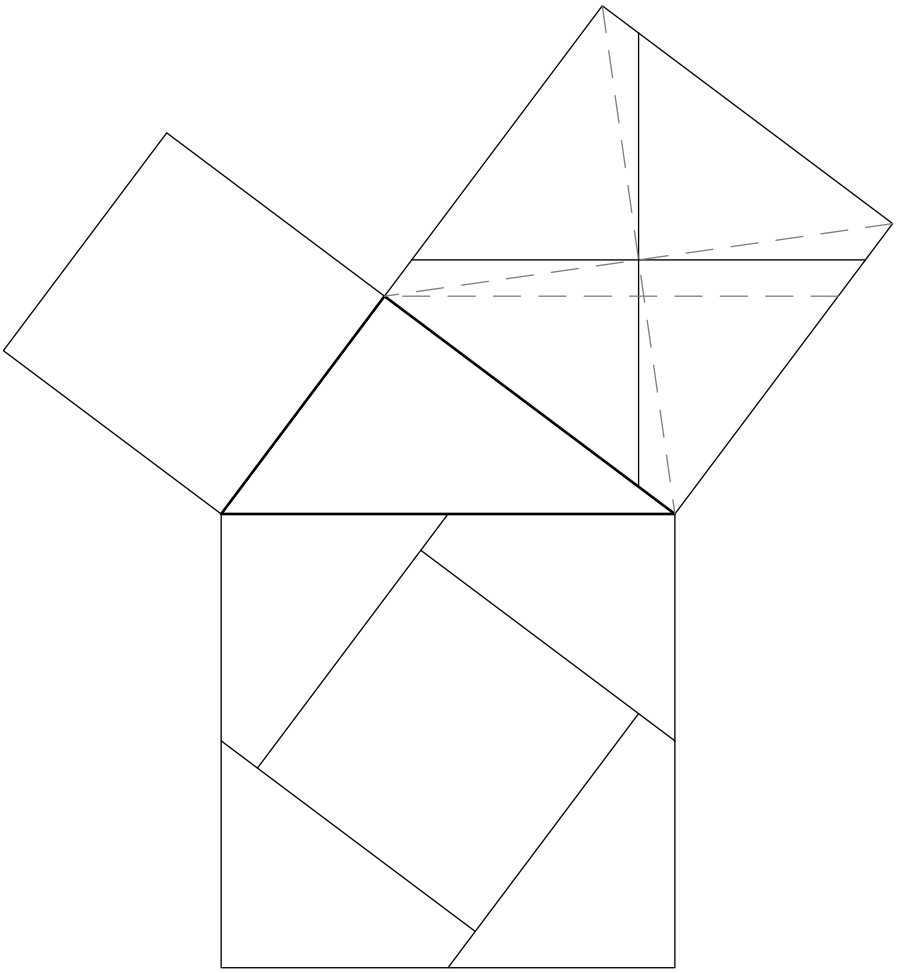
ΕΥΚΛΕΙΔΗΣ

A Euclidean Geometry Drawing Language



Christian Obrecht

Table of Contents

Introduction

This manual describes the second version of the Eukleides language, as implemented in the `eukleides` 1.5.4 interpreter. The first version of the language was implemented in `eukleides` up to 1.0, which is not developed any longer. Even though both versions have rather close designs, there is no backwards compatibility.

Eukleides is a computer language devoted to elementary plane geometry. It aims to be a fairly comprehensive system to create geometric figures, either static or dynamic. It allows to handle basic types of data: numbers and strings, as well as geometric types of data: points, vectors, sets (of points), lines, circles, and conics.

A Eukleides script usually consists in a declarative part where objects are defined, and a descriptive part where objects are drawn. Nonetheless, Eukleides is also a full featured programming language, providing conditional and iterative structures, user defined functions, modules, etc. Hence, it can easily be extended.

The Eukleides distribution provides two distinct interpreters: `eukleides` and `euktopst`, and three shell scripts: `euktoeps`, `euktotex`, and `euktopdf`. The former interpreter produces Encapsulated PostScript (EPS). The later, which is run by the scripts, produces TEXable PSTricks macros. The `euktoeps` script is an alternative to `eukleides` when mathematics typesetting is required. The two other scripts are useful when using Eukleides together with LATEX.

The first version of Eukleides came with a graphical user interface (GUI) named `xeukleides`, allowing to create and view interactive figures. A GUI for the second version will be developed in the future. Yet, the specifications of the language already include interactivity.

Bug reports as well as comments or contributions should be sent to the author of Eukleides, Christian Obrecht (`obrecht` *`at`* `eukleides` *`dot`* `org`).

1 Basics

1.1 General Syntax

Eukleides source files should be ISO 8859-1 encoded text files. Line breaks can be either in DOS, Mac OS or Unix format.

