
```

The EXTEND command can be used to stretch the V-Form node to define V-Forms inside a

V-Form. Thus, the command

```
VDL> EXTEND Yes
```

will clear up the screen, then give out a command menu and a prompt. After giving a series of OPEN commands similar to the above interactive sequence, the screen of Fig. 4.1 is obtained.

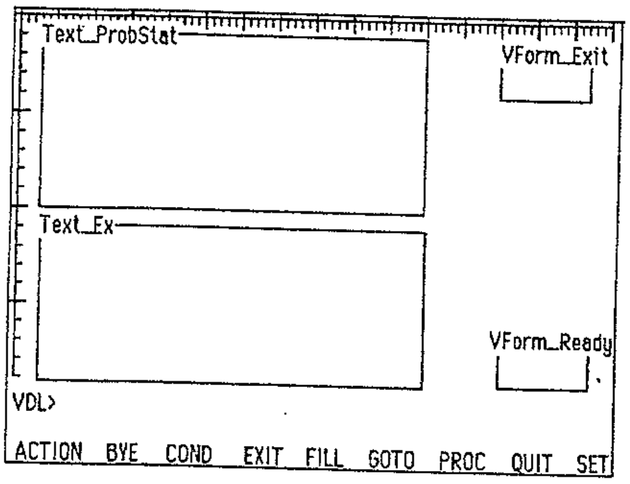


Fig. 4.1 After using EXTEND and a series of OPEN commands
on VForm_Yes of Fig. 3.3

RETURN command can be used to set back to the previous V-Form, for example the selection of

```
VDL> RETURN
```

in Fig. 4.1 will display Fig. 3.3 again.

ERASE command is used to erase the V-Forms on the screen. DEFINE command can be used to change the type of a V-Form. SAVE command can save previous V-Form type just constructed. LOAD command reloads a prewritten V-Form type.

4.2 Editing Phase

The editing phase is to fill in each V-Form instance. The basic commands in editing phase are given in Table 2.

Table 2. Basic commands in VDL Editing Phase

```
<EDITOR>          ::= <EDIT> [<ED_COMMANDS>] <ED_EXIT>
<ED_COMMANDS>     ::= <ED_COMMAND> [<ED_COMMANDS>]
<ED_COMMAND>      ::= <ACTION> | <COND> | <FILL> | <GOTO>
                      | <PROC> | <SET>
<ACTION>          ::= ACTION [{IS | ARE}] <Statements>
<COND>            ::= COND [{IS | ARE}] <Statements>
<FILL>            ::= FILL <CONTENTS> ^Z
<GOTO>            ::= GOTO <IDENTIFIER>
<PROC>            ::= PROC [{IS | ARE}] <Statements>
<SET>             ::= SET <IDENTIFIER> <ATTRIBUTE>
<ATTRIBUTE>       ::= BOLD | FLASH | NORMAL | REVERSE
                      | MIXED
<ED_EXIT>         ::= BYE | EXIT | QUIT
```

FILL command is used to fill values into each atom. Let us consider the template shown in Fig. 3.3. The following interactive sequence of commands:

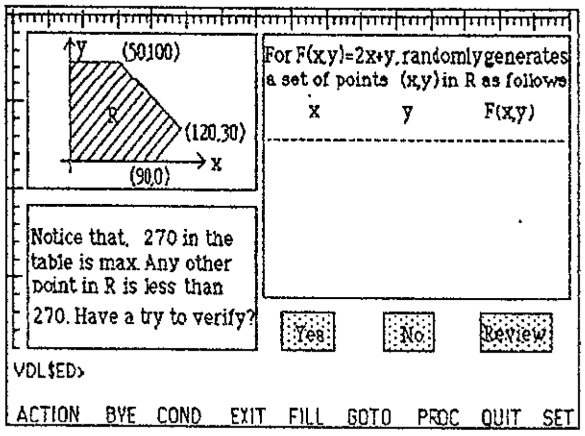
```
VDL> EDIT Title
VDL$ED> FILL  /* cursor is now at the beginning of VForm Text_Title,
                 user can now keyin text 'Linear Programming' */
VDL$ED> SET Title BOLD
VDL$ED> EXIT
VDL> EDIT Snoopy
VDL$ED> FILL
      ...
VDL$ED> EXIT
VDL> EDIT Head
```

```
VDL$ED>
```

will generate a V-Form as shown in Fig.3.1.

If an atom has procedures embedded as in Fig. 4.2, then while in editing the upper-right corner, we can specify the procedures as follows:

```
VDL$ED> PROC        /* Prompt Procedure: ,user can keyin statements as follows */
Procedure: Print x  y  z , VARY x FROM 0 TO 120 BY 20 , y is RANDOM , z is 2x+y.
