

Implementation of Abstract Data Types in Dynamic Sketches for Learning Geometry

Eglė JASUTĖ, Valentina DAGIENĖ

*Vilnius University, Institute of Mathematics and Informatics
Akademijos str. 4, LT-08663 Vilnius, Lithuania
e-mail: egle.jasute@mii.vu.lt, valentina.dagiene@mii.vu.lt*

Received: December 2013

Abstract. A long-term observation of students' usage of a dynamic geometry in a classroom at all grade levels has challenged to develop an approach for learning and understanding mathematics in an easier way for both students and teachers. The paper deals with the results of a study that investigates the process and outcomes of the implementation of abstract data types in dynamic sketches (by composing scripts) for learning geometry. Four abstract data types have been developed and defined using algebraic specifications. The development of a dynamic sketch scenario with the implementation of these abstract data types is presented in detail. An example of creating an interactive microworld, using abstract data types, is presented and discussed as well.

Keywords: abstract data type, ADT, learning geometry, dynamic sketch, interactive microworld.

1. Introduction

The existing literature on students' conceptions of mathematics indicates that students at all grade levels have difficulties with understanding geometric sketches. To improve mathematical skills of all students in a classroom is one of the most difficult problems for mathematics teachers. Information technology is one of the modern tools which can help both teachers and students in the learning process. A lot of various educational software and learning objects are offered for teachers and students. A dynamic geometry is one of such tools for both teachers and students. A dynamic geometry is aimed to improve students' geometric skills and to make their knowledge deeper and more significant (Dagienė and Jasutienė, 2008) (Dagienė *et al.*, 2007). However, a dynamic geometry is a relatively complex tool for a mathematics teacher due to several reasons: first, the dynamic geometry constructions are based on hierarchy, so in order to construct a sketch, a teacher must have (or acquire) enough skills in developing algorithms and programs; second, most tools of the dynamic geometry's software are rather complex for the teacher (Hohenwarter *et al.*, 2007).

Some scientists declare the another problem of the usage of information technology: the usage of digital tools depends on a teacher's disposition. If teachers use an active learning approach and constructive methods of teaching, they are willing to use a dynamic geometry for teaching as well. If teachers are more satisfied with traditional teaching methods, they are not willing to use a dynamic geometry for teaching (Stols and Kriek, 2011).

A long-term observation inspired to develop an approach for making the mathematics studies easier for both students and teachers. The aim of the research is as follows: to construct a method (or a model) of interactive visualization for secondary school geometry. The method should help construct interactive objects, using a dynamic geometry, and direct teachers and students to master their skills in a dynamic geometry (Jasutė and Dagienė, 2012).

A dynamic geometry can be presented as a unique programming application. The authors have noticed that mathematics teachers have difficulty when creating interactive microworlds, using a dynamic geometry. An informatics problem can be recognised here: linking a geometric topic S_i to the interactive microworld M_k ($k = 1, 2, \dots, m$) by using a dynamic geometry. The analysis of the features of a dynamic geometry allows the authors to appeal to the abstract data type (ADT) theory. The authors define ADTs by using the methods of heterogeneous algebra and apply them in writing scenarios for the microworld M_k . Four abstract data types will be developed and an example of creating a microworld using these ADTs will be presented in the paper. The 'Geometer Sketchpad' programme will be used to illustrate the implementation results.

2. Methodology

2.1. Definitions

The authors introduce the main definitions in connection with abstract data types and algebraic specifications in this section.

Definition 1. An abstract data type is a set of data values and associated operations that are precisely specified, independent of any particular implementation.

Definition 2. ADT specification is a representation-independent formal definition of each operation of a data type. Thus, the complete design of a single data type would proceed by first giving its specification, followed by an (efficient) implementation which agrees with the specification.

Definition 3. Algebraic specification is defined as a set of three elements (S, OP, E): a set S of values, a set OP of operation symbols, and a set E of equations or axioms.

Definition 4. Implementation of data abstraction is an assignment of meaning to the values and operations in terms of the values and operations of another data type or set of data types. A correct implementation is an implementation that satisfies the axioms.

Definition 5. A generating function is a function that generates the value of a type.

Definition 6. A transforming function is a function that replaces one value of type T by another value of type T which can be generated by the generating function. This function does not create new values, but the result belongs to the set of values of type T is created by generating functions.

Definition 7. A reflecting function is a function that reflects the value of one type T to the value of some another type.

Definition 8. Homogeneous algebra is defined as a pair $[R, F]$, where R is not an empty set of values, and F is a finite set of functions F_{nj} , each function F_{nj} is a map: $F_{nj}: R_n \rightarrow R$, where n is the operand number of the j^{th} function.

Definition 9. Heterogeneous (or many-sorted) algebra is a pair $[R, F]$, where $R = \{T_1, \dots, T_n\}$ is not an empty value set of data types T_1, \dots, T_n ; and F is a finite set of functions F_{nj} , each function F_{nj} maps: $F_{nj}: T_1 \times T_2 \times \dots \times T_n \rightarrow T_r$, where n is the operand number of the j^{th} function.