Eukleides code consists in directives, statements and comments.

Directives are one-line special instructions starting with either the number sign (`#`) or the at sign (`@`).

Statements end in a semi-colon or a line break. Runs of white space are ignored. A statement may be written on more than one line using a backslash (\) as line continuation character. Statements are either assignments or commands. Command parameters require no parentheses, which differs from the syntax of the first version of the language.

Comments start with a percent sign (`%`) and run to the end of the line.

Identifiers are case sensitive. Valid symbols in identifiers are letters, underscore (`_`), ISO 8859-1 accented letters (`0xC0` to `0xFF`, except `0xD7` and `0xF7`) and (except for the first character) digits and single quote (`'`).

Default keywords are taken from the English vocabulary, hence contain only unaccented letters, but localized keywords may contain accented letters as well.

A variable may contain any type of data: number, string, point, vector, set, line, circle or conic. Assignments to single variables are made using the equal sign (`=`). A variable in use may be unset using the `clear` command.

Angular parameters may be given either in degrees or radians. An angle measure consists in a number followed by either the `deg` keyword or a colon (`:`) or a degree sign (`°`) for degrees or by the `rad` keyword for radians. Usage of colons is deprecated.

In this manual, optional parameters are enclosed in braces.

1.2 Numbers

Numbers are stored in double precision floating point format.

There's no integer type. Whenever an integer argument is expected, the passed value is truncated.

Constant numbers consist in decimal digits and possibly a decimal dot. The dot may start the sequence; in this case the first digit is assumed to be a zero.

Numeric operators

`x + y` Sum of x and y.

`x - y` Difference of x and y.

`x * y` Product of x and y.

`x / y` Quotient of x and y.

`x ^ y` x to the power y.

`x mod y` Remainder after division of x by y.

Numeric functions

`sqrt(x)` Square root of x.

`exp(x)` Base-e exponential of x.

`ln(x)` Natural logarithm of x.

`sin(x), cos(x), tan(x)`
Sine, cosine, tangent of x degrees.

`asin(x), acos(x), atan(x)`
Arcsine, arccosine, arctangent of x (in degrees).

`deg(x)` Radians to degrees conversion.

`rad(x)` Degrees to radians conversion.

`abs(x)` Absolute value of x.

`sign(x)` -1 if $x < 0$, 0 if $x = 0$, 1 if $x > 0$.

`ceil(x)` Smallest integral value greater than or equal to x.

`floor(x)` Largest integral value less than or equal to x.

`round(x)` Integral value nearest to x.

`min(x, y)` Minimum of x and y.

`max(x, y)` Maximum of x and y.

`clamp(x, y, z)`
y if $x < y$, z if $x > z$, x otherwise.

Numeric constant

`pi` Archimedes' constant.

1.3 Strings

Literal strings must be enclosed in double quotes (`"`) or dollar signs (`$`). With `eukleides`, enclosing characters yield no difference but with `euktopst` dollar signs are taken as part of the string. Literal strings may be split in several lines.

Special characters

`%n` Newline (LF).

`%r` Return (CR).

`%t` Tab.

`%"` Double quote (when enclosing character).

`%$` Dollar sign (when enclosing character).

`%%` Percent sign.

String related functions

`length(s)`
Length of string s.

`sub(s, i, j)`
Substring of string s from index i to j. Indices start at 0.

`cat(`*list*`)`

Concatenates *list* into a single string, where *list* is a comma separated sequence of strings, numbers, points or sets. Numbers are formatted using at most 6 digits, with no trailing zeros or decimal point (i.e. the `%g` format for `printf` in C). Points are converted to their Cartesian coordinates. With `eukleides` abscissa and ordinate are simply separated by a white space. With `euktopst` coordinates are formated in the usual mathematical way, using parenthesis and comma.

2 Objects

2.1 Points

Points are stored using an implicit Cartesian coordinate system.

Point related functions

`point(x, y)`
: Point of Cartesian coordinates (x, y).

`point(r, a)`
: Point of polar coordinates (r, a).

`abscissa(A)`
: Abscissa of point A.

`ordinate(A)`
: Ordinate of point A.

`distance(A, B)`
: Distance between point A and point B.

`barycenter(`*`list`*`)`
: Barycenter of a set of weighted points. In the given list, each point has to be followed by its weight.

Example: `barycenter(A, 1, B, 3, C, 2)`.

2.2 Vectors

Vectors are stored using their Cartesian coordinates.

Vector operators

`u + v` Sum of u and v.

`u - v` Difference of u and v.

`k * u` Scalar product of u by k.

`u / k` Scalar quotient of u by k.

`u * v` Dot product of u and v.

Vector related functions

`vector(x, y)`
: Vector of Cartesian coordinates (x, y).

`vector(r, a)`
: Vector of polar coordinates (r, a).