```



For F(x,y)=2x+y, randomly generates a set of points (x,y) in R as follows

```
x          y          F(x,y)
--------------------------------
```

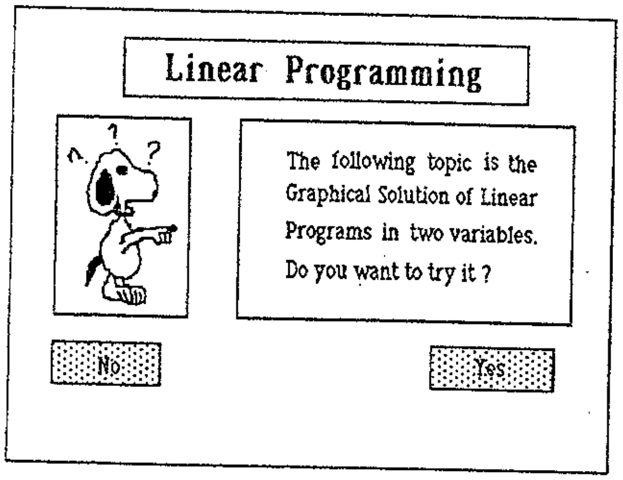
Notice that, 270 in the table is max. Any other point in R is less than 270. Have a try to verify?

```
VDL$ED>
```

Fig. 4.2 Procedures embedded example

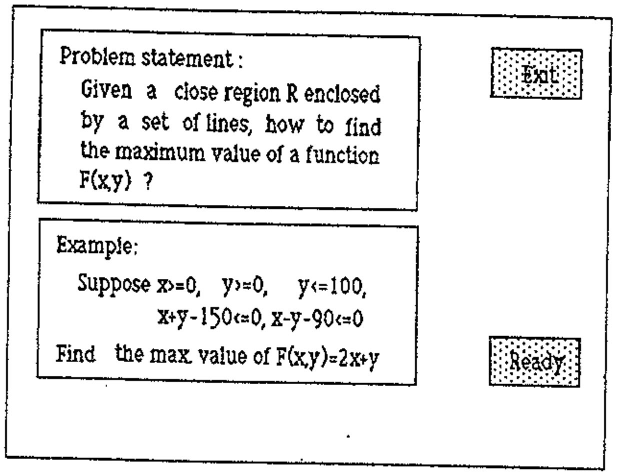
COND command is used to specify the conditions of performing actions/procedures. ACTION command is used to specify actions to be performed. Unconditionly transfer between V-Forms is given by GOTO command.

The resulting CAI_Course V-Form instance is shown in Fig. 4.3. Then, this V-Form instances embedded with procedures/actions can be executed by a VIPS interpreter according to the synthesized control flow of Fig. 4.4.



The following topic is the Graphical Solution of Linear Programs in two variables. Do you want to try it ?

(a) F1

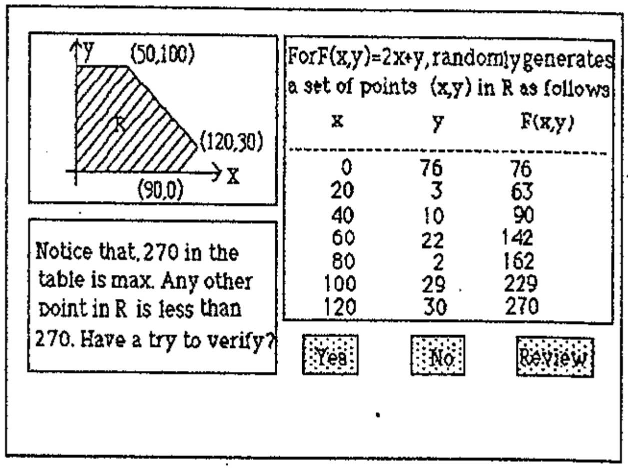


Problem statement:
Given a close region R enclosed by a set of lines, how to find the maximum value of a function F(x,y) ?

