How to TikZ?

An overview

Jan-Philipp Kappmeier

Technische Universität Berlin

21. 02. 2013

A Picture is Worth a Thousand Words

Pictures and figures improve papers and slides for talks.


One picture on each slide

Possible sources:

- pixel graphics from the Internet
- vector graphics manually created using
  - Inkscape
  - xfig
  - TikZ
  - ...

TikZ

TikZ ist *kein* Zeichenprogramm

- A language to create vector graphics
- Two components:
  1. TikZ
     easy to use high-level LaTeX commands
  2. PGF (Portable Graphics Format)
     low-level commands

Pros

- Optics fit to document
- Graphics can contain LaTeX
- Perfect integration to Beamer
- Export from a lot of tools

Cons

- No WYSIWYG
- Slow?

For Starters

How to add a TikZ picture to your document?

1. include the package tikz.
2. include additional TikZ libraries, if necessary
3. write TikZ code
   - graphics are described within `tikzpicture` environment
   - put into a picture environment to add caption, reference etc.
   - inline-TikZ: use the `\tikz` command to create inline graphics like this nice 5-wheel  here.

Example

It is easy to draw a thistle .

```
\tikz{ \filldraw[color=Thistle] circle (0.5ex); }
```

Drawing on Paths

Generel principle

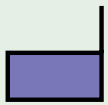
- central syntactic element is a *path*
- a sequence of coordinates and drawing commands
- General syntax:

  `\draw[options] (coordinate) command (coordinate) ...;`

  like moving a pencil to some place and start drawing something.
- different types of paths are started with ...
  - `\draw` – draws lines and shapes
  - `\fill` – fills interior shapes
  - `\filldraw` – draws exterior and interior of shapes
  - `\node` – places a node (containing text) somewhere
  - `\coordinate` – places an invisible, named coordinate somewhere

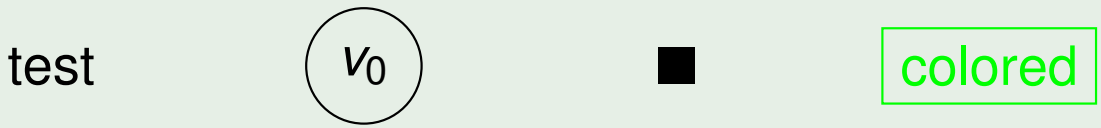
A First Example

Draw geometric forms



```
\filldraw[fill=Periwinkle,thick] (1,0) rectangle +(2,1) -- (3,2);
\draw (0,0) -- (0,2) -- (1,3.25){[rounded corners] -- (2,2)
  -- (2,0)} -- (0,2) -- (2,2) -- (0,0) -- (2,0);
```

Drawing graphs




```
\draw node at (0,0) {test};
\node[draw,circle] at (2,0) {$v_0$};
\node[fill] at (4,0) {};
\node[draw,color=red] at (6,0) [green] {colored};
```

Drawing the 5 Wheel

We are ready to pimp our slides with the COGA-5-Wheel!

Drawing the 5 Wheel

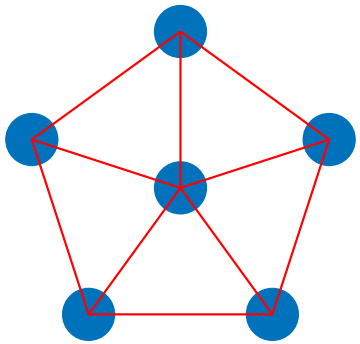
We are ready to pimp our slides with the COGA-5-Wheel!

```
\node[fill,circle,draw,RoyalBlue] at (0,1) {};
\node[fill,circle,draw,RoyalBlue] at (-0.9511,0.3091) {};
\node[fill,circle,draw,RoyalBlue] at (-0.5878,-0.8091) {};
\node[fill,circle,draw,RoyalBlue] at (0.5878,-0.8091) {};
\node[fill,circle,draw,RoyalBlue] at (0.9511,0.3091) {};
\node[fill,circle,draw,RoyalBlue] at (0,0) {};
\draw[red] (0,1) to (-0.9511,0.3091) to (-0.5878,-0.8091)
    to (0.5878,-0.8091) to (0.9511,0.3091) to (0,1);
\draw[red] (0,0) to (0,1) (0,0) to (-0.9511,0.3091) (0,0)
    to (-0.5878,-0.8091) (0,0) to (0.5878,-0.8091) (0,0)
    to (0.9511,0.3091);
```

Notice, here is some serious math going on!

Drawing the 5 Wheel

We are ready to pimp our slides with the COGA-5-Wheel!



Unfortunately we have do to a lot of computation.

➡ Can be done by PGF!

Also, lines are drawn over the nodes.

➡ Solve this using named nodes.

Drawing the 5 Wheel with Less Math

Computations using PGF

- `\pgfmathsetmacro{\x}{computation}`
  Creates a variable `\x` with the result of the computation
- `\pgfmathparse{computation}`
  Stores the result in the variable `\pgfmathresult`

Drawing the 5 Wheel with Less Math

Computations using PGF

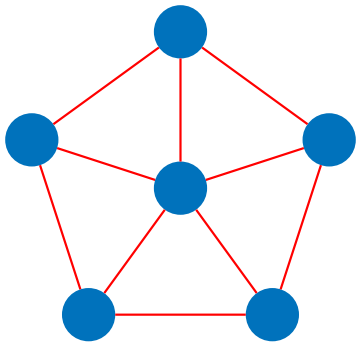
- `\pgfmathsetmacro{\x}{computation}`
  Creates a variable `\x` with the result of the computation
- `\pgfmathparse{computation}`
  Stores the result in the variable `\pgfmathresult`

```
\pgfmathsetmacro{\xa}{cos(90)}
\pgfmathsetmacro{\ya}{sin(90)}
\pgfmathsetmacro{\xb}{cos(90+72)}
\pgfmathsetmacro{\yb}{sin(90+72)}
...
\node[fill,circle,draw,RoyalBlue] (1) at (\xa,\ya) {};
\node[fill,circle,draw,RoyalBlue] (2) at (\xb,\yb) {};
...
\draw[red] (1) to (2) to (3) to (4) to (5) to (1);
\draw[red] (0) to (1) (0) to (2) (0) to (3) (0) to (4) ...
```

Drawing the 5 Wheel with Less Math

Computations using PGF

- `\pgfmathsetmacro{\x}{computation}`
  Creates a variable `\x` with the result of the computation
- `\pgfmathparse{computation}`
  Stores the result in the variable `\pgfmathresult`



Still we need to know a lot about how to compute coordinates on a circle

Polar Coordinates

Polar coordinates

- all coordinates can be defined via polar coordinates
- needed only an angle and the radius (distnace from the origin)
- expressed as `(angle:radius)`

Polar Coordinates

Polar coordinates

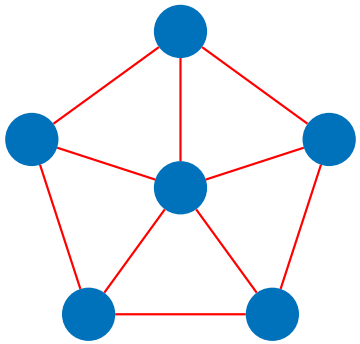
- all coordinates can be defined via polar coordinates
- needed only an angle and the radius (distnace from the origin)
- expressed as `(angle:radius)`

```
\node[fill,circle,draw,RoyalBlue] (1) at (90+0*72:1) {};
\node[fill,circle,draw,RoyalBlue] (2) at (90+1*72:1) {};
\node[fill,circle,draw,RoyalBlue] (3) at (90+2*72:1) {};
\node[fill,circle,draw,RoyalBlue] (4) at (90+3*72:1) {};
\node[fill,circle,draw,RoyalBlue] (5) at (90+4*72:1) {};
\node[fill,circle,draw,RoyalBlue] (0) at (0,0) {};
\draw[red] (1) to (2) to (3) to (4) to (5) to (1);
\draw[red] (0) to (1) (0) to (2) (0) to (3) (0) to (4) ...
```

Polar Coordinates

Polar coordinates

- all coordinates can be defined via polar coordinates
- needed only an angle and the radius (distnace from the origin)
- expressed as `(angle:radius)`



Using Styles to Clean Up Code

- using a lot of parameters creates ugly code
- user defined styles help keeping code clean
- style needs to be changed once only

Using Styles to Clean Up Code

- using a lot of parameters creates ugly code
- user defined styles help keeping code clean
- style needs to be changed once only

Different Types of Styles

1. local styles for one `tikzpicture` environment

```
\begin{tikzpicture}
[stylename/.style={some commands},another/.style=... ]
...
```

Using Styles to Clean Up Code

- using a lot of parameters creates ugly code
- user defined styles help keeping code clean
- style needs to be changed once only

Different Types of Styles

1. local styles for one `tikzpicture` environment
2. global styles for a document

```
\tikzstyle{source}=[draw, circle, fill=green]
```

Using Styles to Clean Up Code

- using a lot of parameters creates ugly code
- user defined styles help keeping code clean
- style needs to be changed once only

Different Types of Styles

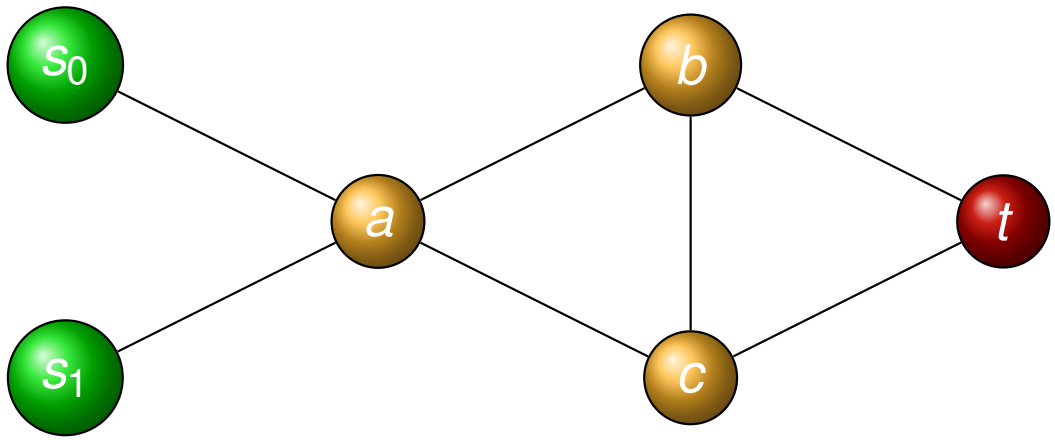
1. local styles for one `tikzpicture` environment
2. global styles for a document
3. styles for element types

```
\begin{tikzpicture}
[every node/.style={fill,circle,inner sep=2}]
...
```

Styles Example

Example

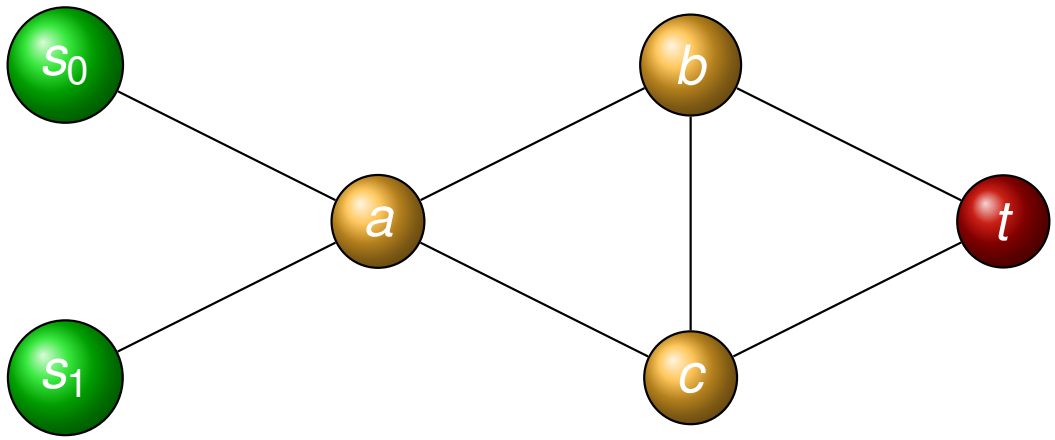
```
[default/.style={draw,fill,circle,shading=ball,
  ball color=Dandelion,text=white},
source/.style={draw,fill,circle,shading=ball,
  ball color=ForestGreen,text=white},
sink/.style={draw,fill,circle,shading=ball,
  ball color=BrickRed,text=white}]
```



Styles Example

Example

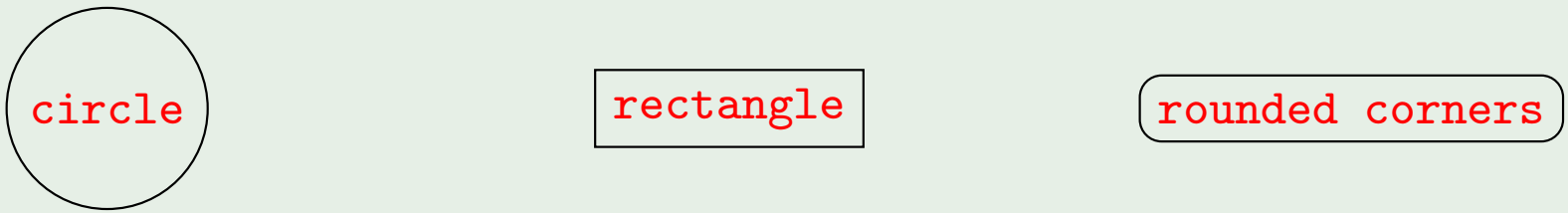
```
\node[source] (1) at (0,1) {$s_0$};
\node[source] (2) at (0,-1) {$s_1$};
\node[default] (3) at (2,0) {$a$};
\node[default] (4) at (4,1) {$b$};
\node[default] (5) at (4,-1) {$c$};
\node[sink] (6) at (6,0) {$t$};
```



Nodes

Parameter

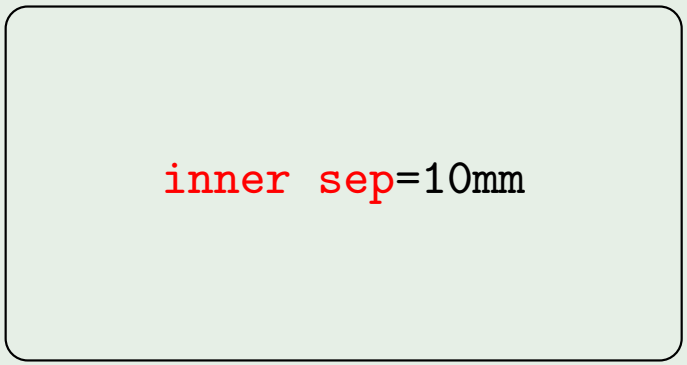
- `circle`, `rounded corners` `rectangle` – the shape

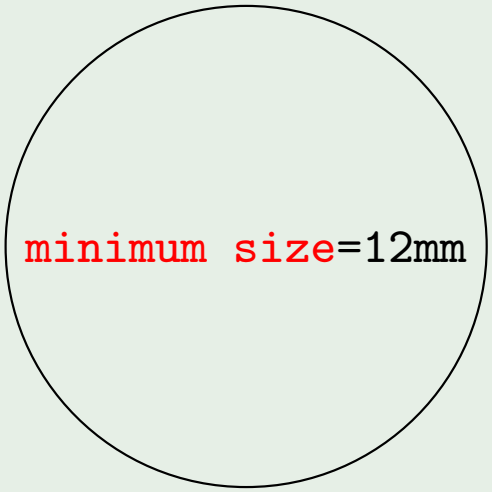


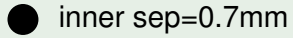
Nodes

Parameter

- `circle`, `rounded corners` `rectangle` – the shape
- `inner sep`, `minimum size` – defining the size







Nodes cont.

- multiline nodes: allows to have several lines of text within one node

```
\node[align=center] {Line 1 \\ Another line};
```

Line 1
Another line

- node labels: can add a label to all corners of the node

```
\node at (0,0) [label=below left:\tiny{$note$}] {A};
```

A
note

- anchors: defines the corner of the node that lies at the specifies position. Default is `center`

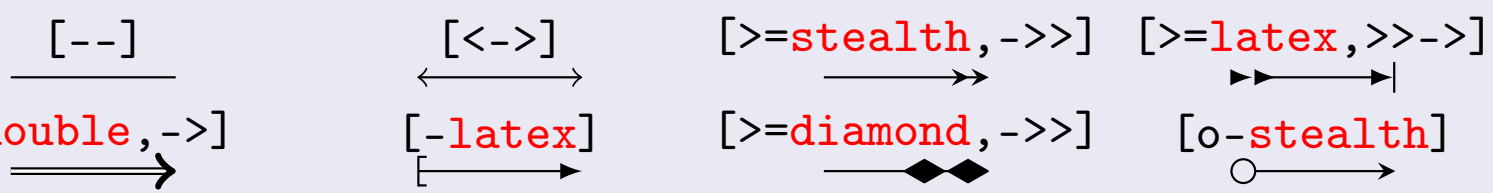
```
\node[draw,anchor=north west] at (0,0) {NW};
\node[draw,anchor=south east] at (0,0) {SE};
```

SE
NW

Arcs

Arc styles

```
\draw[-] (a) to (b);
```



Named Arcs

```
\draw (a) to node[sloped,above] {on top} (a)
```

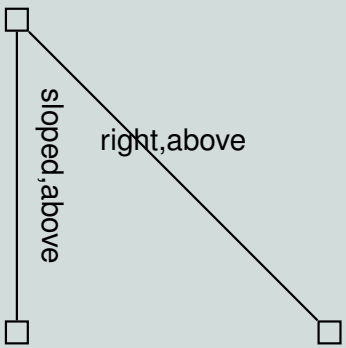
- position of arc label:

  above, below, right, left

- should label be aligned to slope of arc?

  sloped

  Bug: slope is wrong if scaling is used!



Arcs cont.

Bended arcs

- bending arcs

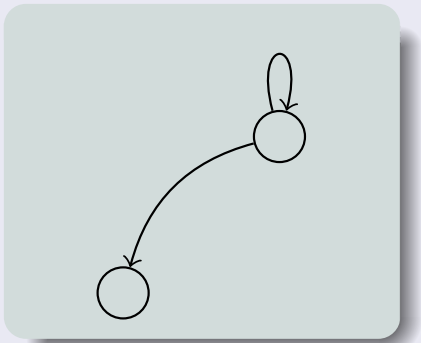
  `[bend left]`, `[bend right]`

- specify angle (at each node!)

  `[bend left=14]`

- loops

  `[loop above]`

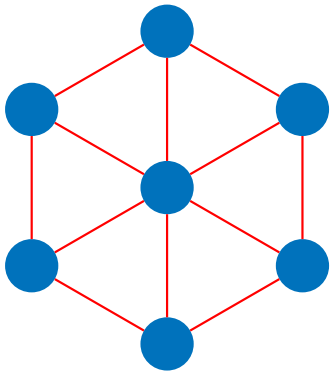


Additional Styles

- `[dashed]` – – – – –
- `[thick]` ———
- `[very thick]` ———

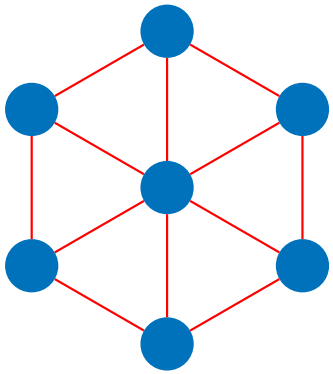
Loops

What happens if we surprisingly should draw a 6 wheel?

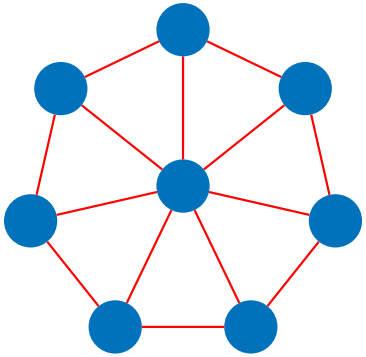


Loops

What happens if we surprisingly should draw a 6 wheel?

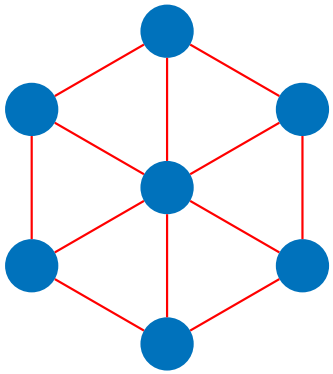


Or even to a 7 wheel?

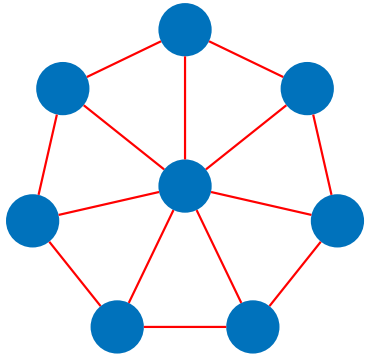


Loops

What happens if we surprisingly should draw a 6 wheel?



Or even to a 7 wheel?



The foreach command

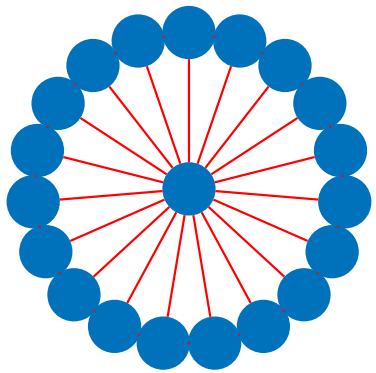
- executes the same commands for all items of a given set
- assigns the value to a variable
- `\foreach \var in \{ item1, item2, ..., itemN\} { }`

Putting Things Together

```
\pgfmathsetmacro{\n}{5}
\pgfmathtruncatemacro{\nodes}{\n-1}
\node[fill,circle,draw,RoyalBlue] (c) at (0,0) {};
\foreach \i in {0,...,\nodes}
  \node[fill,circle,draw,RoyalBlue] (\i) at (90+\i*360/\n:1)
\foreach \i in {0,...,\nodes} {
  \draw[red] (c) to (\i);
  \pgfmathtruncatemacro{\j}{mod(round(1+\i),\n)}
  \draw[red] (\i) -- (\j);
}
```

Putting Things Together

```
\pgfmathsetmacro{\n}{5}
\pgfmathtruncatemacro{\nodes}{\n-1}
\node[fill,circle,draw,RoyalBlue] (c) at (0,0) {};
\foreach \i in {0,...,\nodes}
  \node[fill,circle,draw,RoyalBlue] (\i) at (90+\i*360/\n:1)
\foreach \i in {0,...,\nodes} {
  \draw[red] (c) to (\i);
  \pgfmathtruncatemacro{\j}{mod(round(1+\i),\n)}
  \draw[red] (\i) -- (\j);
}
```



Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.
Give it a first try:

```
\draw[->] (0,0) to (8,0);
\foreach \x in {0, 1, 1.57, 3.14, 2.71} {
  \draw (\x,0.1) to (\x,-0.1);
  \node at (\x, -0.3) {\footnotesize{\x}};
```

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.
Give it a first try:

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.
Better:

Loops cont.

\foreach with more variables

- The `\foreach` command can iterate over tuples
- values are assigned to several variables

Example

Want to highlight specific numbers on the real line.

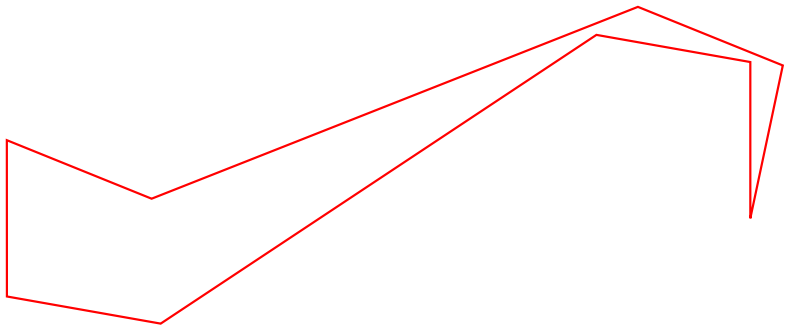
```
\draw[->] (0,0) to (4,0);
\foreach \x / \txt in
{0, 1, 1.57 / $\frac{\pi}{2}$, 3.14 / $\pi$, 2.71 / $e$} {
  \draw (\x,0.1) to (\x,-0.1);
  \node at (\x, -0.3) {\footnotesize{\txt}};
```

Calculate Coordinates

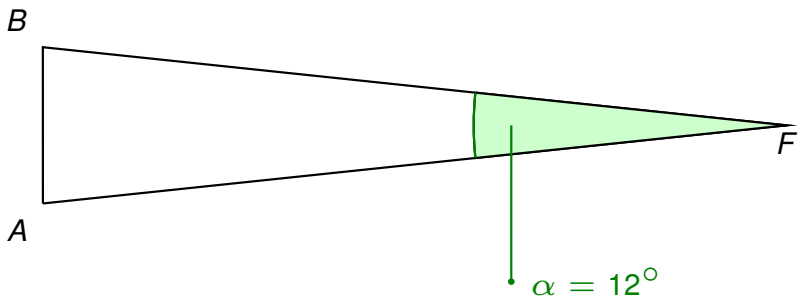
Coordinates

- can be defined using `coordinate`
- like nodes with empty text
- coordinates can be computed (like vector math)
- need to add the package calc (`\includetikzpackage{calc}`)

An Example Using Coordinate Calculation – Background



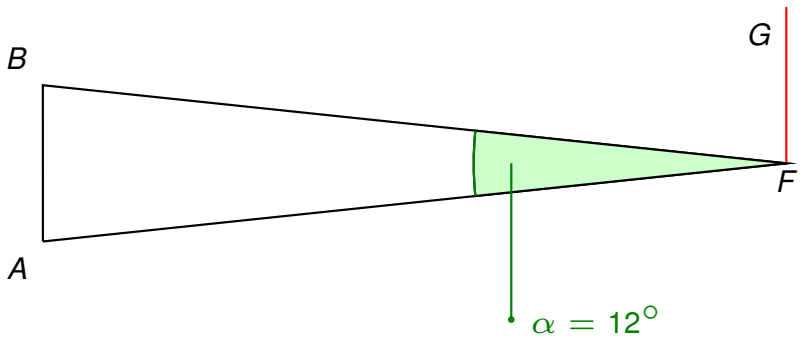
An Example Using Coordinate Calculation – Background



Construction

1. start with an isosceles triangle with base of length *a*

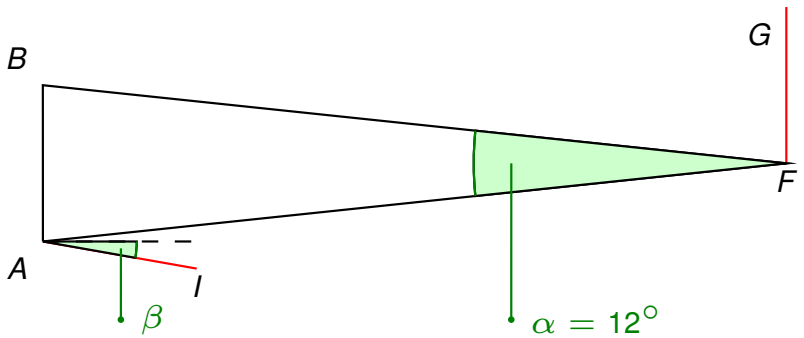
An Example Using Coordinate Calculation – Background



Construction

1. start with an isosceles triangle with base of length a
2. draw a copy of $\overline{AB}$ at F

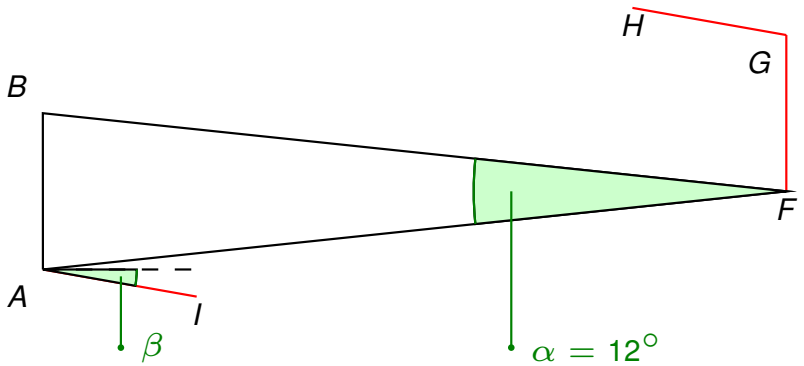
An Example Using Coordinate Calculation – Background



Construction

1. start with an isosceles triangle with base of length a
2. draw a copy of $\overline{AB}$ at F
3. draw segment $\overline{AI}$ of length b with some angle β

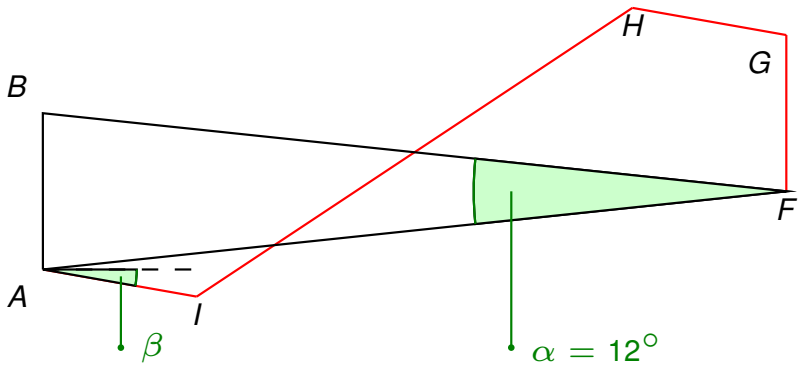
An Example Using Coordinate Calculation – Background



Construction

1. start with an isosceles triangle with base of length *a*
2. draw a copy of $\overline{AB}$ at *F*
3. draw segment $\overline{AI}$ of length *b* with some angle β
4. and copy it to *G*

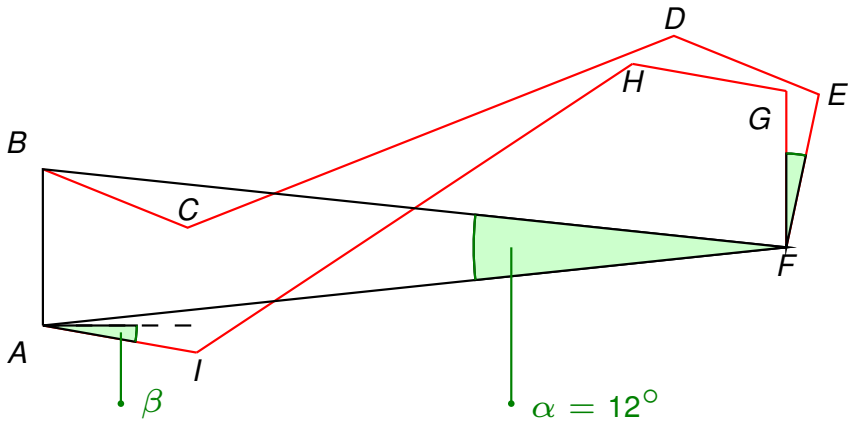
An Example Using Coordinate Calculation – Background



Construction

1. start with an isosceles triangle with base of length *a*
2. draw a copy of $\overline{AB}$ at *F*
3. draw segment $\overline{AI}$ of length *b* with some angle β
4. and copy it to *G*
5. connect *H* and *I*

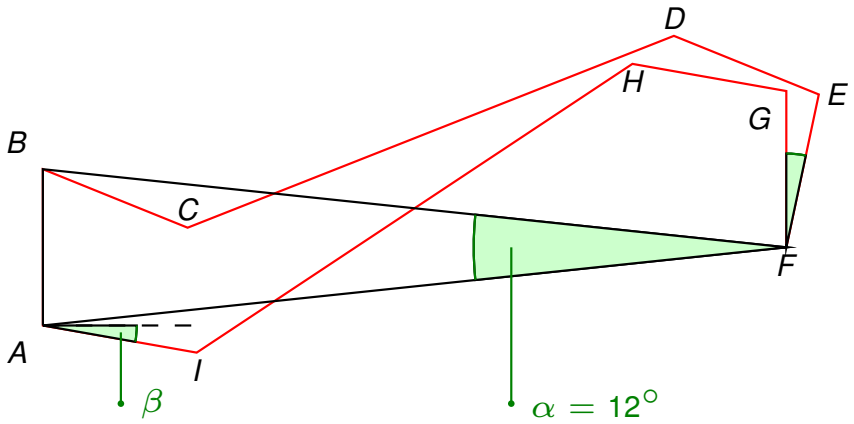
An Example Using Coordinate Calculation – Background