`vector(A, B)`
: Vector going from point A to point B.

`abscissa(u)`
: Abscissa of u.

`ordinate(u)`
: Ordinate of u.

`length(u)`
: Length of u.

`arg(u)` Polar angle of u (in degrees).

`angle(u, v)`
: Angle between u and v (in degrees).

2.3 Sets

A set consists in a finite ordered list of points. Thus a set can also be seen as an open path of line segments or as a polygon.

A set of two points is considered as a single line segment. When a parameter is expected to be a segment, any subsequent point is ignored.

Set operators

`s.t` Concatenates s and t. Each operand can be either a set or a point.

`s[i]` Point of index i in variable s, assuming s contains a set. Indices start at 0.

Set related functions

`set(P)` Singleton containing point P.

`card(s)` Number of elements of set s.

`length(s)`
: Length of path s.

`perimeter(s)`
: Perimeter of polygon s.

`area(s)` Area enclosed by polygon s, provided s is not self-intersecting.

`arg(s)` Polar angle of segment s (in degrees).

`point(s, x)`
: Point of abscissa x on an axis containing segment s. The point is on segment s when x ranges from 0 to 1.

`midpoint(s)`
: Midpoint of segment s.

`bisector(s)`
: Perpendicular bisector of segment s.

`isobarycenter(s)`
: Isobarycenter of set s.

`centroid(s)`
: Centroid of polygon s.

`element(s, i)`
: Point of index i in set s.

`vector(s)`
: Vector going from first to second point of segment s.

`sub(s, i, j)`
: Subset of set s from index i to j.

`polygon(n, O, r, a)`
`pentagon(O, r, a)`
`hexagon(O, r, a)`
: Vertices of a n-sided (or 5-sided or 6-sided) convex regular polygon of center O. The first point has (r, a) as polar coordinates with respect to O. The vertices are ordered anticlockwise.

Set constant

`empty` Empty set.

Set related assignments

Each element of a variable containing a set may be modified individually using its index enclosed in square brackets.

Example: `S[1] = point(2, pi/3)`

When using empty square brackets, the given point is added to the tail of the set.

Example: `S[] = point(0, 0)`

Several elements of a set may be assigned at once to a dot separated list of variables. Exceeding points, if any, are ignored.

Example: `A.B.C.D.E = pentagon(O, 1, 0°)`

2.4 Lines

A line is internally represented by an origin point and an angular direction, i.e. the anticlockwise angle from the horizontal axis to the line. Thus, lines have an implicit orientation.

Line related functions

`line(A, a)`
: Line of origin A and direction a.

`line(A, B)`
: Line of origin A, passing through point B.

`line(A, u)`
: Line of origin A, directed by vector u.

`line(s)` Line passing through segment s. The origin is set to the first point of s.

`parallel(l, A)`
: Parallel to line l of origin A.

`parallel(s, A)`
: Parallel to segment s of origin A.

`perpendicular(l, A)`
: Perpendicular to line l of origin A.

`perpendicular(s, A)`
: Perpendicular to segment s of origin A.

`bisector(l, l')`
: Bisector of lines l and l'. The resulting angular direction is the mean of the directions of l and l'.

`distance(A, l)`
: Distance between point A and line l.

`arg(l)` Polar angle of line l (in degrees).

`point(l, x)`
Point of abscissa x on line l with respect to its implicit origin and orientation.

`abscissa(l, x)`
Point of abscissa x on line l with respect to the implicit coordinate system.

`ordinate(l, y)`
Point of ordinate y on line l with respect to the implicit coordinate system.

`vector(l)`
Unit vector having the same direction than line l.

2.5 Circles

A circle is internally represented by its center and radius.

Circle related functions

`circle(A, x)`
Circle of center A and radius x.

`circle(s)`
Circle of diameter s.

`radius(c)`
Radius of circle c.

`perimeter(c)`
Perimeter of circle c.

`area(c)` Area enclosed in circle c.

`arg(A, c)` Polar angle of point A with respect to the center of c (in degrees).

`point(c, a)`
Point on c with polar angle a with respect to its center.

`center(c)`
Center of circle c.

`line(c, a)`
Tangent line to circle c. The contact point has polar angle a with respect to the center of c.

2.6 Conics

An ellipse is internally represented by its center, major axis, minor axis and direction of its major axis. The associated parametric representation in the coordinate system defined by its axis is :

$$\begin{cases} x = a\cos(t) \\ y = b\sin(t) \end{cases}$$

where a is the major axis, b is the minor axis and t ranges from $-\pi$ to π.

An hyperbola is internally represented by its center, real axis, imaginary axis and direction of its real axis. The associated parametric representation in the coordinate system defined by its axis is :

$$\begin{cases} x = \dfrac{a}{\sin(t)} \\ y = \dfrac{b}{\tan(t)} \end{cases}$$

where a is the real axis, b is the imaginary axis and t ranges from $-\pi$ to π except 0.