Example:
Suppose x>=0, y>=0, y<=100, x+y-150<=0, x-y-90<=0
Find the max. value of F(x,y)=2x+y

(b) F2

Fig. 4.3 V-Form instance of template CAI_Course

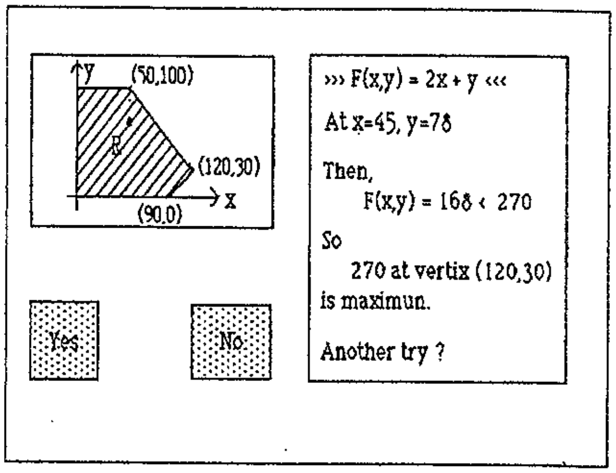


ForF(x,y)=2x+y, randomlygenerates a set of points (x,y) in R as follows

x	y	F(x,y)
0	76	76
20	3	63
40	10	90
60	22	142
80	2	162
100	29	229
120	30	270

Notice that 270 in the table is max. Any other point in R is less than 270. Have a try to verify?

(c) F3



››› F(x,y) = 2x + y ‹‹‹

At x=45, y=78

Then,

F(x,y) = 168 < 270

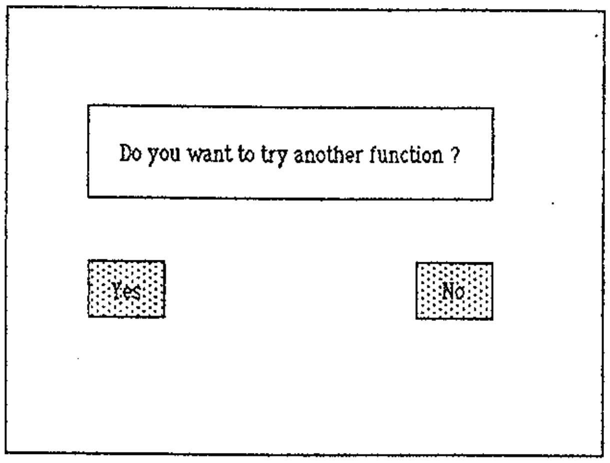
So

270 at vertix (120,30) is maximun.

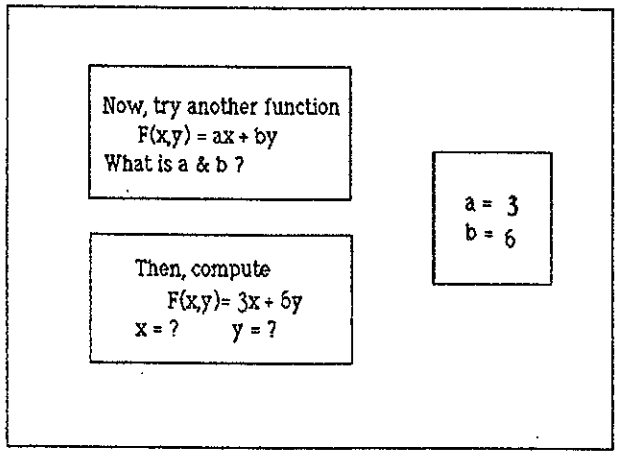
Another try ?

(d) F4

Fig. 4.3 V-Form instance of template CAI_Course (Continued)

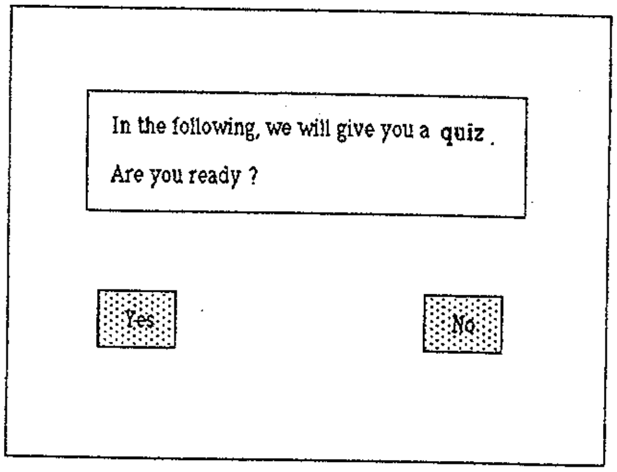


(e) F5



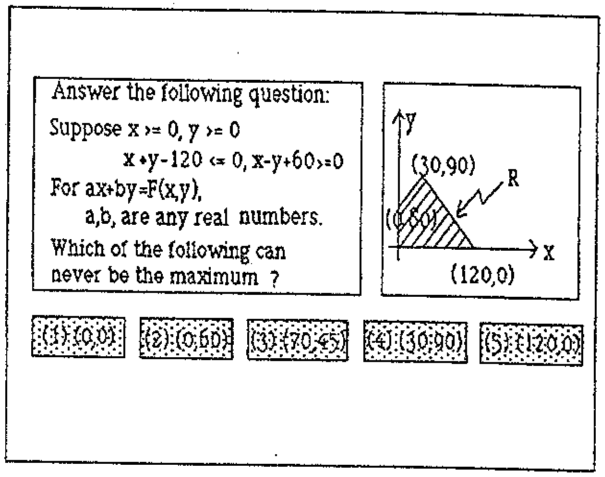
(f) F6

Fig. 4.3 V-Form instance of template CAI_Course (Continued)


In the following, we will give you a **quiz**.