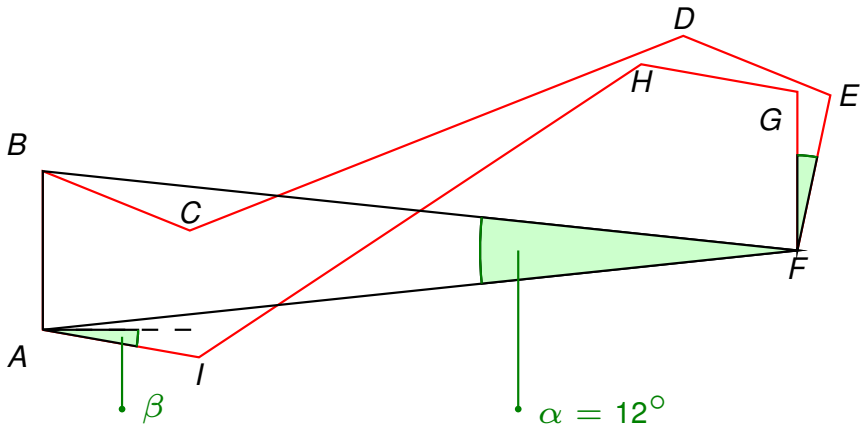
Construction

1. start with an isosceles triangle with base of length a
2. draw a copy of $\overline{AB}$ at F
3. draw segment $\overline{AI}$ of length b with some angle β
4. and copy it to G
5. connect H and I
6. rotate the polygonal path $\overline{FGHIA}$ around A by 12°

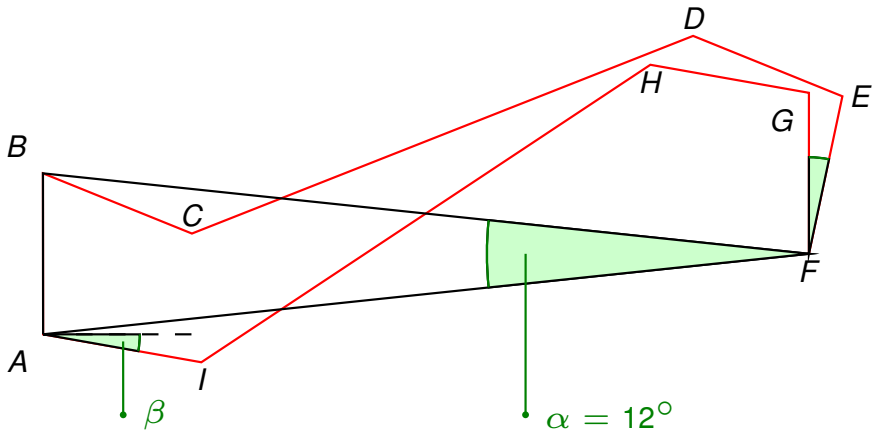
An Example Using Coordinate Calculation – Ti*k*Z realization



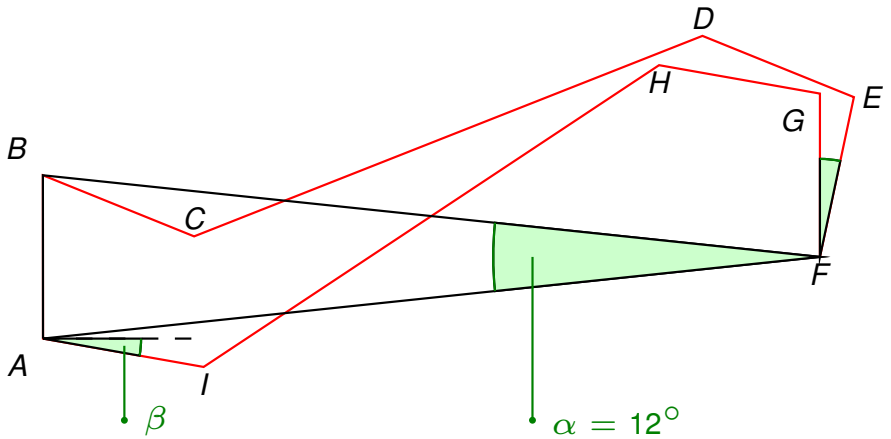
An Example Using Coordinate Calculation – Ti*k*Z realization



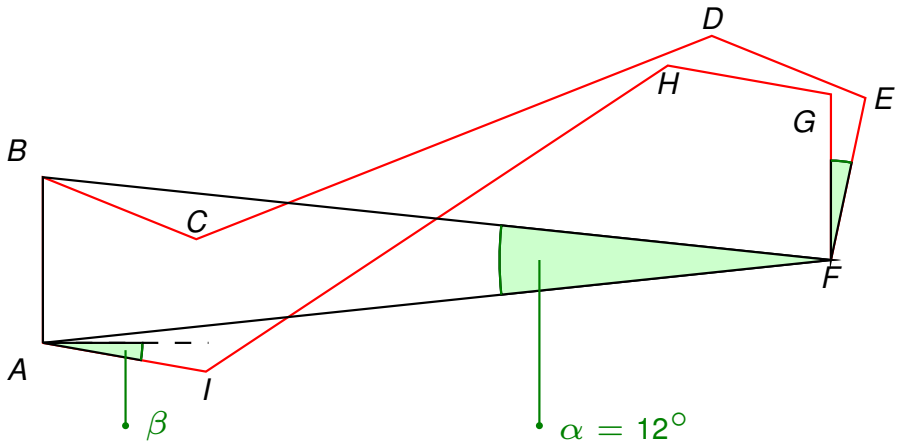
An Example Using Coordinate Calculation – Ti*k*Z realization



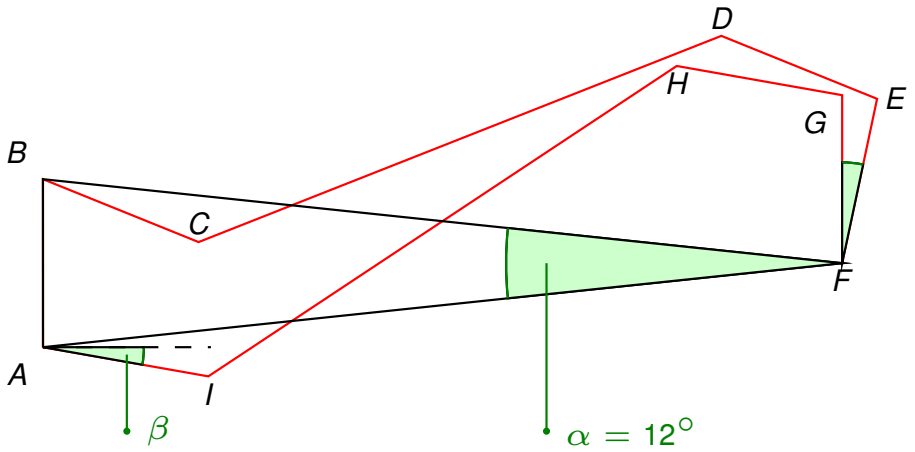
An Example Using Coordinate Calculation – Ti*k*Z realization



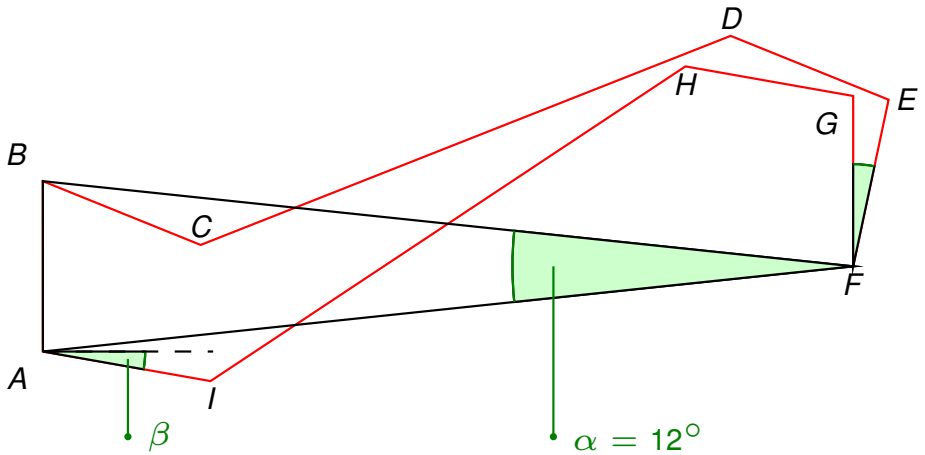
An Example Using Coordinate Calculation – Ti*k*Z realization



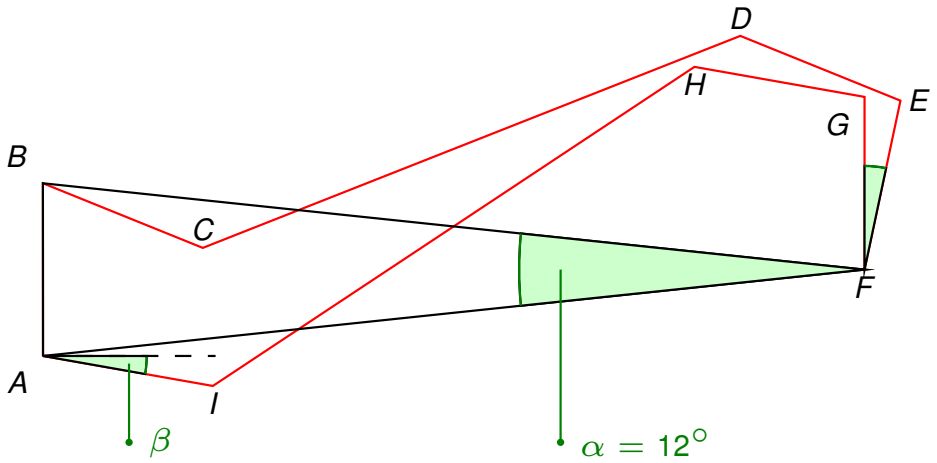
An Example Using Coordinate Calculation – Ti*k*Z realization



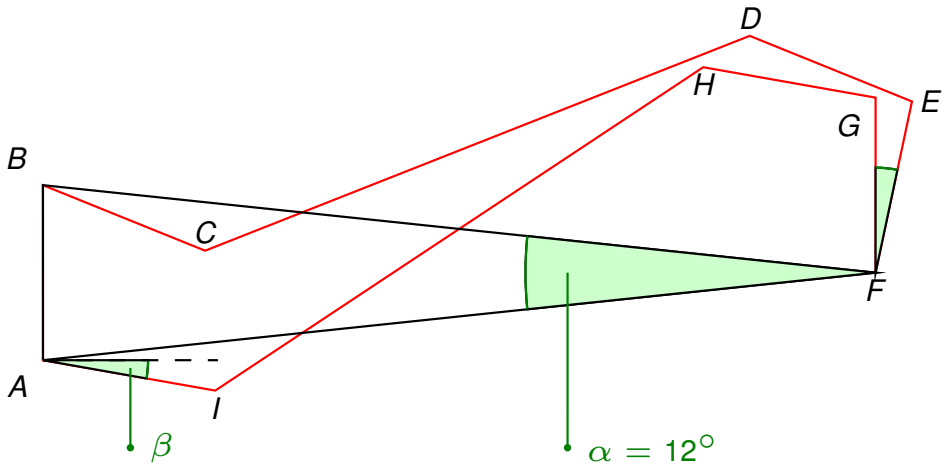
An Example Using Coordinate Calculation – Ti*k*Z realization



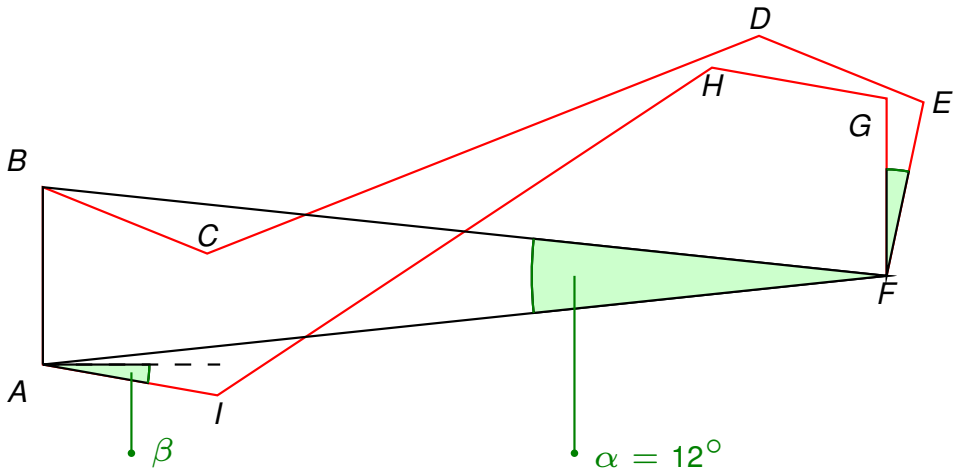
An Example Using Coordinate Calculation – Ti*k*Z realization



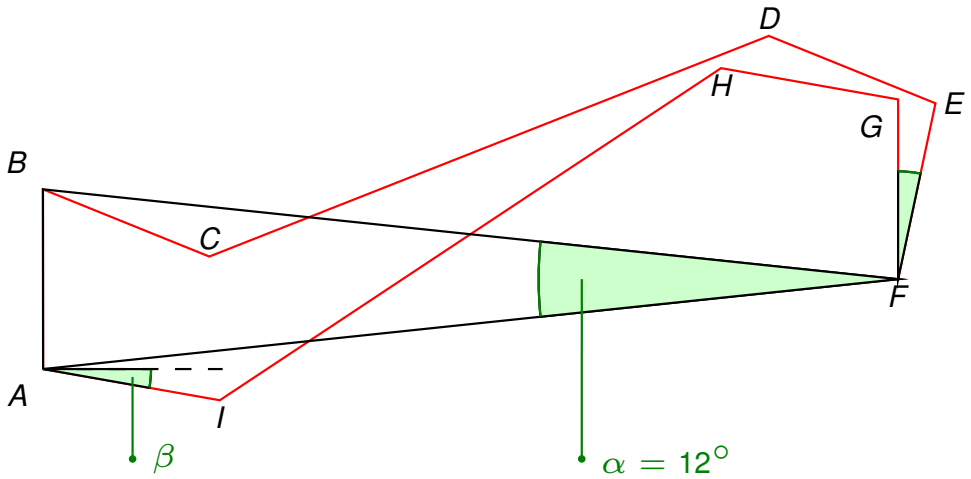
An Example Using Coordinate Calculation – Ti*k*Z realization



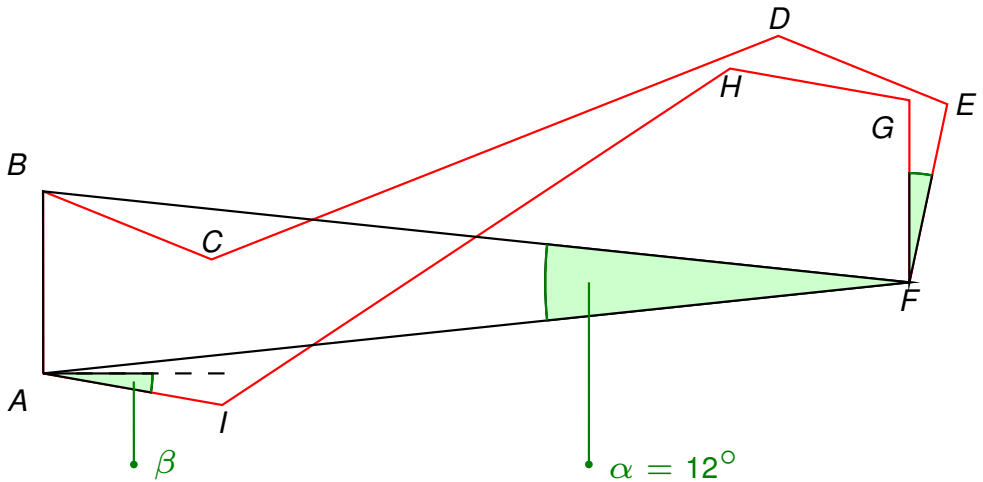
An Example Using Coordinate Calculation – Ti*k*Z realization



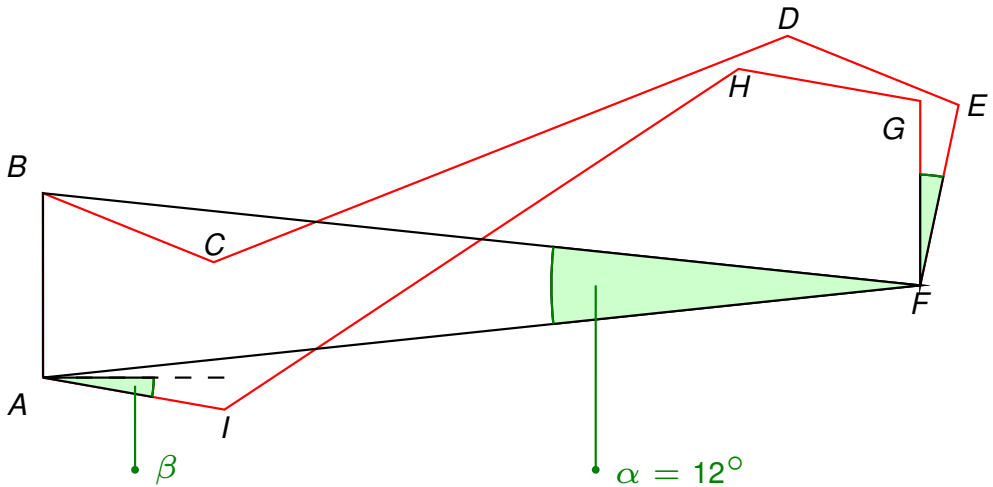
An Example Using Coordinate Calculation – Ti*k*Z realization



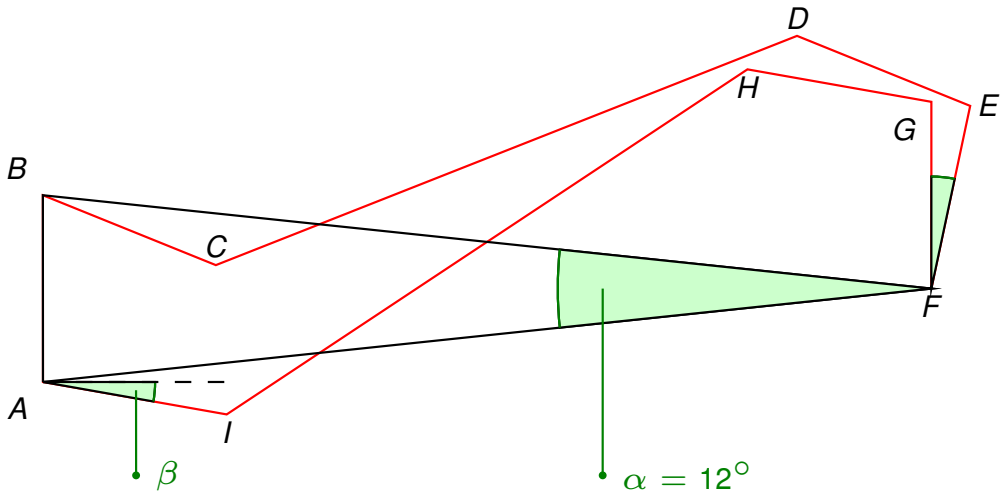
An Example Using Coordinate Calculation – Ti*k*Z realization



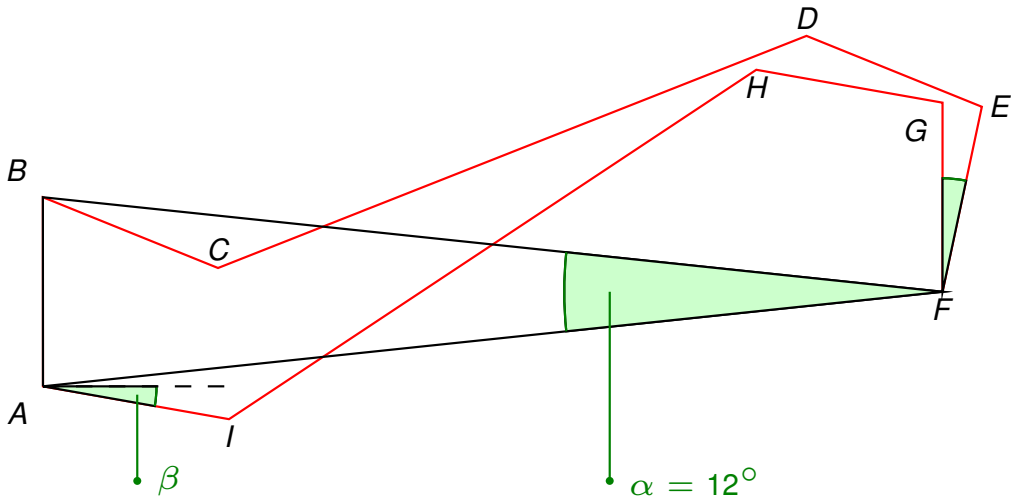
An Example Using Coordinate Calculation – Ti*k*Z realization



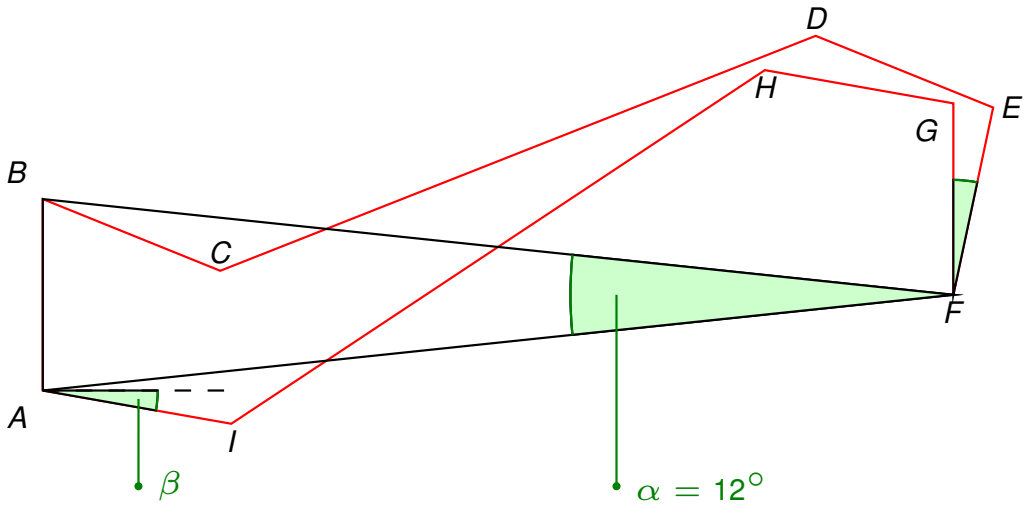
An Example Using Coordinate Calculation – Ti*k*Z realization



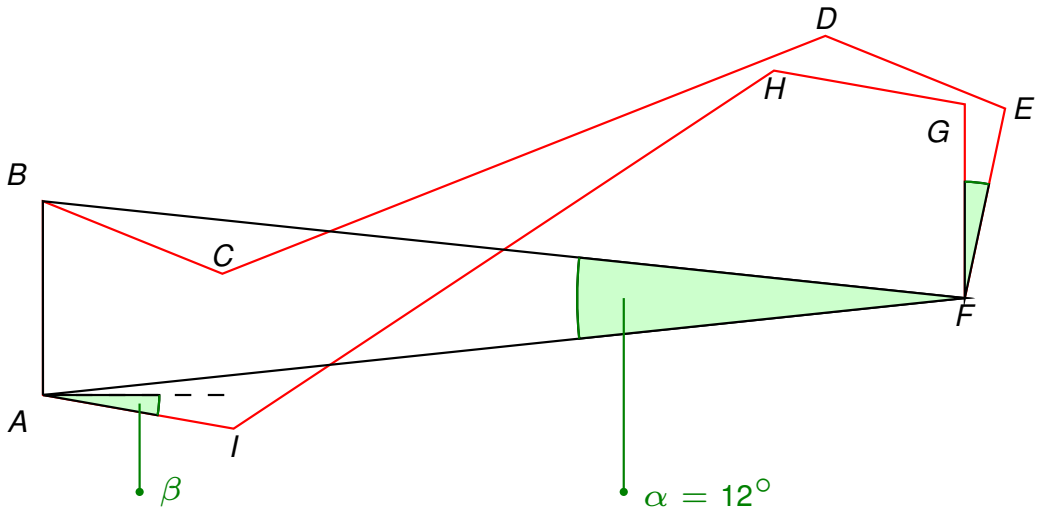
An Example Using Coordinate Calculation – Ti*k*Z realization



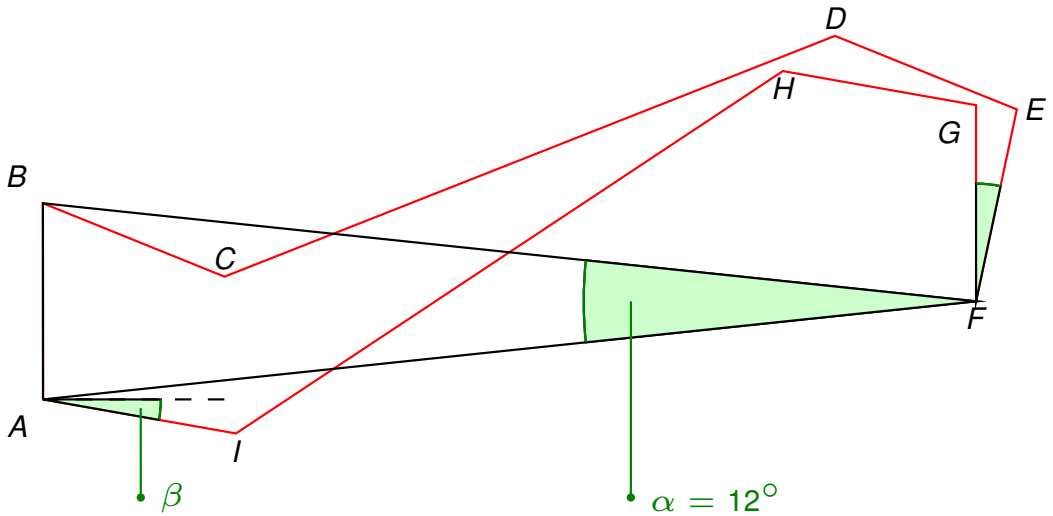
An Example Using Coordinate Calculation – Ti*k*Z realization



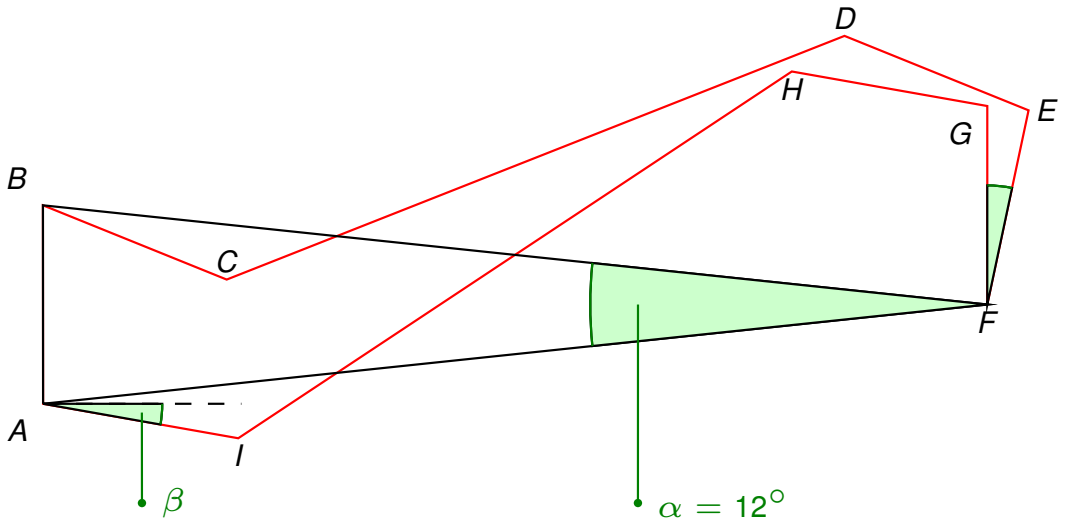
An Example Using Coordinate Calculation – Ti*k*Z realization



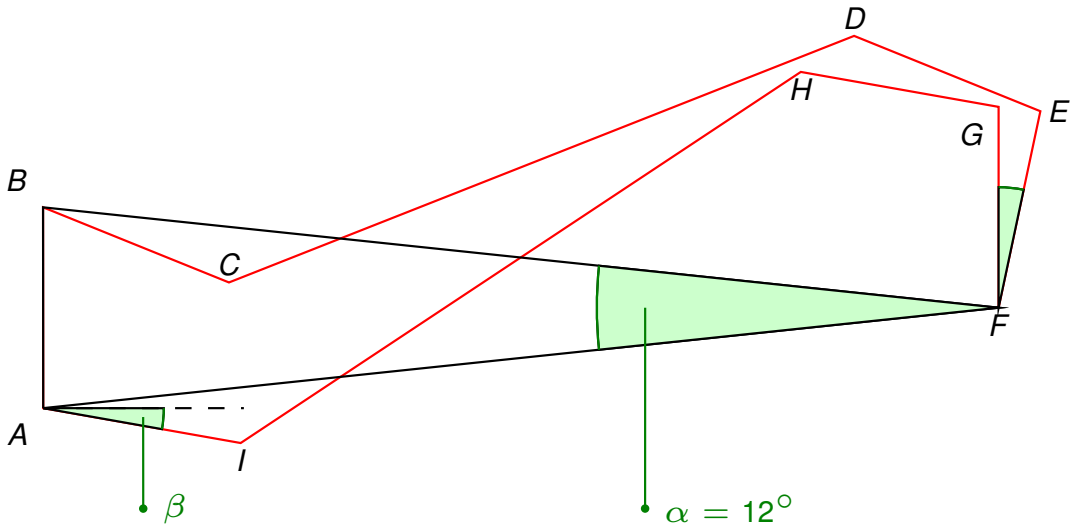
An Example Using Coordinate Calculation – Ti*k*Z realization



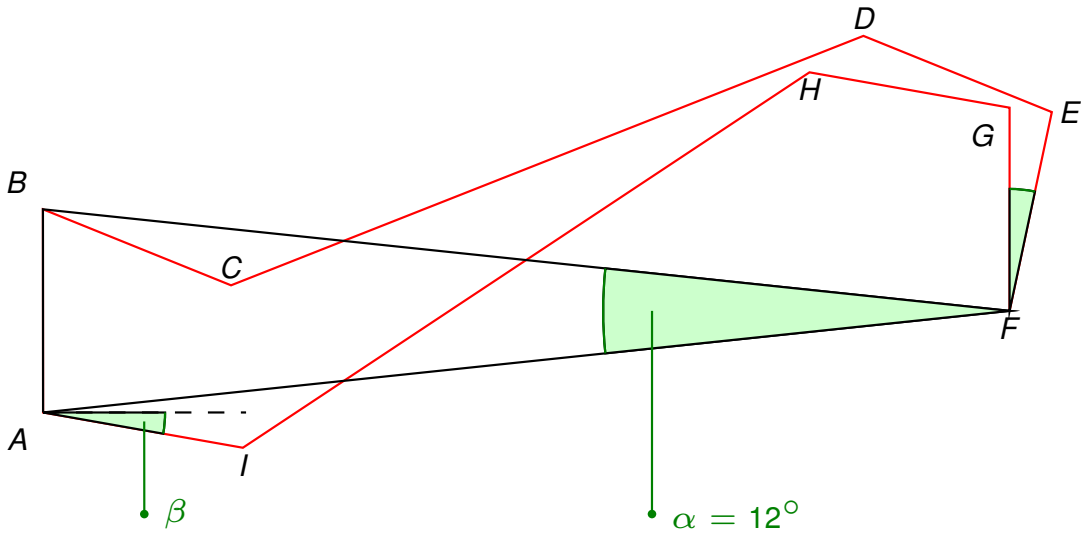
An Example Using Coordinate Calculation – Ti*k*Z realization