2.2. ADT Specifications

ADT is used to reduce and specify a program by defining abstract data values. These values are generated by functions and constructs which are defined by the first line direct equations (Jouannaud and Okada, 1997). Three requirements for ADT are proposed in (Liskov and Guttag, 1986):

- All functions for values of ADT have to be defined in the data type description.
- The user of ADT should not know how ADT values are reflected in the memory of a computer.
- The user of ADT can use only the functions of this ADT with ADT values but cannot operate with the reflections of those values in the memory of computer.

A new ADT has to be described by a formal definition and implementation (Guttag, 1987). ADT can be described by informal and formal methods. In order to avoid ambiguity, the formal methods are used to describe ADT. The syntax specification defines the names, domains, and ranges of the operation type. The semantic specification contains a set of axioms in the form of equations which relate the operations of type to each other (Guttag, 1987). Mostly two specification methods are used: algebraic specification and operational specification (Loeckx, 1987). An operational method of specification is close to realization. Some programming language is used for ADT specification. The set of values of ADT are constructed with the help of data structures of the programming language. The operations are defined with the help of the programming language (Loeckx, 1986). A defect of the operational specification method is its lack of abstraction. This method constitutes an implementation rather than specification (Loeckx, 1986). ADT are mathematical models with associated methods which should be implemented in terms of black boxes (Heberman, 2008). An algebraic specification consists of equalities as described in (Loeckx, 1986). This method uses the axiomatic system and therefore it is independent of implementation (Guttag, 1987) (Guttag and Horning, 1978). Heterogeneous algebra (definition 9) is used for the formal description

of ADT (Ehrig and Mahr, 1985) (Liskov and Guttag, 1986) (Loeckx, 1986). Heterogeneous algebra includes the values of several types. This feature is appropriate for representing data types used in programming. According to the ADT theory only one data type is picked out and it is described by other types which were defined before (Guttag and Horning, 1978). The algebraic specification is more suitable for authors to describe ADT.

It is advisable to group the functions of a descriptive data type for creating systematic axioms: generating, transforming, and reflecting as described in definitions 5, 6, and 7. Sometimes the functions which do not generate a new value but add some feature to the existing value can be accepted as generating. The reflecting function which generates a new value of some ADT from the value of another data type can be accepted as generating as well.

A new data type can be specified systematically, based on the tables of function results. Axioms must be written for every transforming and reflecting function. The number of axioms depends on the number of generating functions with the values suitable for a definable (transforming or reflecting) function (Fig. 1).

A dynamic geometry can be described as an application where the microworld of geometry is constructed using the interaction of ADTs. Each microworld M_k is a dynamic geometry's sketch with ADTs interaction. The particularity of a dynamic geometry forces us to describe step by step scenarios for each M_k . For the formal description of scenarios, the method of heterogeneous algebra will be used. Each scenario is a set of functions F_{nj} , where every function $F_{nj}: T_1 \times T_2 \times \dots \times T_n \rightarrow Tr$; T_1, \dots, T_n are not empty ADTs; n is the number of operands in the j^{th} function.

The analysis of a dynamic geometry software let the authors to compose four main ADTs. They are named in line with their purposes: `geom.obj`, `measurements`, `text.block` and `action.button`.

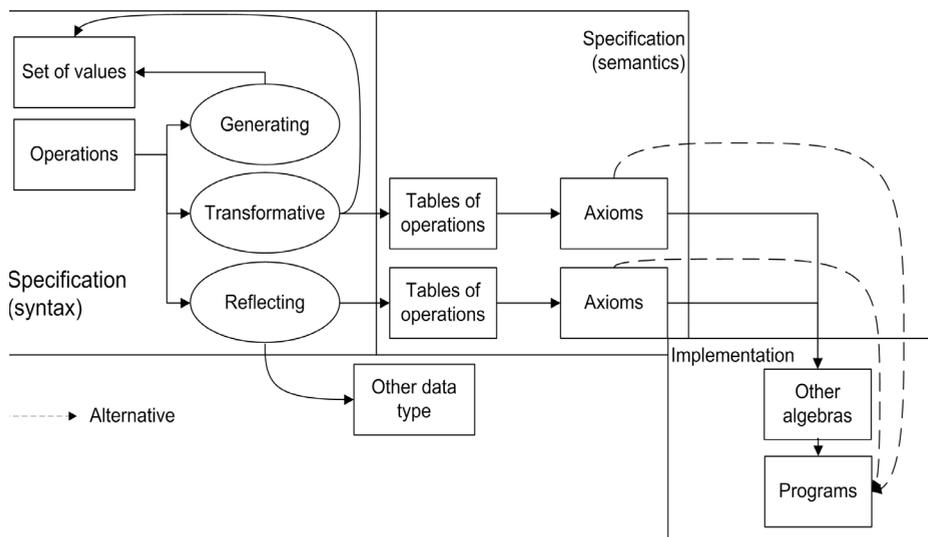


Fig. 1. Systematic method for defining ADT.