A parabola is represented by its focus, parameter and the direction of its axis. The associated parametric representation is :

$$\begin{cases} x = -\dfrac{p\cos(t)}{1+\cos(t)} \\ y = -\dfrac{p\sin(t)}{1+\cos(t)} \end{cases}$$

where p is the parameter and t ranges from $-\pi$ to π. The corresponding coordinate system uses the focus as origin and the axis of the parabola as ordinate axis.

Conic related functions

`ellipse(A, x, y, a)`
: Ellipse of center A, major axis x, minor axis y. The direction of the major axis is a.

`hyperbola(A, x, y, a)`
: Hyperbola of center A, real axis x, imaginary axis y. The direction of the real axis is a.

`parabola(A, x, a)`
: Parabola of summit A and parameter x. The direction of its axis is a.

`parabola(A, l)`
: Parabola of focus A and directrix l.

`conic(A, l, x)`
: Conic of focus A, directrix l and eccentricity x.

`conic(A, B, x)`
: Conic of foci A and B and eccentricity x.

`major(c)` Major axis of conic c if c is an ellipse, real axis if c is an hyperbola, parameter if c is a parabola.

`minor(c)` Minor axis of conic c if c is an ellipse, imaginary axis if c is an hyperbola, 0 if c is a parabola.

`eccentricity(c)`
: Eccentricity of conic c.

`arg(c)` Polar angle of the major axis of c if c is an ellipse, of the real axis if c is an hyperbola, of the axis if c is a parabola (in degrees).

`arg(A, c)` Argument of point A on conic c with respect to its parametric representation. If A is not on c this function uses the projection of A on c with respect to its center for centered conics or its focus for parabolas.

`point(c, a)`
: Point on conic c of argument a with respect to its parametric representation.

`center(c)`
: Center of centered conic c.

`foci(c)` Set containing the foci of conic c.

`line(c, a)`
Tangent line to conic c. The contact point has argument a with respect to the parametric representation of c.

3 Geometry

3.1 Transformations

Transformations consist in either generic transformations, which can be applied on any kind of geometric objects, or projections, which are only defined for points.

Generic transformations

`translation(o, u)`
: Translation of object o using vector u.

`reflection(o, l)`
: Reflection of object o with respect to line l.

`symmetric(o, A)`
: Symmetric (i.e. 180° rotation) of object o with respect to point A.

`rotation(o, A, a)`
: Rotation of object o with respect to point A, using angle a.

`homothecy(o, A, x)`
: Homothecy (i.e. reduction or dilation) of object o with respect to point A, using scale x.

Projections

`projection(A, l)`
: Orthogonal projection of point A on line l.

`projection(A, l, l')`
: Projection of point A on line l using direction of line l'.

3.2 Intersections

Intersection functions provide the points belonging to both of two given geometric objects. Sets are considered as open paths. For line-line intersections the function is point valued, otherwise it is set valued.

Intersection functions

`intersection(l, l')`
: Intersection of line l and line l'. Parallel lines cause an error.

`intersection(l, s)`
: Intersection of line l and set s.

`intersection(l, c)`
: Intersection of line l and circle or conic c.

`intersection(s, s')`
: Intersection of set s and set s'.

`intersection(c, c')`
: Intersection of circle c and circle c'.

`intersection(c, s)`
: Intersection of circle c and set s.

3.3 Triangles

Triangular assignments consist in a list of three variable names followed by either `triangle`, `right`, `isosceles` or `equilateral`, and some optional parameters.

Example: `A B C triangle 6, 5, 4, 0°`

In the following, we will use letter x for the first side of the triangle, i.e. the segment joining the first and the second vertices, y for the second side, i.e. the segment joining the second and the third vertices, and z for the third side. Likewise, we will use letter u for the first angle, v for the second angle, and a for the direction of the first side.

In a triangular assignment one or two vertices may be predefined, i.e. the first or the first and the second variables already contain points. If no vertices are predefined then the first variable is set to the origin of the implicit coordinate system. The direction of the first side is horizontal unless specified by an angular value at the end of the parameter list. If two vertices are predefined then the parameters giving the length and the orientation of the first side have to be omitted.

If no parameter is given, the length of the first side is set to 6.

Generic triangles

```
triangle { x { , a } }
triangle { x, } y, z { , a }
triangle { x, } u, v { , a }
triangle { x, } u, z { , a }
triangle { x, } z, v { , a }
```

The first assignment yields an optimal scalene triangle.

Right triangles

```
right { x { , a } }
right { x, } y { , a }
right { x, } u { , a }
```

With these assignments the resulting triangle has a right angle at its second vertex. The first assignment yields a right triangle with sides proportional to 4-3-5.