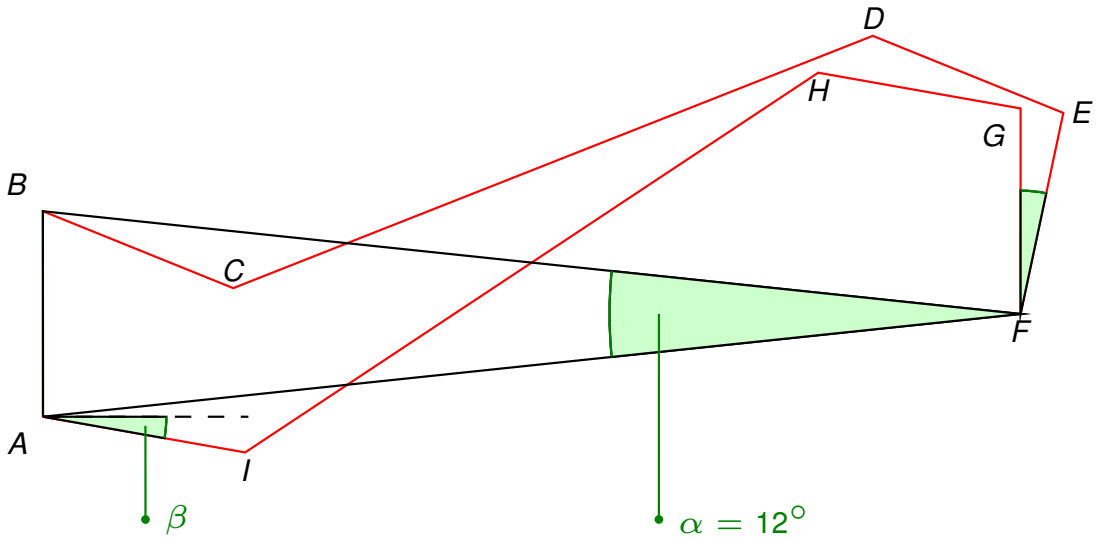
An Example Using Coordinate Calculation – Ti*k*Z realization



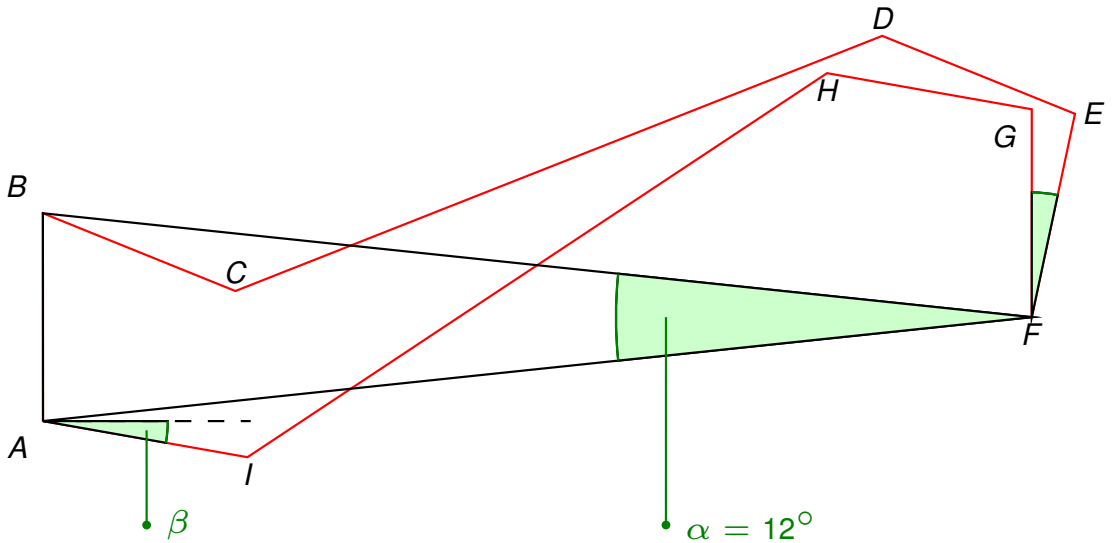
An Example Using Coordinate Calculation – Ti*k*Z realization



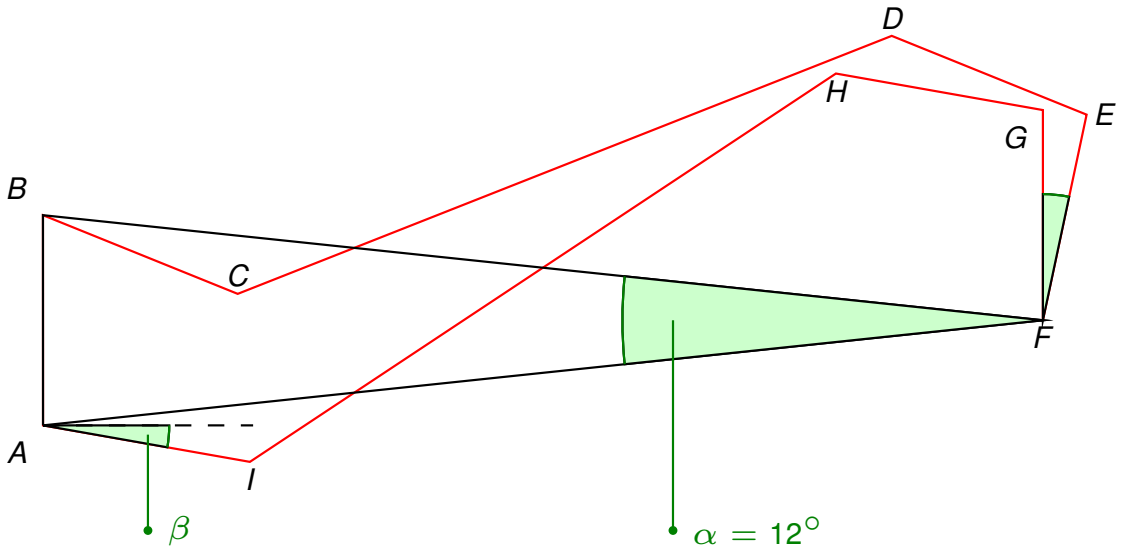
An Example Using Coordinate Calculation – Ti*k*Z realization



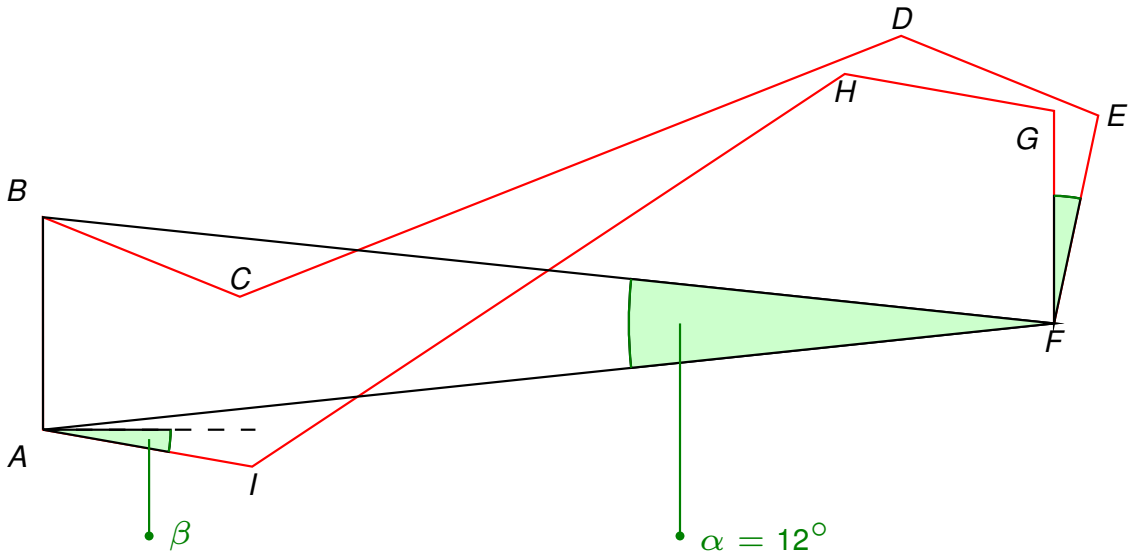
An Example Using Coordinate Calculation – Ti*k*Z realization



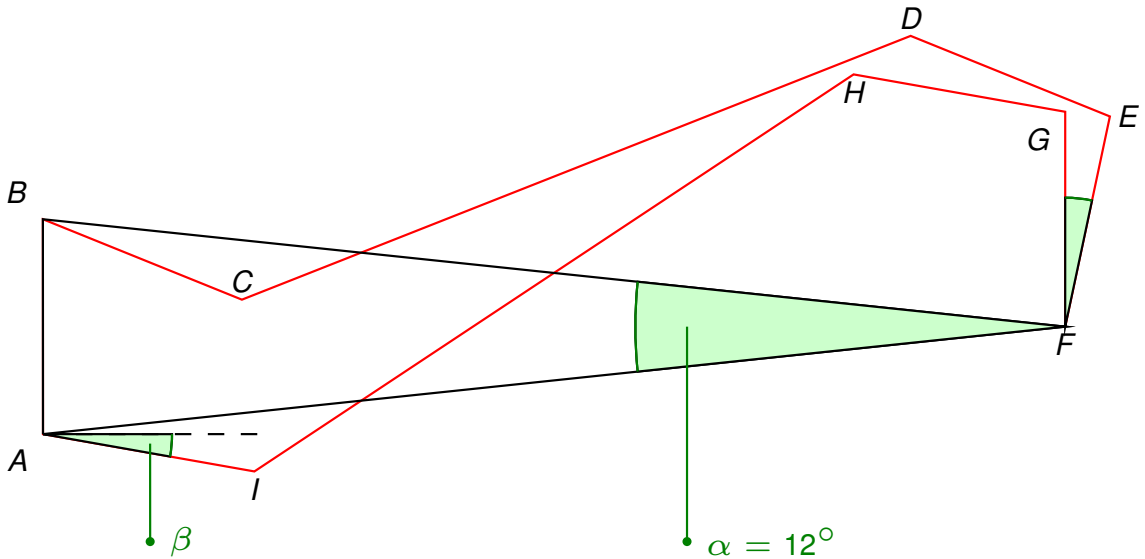
An Example Using Coordinate Calculation – Ti*k*Z realization



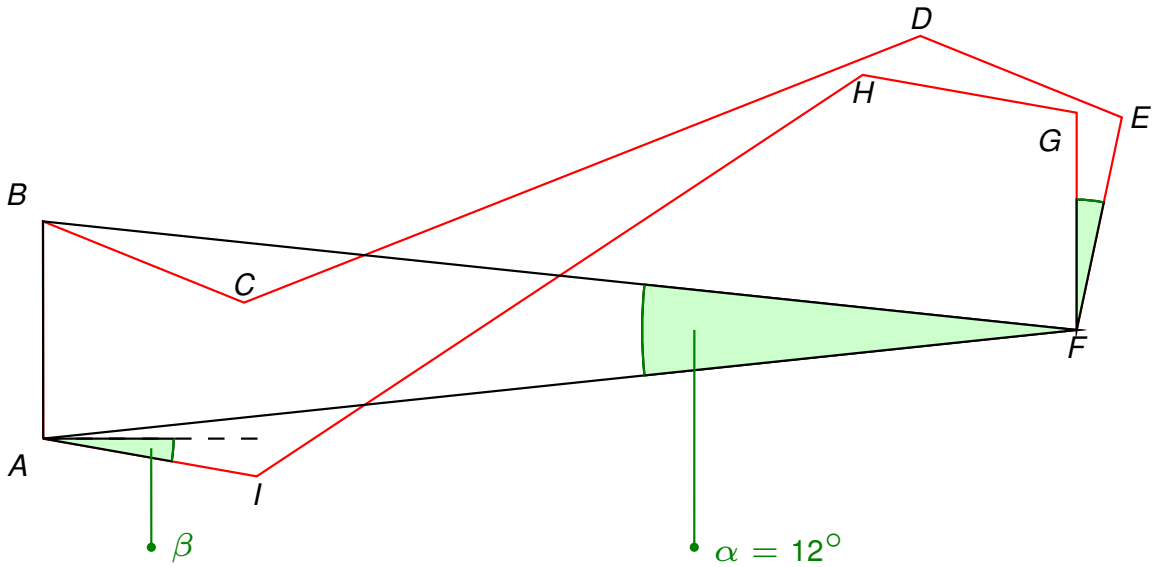
An Example Using Coordinate Calculation – Ti*k*Z realization



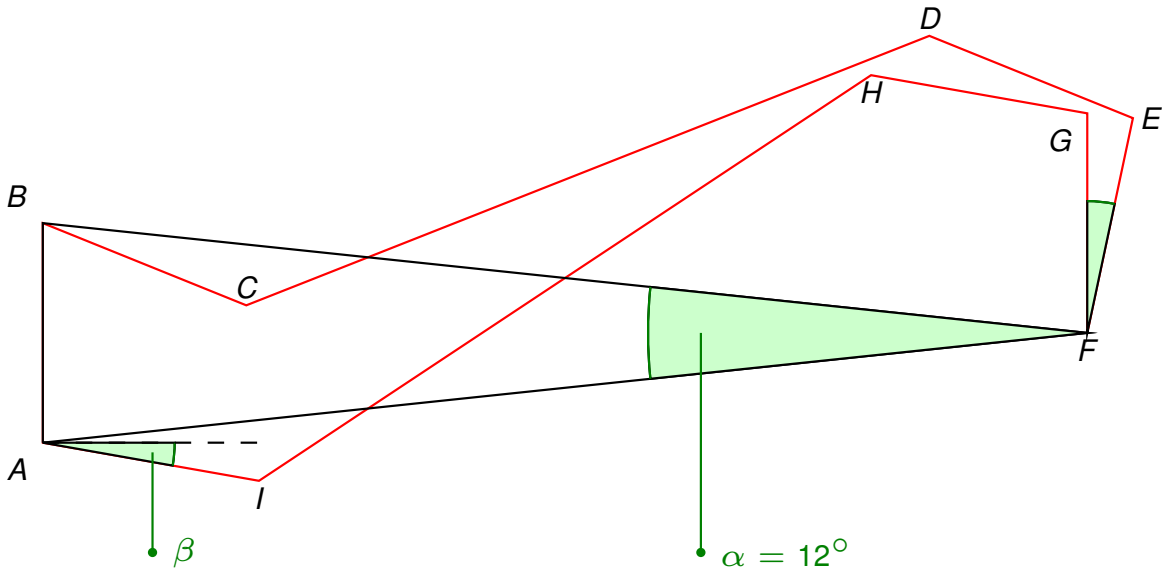
An Example Using Coordinate Calculation – Ti*k*Z realization



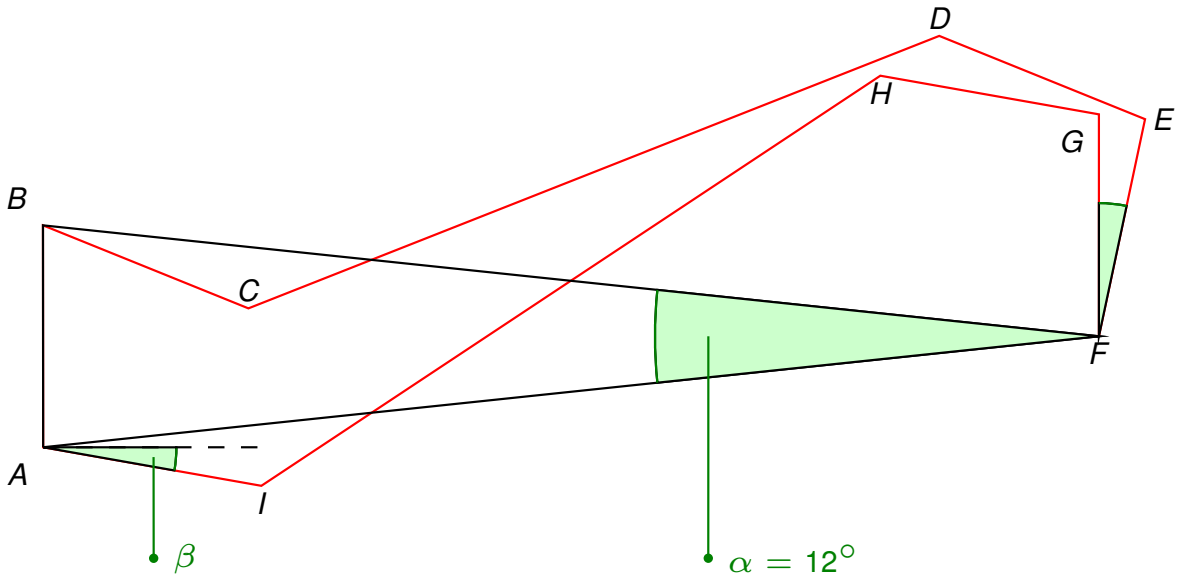
An Example Using Coordinate Calculation – Ti*k*Z realization



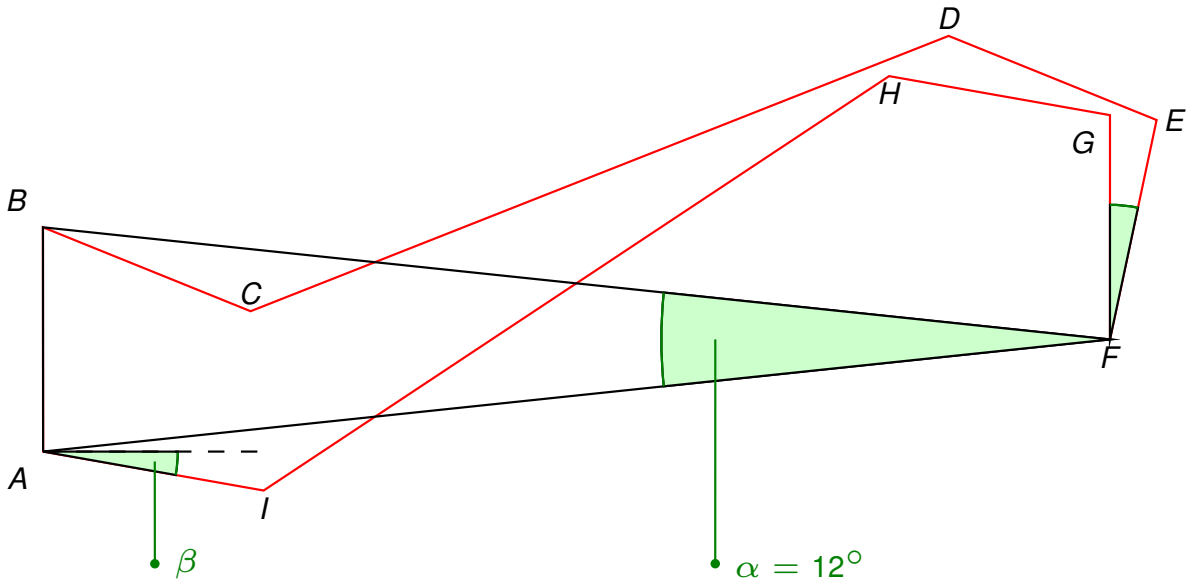
An Example Using Coordinate Calculation – Ti*k*Z realization



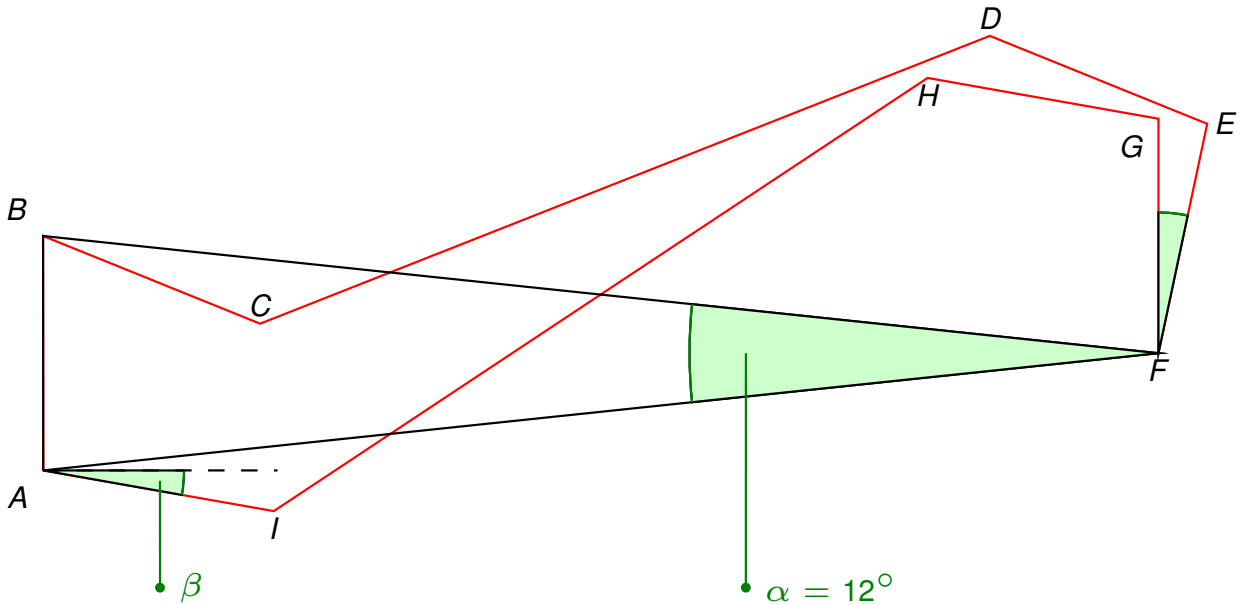
An Example Using Coordinate Calculation – Ti*k*Z realization



An Example Using Coordinate Calculation – Ti*k*Z realization

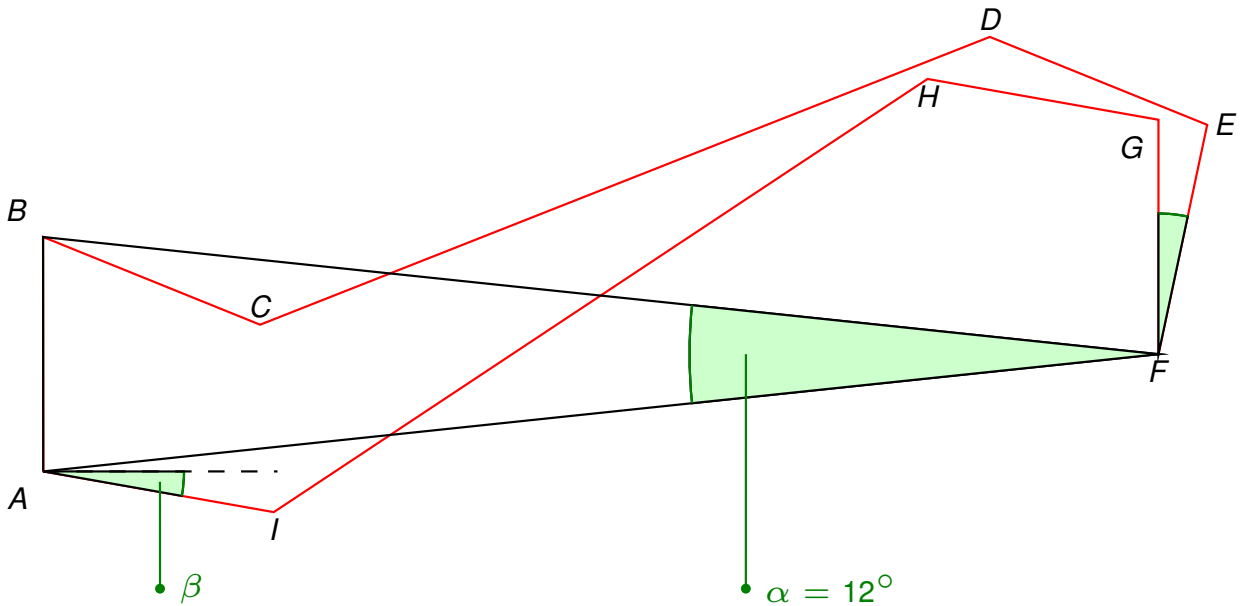


An Example Using Coordinate Calculation – Ti*k*Z realization



$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

An Example Using Coordinate Calculation – Ti*k*Z realization



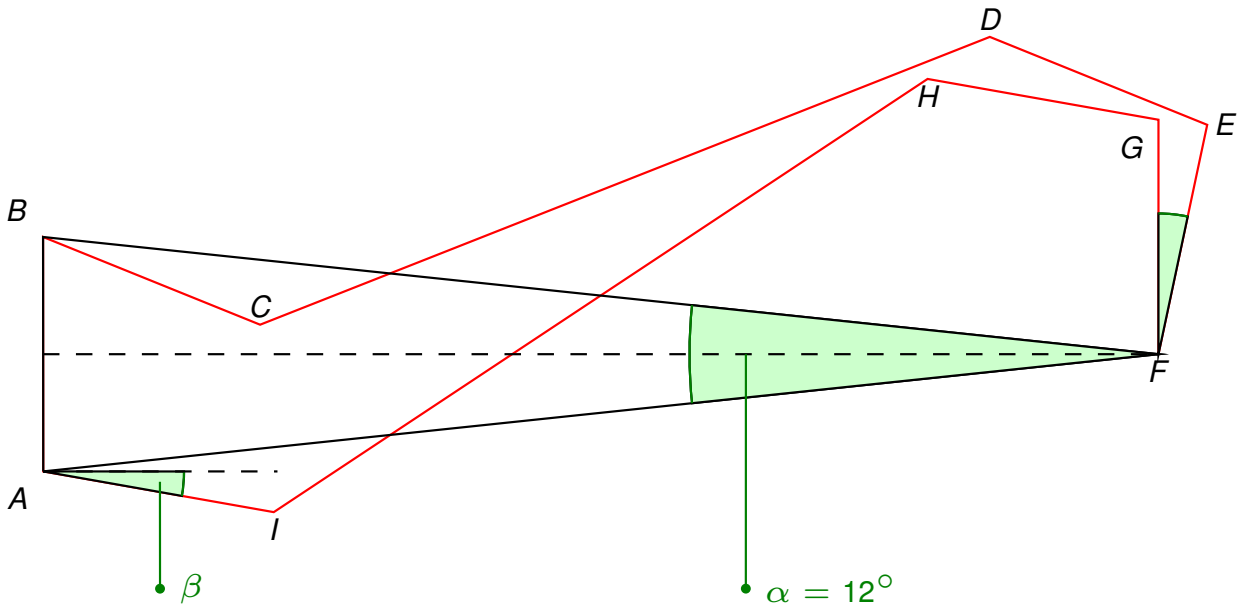
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Define Variables to use:

```
\def\a{0.5}
\def\b{0.5}
\def\bAngle{-10}
```

An Example Using Coordinate Calculation – Ti*k*Z realization



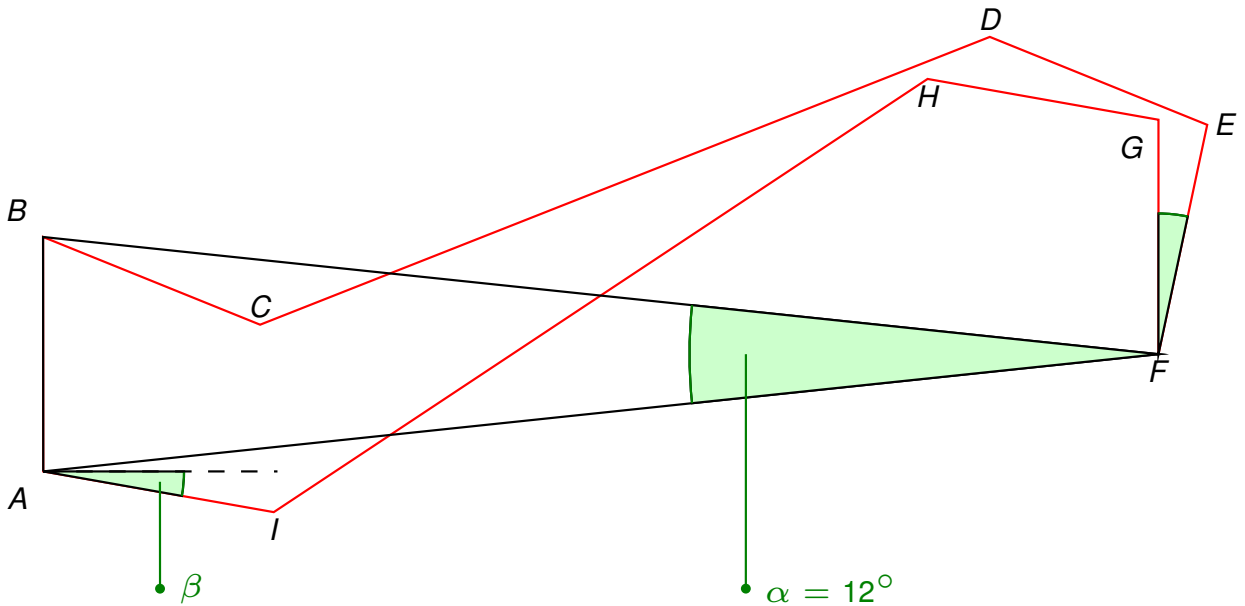
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute Distance of Point F from $\overline{AB}$:

```
\pgfmathsetmacro{\hyp}{\a*0.5 / cos(84)}
\pgfmathsetmacro{\len}{sqrt(\hyp*\hyp-0.25*\a*\a)}
```

An Example Using Coordinate Calculation – Ti*k*Z realization



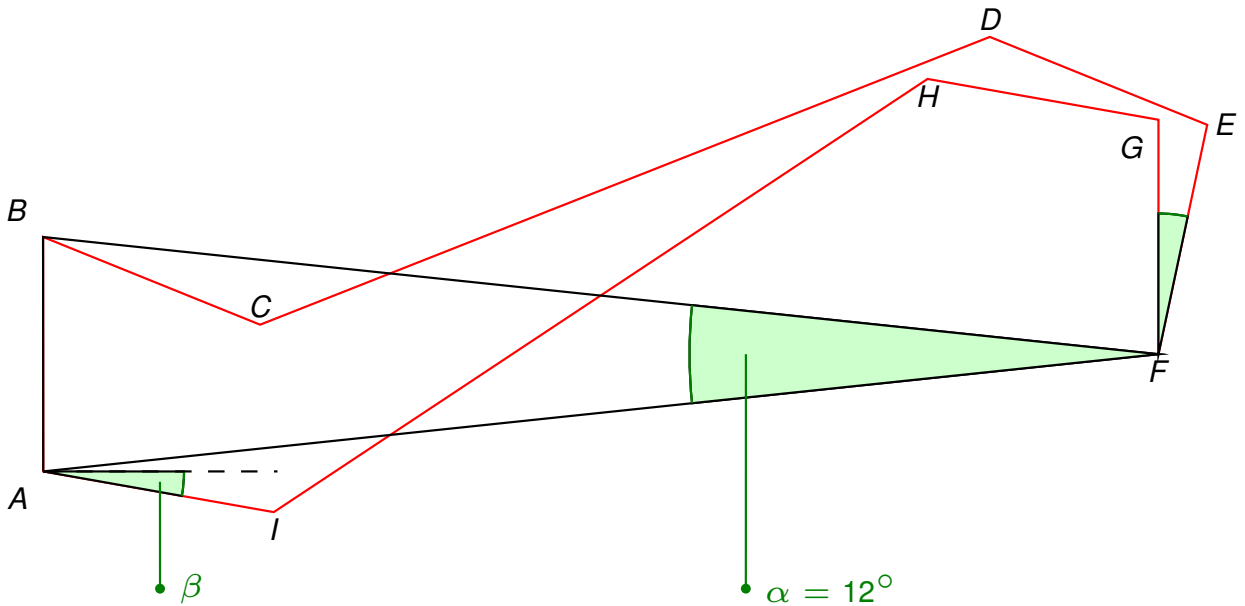
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute the locations of the points:

```
\coordinate (A) at (0,0);          % start coordinate
\coordinate (B) at ($ (A) + (0,\a) $);
\coordinate (F) at ($ (B) + (\len,0.5*\a) $);
\coordinate (G) at ($ (F) + (0,\a) $);
```