2.3. Specification of ADT in Dynamic Geometry Scenarios

The main ADT in the dynamic geometry sketch is `din.geom`. It will be defined when all ADTs are described. The center of an interactive microworld is a dynamic drawing which consists of geometric objects. The `geom.obj` is the first type described. This type can be described by homogeneous algebra, but it needs to be related with other definable types. Therefore the authors define it in the notation of heterogeneous algebra. The set of values of this data type is infinite and all the values are created by generating functions (see Annex). The set of values includes all objects from the point to complicated constructions (bisectors of a triangle, the tangent of two circles, etc.). The set of values is denoted as R1. All the values of set R1 can be produced by a finite set of functions (see Annex). Some general constants of set R1 of type `geom.obj` have to be written: `no`, `A`, `segment_AB`, `ray_AB`, `line_AB`, `vector_AB`, `arc_AB`, `arc_ABC`, `circle_AB`, `circle.sector_ABC`, `circle.segment_ABC`, `circle.sector_AB`, `circle.segment_AB`, `circ.interior_AB`, `pol.interior_Ak`. These constants are used in other specifications of definable data types as well. Variables of type `geom.obj` are defined: `r`, `l`. The variables can reduce the number of axioms for the described ADT (see Annex).

Functions are grouped into generating functions F1 and transforming and reflecting functions F2 (Fig. 2).

Some of functions F1 do not create new values. They assign features to the existing values:

- Function `select`. All functions in a dynamic geometry are used with selected objects. The result of the function `select` is labelled by subscript 1. This function will be used in all abstract data types that will be described.
- Function `hide`. Such an object exists, but it is not seen on the screen. The result of the function `hide` is labelled by subscript 2. This function will be used in all the

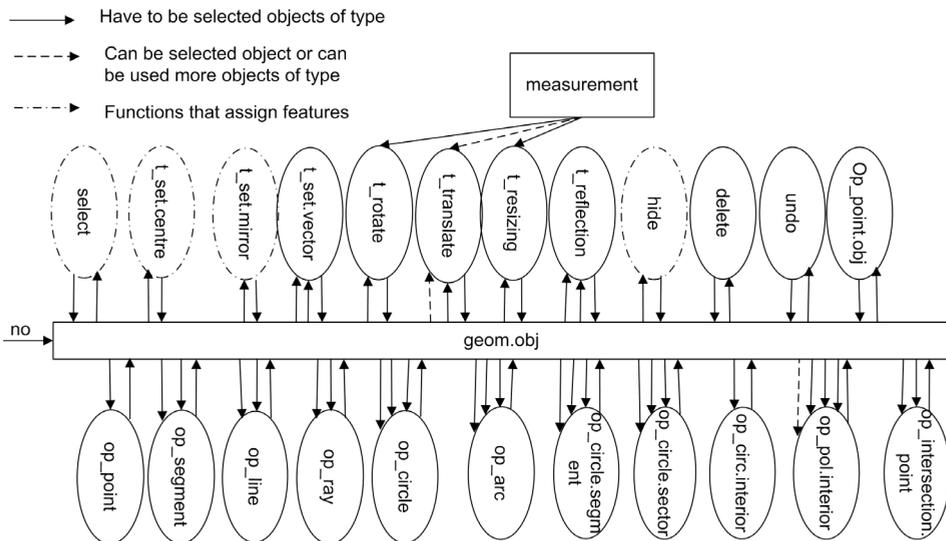


Fig. 2. Graphical visualization of ADT `geom.obj`.

abstract data types that will be described.

- Function `t_set.mirror` assigns a symmetric line feature to the straight object. The result of this function is used to create some reflected object. The result of the function `t_set.mirror` is labelled by subscript 3.
- Function `t_set.centre` assigns the central feature to the point. The result of this function is used to turn the object. The result of function `t_set.centre` is labelled by subscript 4.

This type has only three transforming functions: `undo`, `op_point.obj` and `op_intersection.point`. The axioms for these transforming functions were written down (see Annex).

Another ADT for a dynamic geometry is named by `measurement`. This type needs a method of heterogeneous algebra for its specification. This type consists of two types: the type of real numbers and the type of measurement units. The set of values of this type is the Cartesian multiplication of two sets {real} and {no, cm, inches, pixel, degree, direct degree}. The set of values is infinite. Formally this set of `measurement` values is named by `R2`. All functions of real numbers are suitable for this type. The authors use variables to represent the values of this type: a variable `a` of the type `real` and variables `m`, `n`, `d` of the type `measurement`.

Functions of this type are distributed in two groups: generating and transforming functions `F3`, and transforming functions `F4` which reflect values to that of the data type `geom.obj` (Fig. 3).

All the values of type `measurement` can be generated by seven functions: `m_parameter`, `op_unit`, `sign '+'`, `sign '-'`, `t_set.distance`, `t_set.angle`, `t_set.ratio`. The generating function `t_set.distance` assigns a feature to measurements. The result of this function is labelled by subscript 7. The reflecting functions `t_set.angle` and `t_set.ratio` assign features to the existing values. The results of these functions are labelled by subscripts 5 and 6, respectively. Axioms are constructed for the rest transforming and reflecting functions (see Annex).