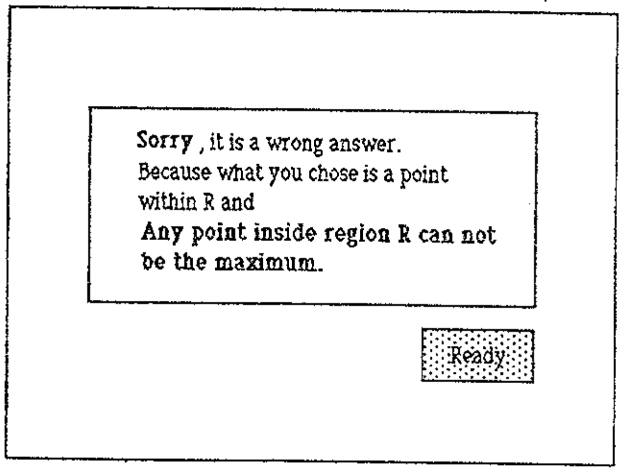
Are you ready ?

(g) F7

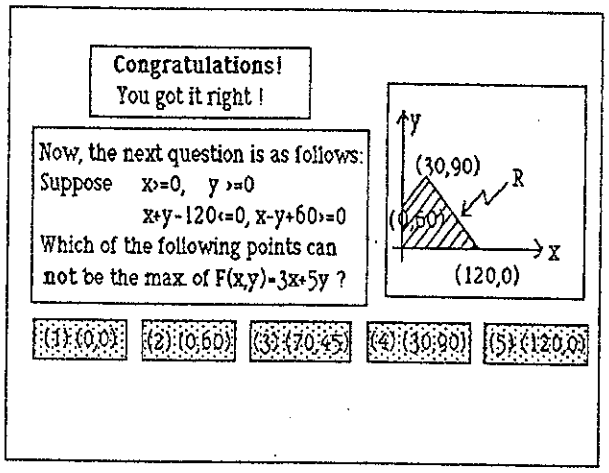

Answer the following question:
Suppose x >= 0, y >= 0
x+y-120 <= 0, x-y+60>=0
For ax+by=F(x,y),
a,b, are any real numbers.
Which of the following can never be the maximum ?

(h) F8

Fig. 4.3 V-Form instance of template CAI_Course (Continued)


Sorry , it is a wrong answer.
Because what you chose is a point
within R and
**Any point inside region R can not
be the maximum.**

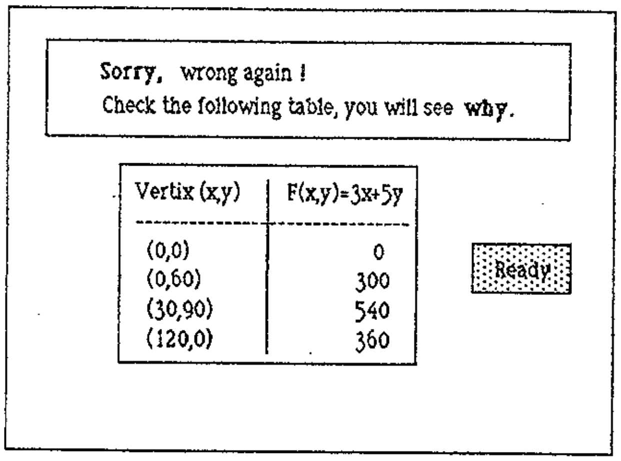
(i) F9


Congratulations!
You got it right !

Now, the next question is as follows:
Suppose x>=0, y >=0
x+y-120<=0, x-y+60>=0
Which of the following points can
not be the max. of F(x,y)=3x+5y ?