An Example Using Coordinate Calculation – Ti*k*Z realization



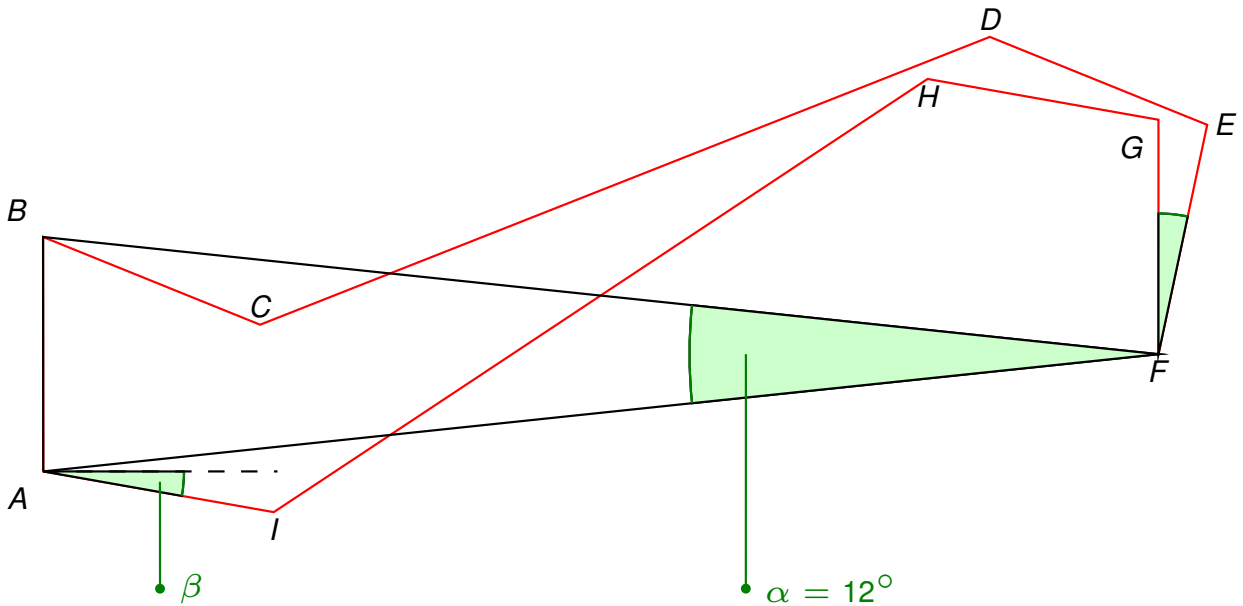
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute the locations of the points:

```
\coordinate (Htemp) at ($ (G) - (\b,0) $);
\coordinate (H) at ($ (G)!1!\bAngle:(Htemp) $);
\coordinate (Itemp) at ( $ (A) + (\b,0)$ );
\coordinate (I) at ($ (A)!1!\bAngle:(Itemp) $);
```

An Example Using Coordinate Calculation – Ti*k*Z realization



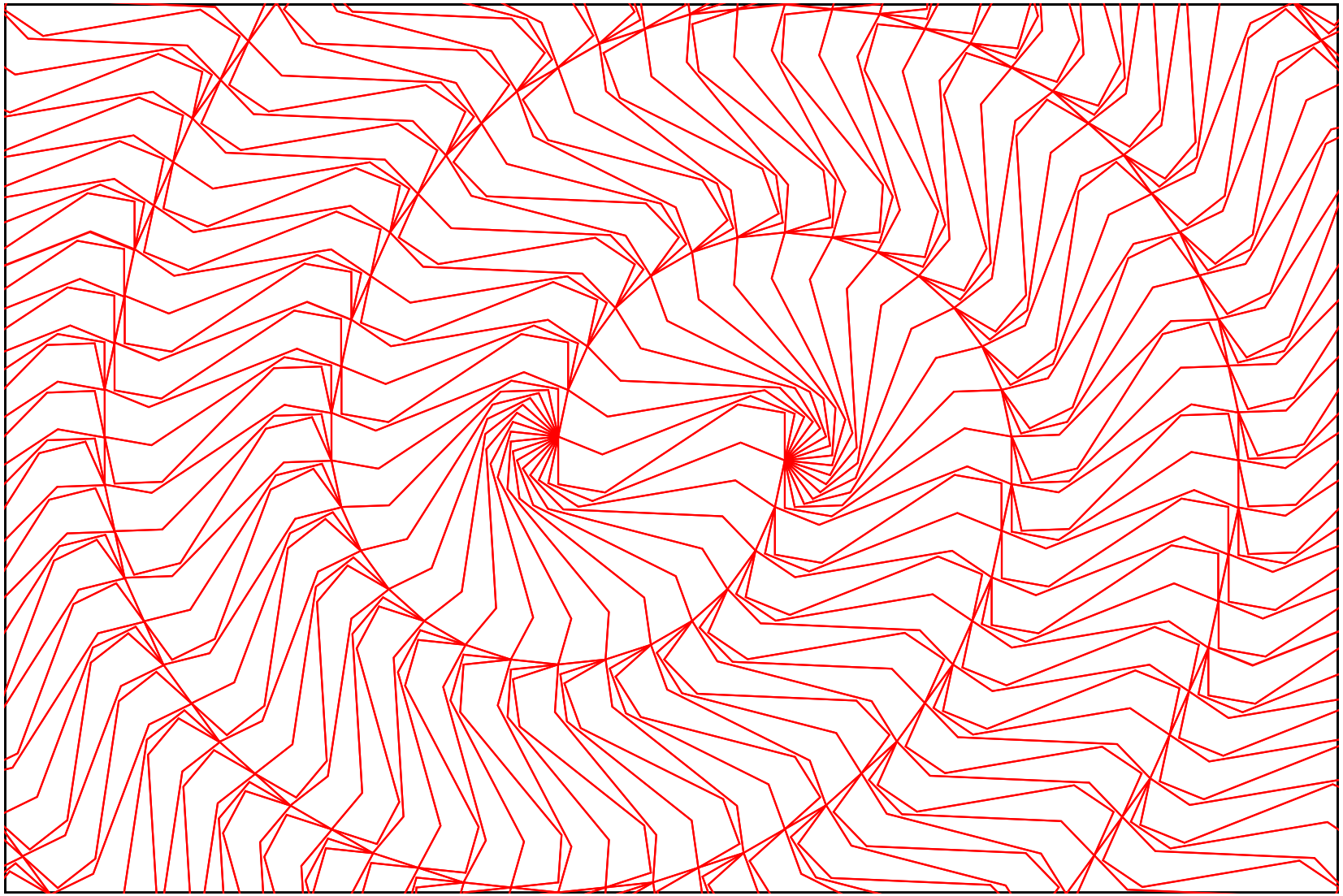
$$a = \overline{AB} = \overline{FG}$$
$$b = \overline{AI} = \overline{HG}$$

Example

Compute the locations of the points:

```
\coordinate (E) at ($ (F)!1!-12:(G) $);
\coordinate (D) at ($ (F)!1!-12:(H) $);
\coordinate (C) at ($ (F)!1!-12:(I) $);
```

The Voderberg Spiral



Scopes in TikZ

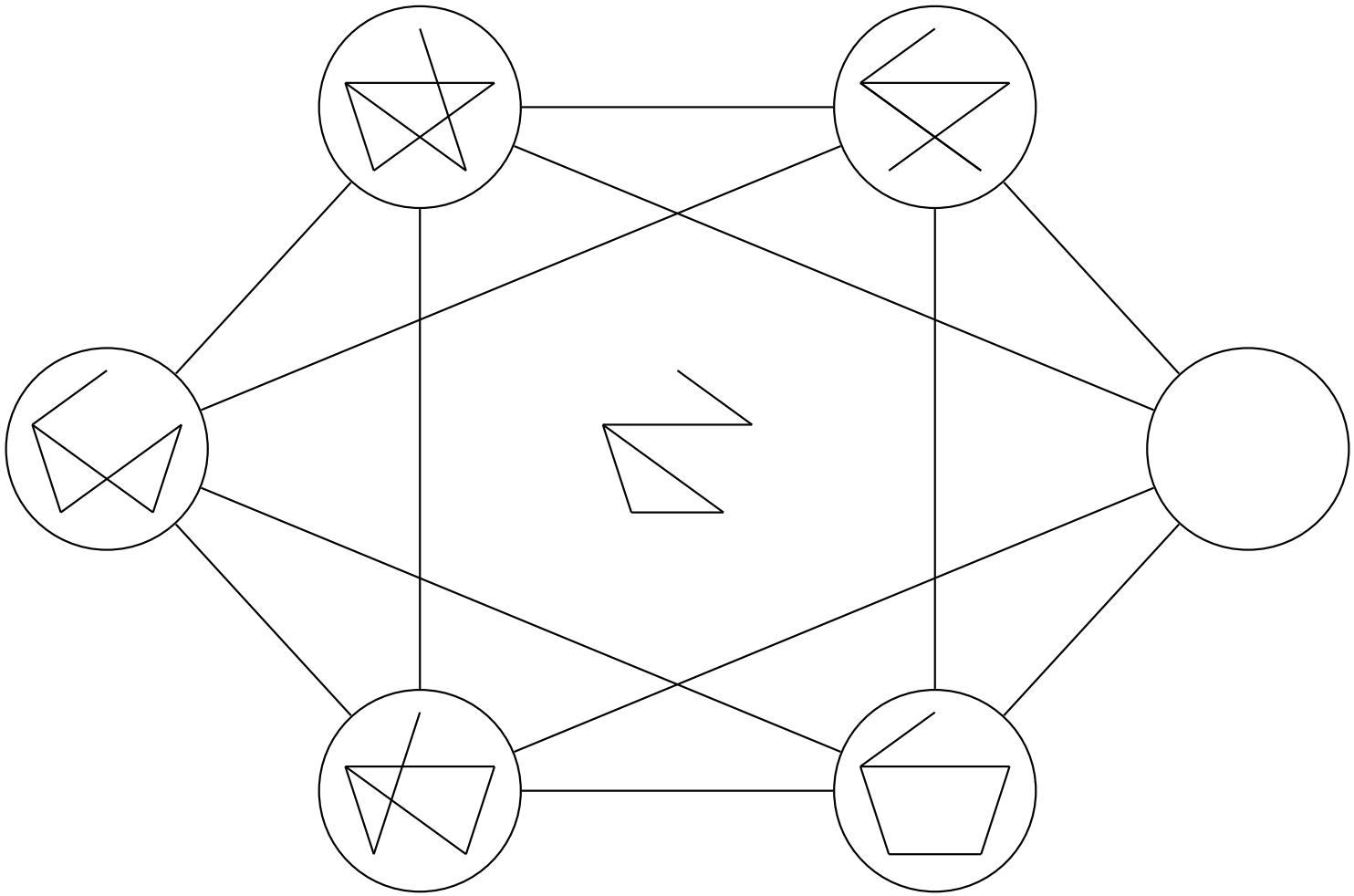
- TikZ allows scopes, just like e. g. JAVA
- scopes can alter the drawing projection
- that means rotating, moving or scaling etc.

Possible commands are:

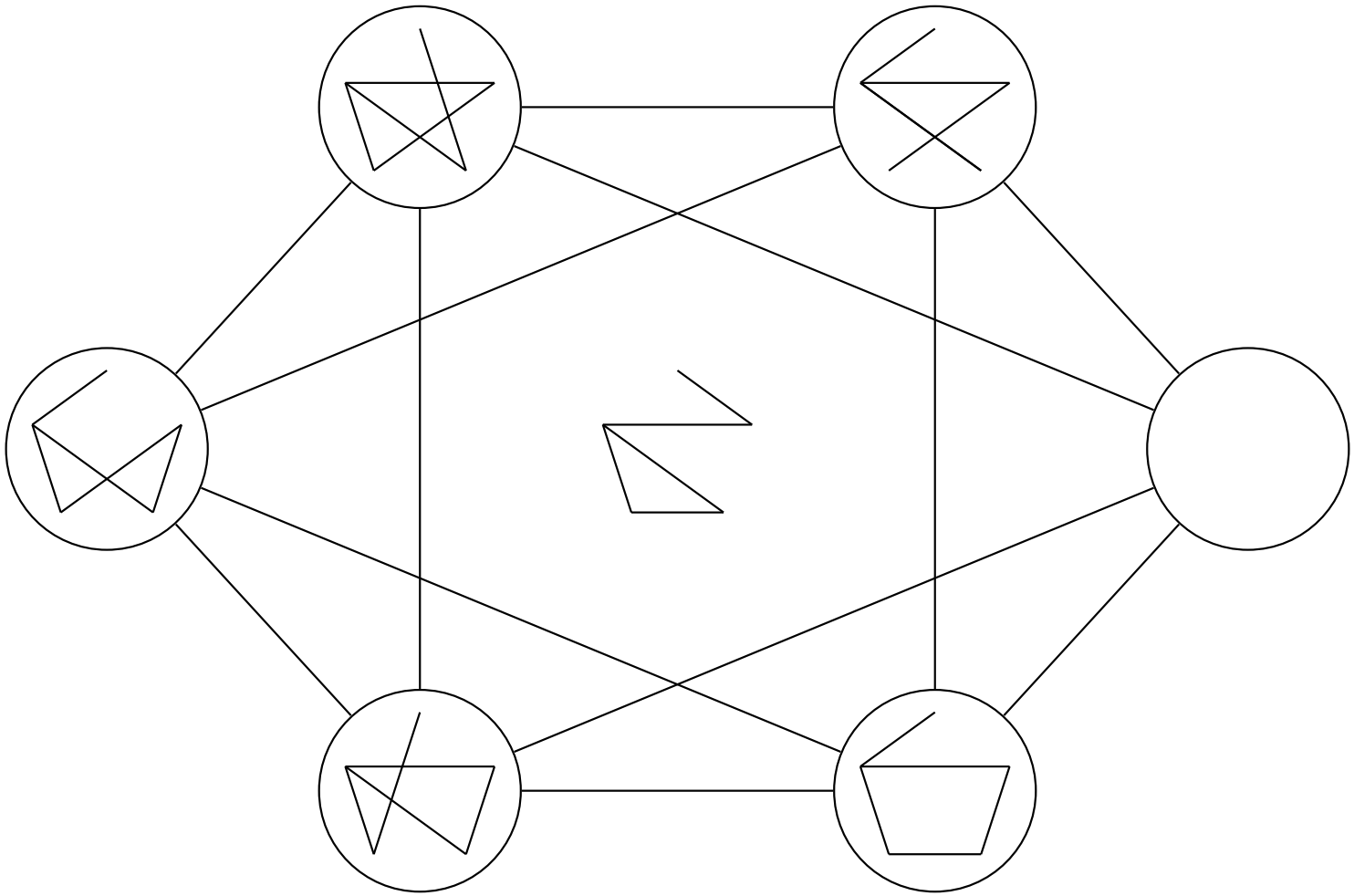
- `xscale`, `yscale` and `scale` for scaling
- `rotate` for rotation around an angle
- `xshift`, `yshift` and `shift` for movements of the origin

```
\begin{scope}[rotate=30,xscale=0.5,shift={(0:\s)}]
...
\end{scope}
```