Isosceles triangles

```
isosceles { x { , a } }
isosceles { x, } y { , a }
isosceles { x, } u { , a }
```

With these assignments the resulting triangle is isosceles at its third vertex. The first assignment yields a golden triangle.

Equilateral triangles

```
equilateral { x { , a } }
```

Triangle related functions

`angle(A, B, C)`
: Degree measure of angle ABC.

`height(A, B, C)`
: Height of triangle ABC with respect to vertex A.

`orthocenter(A, B, C)`
: Orthocenter of triangle ABC.

`altitude(A, B, C)`
: Altitude of triangle ABC with respect to vertex A.

`bisector(A, B, C)`
: Bisector of angle ABC.

`median(A, B, C)`
: Median of triangle ABC with respect to vertex A.

`circle(A, B, C)`
: Circumcircle of triangle ABC.

`incircle(A, B, C)`
: Incircle of triangle ABC.

3.4 Quadrilaterals

Quadrilateral assignments consist in a list of four variable names followed by either `parallelogram`, `rectangle` or `square`, and some optional parameters.

Example: `A B C D parallelogram 6, 3, 60°`

In the following, we will use letter x for the first side of the quadrilateral, i.e. the segment joining the first and the second vertices, y for the fourth side, i.e. the segment joining the first and the fourth vertices, u and v the corresponding vectors. Likewise, we will use letter a for the first angle, and b for the angular direction of the first side.

In quadrilateral assignments, one, two or three vertices may be predefined. Like with triangular assignments, if no vertices are predefined then the first variable is set to the origin of the implicit coordinate system. The direction of the first side is horizontal unless specified by an angular value at the end of the parameter list. If two vertices are predefined then the parameters giving the length and the orientation of the first side have to be omitted. If three vertices are predefined then only `parallelogram` is valid.

Generic quadrilateral assignments

`parallelogram { x, } y, a { , b }`
`rectangle { x, } y, { , b }`
`square { x { , b } }`

Default quadrilateral assignments

Default assignments, i.e. assignments without parameters, are only valid with at most one predefined vertex.

`parallelogram`
: Assigns a parallelogram such as $x = 5$, $y = 4$ and $a = 75°$.

`rectangle`
: Assigns a golden rectangle such as $x = 6$.

`square` Assigns a square such as $x = 4$.

Vector based quadrilateral assignment

This assignment is only valid with at most one predefined vertex.

`parallelogram u, v, a`

3.5 Locus Assignments

A locus assignment is useful to generate a set of points giving an approximation of a locus. It consists in a locus statement followed by a block of instructions delimited by `end`. The instruction block must contain a put statement, i.e. the `put` keyword followed by a point-valued expression. The syntax for locus statements is:

```
locus l(t = a to b { step n })
```

The associated instruction block is repeated n times with values of number t increasing from a to b. Each put statement appends a point to the resulting set l. Default value for n is 120.

Examples

A cardioid.

```
locus C(t = 0 to 360)
  put point(sin(t/2), t°)
end
```

A quadratrix generated from the intersection of two uniformly-moving lines: one angular, one linear (original script by Robert D. Goulding).

```
locus q(t = 10^-3 to 1)
  l1 = line(point(0, 0), t*pi/2 rad)
  l2 = line(point(0, t), 0 rad)
  put intersection(l1, l2)
end
```

4 Drawing

4.1 Drawing Commands

A drawing command is either a single or a multiple drawing statement. A single drawing statement consists in the `draw` keyword followed by a drawable object and a possibly empty comma separated list of aspect parameters.

Example: `draw point(2, 3) red`

A multiple drawing statement consists in the `draw` keyword followed by a list of global aspect parameters and a list of drawable objects (with possibly local aspect parameters) delimited by `end`.

Example:

```
draw blue
  A.B
  C.D dashed, black
  E.F dotted
end
```

Local parameters override global ones.

Aspect parameters

Color `black` (default), `darkgray`, `gray`, `lightgray`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow`.

Scale factor
A number greater than 0 (default: 1).

Point shape
`dot` (default), `disc`, `box`, `plus`, `cross`.

Line style `full` (default), `dashed`, `dotted`.

Partition `entire` (default), `half`.

Direction `forth` (default), `back`.

Endings `none` (default), `arrow`, `arrows`.

Conics draw step
An angular parameter (default: 3°) setting the current drawing step (with respect of the parametric representation).

Font `font(`*s*`)` where *s* is a string containing the font description.

Font description

With `eukleides` the font description string has to follow the standard PostScript format, i.e. `"`*`name-face-size`*`"`.

Example: `"Helvetica-Bold-12"`

The default font is `"NewCenturySchlbk-Roman-10"`.