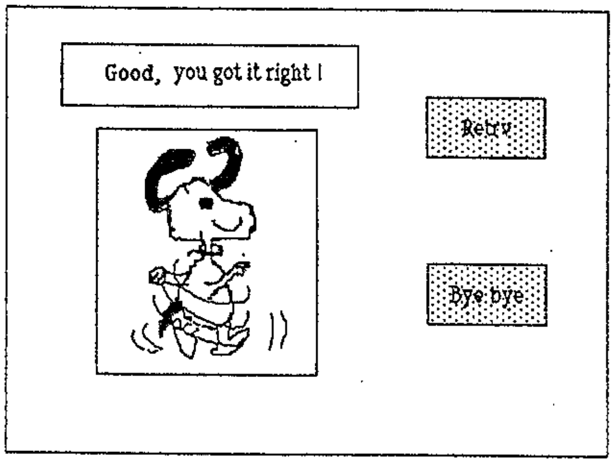
(j) F10

Fig. 4.3 V-Form instance of template CAI_Course (Continued)


Sorry, wrong again !
Check the following table, you will see **why**.

Vertix (x,y)	F(x,y)=3x+5y
(0,0)	0
(0,60)	300
(30,90)	540
(120,0)	360

(k) F11


Good, you got it right !

(l) F12

Fig. 4.3 V-Form instance of template CAI_Course (Continued)

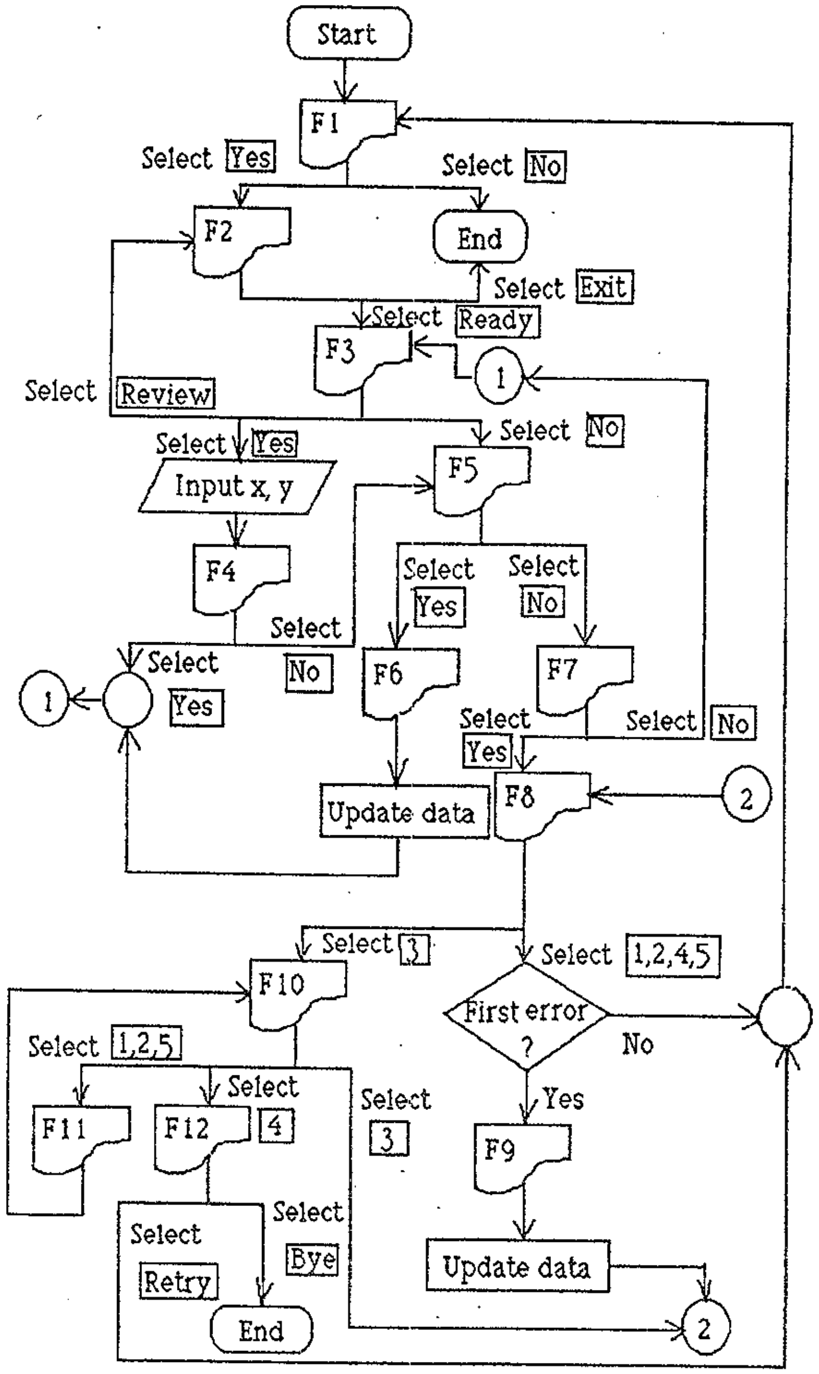


Fig. 4.4 Control flow of CAI_Course

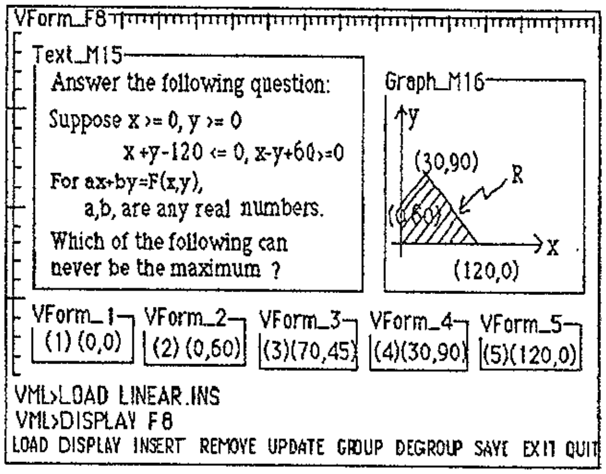
5. THE V-FORM MANIPULATION LANGUAGE

VML is a language to manipulate V-Form instances gotten from VDL. As described earlier, a V-Form instance can be regarded as a user-defined program. Thus, VML, similar to a program editor, can manipulate V-Form instances at will. It provides users the following opeartions: (1)Displaying, (2)Inserting, (3)Removing, (4)Modifying, (5)Grouping, (6)Degrouping. Like other form manipulators[14,15,16,18,19,20], the embedded components (conditions, actions and procedures) are manipulated by the operations(1), (2), (3) and (4), while Grouping and Degrouping allow users to restructure the existing V-Form instances. The basic commands of VML are shown below.

Table 3. Basic commands of VML

```
<VML>         ::= <BEGIN> <COMMANDS> <END>
<COMMAND>     ::= <LOAD> | <DISPLAY> | <INSERT> | <REMOVE>
                  | <UPDATE> | <GROUP> | <DEGROUP> | <SAVE>
<EMBEDDED>    ::= COND | ACTION | PROC