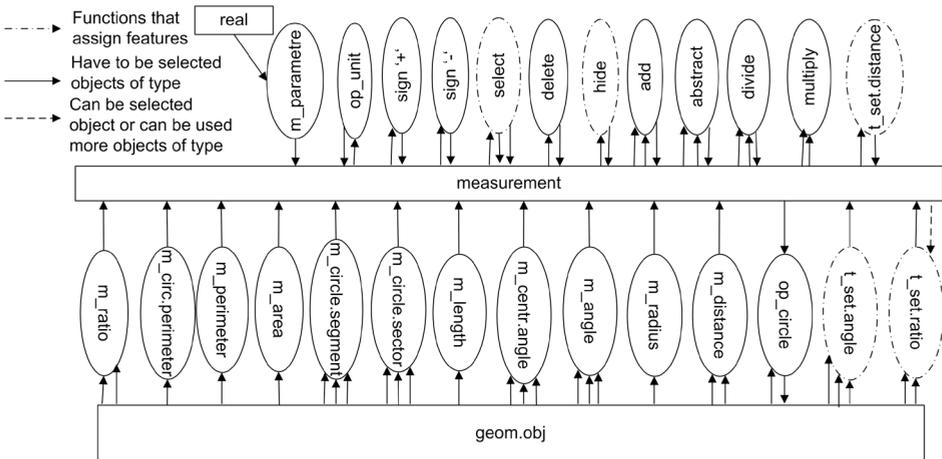


Fig. 3. Graphical visualization of relations between ADTs `geom.obj` and `measurement`.

The third defined type of ADT is `text.block`. The set of values of this type is infinite and the values are text blocks. This set was named by R3. Every block of this type consists of characters. To this end the authors use a data type `char` with generating functions `c_add`, `c_delete`, and `c_select` (Fig. 4). This type has a finite set of functions: generating functions F5 and reflecting functions F6 (see Annex).

The last created ADT is `action.button`. The set of values of this type is finite: $R4 = \{no, ah, as, aa, aac, seq\}$. All values are generated by the reflecting functions.

All the described ADTs are composed in heterogeneous algebra. In this way, the authors have obtained a new ADT named `din.geom`. A set of values of this type is $R = \{R1, R2, R3, R4\}$, where set R1 consists of values of the type `geom.obj`, set R2 consists of values of the type `measurement`, set R3 consists of values of the type `text.block` and set R4 consists of values of the type `action.button`.

All the functions described for each data type are suitable for ADT: $F = \{\{F1\}, \{F2\}, \{F3\}, \{F4\}, \{F5\}, \{F6\}, \{F7\}, \{F8\}\}$ (Fig. 6).

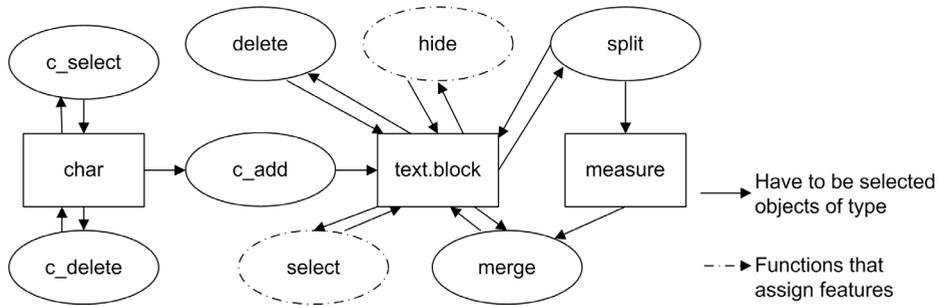


Fig. 4. Graphical visualization of the type `text.block`.

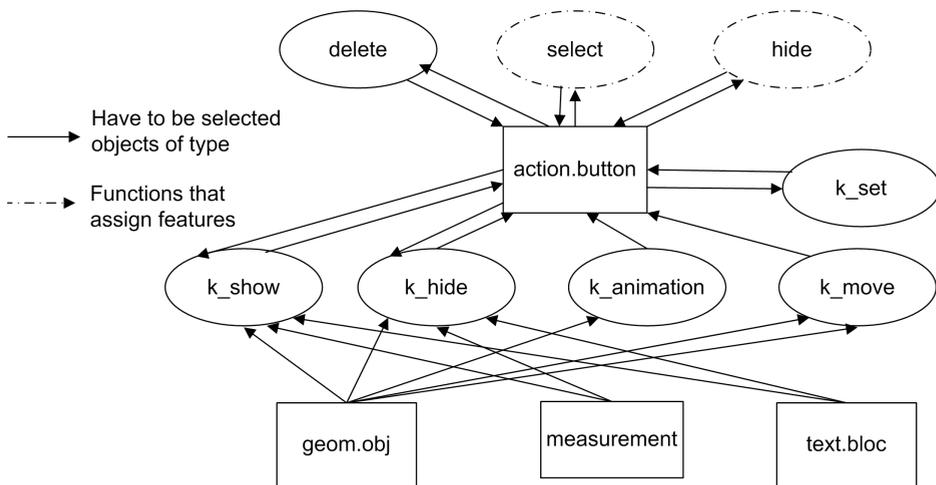


Fig. 5. Graphical visualization of ADT `action.button`.