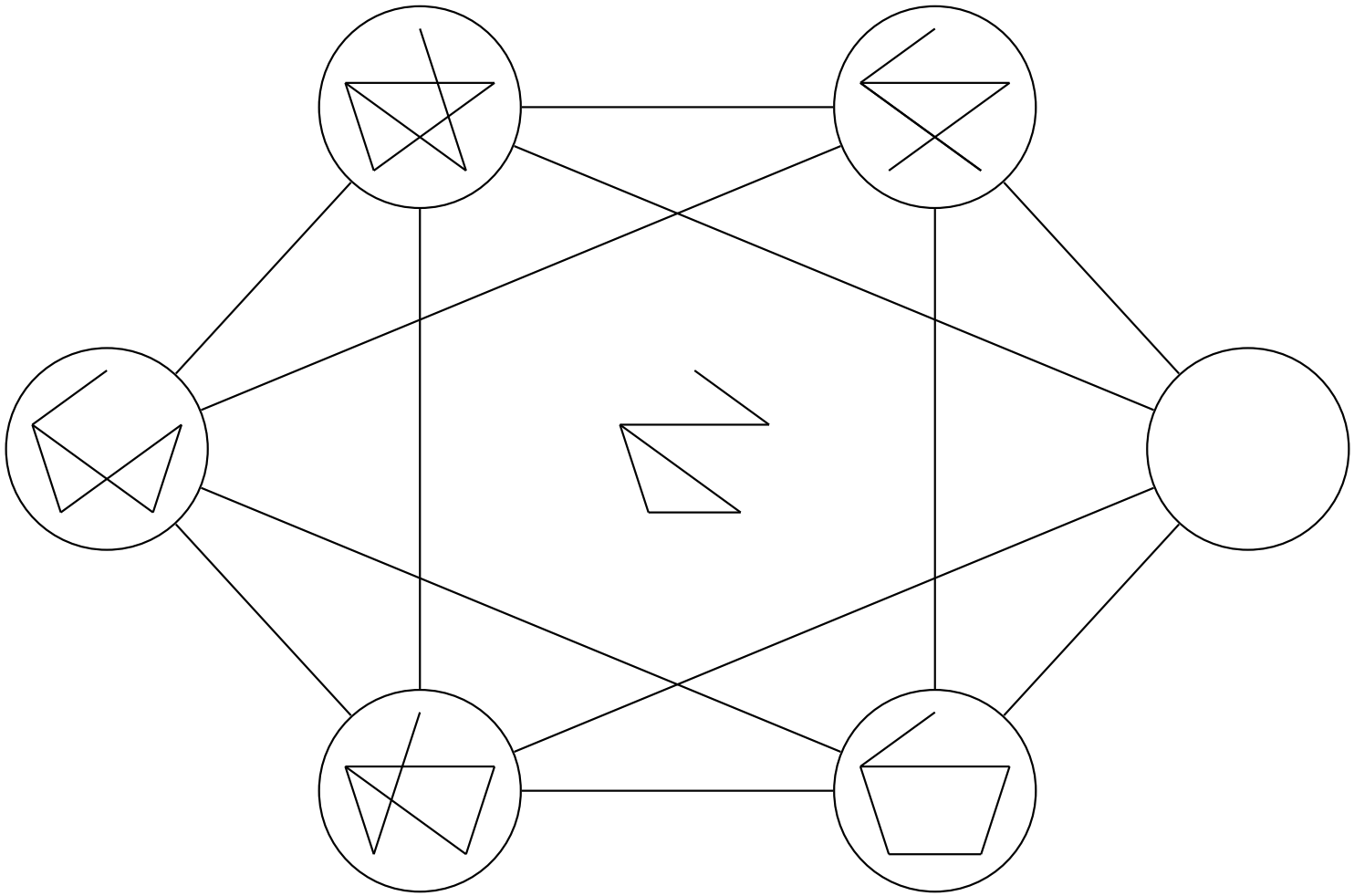
Example for Scopes



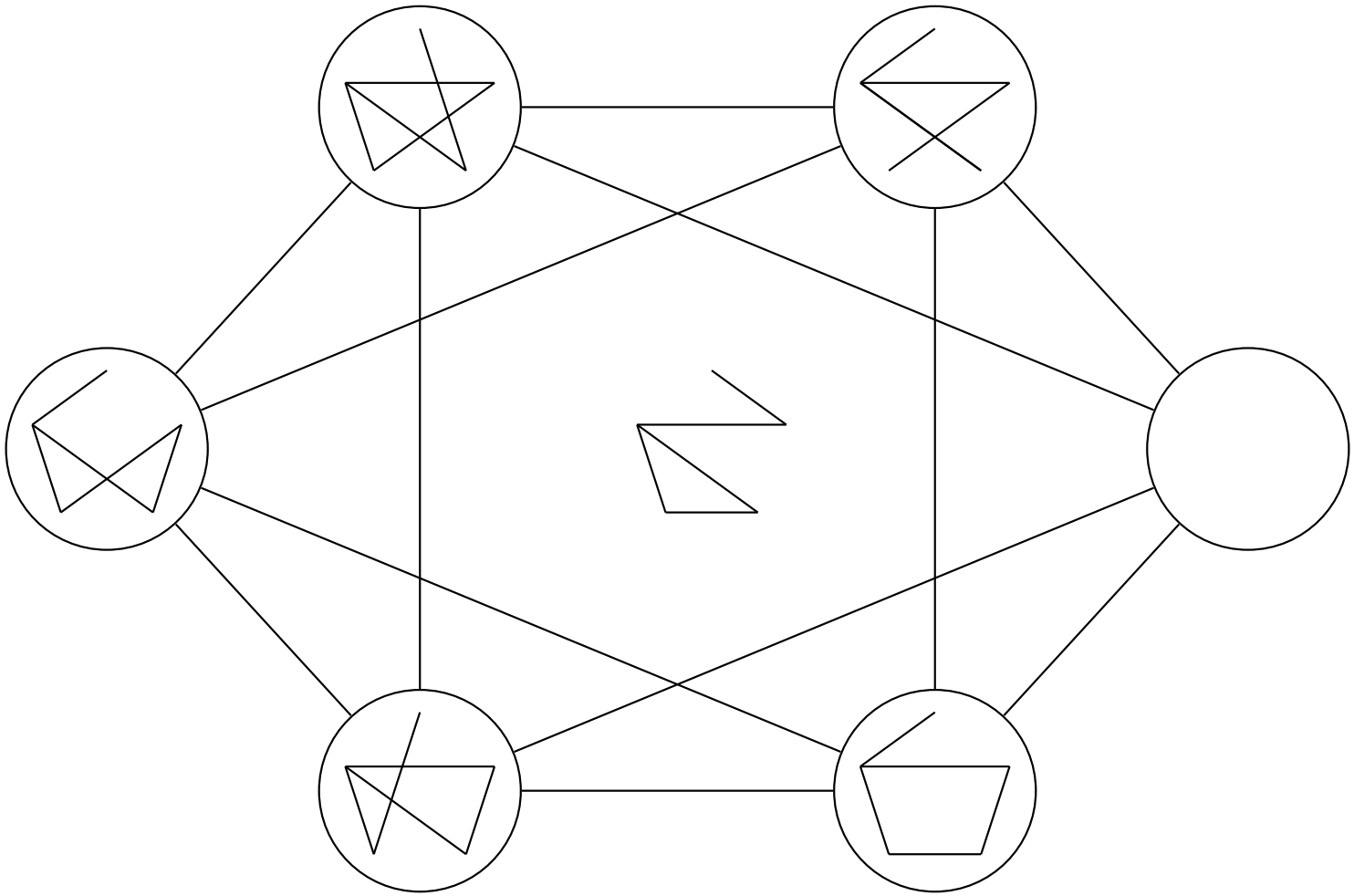
Example for Scopes



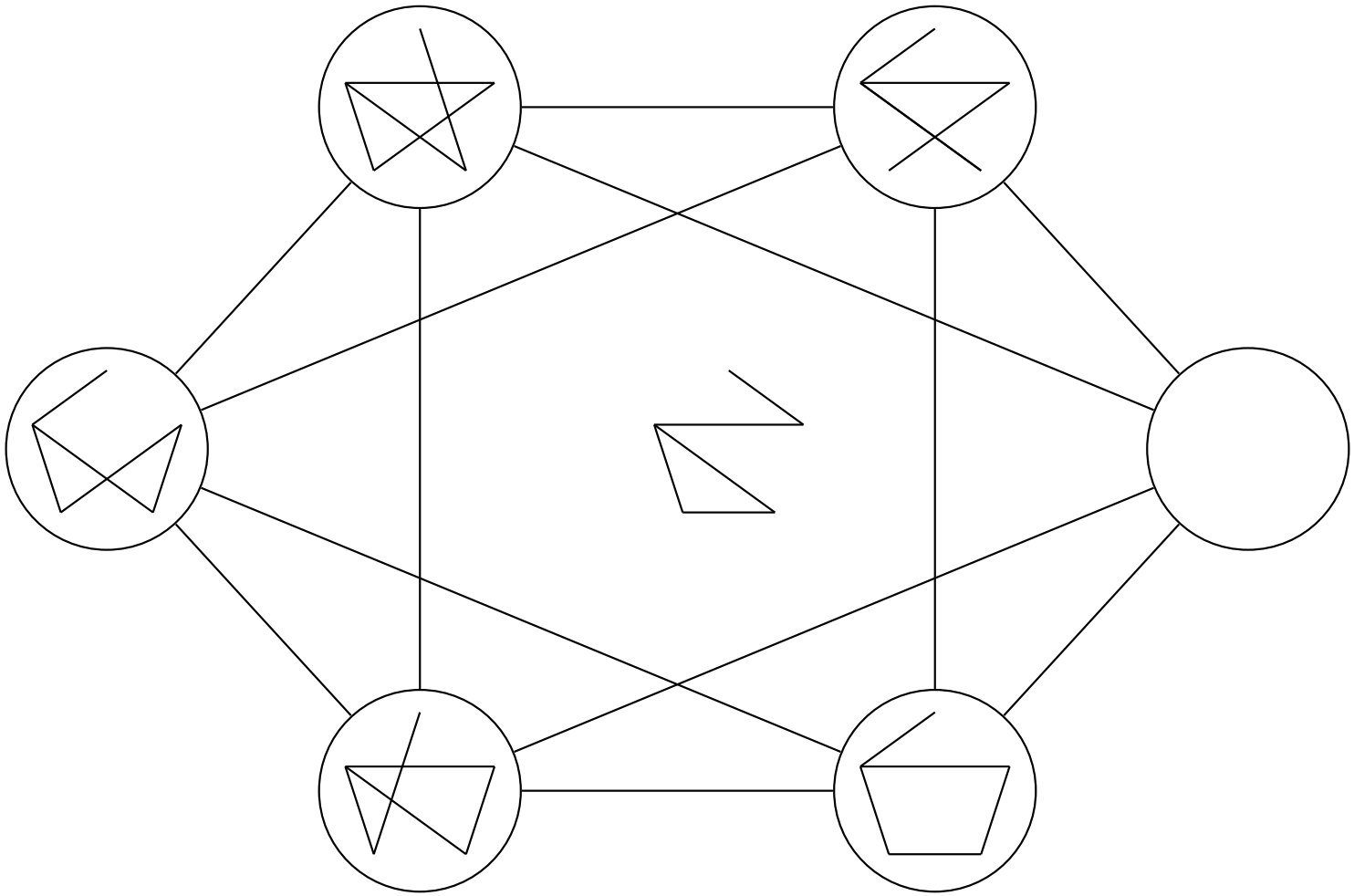
Example for Scopes



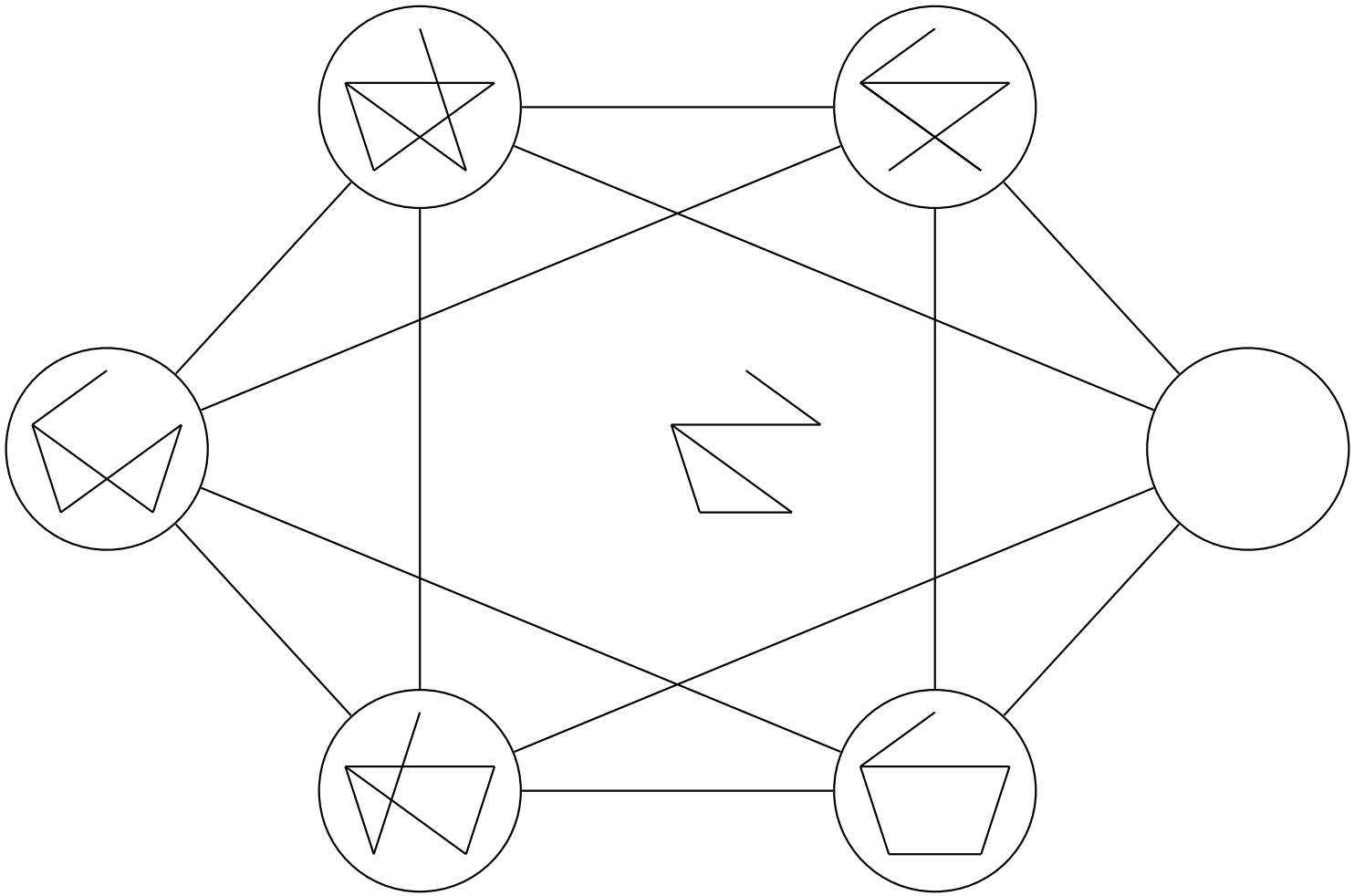
Example for Scopes



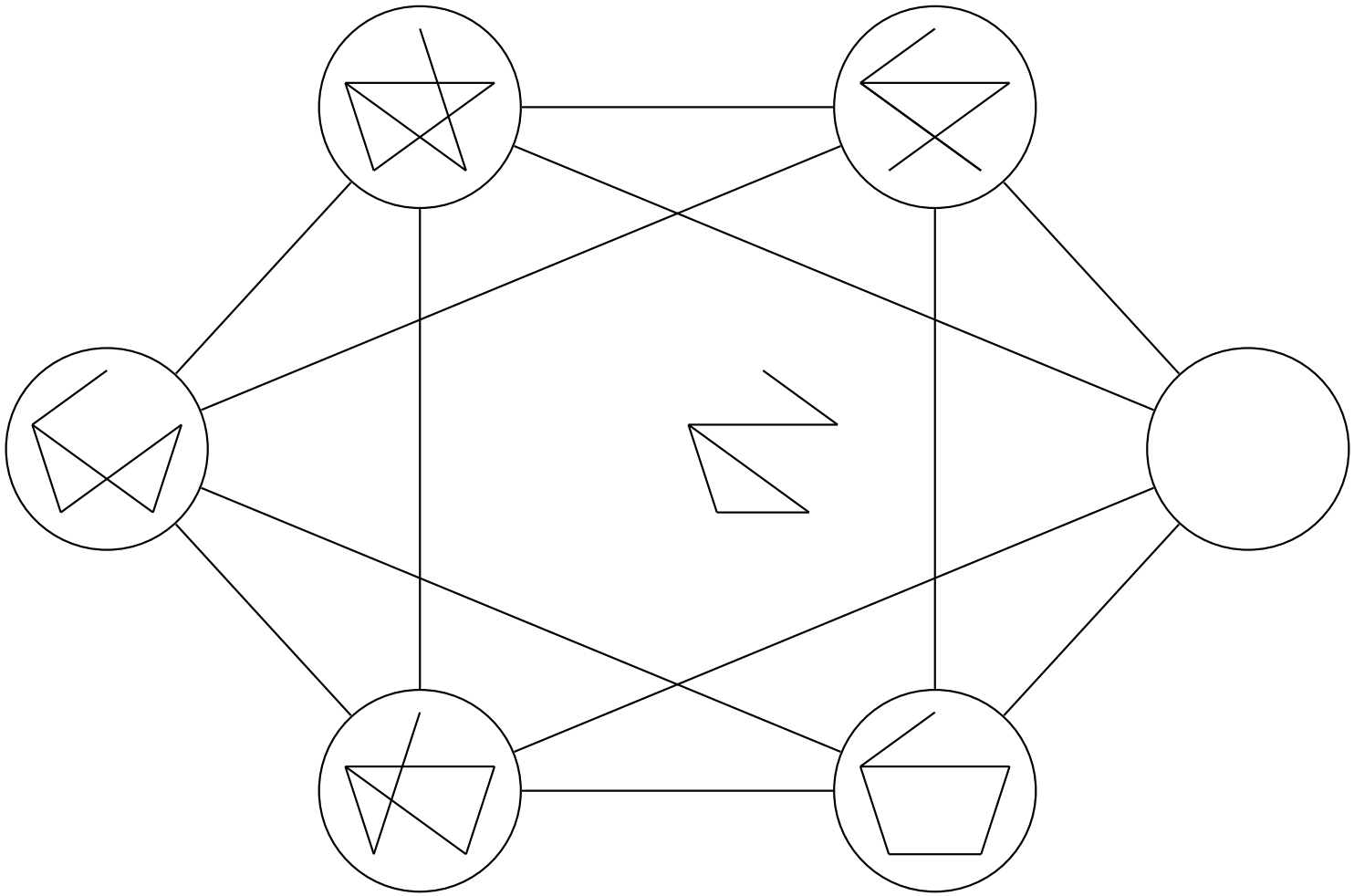
Example for Scopes



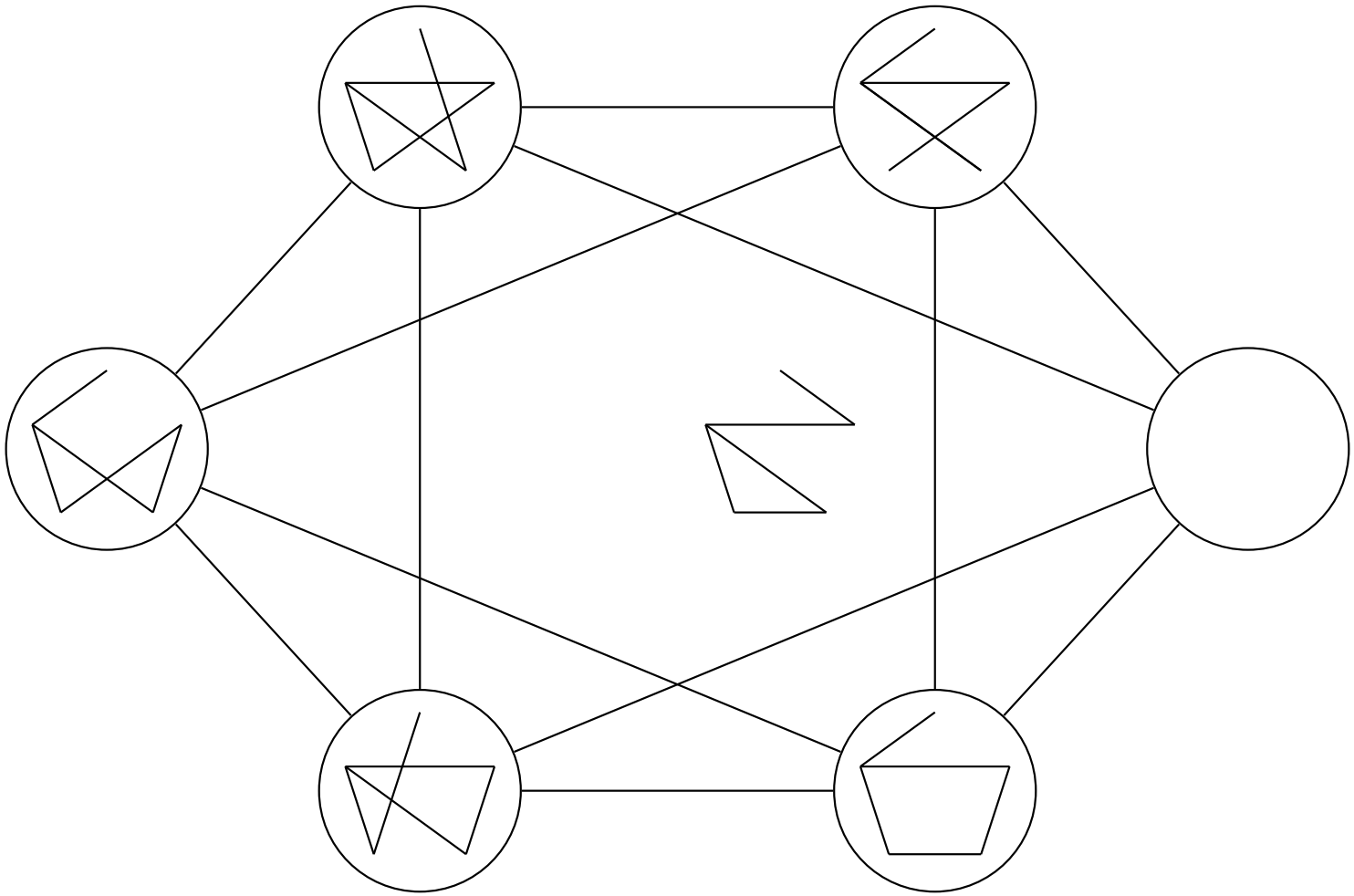
Example for Scopes



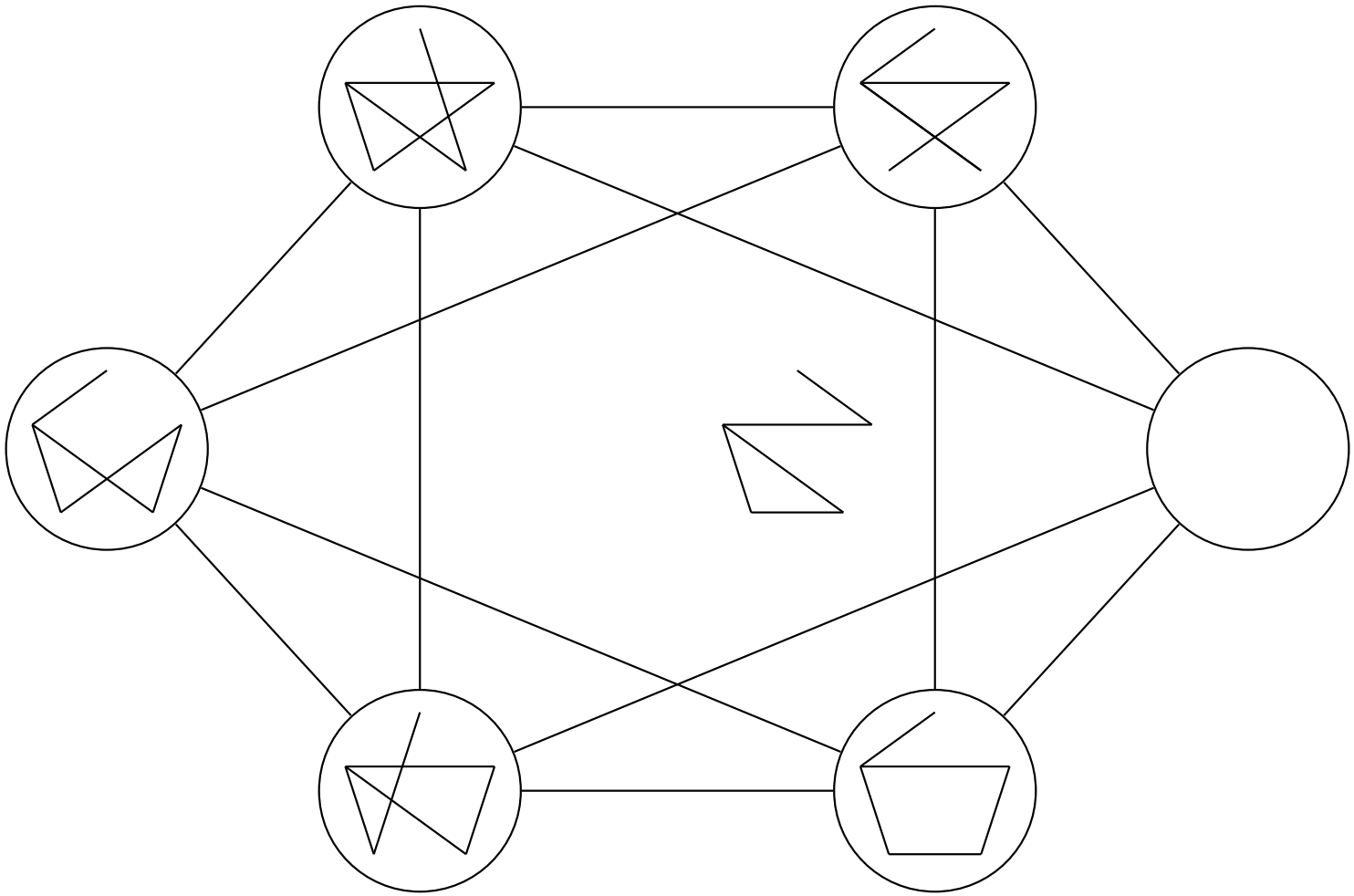
Example for Scopes



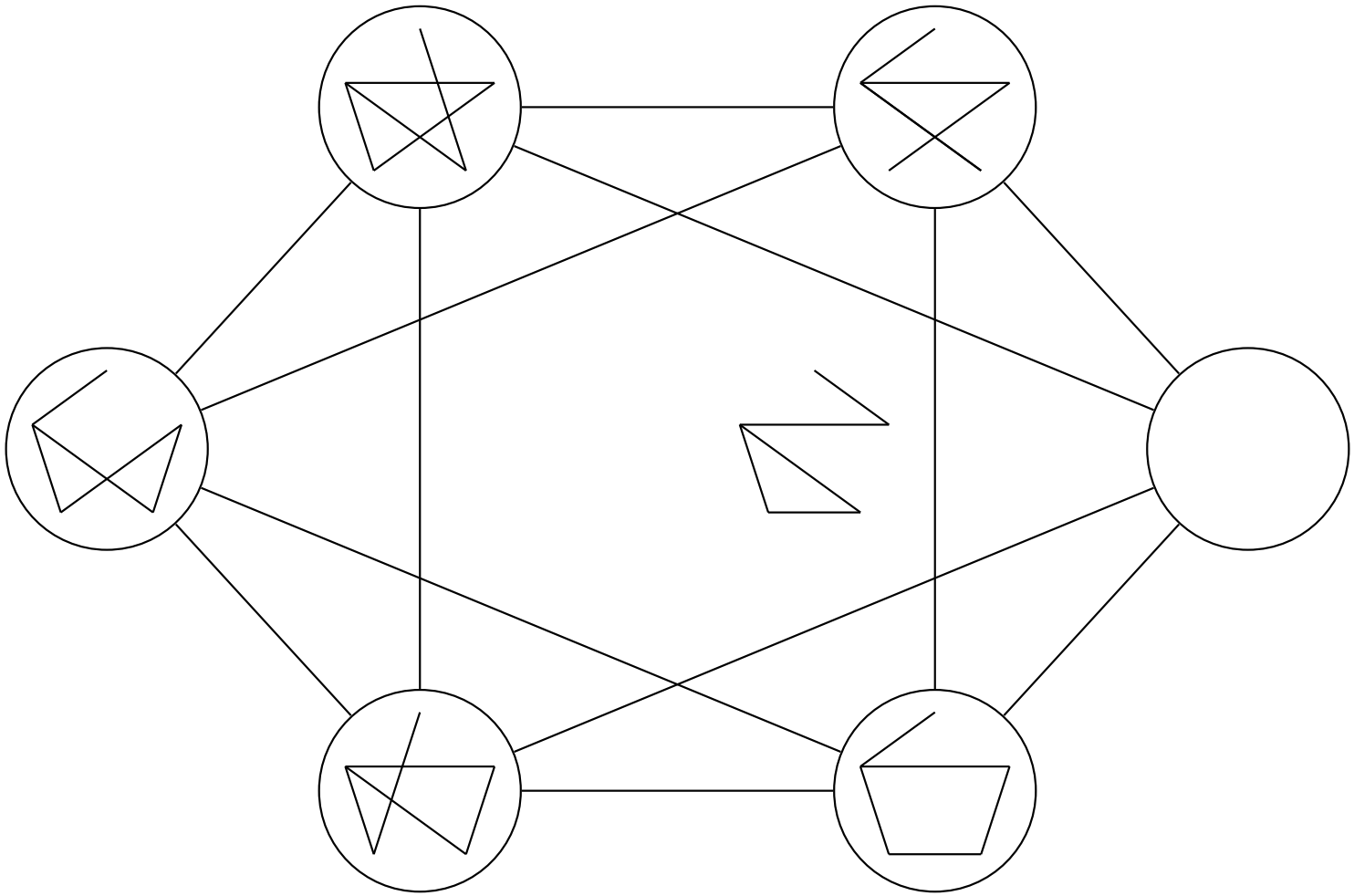
Example for Scopes



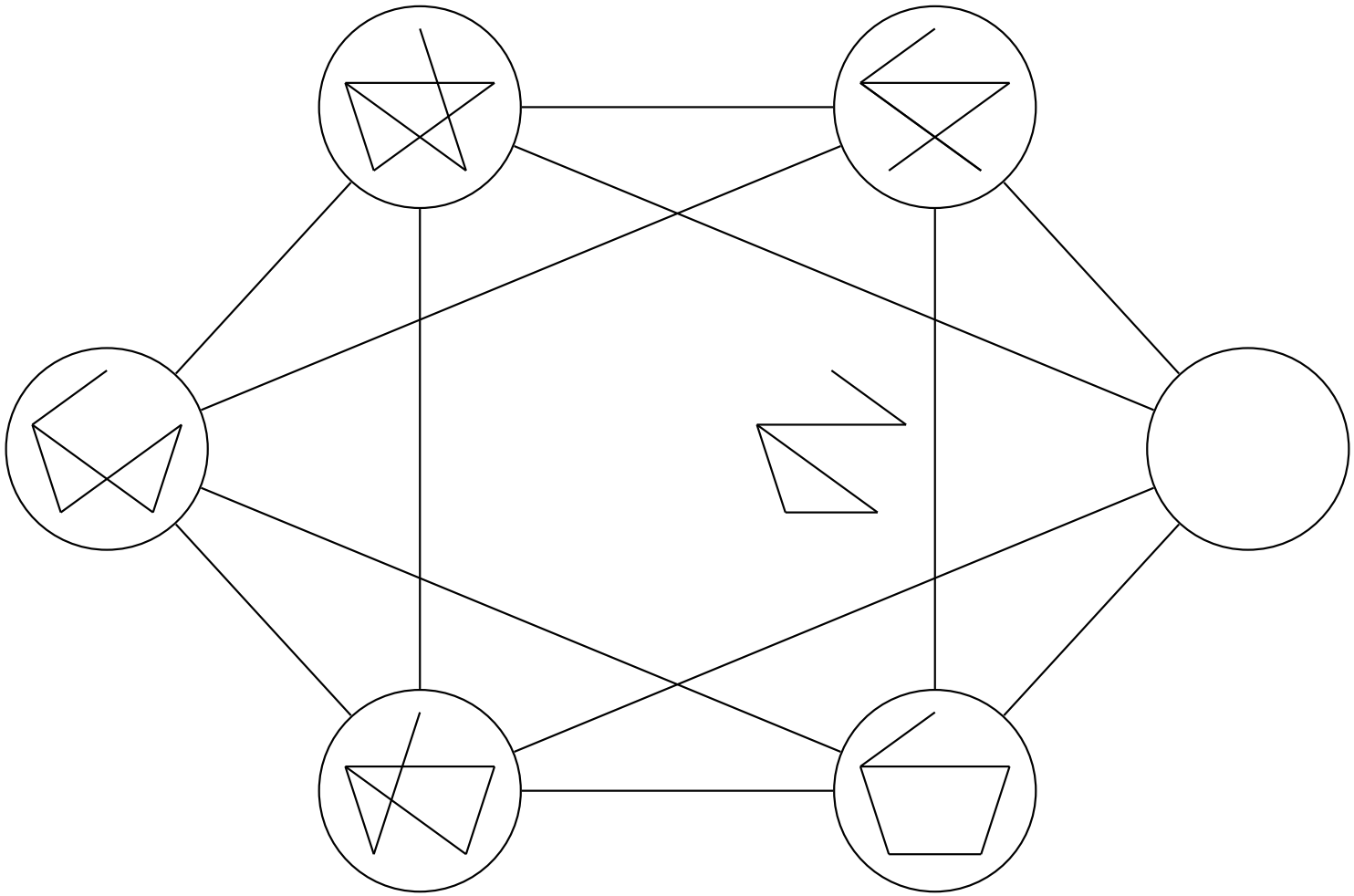
Example for Scopes



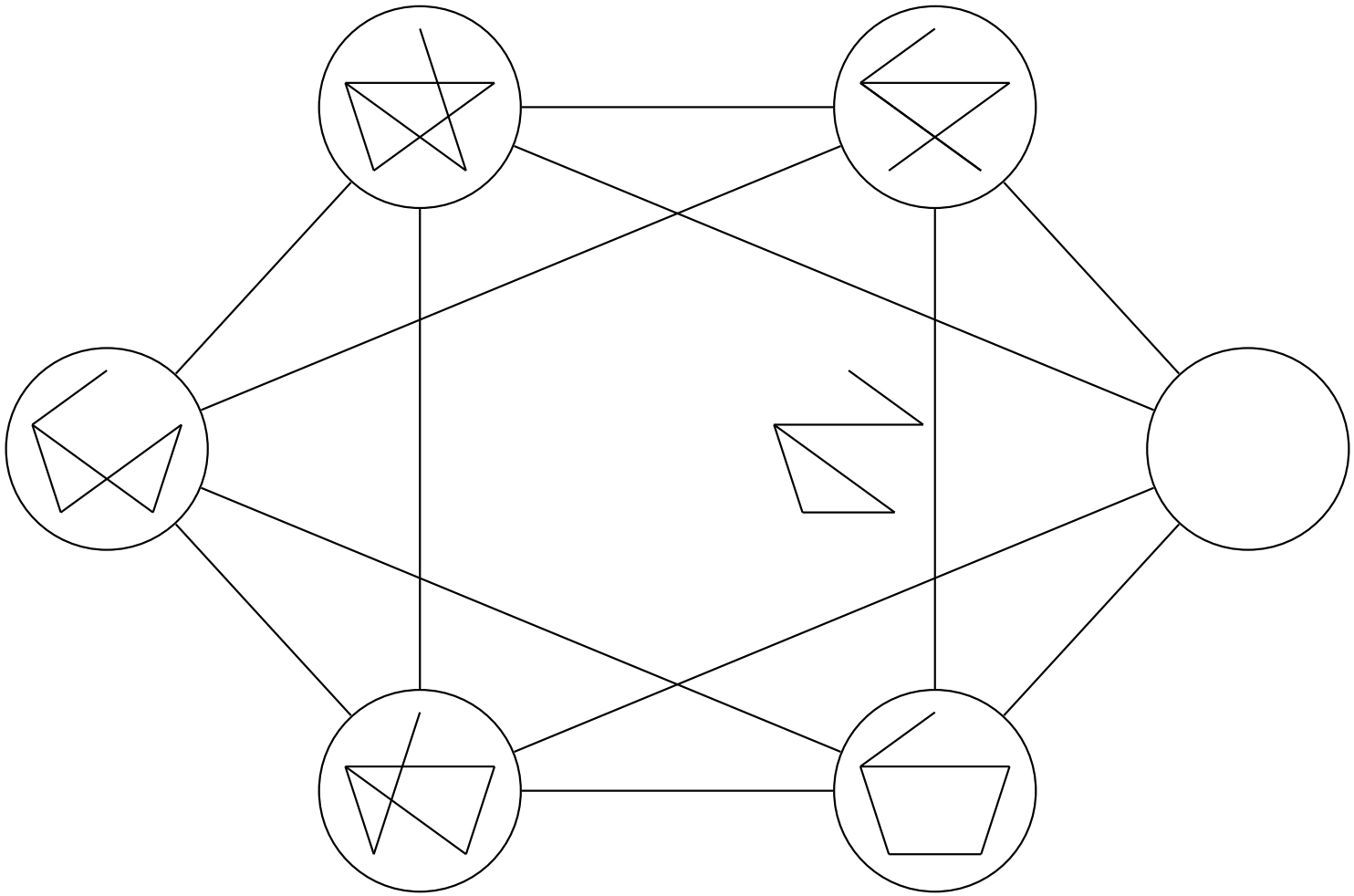
Example for Scopes



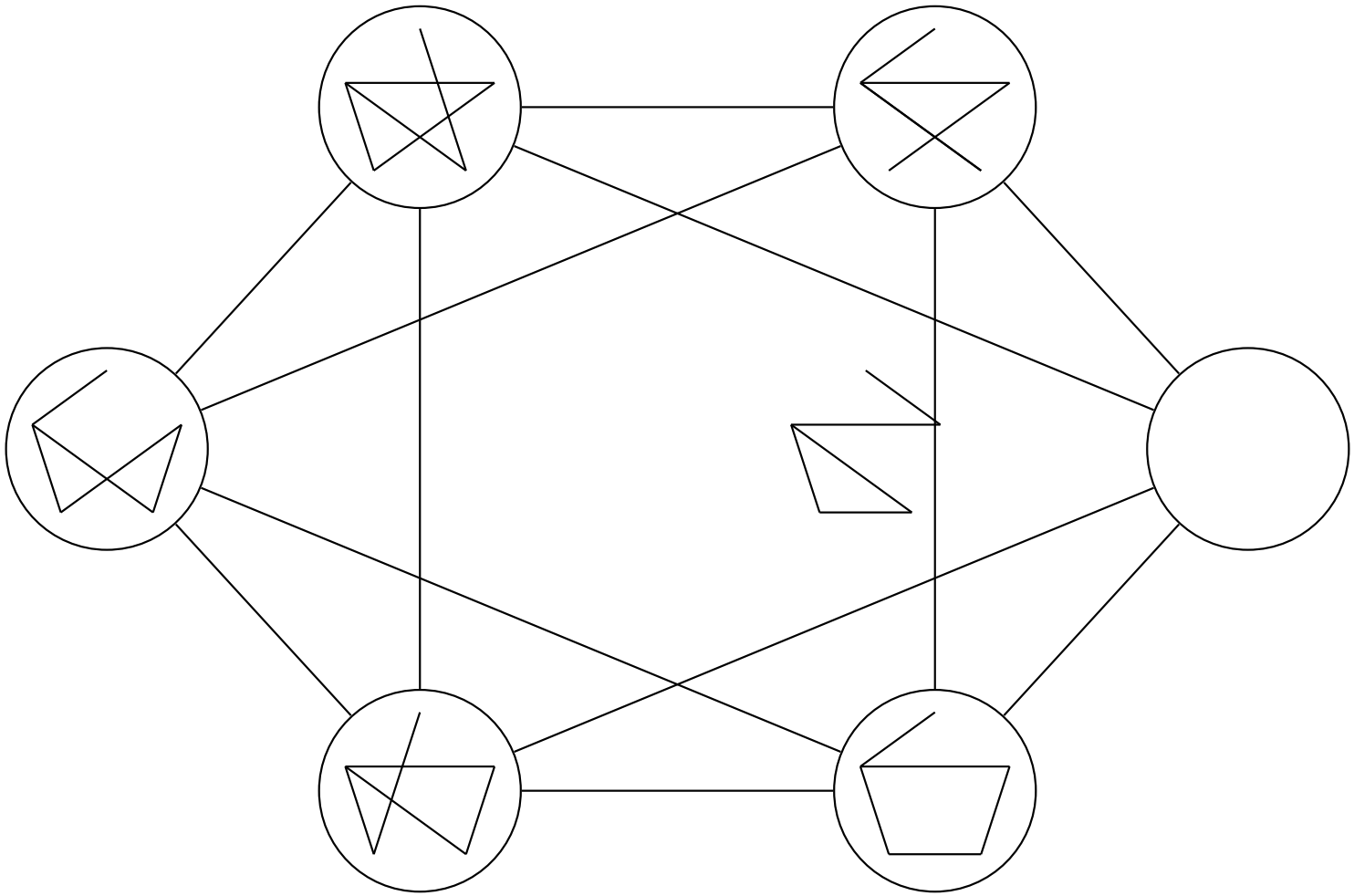
Example for Scopes



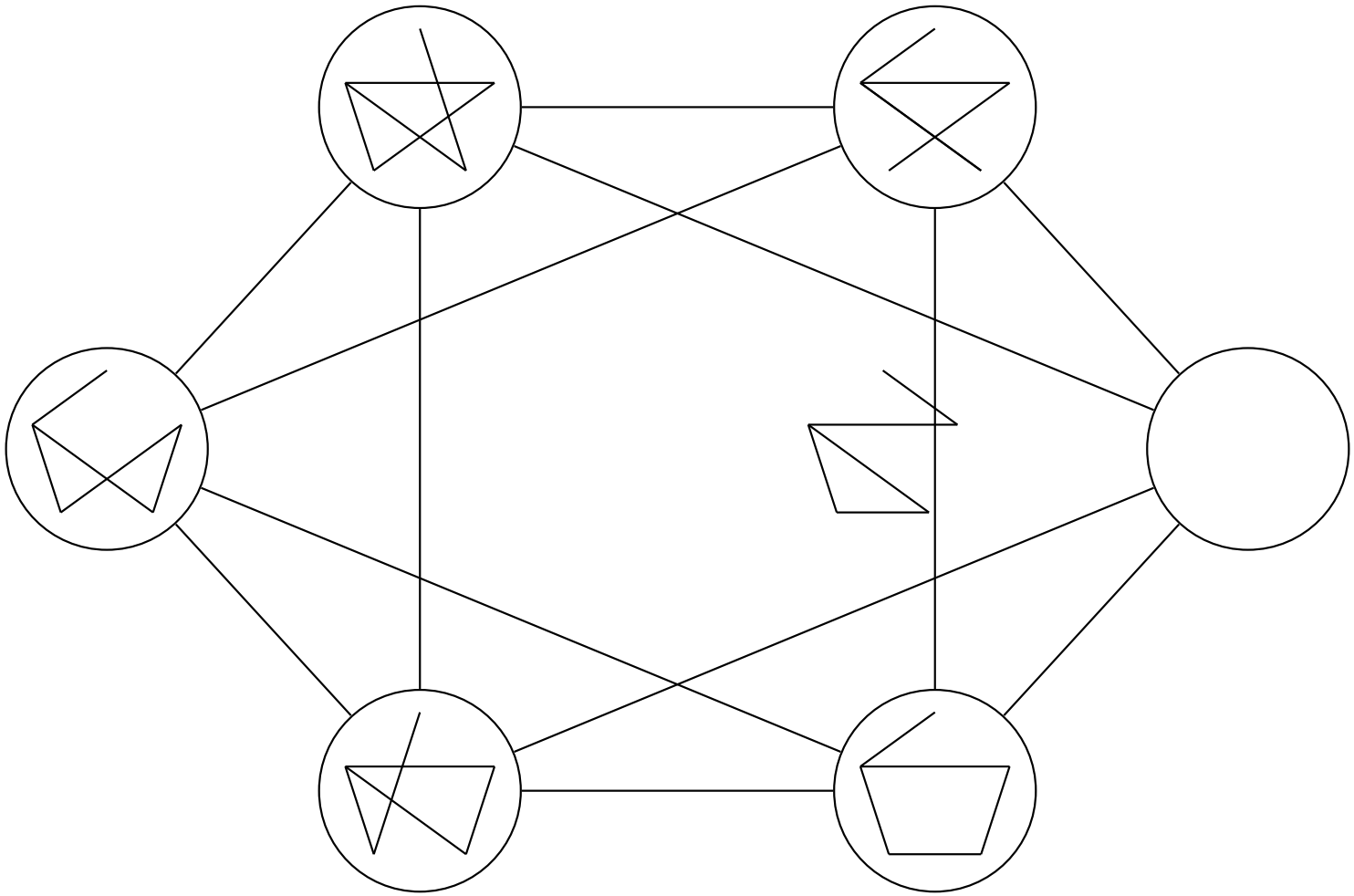
Example for Scopes



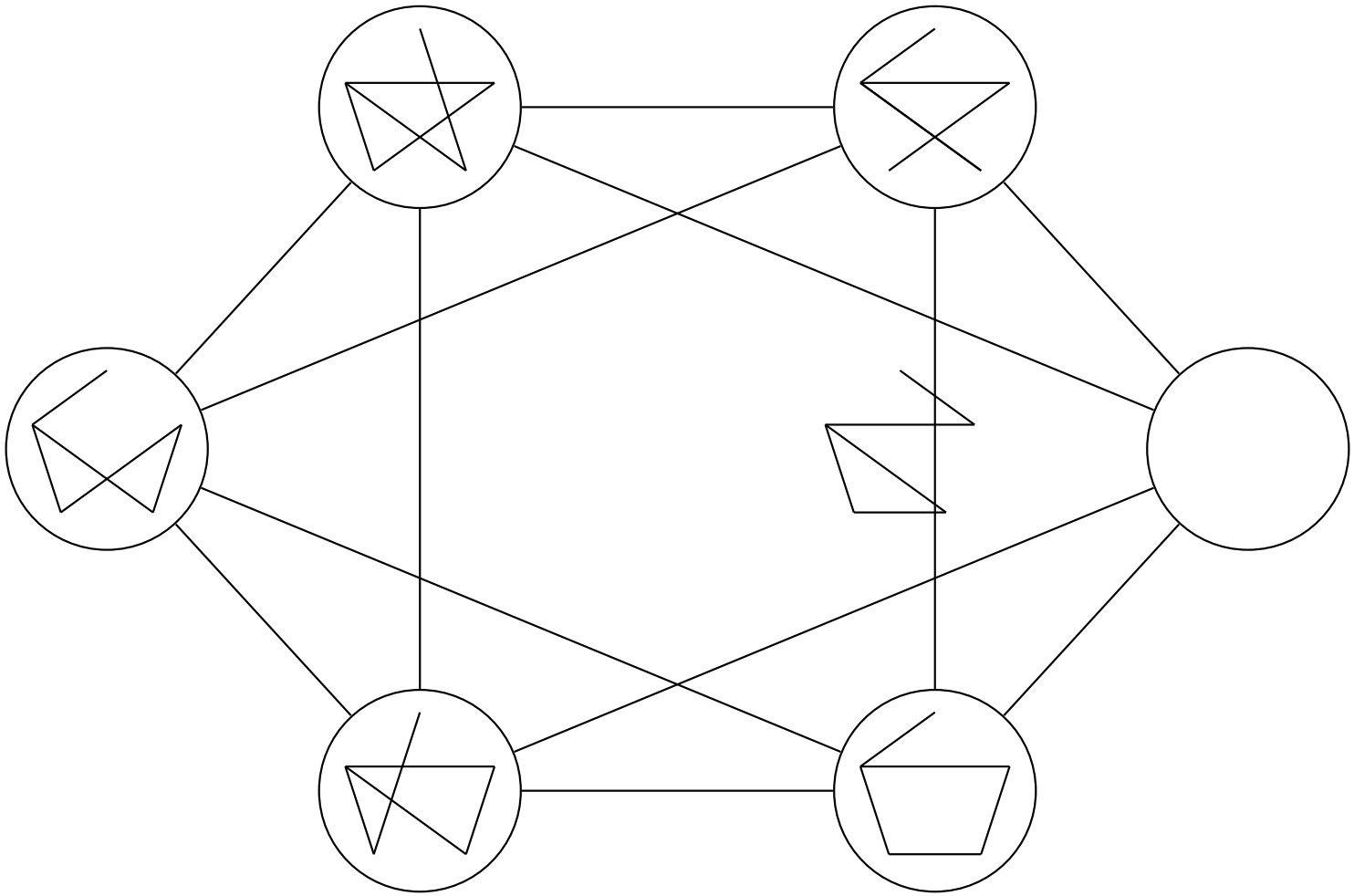
Example for Scopes



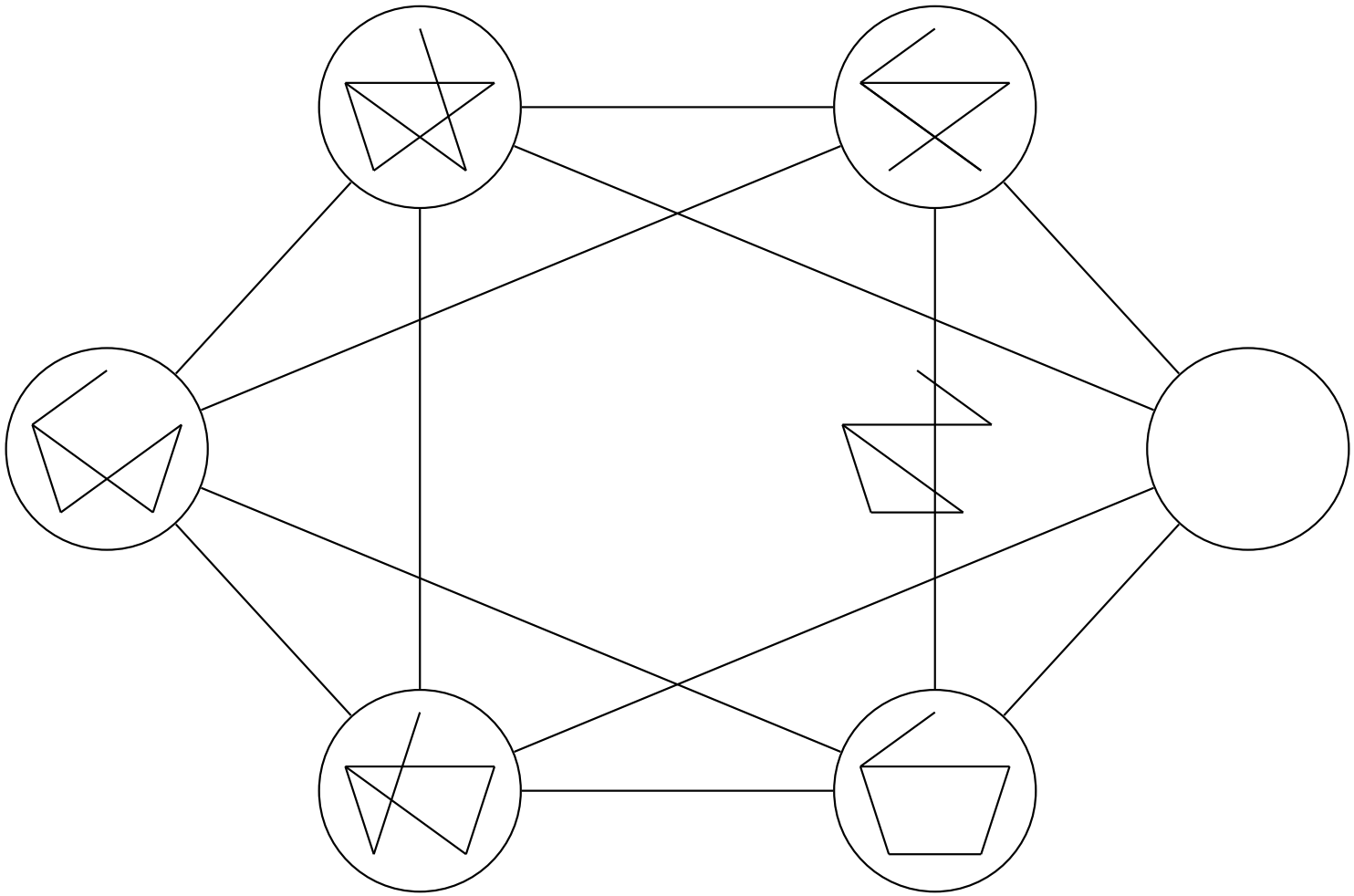
Example for Scopes



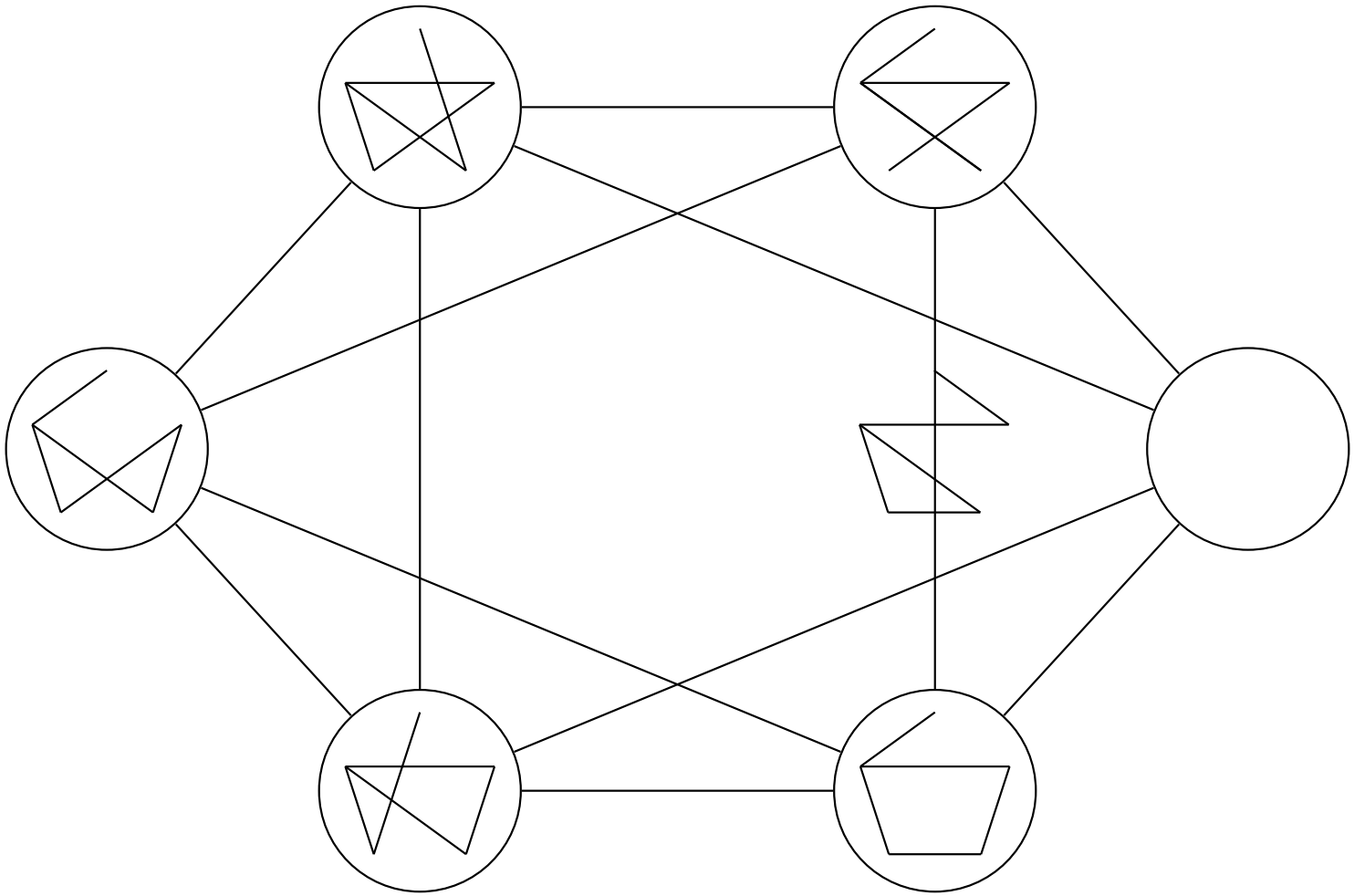
Example for Scopes



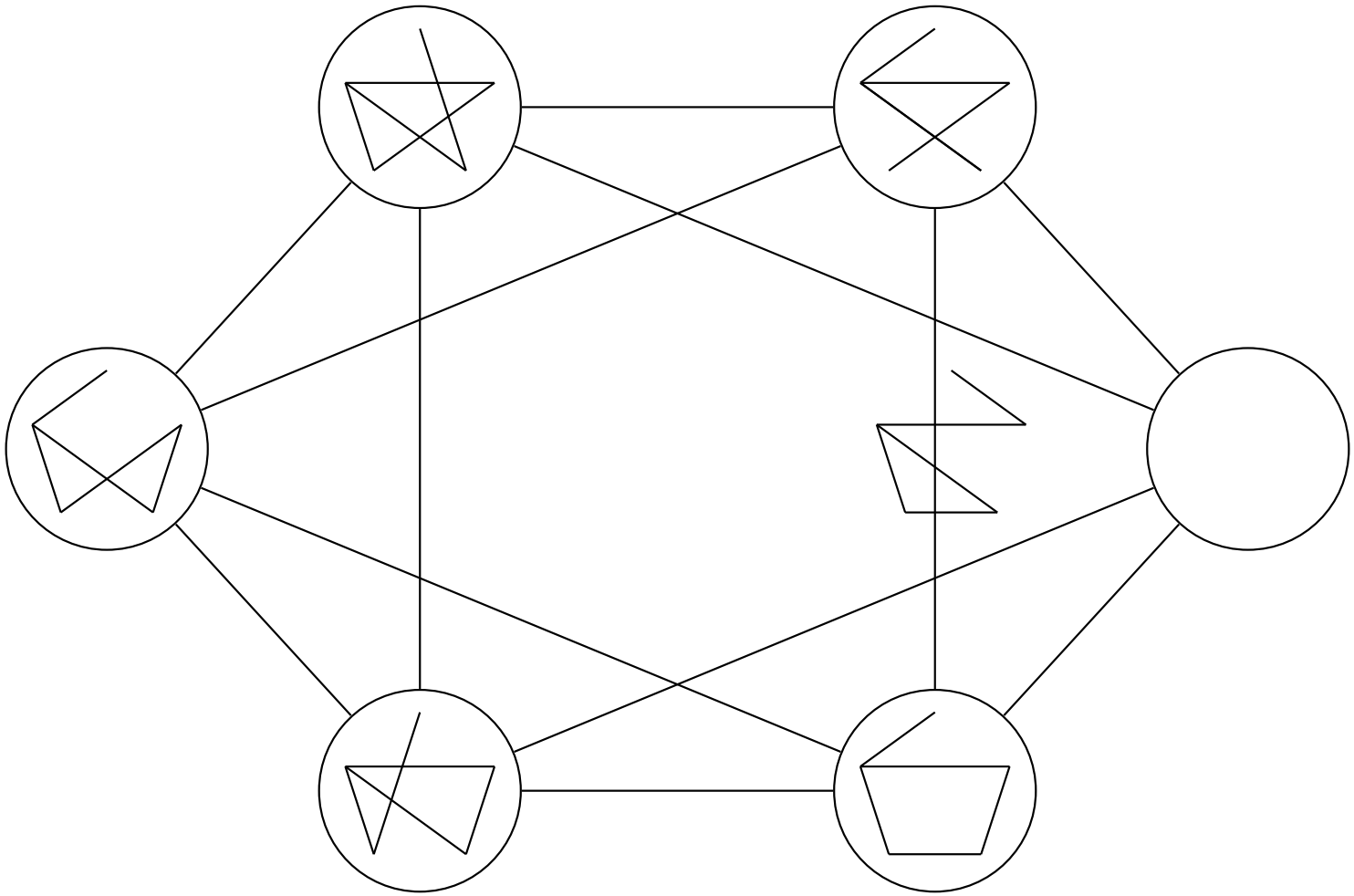
Example for Scopes



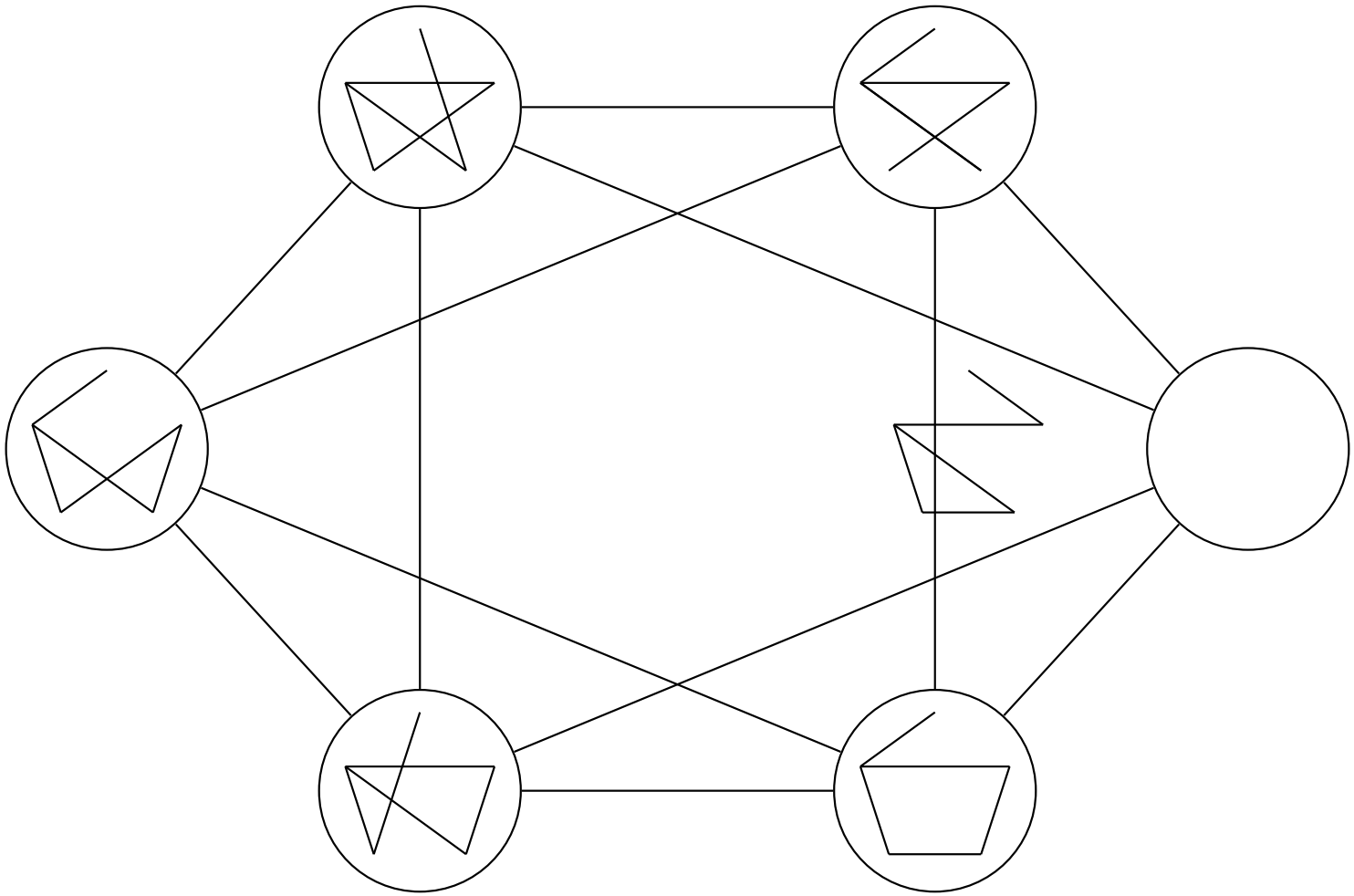
Example for Scopes



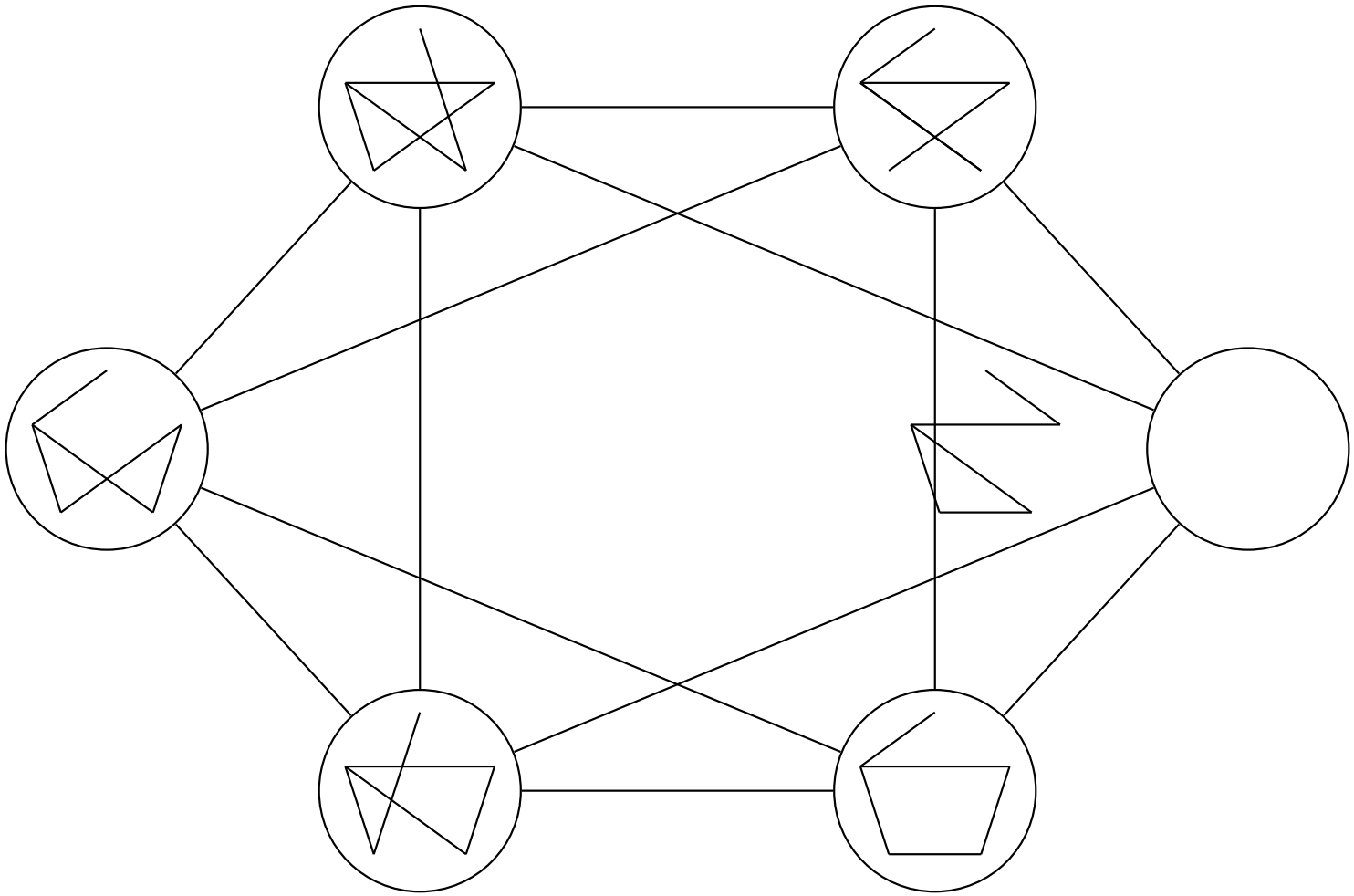
Example for Scopes



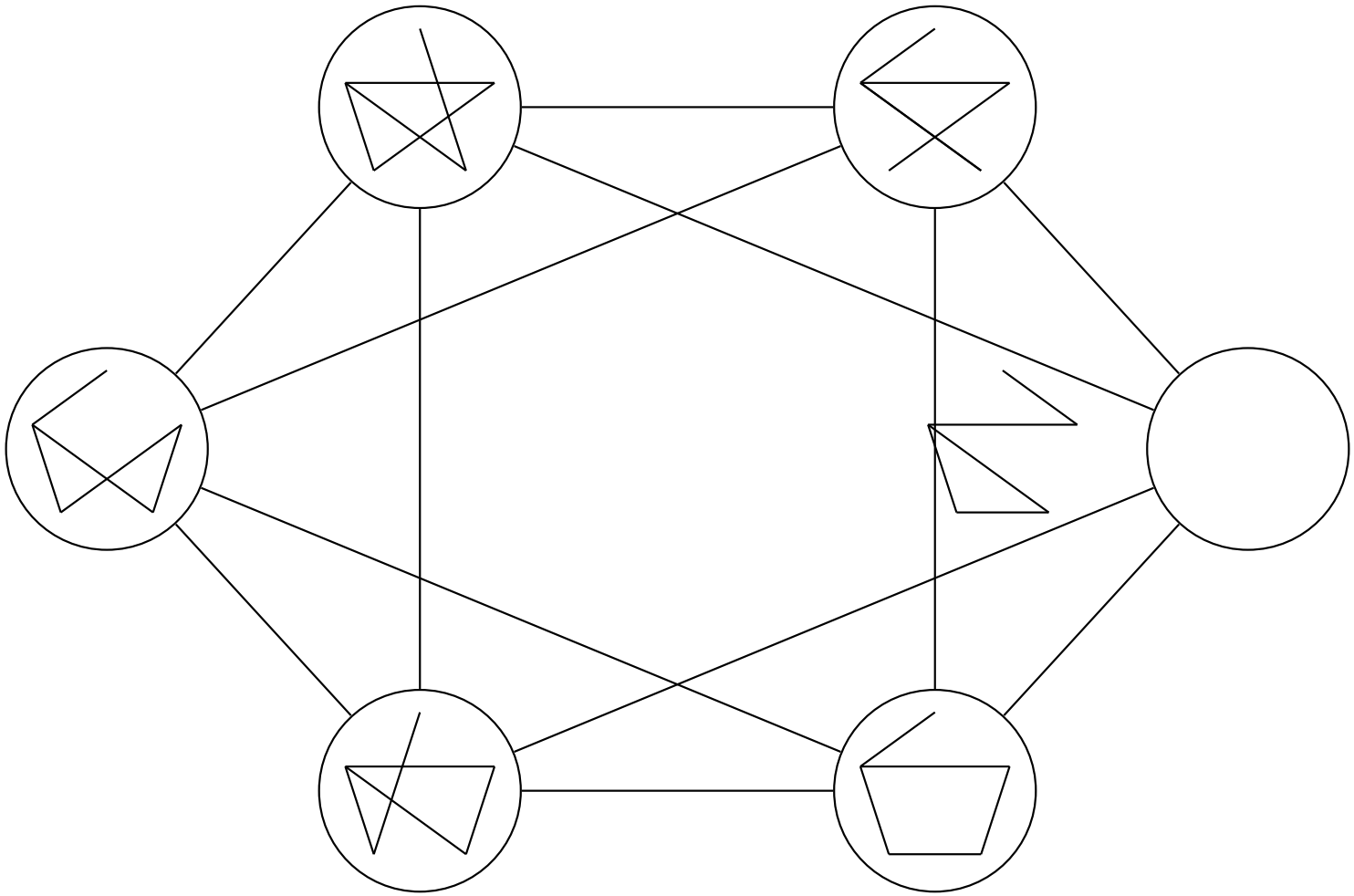
Example for Scopes



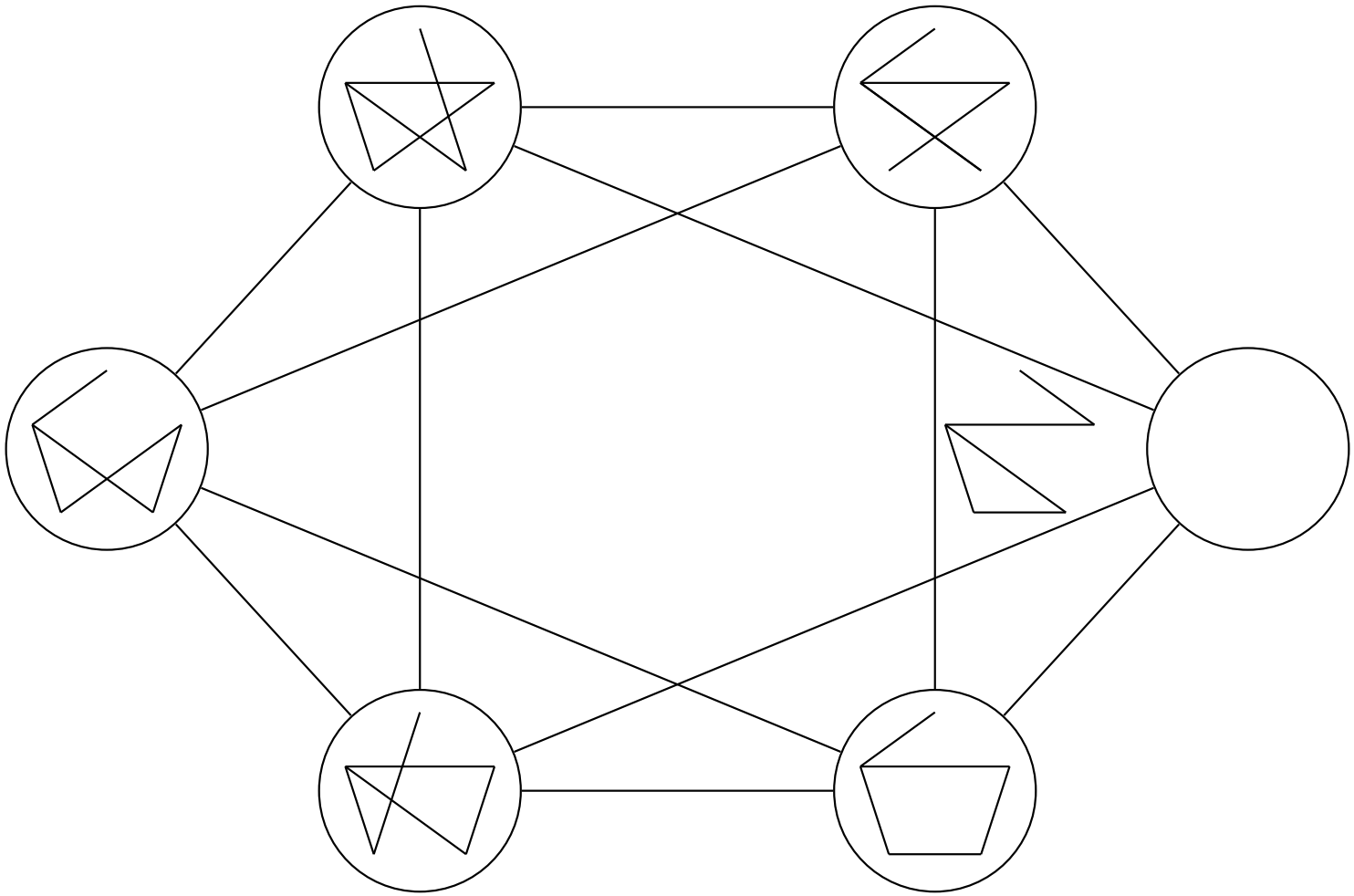
Example for Scopes



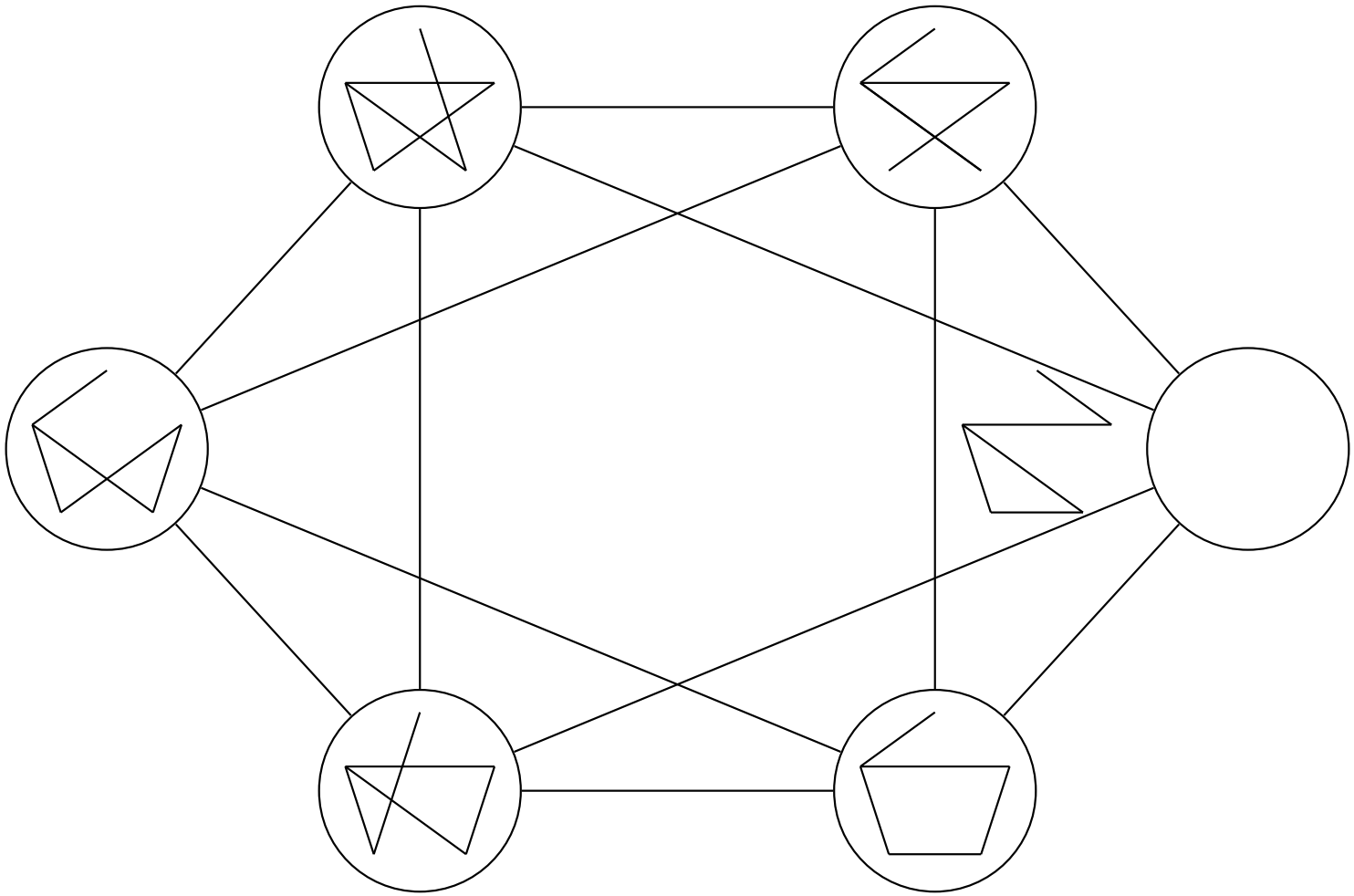
Example for Scopes



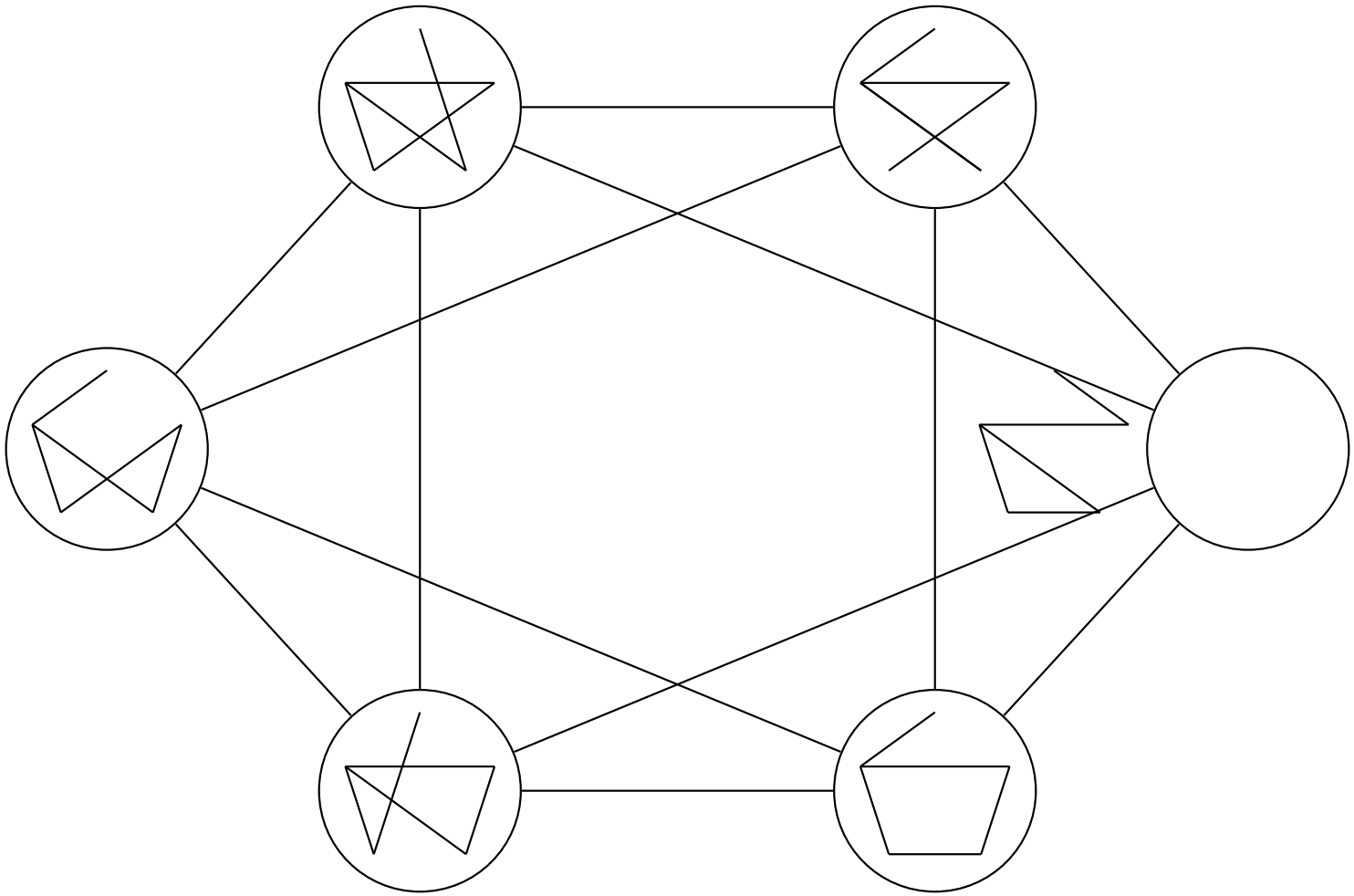
Example for Scopes



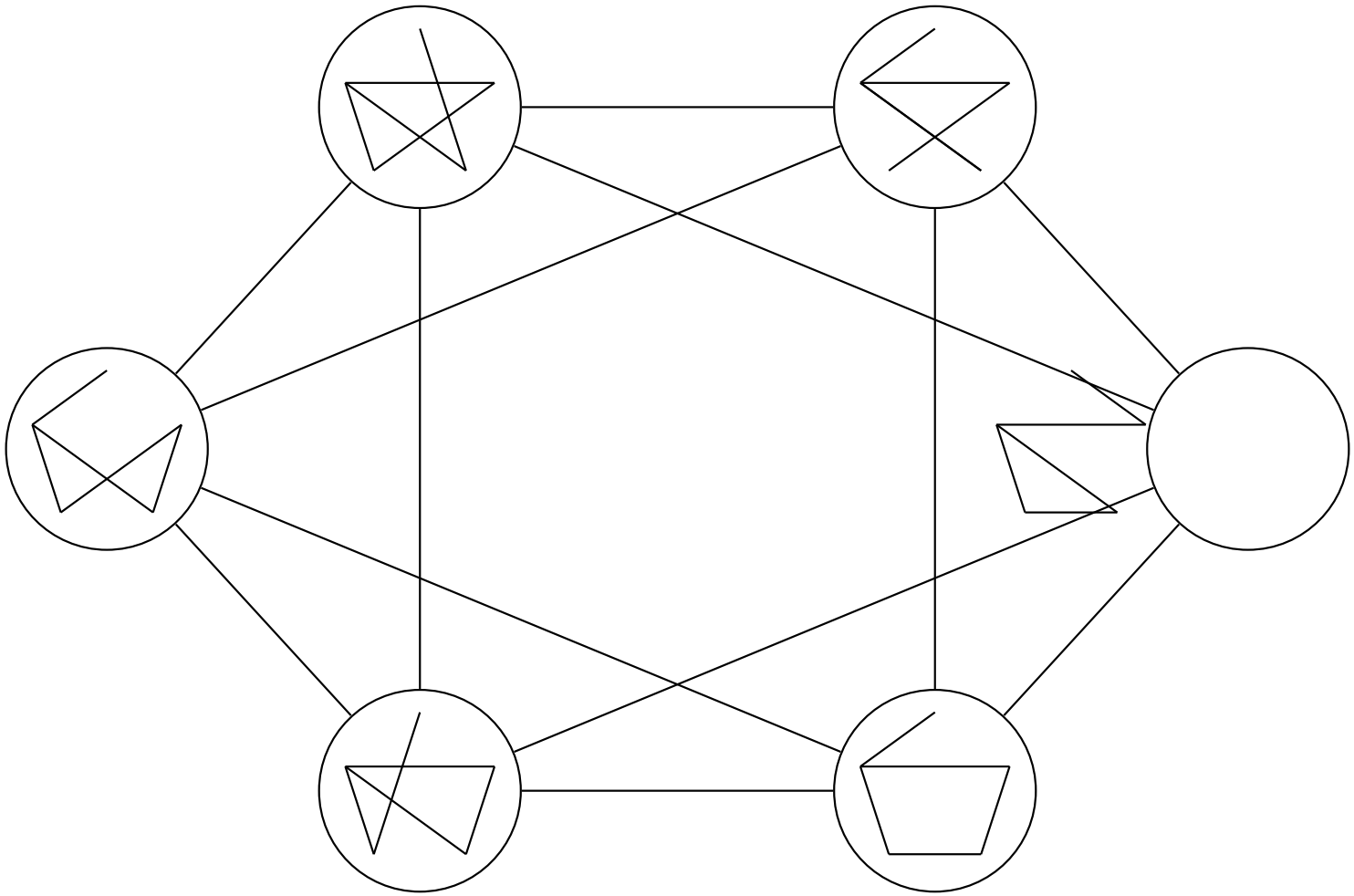
Example for Scopes



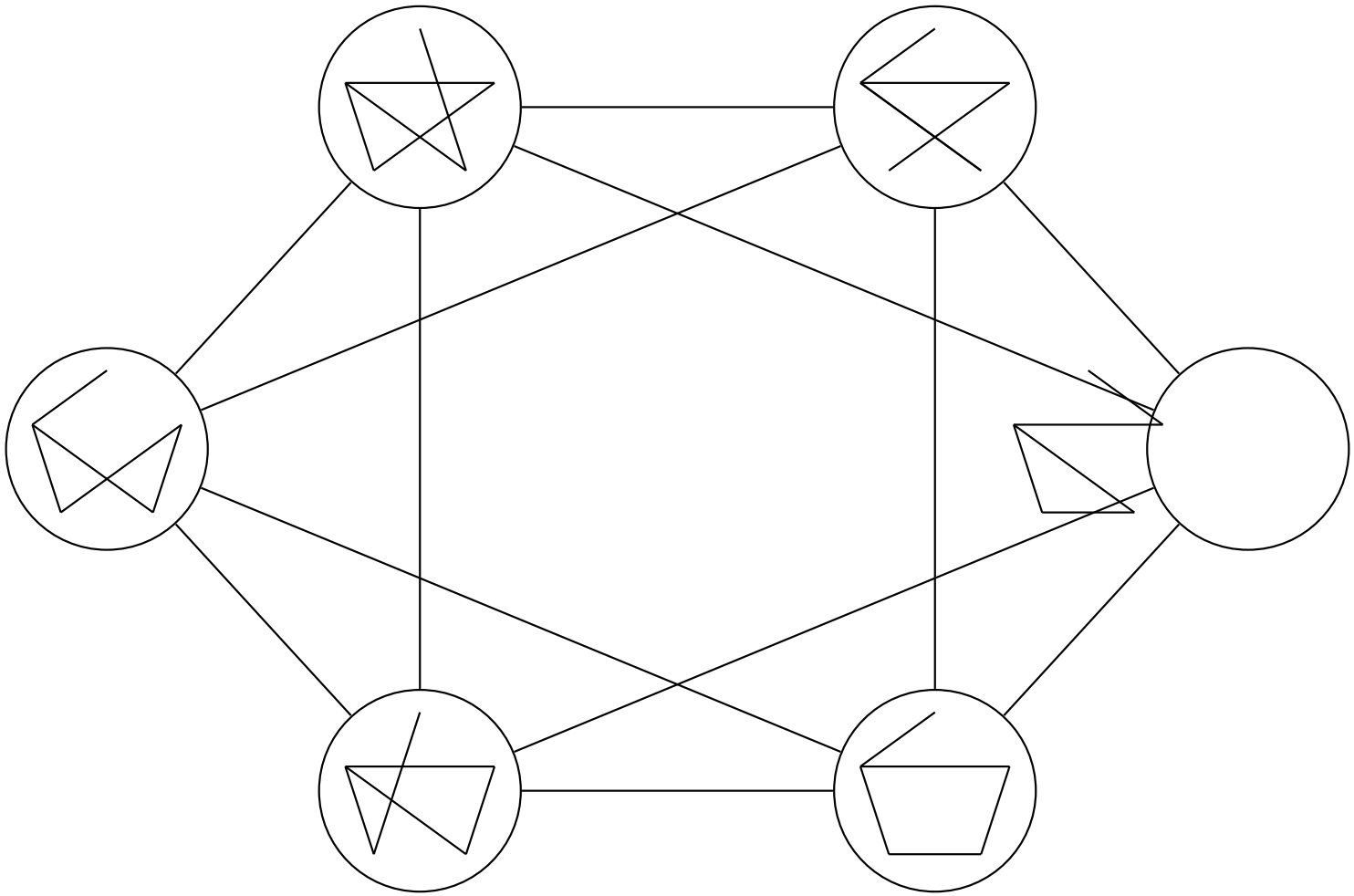
Example for Scopes



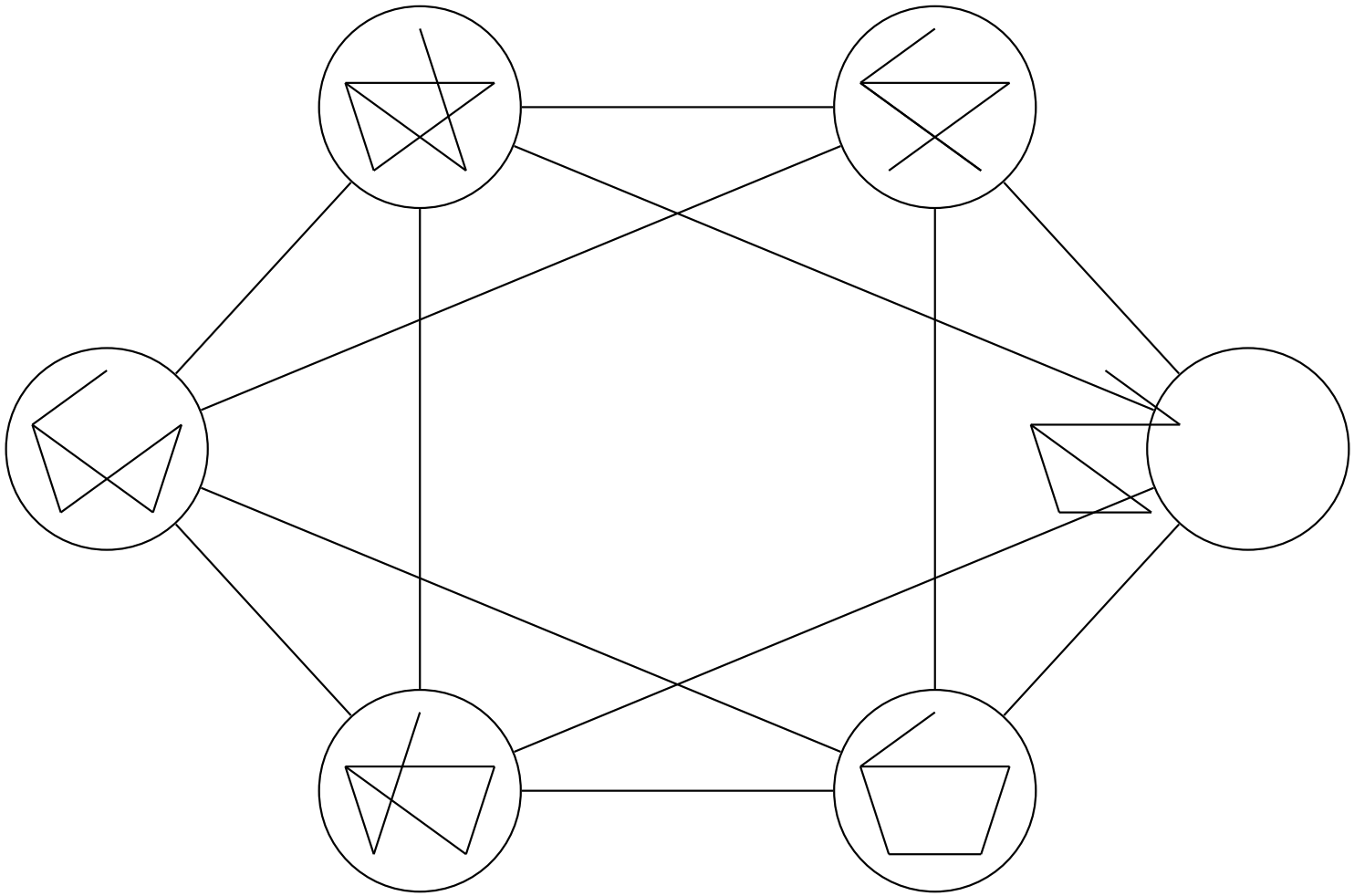
Example for Scopes



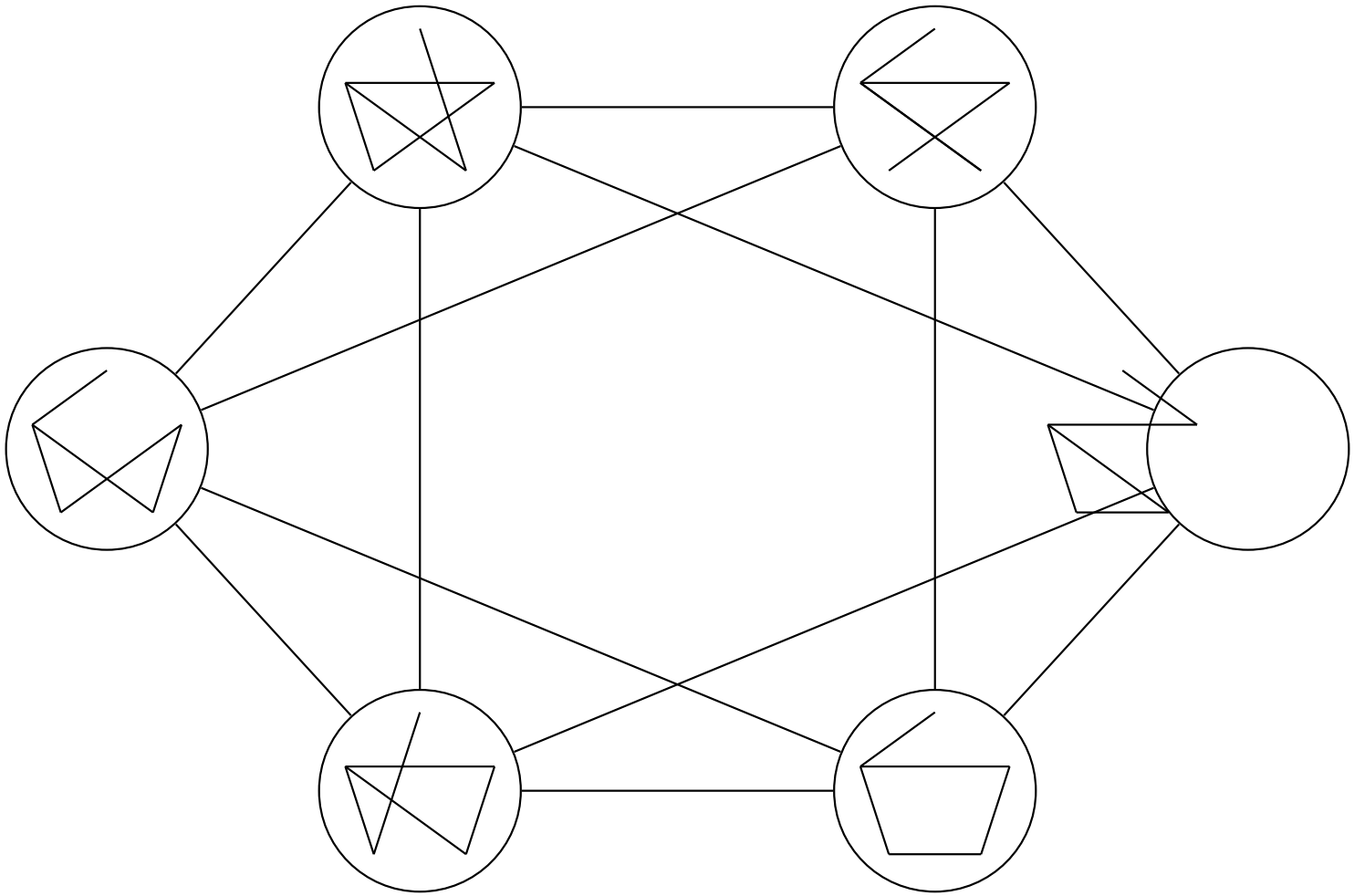
Example for Scopes



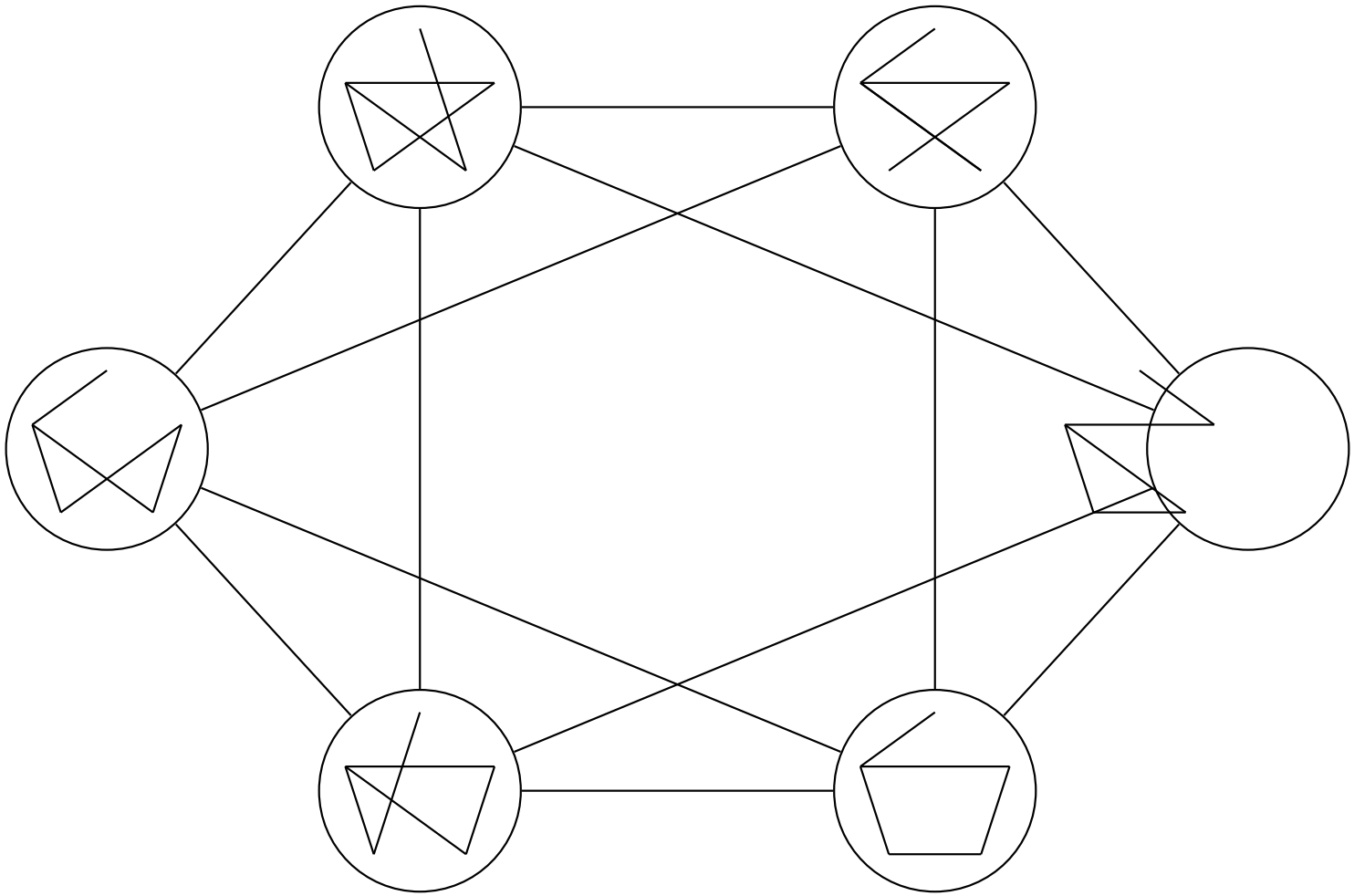