<INSERT>      ::= INSERT <TYPE1> <IDENTIFIER> <DESTINATION>
                    AT <POSITION> WITH <SIZE>
                  | INSERT <EMBEDDED> <DESTINATION>
<TYPE1>       ::= TEXT | GRAPH | BIT-MAP | ANIMATION | VOICE
                  | RULE
<REMOVE>      ::= REMOVE <IDENTIFIER> FROM <IDENTIFIER>
                  | REMOVE <EMBEDDED> <NUMBERS>
                   FROM   <IDENTIFIER>
<UPDATE>      ::= UPDATE <IDENTIFIER>
                  | UPDATE <IDENTIFIER> <EMBEDDED> <NUMBERS>
<GROUP>       ::= GROUP <IDENTIFIER> IN <IDENTIFIER>
                    AT <POSITION>  WITH <SIZE> <IDENTIFIER>
<DEGROUP>     ::= DEGROUP <IDENTIFIER> FROM <IDENTIFIER>
                    TO <FILENAME>
<SAVE>        ::= SAVE <FILENAME>
```

Fig.5.1 shows an example of displaying operation. In Fig.5.1(a), LOAD command loads the entire CAI_Course V-Form instance into the memory and displays the root V-Form on the screen, then DISPLAY command displays V-Form (F8) on screen as shown. After keyin another DISPLAY command to display the condition part of VForm_2, the resultant screen is shown in Fig.5.1(b). The usage of other operations (Inserting, Removing and Modifying) is similar and need no further explanations.

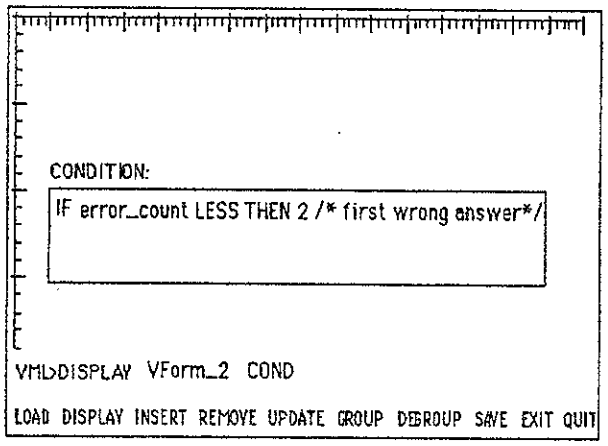


Answer the following question:
Suppose x >= 0, y >= 0
x+y-120 <= 0, x-y+60>=0
For ax+by=F(x,y),
a,b, are any real numbers.