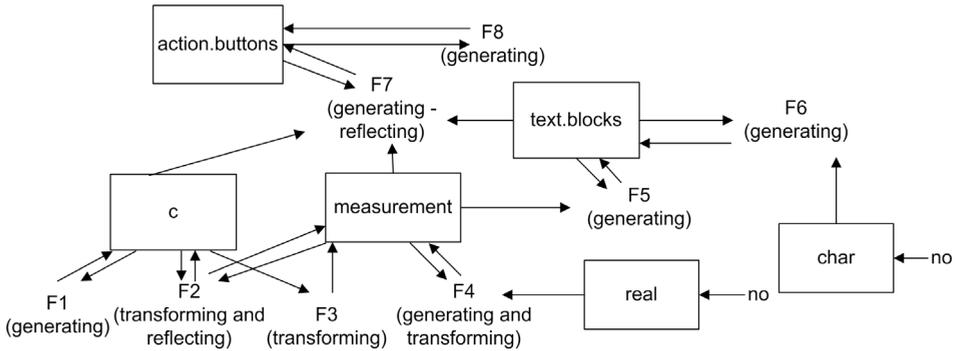


Fig. 6. Schema of the ADT syntax in a dynamic geometry.

3. Example of ADT Implementation in a Dynamic Geometry

As the authors described above, in order to create the microworld M_k in a dynamic geometry application, it is necessary to describe that step-by-step. All the steps are functions of the ADT `din.geom`.

For example, the authors illustrate how to create a microworld for the learning topic: ‘The area of a triangle when two sides and the included angle are given’.

To create this scenario the authors need an additional scenario `angle.arc` to construct an arc to mark the angle in the drawing of the triangle. This scenario will be used three for times in the main scenario. The same constants are used just like in the ADT descriptions above.

```
Name: angle.arc
1.op_point(no)=A1
2.op_point(no)=B1
3.op_point(no)=C1
4.op_segment(A1,B1)=segment_AB1
5.op_segment(B1,C1)=segment_BC1
6.t_translate(B1,0,3)=B1
7.op_circle(B1,B1)=circle_B1B1
8.op_intersection.point(segment_AB1,circle_B1B1)=E
9.op_intersection.point(segment_AB1,circle_B1B1)=F
10.op_arc(E,F,circle_B1B1)=arc_EF
```

The scenario of microworld in heterogeneous algebra axioms is written below:

```
Name: area.triangle.trig
1.op_point(no)=A1 {geom.obj→geom.obj}
2.op_point(no)=B1
3.op_point(no)=C1
4.op_segment(A1,B1)=segment_AB1
5.op_segment(B1,C1)=segment_BC1
```

```

6.op_segment(A1,C1)=segment_AC1
7.angle.arc(A1, B1, C1)=arc_EF {script}
8.angle.arc(B1, A1, C1)=arc_EF {script}
9.angle.arc(A1, C1, B1)=arc_EF {script}
10.m_segment(AB1)=a {geom.obj→measurement}
11.m_segment(BC1)=b
12.m_segment(AC1)=c
13.m_angle(A1, B1, C1)=β
14.m_angle(A1, C1, B1)=γ
15.m_angle(C1, A1, C1)=α
16.c_add(empty)=t1{char→text.block}
17.c_add(empty)=t2{char→text.block}
18.c_add(empty)=t3{char→text.block}
19.c_add(empty)=t4{char→text.block}
20.m_area(op_pol.interior(A31))=S {geom.obj→measurement}
21.merge(t1,b,t2,c,t3,α,t4,S)=t7{text.block→text.block}
22.merge(t1,a,t2,b,t3,γ,t4,S)=t7
23.merge(t1,a,t2,c,t3,β,t4,S)=t7
24.c_add(empty)=t8{char→text.block}
25.k_hide(t8)=ah {text→action.button}
26.k_show(t8)=as {text→action.button}
27.k_animation(A1)=aa1 {geom.obj→action.button}
28.k_animation(B1)=aa2
29.k_animation(C1)=aa3
30.op_point(no)=X {geom.obj→geom.obj}
31.op_point(no)=Y
32.op_point(no)=Z
33.k_action(A1, X1)=aac1 {geom.obj→action.button}
34.k_action(B1, Y1)=aac2
35.k_action(C1, Z1)=aac3
36.k_sequence(aac1,aac2,aac3,ah)=Start{action.button→action.
    button}
37.c_add(empty)=t9{char→text.block}

```

The result of the scenario implemented in a dynamic geometry is shown in Fig. 7. The points A, B and C are movable. The measures of sides and angles change when the buttons between measures of sides and angles are clicked. The start position can be restored when the button in the left angle of the sketch is clicked.