With `euktopst` the font description string should be a sequence of parameterless commands (without the leading backslash) appropriate to the TEX format in use, e.g. `"bf"` with plain TEX or `"sffamily\bfseries"` with LATEX.

Single drawing statements

`draw A` *list*
: Draws point A, where *list* may contain color, scale and shape parameters.

`draw v O` *list*
: Draws vector v from point O, where *list* may contain color, scale and style parameters. The scale factor determines the line thickness.

`draw s` *list*
: Draws the open path corresponding to set s, where *list* may contain color, scale, style, direction and endings parameters.

`draw (s)` *list*
: Draws the polygon corresponding to set s, where *list* may contain color, scale, style, direction and endings parameters.

`draw [s] {` *color* `}`
: Fills the polygon corresponding to set s.

`draw [s] a` *list*
: Hatches the polygon corresponding to set s, where *list* may contain color and scale parameters. The angular parameter a determines the direction of the hatches. The scale factor determines the spacing of the hatches (default: 1.5 mm).

`draw l` *list*
: Draws line l where *list* may contain color, scale, style, partition and direction parameters. Using `half` yields the ray having same origin and direction than line l, using both `half` and `back` yields the ray with reverse direction.

`draw c` *list*
: Draws circle c where *list* may contain color, scale and style parameters.

`draw c a b` *list*
: Draws arc of circle c from polar angle a to polar angle b (with respect to the center of c) where *list* may contain color, scale, style, direction and endings parameters.

`draw [c] {` *color* `}`
: Fills circle c.

`draw [c] a` *list*
: Hatches circle c where *list* may contain color and scale parameters. The angular parameter a determines the direction of the hatches. The scale factor determines the spacing of the hatches (default: 1.5 mm).

`draw c` *list*
: Draws conic c where *list* may contain color, scale, style and step parameters.

`draw c a b` *list*
: Draws conic c from argument a to argument b (with respect to the parametric representation of c) where *list* may contain color, scale, style, direction, endings and step parameters.

`draw l A a` *list*
: Writes string l with respect to point A in direction a, where *list* may contain color, scale and font parameters. The scale factor determines the distance from the center of the string to the point.

`draw l s a` *list*
: Writes string l with respect to the middle of segment s in direction a, where *list* may contain color, scale and font parameters. The scale factor determines the distance from the center of the string to the middle of the segment (default: 3 mm).

Output setting commands

`scale z` Sets the value of the length unit. (Default: 1 cm.)

`box x, y, x', y' { , z }`
`frame x, y, x', y' { , z }`
Sets the coordinates of the lower left and upper right corner of the drawing frame, with optional scale factor z. Default coordinates are $(-2,-2)$ and $(8,6)$. With `eukleides` and `euktopst`, both commands yield the same result.

4.2 Label Commands

Label commands are useful to mark segments or angles with usual symbols, or to write point names. As for drawing commands, a label command is either a single or a multiple statement.

Example: `label B.C double`

Example:

```
label blue
  A.B
  B.C cross, black
  E.F double
end
```

Local parameters override global ones.

Label parameters

Segment mark shape
`simple` (default), `double`, `triple`, `cross`.

Angle mark shape
`simple` (default), `double`, `triple`, `right`, `forth`, `back`.

Angle mark decoration
`none` (default), `dotted`, `dashed`.

Color `black` (default), `darkgray`, `gray`, `lightgray`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow`.

Scale factor
A number greater than 0 (default: 1).

Font `font(s)` where s is a string containing the font description.

Single label statements

`label s` *list*
Marks segment s where *list* may contain shape, color and scale parameters.

`label B, A, C` *list*
Marks angle BAC (i.e. angle from ray AB to ray AC in anticlockwise direction) where *list* may contain shape, decoration, color and scale parameters. The `forth` and `back` shape yield arrowed angle marks (respectively in direct and reverse direction). The `dotted` decoration adds a dot inside the angle mark. The `dashed` decoration adds a tick on the angle mark.

`label P a` *list*
Assuming variable P contains a point, writes the name of P in direction a from P, where *list* may contain color, scale and font parameters. The scale factor determines the distance from the center of the label to the point (default: 3 mm).

5 Programming

5.1 Input and Output

Input commands

`read s` Opens the file whose name is string *s* for reading.

`close` Closes the previously opened data file.

Standard input is used unless a data file is specified at command line invocation or open for reading.

Input functions

`number(s)`
: Reads a number from current input.

`string(s)`
: Reads a string from current input.

String *s* is used as a prompt.

Example:

```
l = number("Length? ")
w = number("Width? ")
draw rectangle(l, w)
```

When running in batch mode with no defined data file, the `number` function returns 0 and the `string` function returns the empty string.

Output commands

Output commands take a comma separated list of printable objects as arguments. Printable objects are strings, numbers, points and sets. They are formatted the same way as for conversion to string.