Example for Scopes



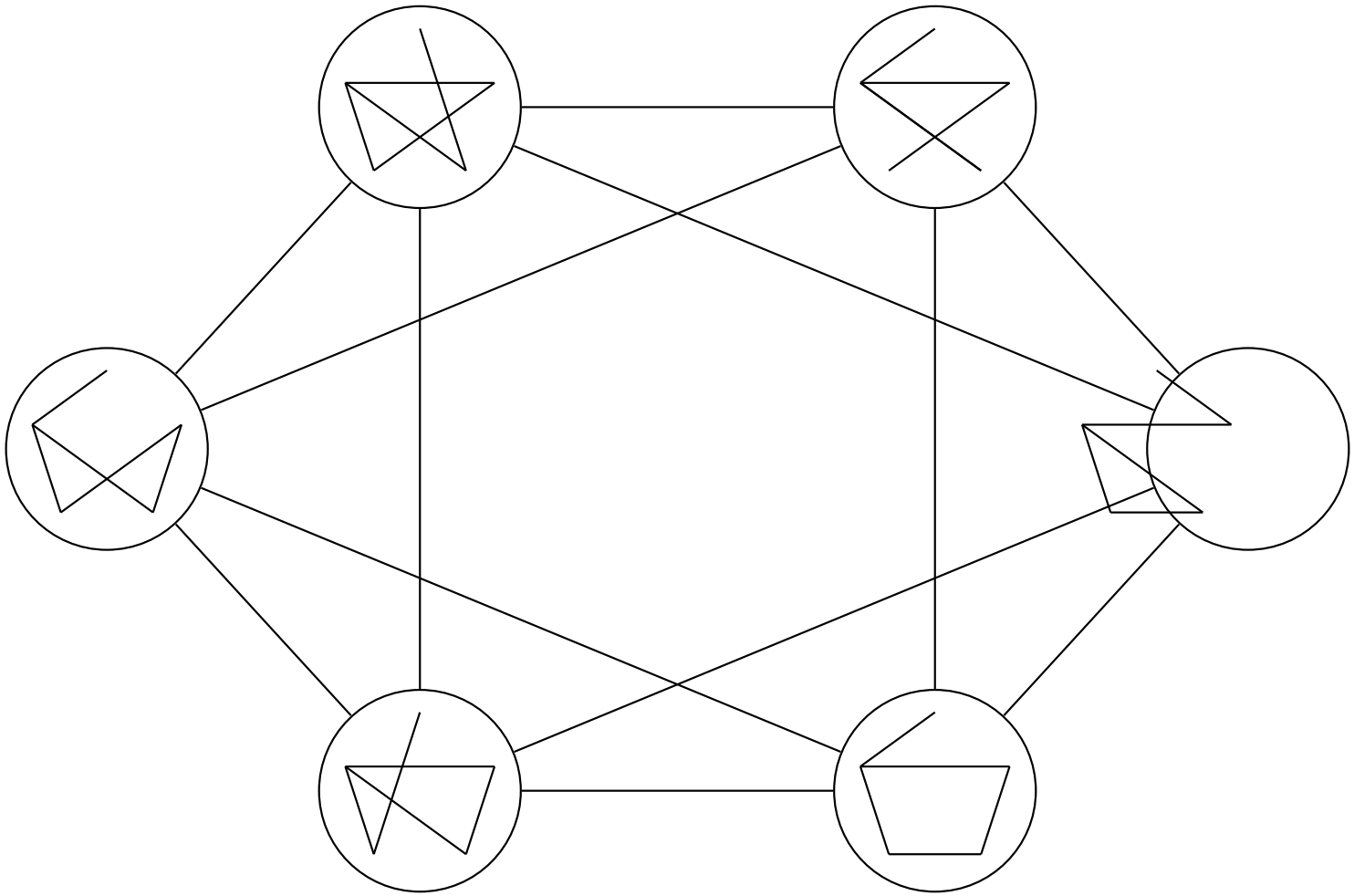
Example for Scopes



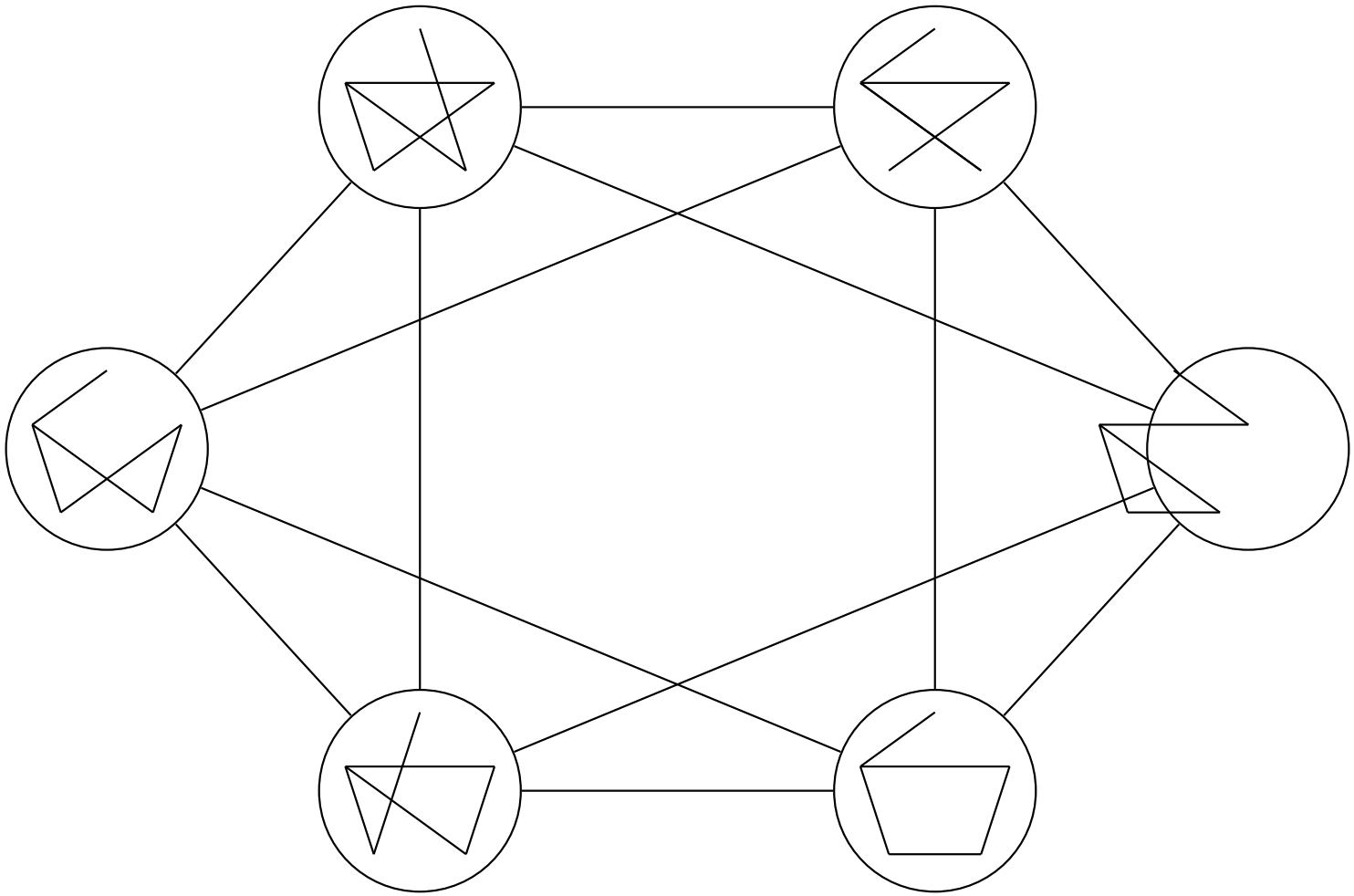
Example for Scopes



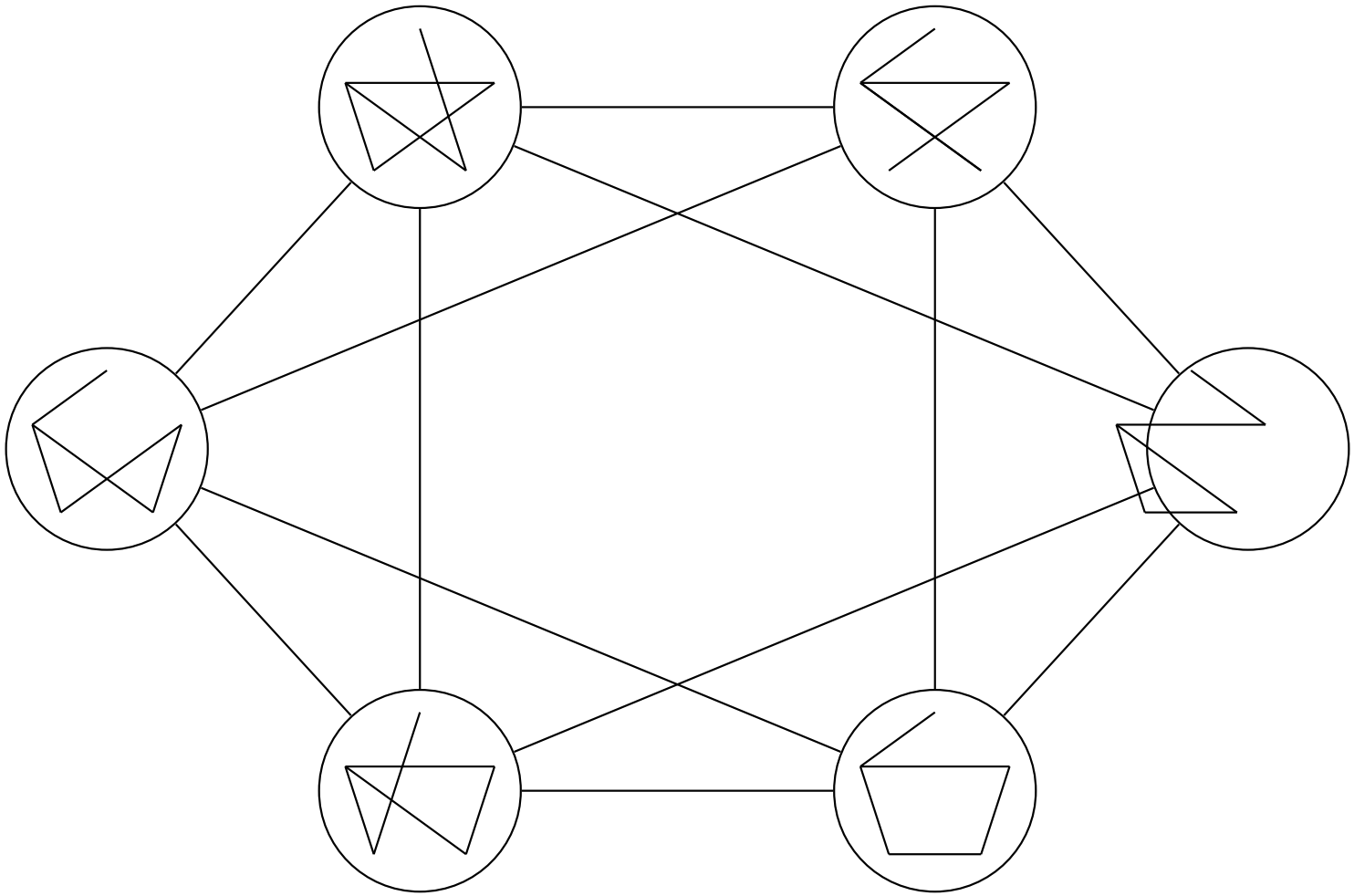
Example for Scopes



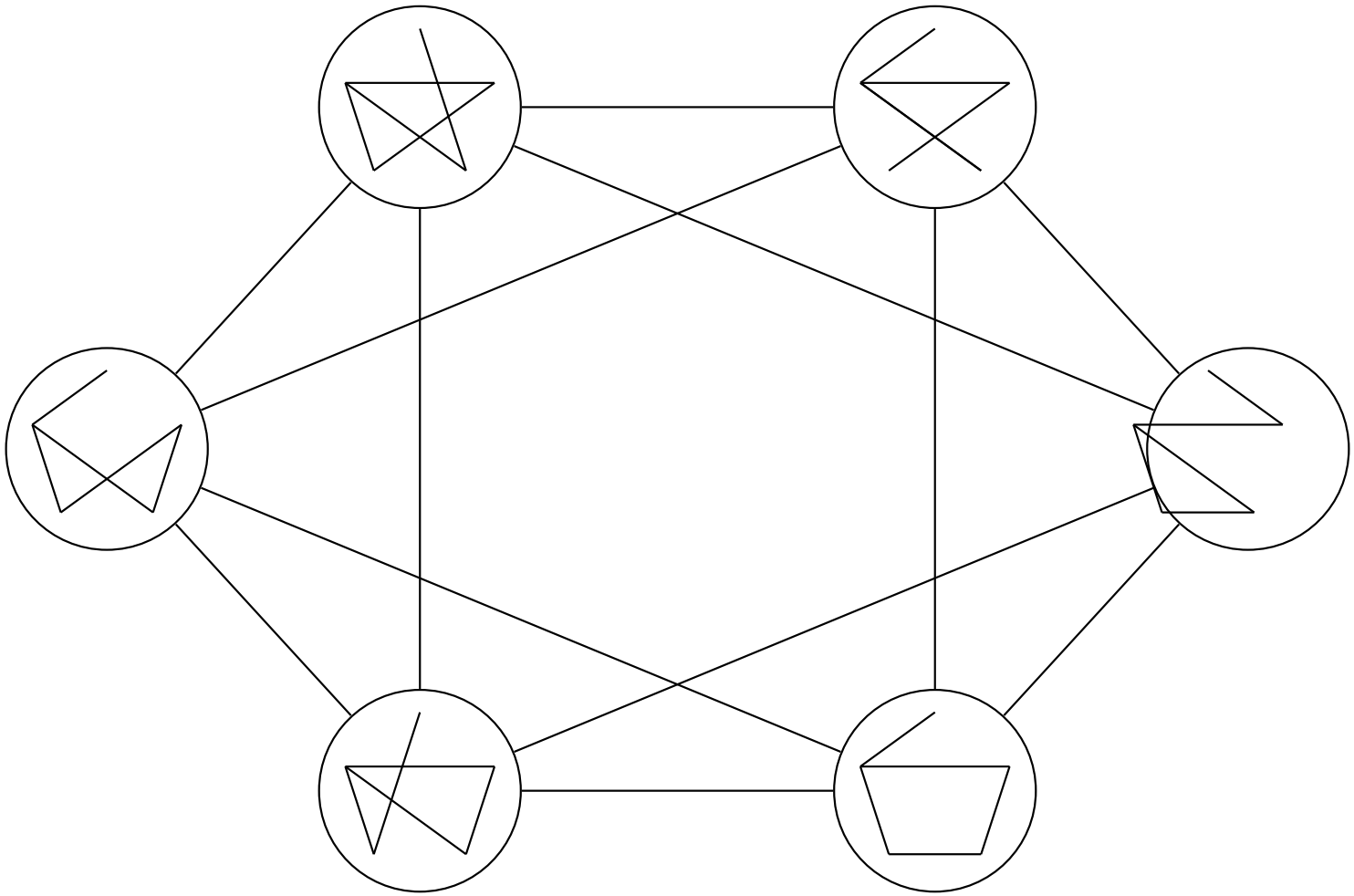
Example for Scopes



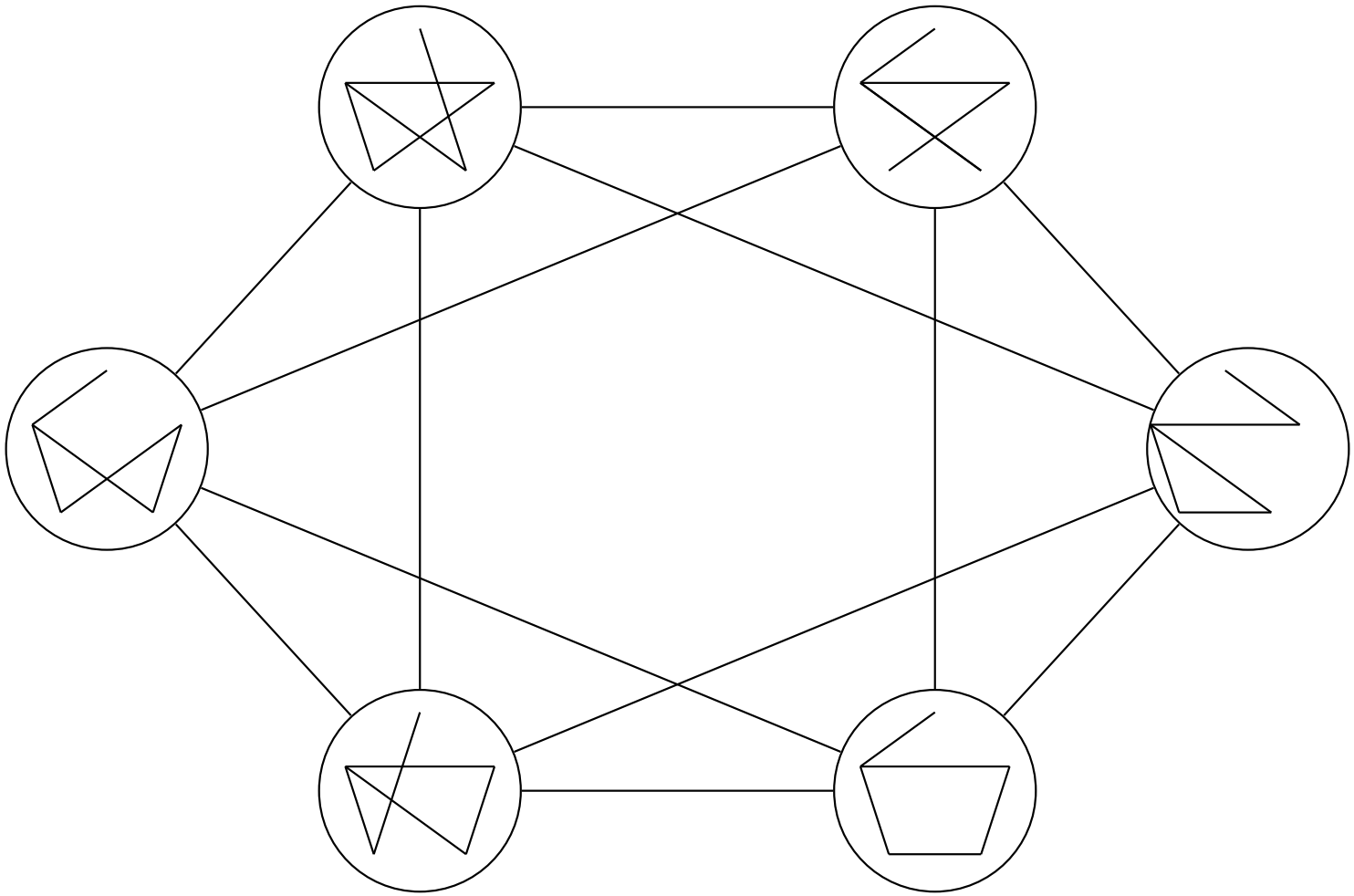
Example for Scopes



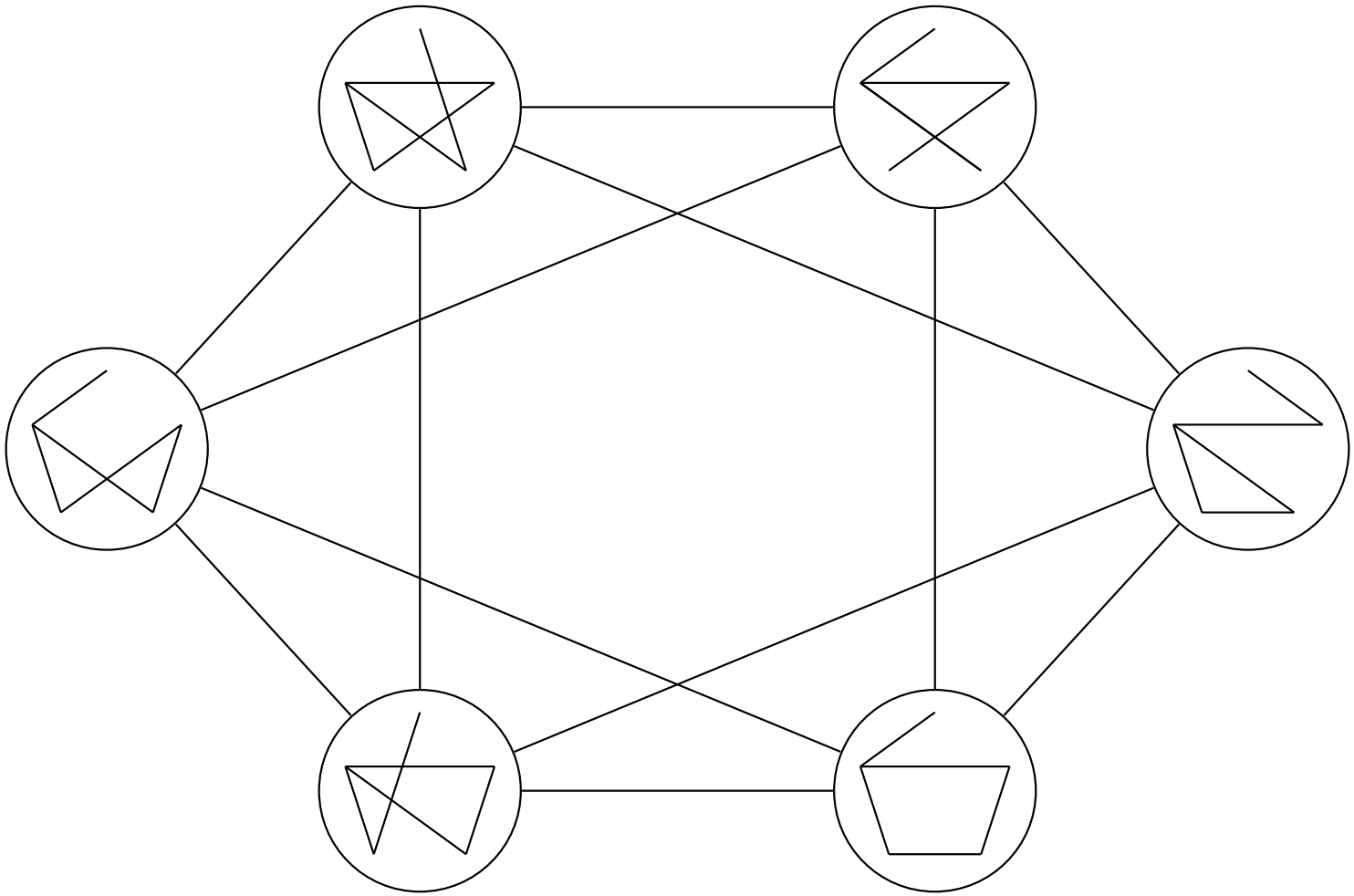
Example for Scopes



Example for Scopes



Example for Scopes



Two Ways to Add Animations

Beamer

- animations can be used once
- one slide in the file per frame

Animation Package

- animations can be restarted
- infinite loops possible
- embedded into a single slide

Need a lot of computation!
Take care with integral/floating point calculations!

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

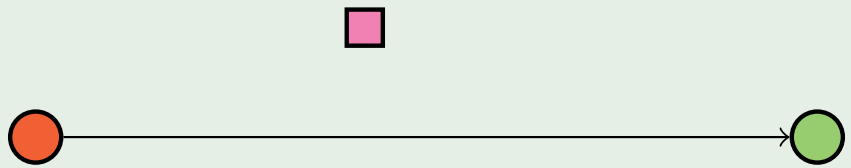
Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

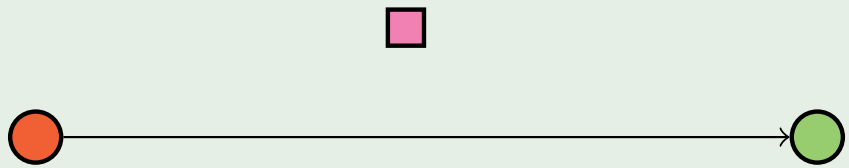


Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

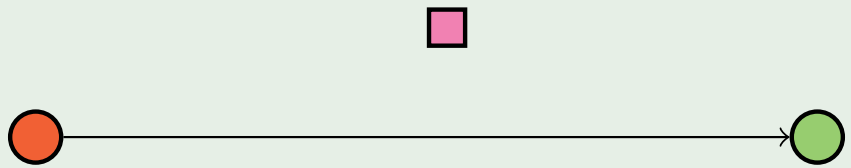


Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

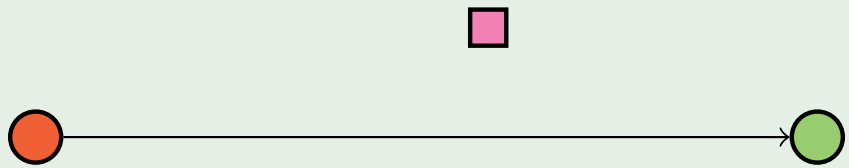


Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc



Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

Animation Parameter

- needs a variable: `newdimen varname`
- animated frames defined by `animate<2-19>`
- variable contains values as in `animatedvalue`

Animations Using Beamer Style

Packet on an arc

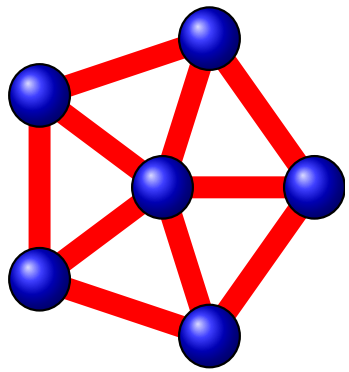
```
\newdimen\pos
\animate<2-19>
\animatevalue<1-20>{\position}{0cm}{5cm}
\begin{tikzpicture}
  \node (x) at (\position,0.7) [draw,thick,fill=CarnationPink