4. Conclusions and Discussions

Information technologies became an important part of the mathematics education process. One of such tools is a group of programs called a dynamic geometry. But the usage of a dynamic geometry is too complex for a large number of teachers in school. The

Keisdami trikampį stebėkite, kaip galima apskaičiuoti jo plotą, remiantis dviejų kraštinių ilgiais ir kampo tarp tų kraštinių sinusu

Pradinė padėtis

a = 4,0	Keisti a, b	α = 102°
b = 2,2	Keisti b, c	β = 33°
c = 2,9	Keisti a, c	γ = 45°

$$S = \frac{1}{2} \cdot b \cdot c \cdot \sin \alpha = \frac{1}{2} \cdot 2,2 \cdot 2,9 \cdot \sin 102^\circ = 3,2$$

$$S = \frac{1}{2} \cdot b \cdot a \cdot \sin \gamma = \frac{1}{2} \cdot 2,2 \cdot 4,0 \cdot \sin 45^\circ = 3,2$$

$$S = \frac{1}{2} \cdot c \cdot a \cdot \sin \beta = \frac{1}{2} \cdot 2,9 \cdot 4,0 \cdot \sin 33^\circ = 3,2$$

Rodyti išvadą

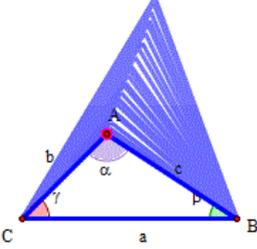


Fig. 7. The scenario implementation in the Geometer Sketchpad.

digital competency of most mathematics teachers is imperfect. The authors have found a way how to help teachers to use a dynamic geometry and to make learning deeper and more attractive for students. The method has been developed for creating interactive microworlds with a dynamic geometry. The core of the method is to map some geometric topic S_i to interactive microworld M_i . This step was formalized by using the abstract data type theory. More than 500 interactive microworlds have been developed, based on this formalized method for mathematics teachers and students in Lithuanian schools.

The described ADT can be expanded by adding more ADTs. The authors described ADT to create microworlds for learning the geometry only. ADT for creating microworlds for learning algebra or mathematics analysis can be described and added to the type `dim.geom` in the presented way.

The authors choose the algebraic specification for formalizing ADT. This specification method is abstract and can be realized in any dynamic geometry. And ADT can be expanded, depending on tools of a dynamic geometry, if necessary.

Annex

Specification of the ADT `geom.obj`

Syntax.

```
F1={op_point: → geom.obj
  select: geom.obj → geom.obj
  delete: geom.obj → geom.obj
  hide: geom.obj → geom.obj
```

```

t_set.mirror: geom.obj → geom.obj
t_set.centre: geom.obj → geom.obj
op_segment: geom.obj x geom.obj → geom.obj
op_ray: geom.obj x geom.obj → geom.obj
op_line: geom.obj x geom.obj → geom.obj
op_circle: geom.obj x geom.obj → geom.obj
op_arc: geom.obj x geom.obj x geom.obj → geom.obj
op_circle.sector: geom.obj x geom.obj x geom.obj → geom.obj
op_circle.segment: geom.obj x geom.obj x geom.obj → geom.obj
op_circ.interior: geom.obj → geom.obj
op_pol.interior: geom.obj x geom.obj x...x geom.obj → geom.obj
t_set.vector: geom.obj x geom.obj → geom.obj
t_translate: geom.obj x x geom.obj → geom.obj
t_reflection: geom.obj x geom.obj → geom.obj
t_resizing: geom.obj x measurement → geom.obj
t_rotate: geom.obj x measurement → geom.obj
t_translate: geom.obj x measurement → geom.obj}
F2={undo: geom.obj → geom.obj
op_point.obj: geom.obj → geom.obj
op_intersection.point: geom.obj x geom.obj → geom.obj}

```

Semantic.

```

op_intersection.point(no,no)=no
op_intersection.point(no,op_point(A))=?
op_intersection.point(op_point(A),op_point(B))=?
op_intersection.point(op_point(A),r1)=?
op_intersection.point(r1,l1)=E
undo(select(r))=r
undo(hide(r))=r
undo(op_point(A))=r
undo(op_segment(A,B))=(A1,B1)
undo(op_line(A,B))=(A1,B1)
undo(op_ray(A,B))=(A1,B1)
undo(op_circle(O,B))=(O1,B1)
undo(op_circle(O,B))=(O1,segment_AB1)
undo(op_arc(A,B,C))=(A1,B1,C1)
undo(op_arc(A,B))=(A1,B1)
undo(circle.sector_ABC)=arc_ABC
undo(circle.sector_AB)=arc_AB
undo(circle.segment_ABC)=arc_ABC
undo(circle.segment_AB)=arc_AB
undo(op_circ.interior(O,B))=circle_OB1
undo(op_pol.interior(Ak))=(Ak1)
undo(r2)=r1

```

```

undo(delete(r1))=r1
undo(t_set.vector(A,B)=(A,B)
undo(t_set.mirror(A,B)=segment_AB1
undo(t_set.mirror(A,B)=line_AB1
undo(t_set.mirror(A,B)=ray_AB1
undo(t_set.centre(A)=A1
undo(t_translate(r))=r1
undo(t_reflection(r,ray_AB))=r1
undo(t_reflection(r,line_AB))=r1
undo(t_reflection(r,segment_AB))=r1