Which of the following can
never be the maximum ?

(1) (0,0) | (2) (0,60) | (3)(70,45) | (4)(30,90) | (5)(120,0)

```
VML>LOAD LINEAR.INS
VML>DISPLAY F8
```

(a) A V-Form


CONDITION:
LOAD DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE EXIT QUIT

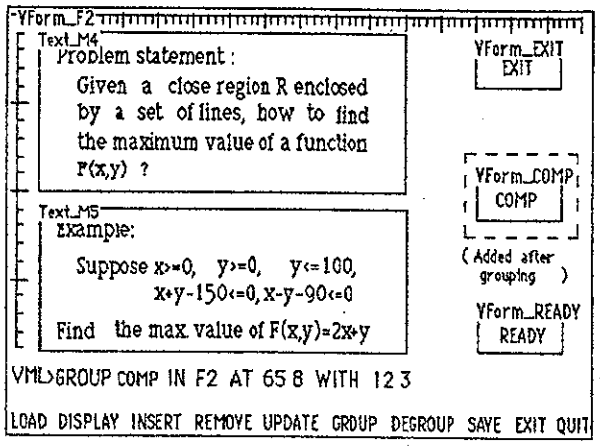
```
IF error_count LESS THEN 2 /* first wrong answer*/
```

```
VML>DISPLAY VForm_2 COND
```

(b) Condition part of VForm_2

Fig. 5.1 Example of Displaying Operations

Fig. 5.2 shows an example of grouping. The grouping operation adds a complementary course, V-Form instance COMP.INS as in Fig. 5.2(b), to the node F2 of the V-Form instance LINEAR.INS. The new structural relationship between two grouped V-Form instances can be shown in Fig. 5.2(a). Fig. 5.3 shows the logical structure after grouping. Degrouping is just the reverse of grouping. Notice that these two operations can only operate on independent V-Form instances, i.e, the grouped/degrouped V-Form instances must be structurally unrelated.



Problem statement :

Given a close region R enclosed by a set of lines, how to find the maximum value of a function F(x,y) ?

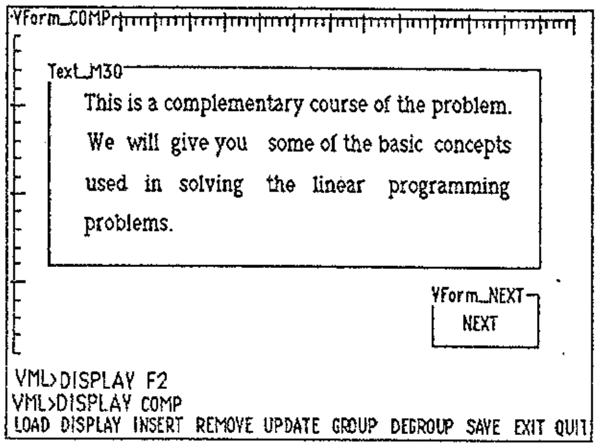
Example:

Suppose x>=0, y>=0, y<=100,
x+y-150<=0, x-y-90<=0

Find the max. value of F(x,y)=2x+y

```
VML>GROUP COMP IN F2 AT 65 8 WITH 12 3
```

(a)V-Form F2


VForm_COMP
Text_M30
VForm_NEXT
NEXT
LOAD DISPLAY INSERT REMOVE UPDATE GROUP DEGROUP SAVE EXIT QUIT

This is a complementary course of the problem. We will give you some of the basic concepts used in solving the linear programming problems.

```
VML>DISPLAY F2
VML>DISPLAY COMP
```

(b)V-Form COMP in COMP.I

Fig. 5.2 Example of Grouping

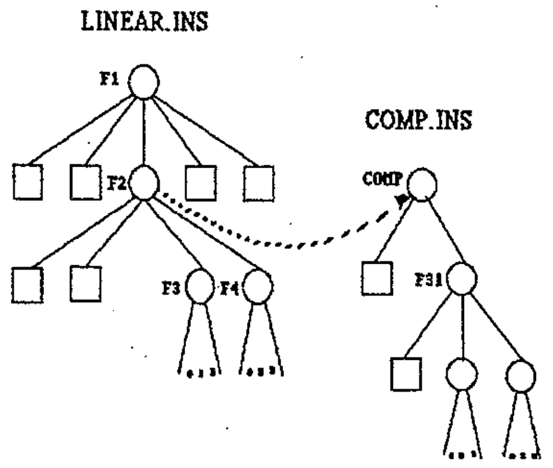


Fig. 5.3 Logical structure of LINEAR.INS after Grouping.