  \node (a) at (0,0) [draw,circle,fill=RedOrange,thick] {};
  \node (b) at (5,0) [draw,circle,fill=YellowGreen,thick] {}
    edge [<-] (a);
\end{tikzpicture}
```

Animations with the Animate Package

```
\begin{animateinline}[autoplay,loop]{24}
  \multiframe{48}{rAngle=0+1.5}{
    \animateLogo{\rAngle}
  }
\end{animateinline}
```

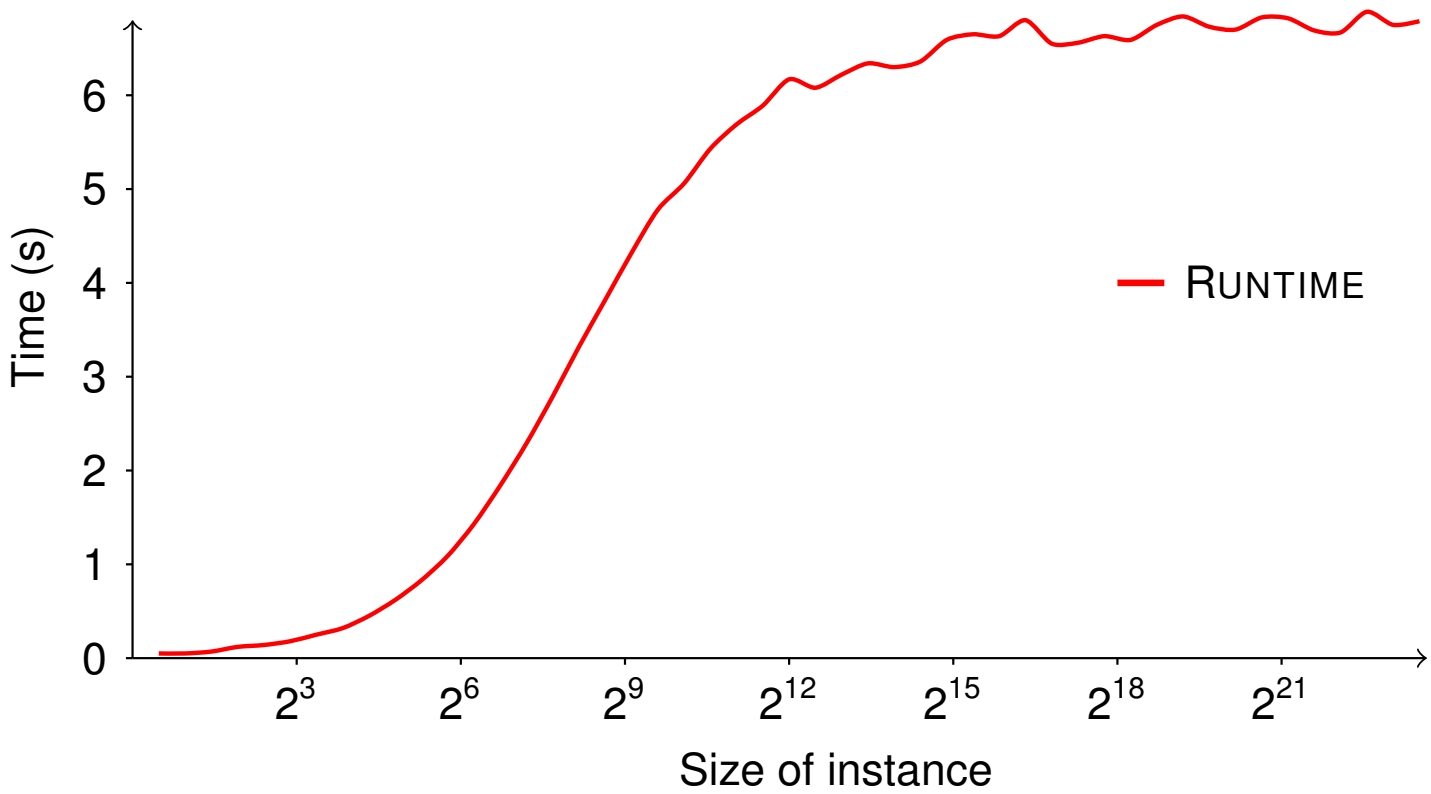
Animations with the Animate Package

```
\begin{animateinline}[autoplay,loop]{24}
  \multiframe{48}{rAngle=0+1.5}{
    \animateLogo{\rAngle}
  }
\end{animateinline}
```



Plotting in TikZ

Something like this is possible in TikZ:



But: quite lengthy code, as axes and legend have to be drawn manually

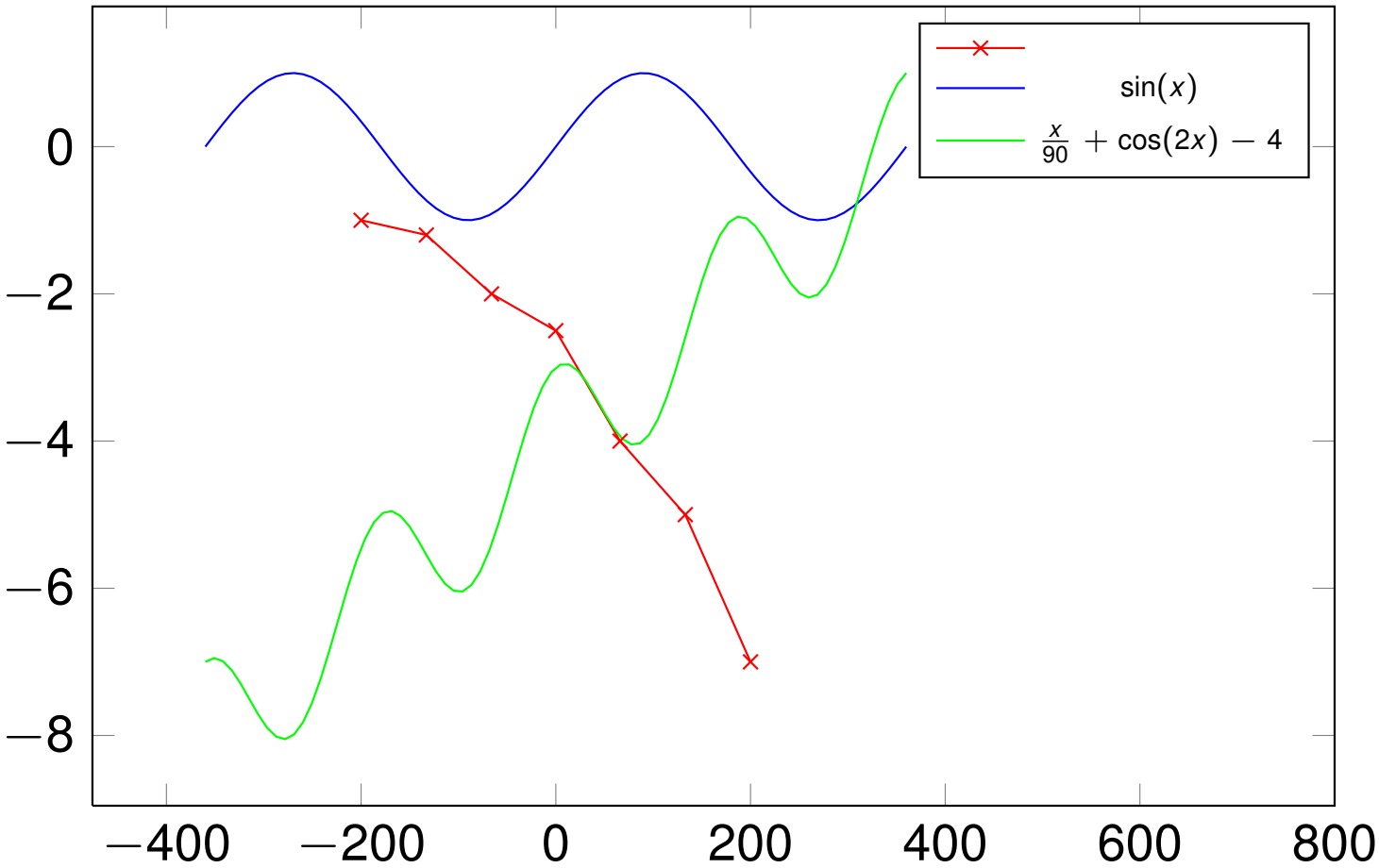
Plotting using PGFPLOTS

The PGFPLOTS package

- package specialized for drawing plots
- based upon PGF/Ti*k*Z
- available at `http://sourceforge.net/projects/pgfplots`
- the manual is as good as the one of Ti*k*Z

On the following slides, there will be just three examples. For more, have a look in the manual.

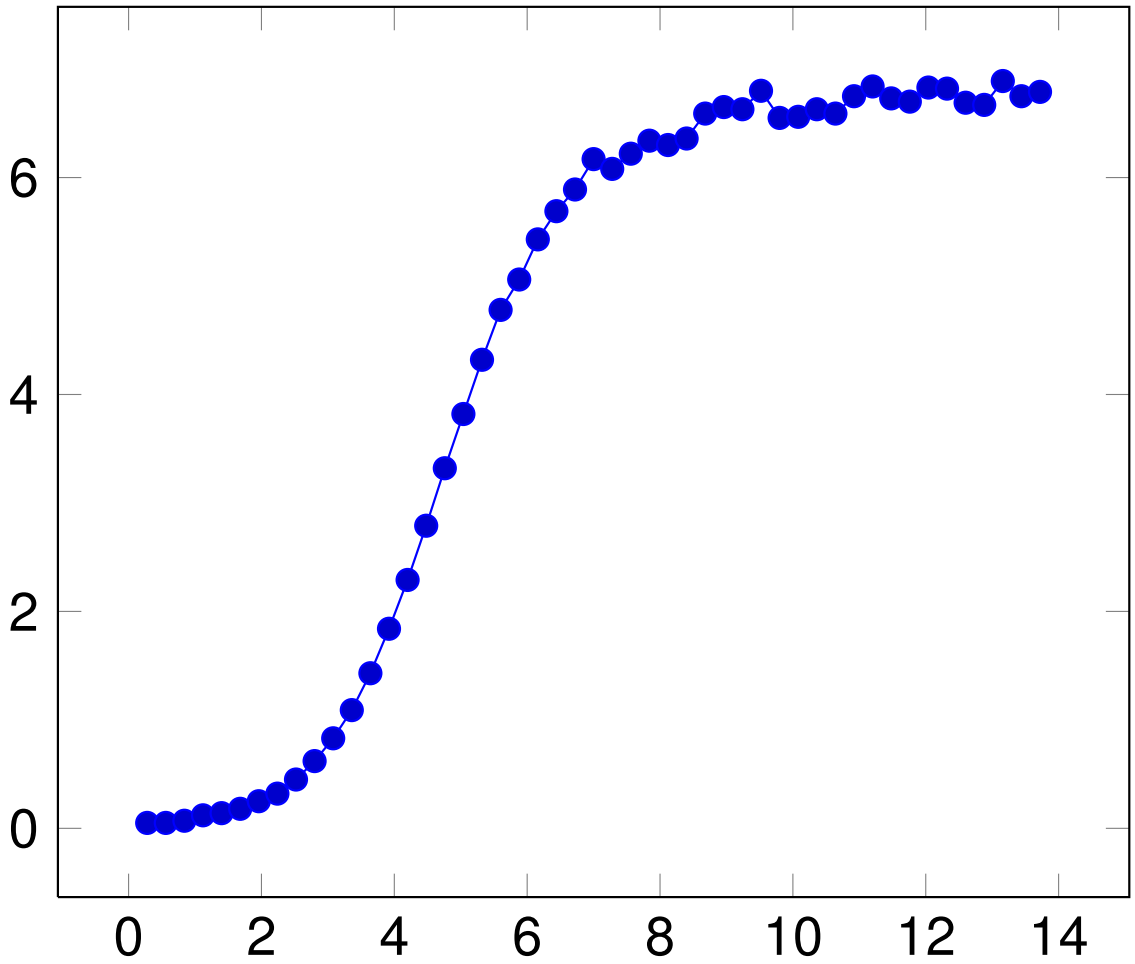
A Starting Example



A Starting Example

```
\begin{tikzpicture}
  \begin{axis}[domain=-360:360,samples=80,
               width=10cm,height=7cm,xmax=800]
    \addplot[color=red,mark=x] coordinates {
      (-200,-1)
      (-133,-1.2)
      (-66,-2)
      (0,-2.5)
      (66,-4)
      (133,-5)
      (200,-7)
    };
    \addplot[color=blue] {sin(x)};
    \addplot[color=green] {-4+x/90+cos(x*2)};
  \end{axis}
\end{tikzpicture}
```

Plotting from Files



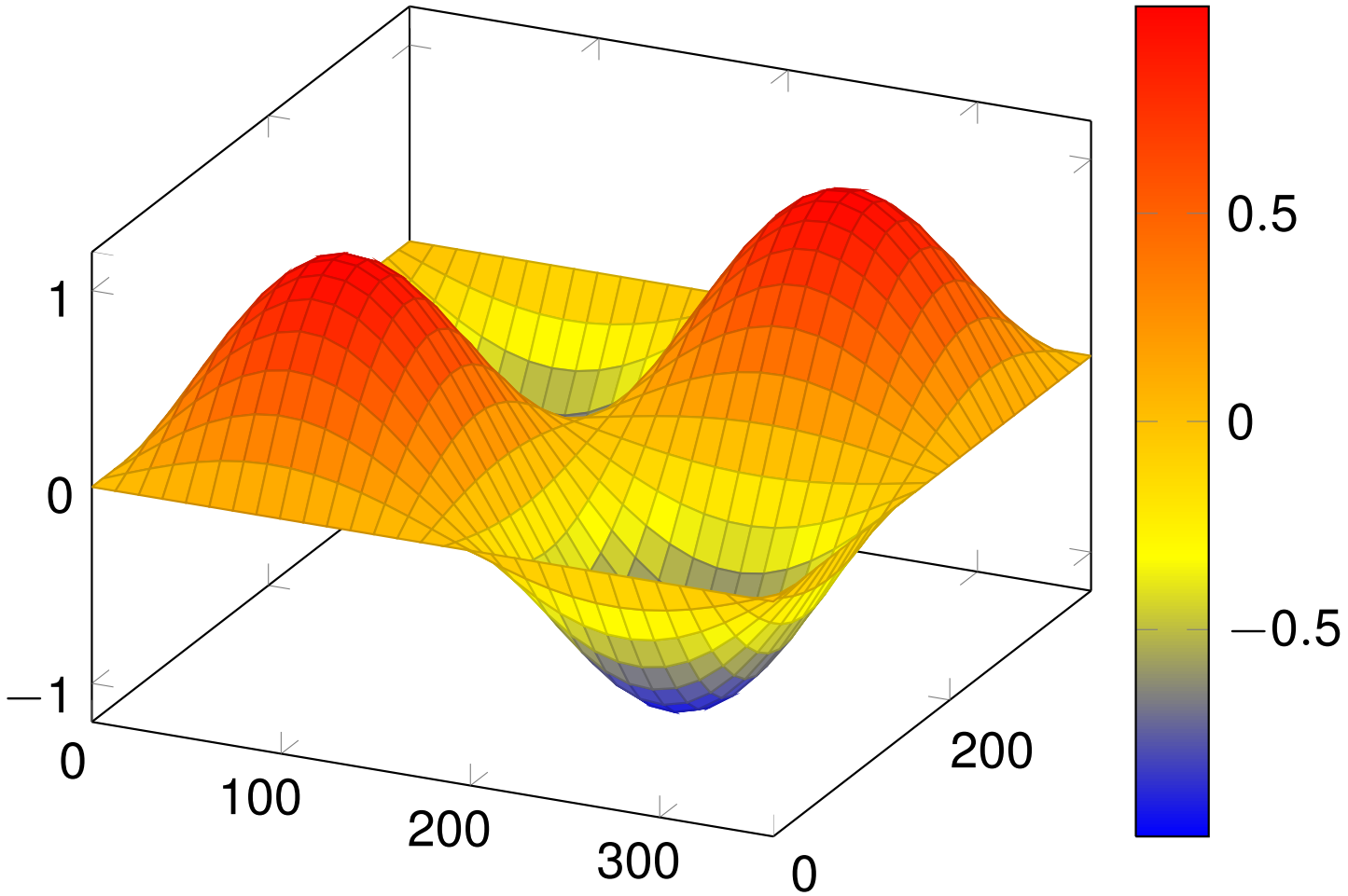
Plotting from Files

```
\begin{tikzpicture}
  \begin{axis}
    \addplot file {charts/data.table};
  \end{axis}