```

Specification of the ADT measurement

Syntax.

```

F4={m_parameter: → real
op_unit: real → measurement
sign'+': measurement → measurement
sign'-\': measurement → measurement
delete: measurement → measurement
select: measurement → measurement
hide: measurement → measurement
add: measurement x measurement → measurement
subtract: measurement x measurement → measurement
multiply: measurement x measurement → measurement
divide: measurement x measurement → measurement
t_set.distance: measurement →measurement}
F3={m_length: geom.obj → measurement
m_distance: geom.obj x geom.obj → measurement
m_ray: geom.obj → measurement
m_angle: geom.obj x geom.obj x geom.obj → measurement
m_centr.angle: geom.obj x geom.obj x geom.obj → measurement
m_circle.sector: geom.obj → measurement
m_circle.segment: geom.obj → measurement
m_area: geom.obj → measurement
m_perimeter: geom.obj x geom.obj → measurement
m_circl.perimeter:geom.obj→measurement
m_ratio: geom.obj x geom.obj → measurement
t_set.angle: geom.obj x geom.obj x geom.obj → measurement
t_set.ratio: geom.obj x geom.obj → measurement}

```

Semantic.

```

A, B, C, O: geom.obj
m, n, x, y, a: measurement
undo(m_parametre(no))=no

```

```

undo (op_unit (a) )=a
undo (sign' +' (m) )=m
undo (sign' -' (m) )=m
undo (t_set.distance (m1) )=m1
undo (select (m) )=m
undo (hide (m) )=m
undo (m_length (op_segment (A,B) ) )=op_segment (A,B)
undo (m_distance (op_point (A) , op_point (B) ) )=op_point (A) , op_
point (B)
undo (m_distance (op_point (A) , op_segment (AB) ) )=op_point (A) , op_
segment (AB) )
undo (m_distance (op_point (A) , op_ray (AB) ) )=op_point (A) , op_ray
(AB)
undo (m_distance (op_point (A) , op_line (AB) ) )=op_point (A) , op_
line (AB)
undo (m_radius (op_circle (OB) ) )=op_circle (OB)
undo (m_angle (op_point (A) , op_point (B) , op_point (C) ) )=op_point
(A) , op_point (B) , op_point (C)
undo (m_centr.angle (op_point (A) , op_point (B) , op_circle (O,C) ) )=
op_point (A) , op_point (B) , op_point (C)
undo (m_circle.sector (op_point (A) , op_point (B) , op_circle (O,C)
) )=op_point (A) , op_point (B) , op_circle (O,C) )
undo (m_circle.segment (op_point (A) , op_point (B) , op_circle (O,C)
) )=op_point (A) , op_point (B) , op_circle (O,C)
undo (m_area (op_circle (O,C) ) )=op_circle (O,C)
undo (m_area (op_circ.interior (O,C) ) )=op_circ.interior (O,C)
undo (m_area (op_pol.interior (Ak) ) )=op_pol.interior (Ak)
undo (m_perimeter (op_pol.interior (Ak) ) )=op_pol.interior (Ak)
undo (m_circl.perimeter (op_circle (O,C) ) )=op_circle (O,C)
undo (m_circ.perimeter (op_circ.interior (O,C) ) )=op_circ.
interior (O,C)
undo (m_ratio (op_segment (A,B) , op_segment (C,D) ) )=op_segment
(A,B)
undo (t_set.angle (op_point (A) , op_point (B) , op_point (C) ) )=op_
point (A) , op_point (B) , op_point (C)
undo (t_set.ratio (divide (x,y) ) )=divide (x,y)

```

Specification of the ADT text.block

Syntax.

```

F5={empty: → text.block
add: char → text.block
delete: text.block → text.block
select: text.block → text.block

```

```

hide: text.block → text.block
merge: text.block text.block → text.block
select: text.block → text.block
delete: text.block → text.block }
F6={merge: measurement x text.block → text.block
split: text.block → measurement x text.block}

```

Semantic.

```

t: text.block
m: measurement
split(merge(m,t))=m,t
undo(empty)=?
undo(c_add(t))=empty
undo(select(t))=t
undo(hide(t))=t
undo(split(merge(m,t))=merge(m,t)
undo(delete(t))=t

```

Specification of the ADT action.button*Syntax.*

```

F7={delete: → action.button → action.button
select: action.button → action.button
hide: action.button → action.button
k_sequence: action.button → action.button }
F8={k_hide: geom.obj x measurement x text.block x action.
button → action.button
k_show: geom.obj x matai x tekstai x action.button → action.
button
k_animation: geom.obj → action.button
k_action: geom.obj x geom.obj → action.button}

```

Semantic.

```

r, A, B, C, D: geom.obj,
t: text.bock,
m: measurement
a, b, c, d: action.button
undo(k_show(a,t,m,r))=a,t,m,r
undo(k_hide(a,t,m,r))=a,t,m,r
undo(k_animation(r,no))=r
undo(k_animation(op_point(A),op_segment(B,C)))=op_point(A),
op_segment(B,C)
undo(k_animation(op_point(A),op_ray(B,C)))=op_point(A),op_
ray(B,C)