6. CONCLUSION

This paper has presented a visual programming synthesizer which allows users to define objcets in what-you-sketch-is-what-you-get manner and synthesizes the sketched objects into a program automatically. The contents of objects may include text, static graphics, dynamic graphics, rules, and voices. In order to exploit the objects visually, we have proposed a V-Form model in which the contents and structure of objects are explicitly incorporated. Based on the V-Form model, a V-Form Definition Language and a V-Form Manipulation Language are designed to support a visual interface between users and the system. The system, then, structuralize these V-Forms into an unique internal representation which can be easily interpreted by an interpreter. By this way, an automatic program synthesizer is obtained.

Since VIPS provides the users a feasible programming environment to develop their application programs visually, users, with or without any programming language background, can describe their specialized knowledge as the process of how to get the desired applications. This will release users from doing arduous and dirty works.

VIPS is only an experimental system in the study of visual programming techniques at the CS Department of National Taiwan University. We expect to extend its capabilities so that more knowledgable objects can be processed. But first, a theoretical understanding of visual perception is needed so that a designer can devise knowledge representations for wider objects in a more general manner.

REFERENCES

[1] Anderson, J. R., Boyle, C. F. and Yost, G., 'The Geometry Tutor,' IJCAI, LosAngles, U.S.A., pp. 1-7 (1985).

[2] Clocksin, W. F. and Mellish, C. S., *Programming in Prolog*, 2nd ed., Springer-Verlag, Berlin, 1984.

[3] Ellis, C. A. and Nutt, G. J., 'Office Information Systems and Computer Science,' Computing Surveys, Vol. 12, No. 1, pp. 27-60 (March 1980).

[4] Fu, K. S. and Booth, T. L., 'Grammatical Inference: Introduction and Survey -- Part I,' IEEE Trans. on System, Man, and Cybernetics, Vol. SMC-5, No. 1, pp. 95-111 (1975).

[5] Gray, M. D., *Logic, Algebra and Database*, Halsted Press, 1984.

[6] Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Language, and Computation*, Addison-Wesley, 1979.

[7] Jacob, Robert J. K., 'A State Transition Diagram Language for Visual Programming,' IEEE Computer, Vol. 18, No. 8, pp. 51-59 (1985).

[8] Kearsley, G., et al., 'Authoring Systems in Computer Based Education' Comm. of the ACM, Vol. 25, No. 7, pp. 429-437 (1982).

[9] King, K. J. and Maryanski, F. J., 'Information Management trends in Office Automation,' Proceedings of the IEEE, Vol. 71, No. 4, pp. 519-528 (1983).

[10] Kitagawa, H., et al., 'Form Document Management System SPECDOQ - Its Architecture and Implementation,' Second ACM-SIGOA Conference on Office Information System, Vol. 5, No. 1-2, pp. 132-142 (June 1984).

[11] Kowalski, R. A., 'Predicate Logic as Programming Language,' Proc. of IFIP 74, Stockholm (1974).

[12] Moriconi, M. and Hare, D. F., 'Visualizing Program Designs Through PegSys,' IEEE Computer, Vol. 18, No. 8, pp. 72-85 (1985).

[13] Reiser, B. J., Anderson, J. R. and Farrel, R. G., 'Dynamic Student Modelling in an Intelligent Tutor for LISP Programming,' IJCAI, LosAngles, U. S. A., pp. 8-14 (1985).

[14] Shu, N. C., et al., 'Specification of Forms Processing and Business Procedures for Office

Automation,' IEEE Transactions on Software Engineering, Vol. SE-8, No. 5, pp. 499-512 (Sep. 1982).

[15] Shu, N. C., 'A Forms-oriented and Visual-directed application development system for non-programmers,' IEEE Computer Society Workshop on Visual Language, pp.162-170 (Dec. 1984).

[16] Shu, N. C., 'FORMAL: A Form-Oriented, Visual-Directed Application Development System,' IEEE Computer, Vol. 18, No. 8, pp. 38-49 (1985).

[17] Sugihara, K., et al., 'An Approach To The Design of a Form Language,' IEEE Computer Society Workshop on Visual Language, pp.171-176 (Dec. 1984).

[18] Tsichritzis, D. C., 'Form Management,' Comm. of the ACM, Vol. 25, No. 7, pp. 453-478 (1982).

[19] Zloof, M. M., 'Query-by-Example: A Data Base Language,' IBM System Journal, Vol. 16, No. 4, pp. 324-343 (1977).

[20] Zloof, M. M., 'QBE/OBE: A Language for Office and Business Automation,' IEEE Computer, Vol. 14, No. 5, pp. 13-22 (1981).