`write s` Opens the file whose name is string *s* for writing. If the file exists, its content will be overwritten. Otherwise, it is created.

`append s` Opens the file whose name is string *s* for appending. The file is created if it does not exist. Writing starts at the end of the file.

`release` Closes previously opened result file.

`print` *`list`*
: Writes *list* to current result file, or to standard output if none is open.

`error` *`list`*
: Writes *list* to standard error stream.

`output` *`list`*
: Writes *list* to output file. This may be useful to include low level PostScript or PSTricks command invocations.

5.2 Conditional Statements

Syntax for conditional statement is:

```
if assertion block { { elseif  assertion block } ... else block } end
```

Example:

```
if x < 0
  print "x is negative."
elseif x > 0
  print "x is positive."
else
  print "x is null."
end
```

Using the `stop` command within a conditional statement allows to abort the execution of a script when some condition is met.

Boolean operators

`not a` Negation of assertion a.

`a and b` Disjunction of assertions a and b.

`a or b` Conjunction of assertions a and b.

Boolean constants

`true`

`false`

Comparison operators

`a == b` Checks whether objects a and b are equal, i.e. have the same internal representation. Objects may be numbers, strings, points, vectors, sets, lines, circles or conics.

`a != b` Negation of the former.

`x < y`
`x <= y`
`x > y`
`x >= y` Comparison of numbers x and y.

Set assertions

`A in s` Checks whether point A belongs to set s.

`empty(s)` Checks for emptiness of set s.

Geometric assertions

`A on s` Checks whether point A is on object s, where s may be a set (considered as an open path), a line, a circle or a conic.

`collinear(A, B, C)`
Checks whether points A, B and C are collinear.

`collinear(u, v)`
Checks whether vectors u and v are collinear, i.e. have same or opposite directions.

`parallel(l, l')`
`perpendicular(l, l')`
Direction comparison of lines l and l'.

`ellipse(c)`
`hyperbola(c)`
`parabola(c)`
: Type checking of conic c.

`isosceles(A, B, C)`
`equilateral(A, B, C)`
`right(A, B, C)`
: Type checking of triangle ABC.

`parallelogram(A, B, C, D)`
`rectangle(A, B, C, D)`
`square(A, B, C, D)`
: Type checking of quadrilateral $ABCD$.

Output format flags

`eps` True with `eukleides`.

`pstricks` True with `euktopst`.

`display` True within GUI.

Ternary operator

assertion `? x | y`
: Number x if *assertion* is true, number y otherwise.

5.3 Iterative Statements

`while` *assertion block* `end`
: Repeats *block* while *assertion* is true.

`for i = a to b { step c }` *block* `end`
: Repeats *block* while incrementing number i by c (default: 1), from a to b. Numbers b and c are evaluated at each step. Iteration ends as soon as i is greater than or smaller than b, depending on the sign of c.

`for P in s` *block* `end`
: Repeats *block* while point P runs through set s.

Example:

```
O = point(3,2)
H = hexagon(O, 3, 0°)
draw (H)

for P in H
  draw O.P dotted
end
```

5.4 Function Definitions

A command or function definition consists in a header followed by a block of instructions delimited by `end`.

A command definition header consists in a command name followed by a comma separated parameter declaration list enclosed in parenthesis. A parameter declaration consists in a parameter type: `number`, `point`, `vector`, `set`, `line`, `circle`, `conic` or `string`, followed by a parameter

name. A function definition header is identical to a command definition header except that it starts with a return type.

A function definition must contain at least a return statement, which consists in the `return` keyword followed by an appropriate expression. Command definitions may contain empty return statements.

Example:

```
number square(number x)
  return x*x
end
```

Variables may be declared local to a command or function definition using the `local` keyword.

Example:

```
vector conjugate(vector v)
  local x, y
  x = abscissa(v)
  y = ordinate(v)
  return vector(x, -y)
end
```

5.5 Modules

Eukleides scripts may be modular using include directives. An include directive consists in a line starting with the at sign (`@`) followed without white space by the path to the file (either absolute or relative). To avoid circular references, only 20 levels of inclusion are allowed.

5.6 Interactive Variables

The GUI for the first version of the Eukleides language, named `xeukleides`, allowed to interactively modify the value of numeric variables using the keyboard. The planned GUI for the second version will have similar features.

Mobile points

A mobile point is a point-valued variable which is bound to a state (A to Z) and to the keyboard arrows. To switch to a given state, the user has to hit the corresponding key.

Syntax for a mobile point declaration is:

`mobile` *var* `{ (`*state*`) }` `=` *point*
`mobile` *var* `( { `*state*`, } z)` `=` *point*
`mobile` *var* `( { `*state*`, } x, y, x', y' { , z } )` `=` *point*

unless *var* already contains a point, in which case the initialisation part has to be omitted.