```

```

undo(k_animacija(op_point(A), op_line(B,C)))=op_point(A), op_
line(B,C)
undo(k_animacija(op_point(A), op_circle(O,B)))=op_point(A),
op_circle(O,B)
undo(k_animacija(op_point(A), op_arc(B,C,D)))=op_point(A),
op_arc(B,C,D)
undo(k_action(op_point(A), op_point(B)))=op_point(A), op_point
(B)
undo(k_sequence(a,b,c,d))=a,b,c,d

```

References

- Dagienė, V., Jasutienė, E. (2008). Developing dynamic sketches for teaching mathematics in basic schools. In: *The 17th ICMI Study: Technology Revised*. Hanoi University of Technology, Vietnam, 120–127.
- Dagienė, V., Jasutienė, E., Jevsikova, T. (2007). An approach to combine learning entities to support mathematics curriculum in schools. In: Benzie, D., Iding, M. (Eds.), *Informatics, Mathematics and ICT: A “golden triangle”*. CD: Proceedings of the Working Joint IFIP Conference, Northeastern University Boston, Massachusetts, USA 27th–29th June 2007. Boston, MA.
- Ehrig, H., Mahr, B. (1985). *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer.
- Ehrig, H., Mahr, B., Classen, I., Orejas, F. (1992). Introduction to algebraic specification. Part 1. Formal methods for software development. *The Computer Journal*, 35(5), 460–467.
- Guttag, J.V. (1987). Abstract data types and software validation. *Communications of the ACM CACM*, 21(12), 1048–1064.
- Guttag, J.V., Horning J.J. (1978). The algebraic specification of abstract data types. *Acta Informatica*, 10, 27–52.
- Heberman, B. (2008). Formal and practical aspects of implementing abstract data types in the Prolog instruction. *Informatica*, 2008, 19, (1), 17–30.
- Hohenwarter, J., Hohenwarter, M., Lavicza, Z. (2009). Introducing dynamic mathematics software to secondary school teachers: the case of GeoGebra. *Journal of Computers in Mathematics and Science Teaching*, 28(2), 135–146.
- Jasutė, E., Dagienė, V. (2012). Constructionist learning of geometry. In: *Proceedings of conference Constructionism 2012: Theory, Practice and Impact, August 21–25, 2012*. The education technology Lab, Athens, 386–396.
- Jouannaud, J., Okada, M. (1997). Abstract data type systems. *Theoretical computer science*, 173(2), 349–391.
- Liskov, B., Guttag, J. (1986). *Abstraction and Specification in Program Development*. MIT Press.
- Loeckx, J. (1986). The algorithmic specification method of abstract data types: an overview. In: *Mathematical Methods of Specification and Synthesis of Software Systems ‘85*, (Lecture Notes in Computer Science, vol. 215), 194–200.
- Loeckx, J. (1987). Algorithmic specifications: a constructive specification method for abstract data types. *ACM Transactions on Programming Languages and Systems*, 9(4), 646–661.
- Stols, G., Kriek, J. (2011). Why don’t all maths teachers use dynamic geometry software in their classrooms? *Australasian Journal of Educational Technology*, 27(1), 137–151.

E. Jasutė has been teaching Mathematics and Computer Science at Vilnius Jesuit High School since 1997. Since 2001 she has also been working as a researcher at Vilnius University Institute of Mathematics and Informatics. Her studies focus on the usage of IT in education and their influence on learning results. She became a PhD student in 2010. She works on the creation of interactive tools for constructionist learning of Geometry. E. Jasutė together with several co-authors has developed a few interactive learning tools related to Dynamic Geometry. She has written several methodical and scientific articles on the studied subject and localized educational and non-educational software and has participated in several European projects, as well.

V. Dagiienė (Professor D.Sc.) is a head of a department in Vilnius University Institute of Mathematics and Informatics. She has published over 100 scientific papers, over 100 methodical works, and more than 50 textbooks in the field of informatics. She is actively involved in various national and international committees as well as work groups on informatics education. V. Dagiienė is a member of International Committee of Olympiads in Informatics and chairs the committee of the conference on Olympiads in Informatics, which has been organized since 2007. In 2004 she founded International Contest on Informatics and Computer Fluency BEBRAS which runs every year and involves about 30 countries. She is a founder of two international journals, “Informatics in Education” and “Olympiads in Education”.

Abstrakčiųjų duomenų tipų realizavimas dinaminių brėžinių scenarijuose

Eglė JASUTĖ, Valentina DAGIENĖ

Ilgalaikis dinaminės geometrijos naudojimo visų klasių pamokose stebėjimas inspiravo sukurti metodą, kuris mokytojams ir mokiniams palengvintų matematikos mokymąsi, tam naudojant dinaminę geometriją. Šiame straipsnyje pateikiami tyrimo, kurio metu abstraktieji duomenų tipai realizuojami dinaminiuose brėžiniuose (konstruojant scenarijus), procesas ir rezultatai. Apibrėžti keturi abstraktieji duomenų tipai naudojant algebrines specifikacijas. Pateikiama detali abstrakčiųjų duomenų tipų realizacija naudojantis dinaminės geometrijos priemonėmis. Taip pat pateiktas interaktyvaus mikropasaulio kūrimo dinaminėje geometrijoje, naudojant abstrakčius duomenų tipus, pavyzdys.