A state identifier consists in the number sign (`#`) followed by the corresponding uppercase letter. When omitted, the variable is bound to the state corresponding to the first letter of its name. Number z is the amount by which the appropriate coordinate is incremented or decremented for each stroke on the arrow keys (default: 0.1). Numbers x and x' are the minimal and maximal abscissa, numbers y and y' are the minimal and maximal ordinate. By default there are no boundaries.

Example: `mobile P_0(#A, 0.2)`

Interactive numeric variables

An interactive numeric variable is bound to a state (A to Z) and to either the horizontal or the vertical keyboard arrows.

Syntax for interactive variable declaration is:

`horizontal` *variable* `(`*state* `{ , x, y } { , z }) = t`
`vertical` *variable* `(`*state* `{ , x, y } { , z }) = t`

Numbers x and y are the lower and upper bound. By default there are no boundaries. Number z is the amount by which the variable is incremented or decremented (default: 0.1).

Example: `horizontal x(#A, -1, 1)`

Initialisation directives

An initialisation directive is equivalent to keystrokes prior to execution. It consists in a single line starting with a number sign followed by a space separated list of pairs of letters and integers. Lowercase letters correspond to horizontal arrows, uppercase to vertical. Negative integers correspond to left or down directions, positive to right or up.

Example: `# a 5 A 10 b -4`

This initialisation directive is equivalent to 5 strokes on the right arrow key and 10 strokes on the up arrow key in A state and 4 strokes on the left arrow key in B state.

An initialisation directive may also be given as a command-line parameter (using the `--interactive` option. If several initialisation directives are present, only the last one is taken into account.

Animation

With `eukleides`, the `--animate` option allows to generate PostScript files containing several pages which may be converted into animated GIFs (using ImageMagik's `convert` for instance). The parameter for this option consists in a single letter followed by an integer number (without space), with the same meaning as for initialisation directives.

Example: `eukleides --animate=A20 figure.euk`

The former command yields a 20 pages PostScript file, each step corresponding to a stroke on the up arrow key.

6 Usage

6.1 Invocation

Command-line invocation for `eukleides`, `euktopst`, or `euktoeps` is:

program { *option* ... } *file_name*

The name of the generated output is formed of the base name of the input file and the `.eps` or `.ps` suffix for `eukleides`, the `.pst` suffix for `euktopst`, or the `.eps` suffix for `euktopst`. The base name is the file name deprived of the `.euk` suffix. If the `.euk` suffix is not present, the base name is the file name itself.

Common options

`-l, --locale{=`*lang*`}`
: Use keywords localized in language *lang*. With no argument given, the current locale is set to the value of the LANG environment variable. This feature may be disabled.

`-#, --interactive=`*string*
: Modify interactive variables.

`-v, --version`
: Print version number and exit.

`-h, --help`
: Print a short help and exit.

Options specific to `eukleides` and `euktopst`

`-o, --output{=`*output_file*`}`
: Set an output file name. With no argument given, the output stream is set to standard output.

`-b, --batchmode{=`*data_file*`}`
: Don't stop for input. If given, use *data_file* instead of standard input.

Option specific to `eukleides`

`-a, --animate=`*string*
: Animate interactive script.

Options specific to `euktoeps`

`-i, --include=`*string*
: Include LaTeX directives in preamble.

`-d, --data=`*data_file*
: Use specific data.

6.2 TeX

The `eukleides` package that comes with the Eukleides distribution provides a convenient way to include geometric figures in LaTeX documents.

Eukleides figures must be enclosed in `eukleides` environments. When first running `latex` or `pdflatex`, the enclosed codes are saved to individual files, named after the TeX source. For instance, *mydoc.tex* would yield *mydoc-fig1.pst*, *mydoc-fig2.pst*, etc.

When using `latex`, the figure files should then be transformed in TEX files with the `euktotex` script (which takes the TEX source name as argument), and `latex` run again. It should be noted that the PSTricks package is automatically loaded if necessary.

When using `pdflatex`, due to incompatibilities with PSTricks, the figure files should be transformed in PDF pictures with the `euktopdf` script (which likewise takes the TEX source name as argument), and `pdflatex` run again. If specific packages are required to typeset the figures, the corresponding `\usepackage` directives have to be enclosed in a `packages` environment in the preamble. This has no effect when using `latex`.

6.3 Localized Keywords

Localized versions of the keywords are available (currently German and French). The `--locale` option allows to specify the desired locale. With no argument given, the current locale is set to the value of the LANG environment variable. Otherwise the argument has to be a valid locale identifier, e.g. `fr_FR` or `fr_FR.utf8`, depending on the default charmap. This feature may be disabled at build time.

Conversion tables for each language are provided with the Eukleides distribution.

Command Index

N

O

P

R

S

T

V

W

Y

Concept Index