\end{tikzpicture}
```

File features

- reads gnuplot style files with datapoints specified as `x y i` with x and y beeing floating point values
- also specific rows of a table can be read
- for details, see the manual

Plots in 3D



Plots in 3D

```
\begin{tikzpicture}
  \begin{axis}
    \addplot3[surf,domain=0:360,samples=50]
      {sin(x)*sin(y)};
  \end{axis}
\end{tikzpicture}
```

Take care, high sample values are not possible due to memory limitations!

If You Still Miss a Powerpoint Feature

Do you really miss anything?

If You Still Miss a Powerpoint Feature

Do you really miss anything?

If you really, really need, just do, it's easy!

If You Still Miss a Powerpoint Feature

Do you really miss anything?

If you really, really need, just do, it's easy!

```
\node[starburst,fill=yellow,draw=red,line width=2pt] at
```

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds! };
```

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds! };
```

`cloud callout` – the shape name

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn’t mind having\\these fancy clouds! };
```

`cloud puffs` – the number of puffs of the cloud

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn’t mind having\\these fancy clouds! };
```

`cloud puff arc` – the angle between two meeting puffs

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds! };
```

`callout pointer segments` – the number of round bubbles

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds! };
```

`callout relative pointer` – the angle and distance of the pointer

The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds! };
```

`aspect` – ratio between width and height

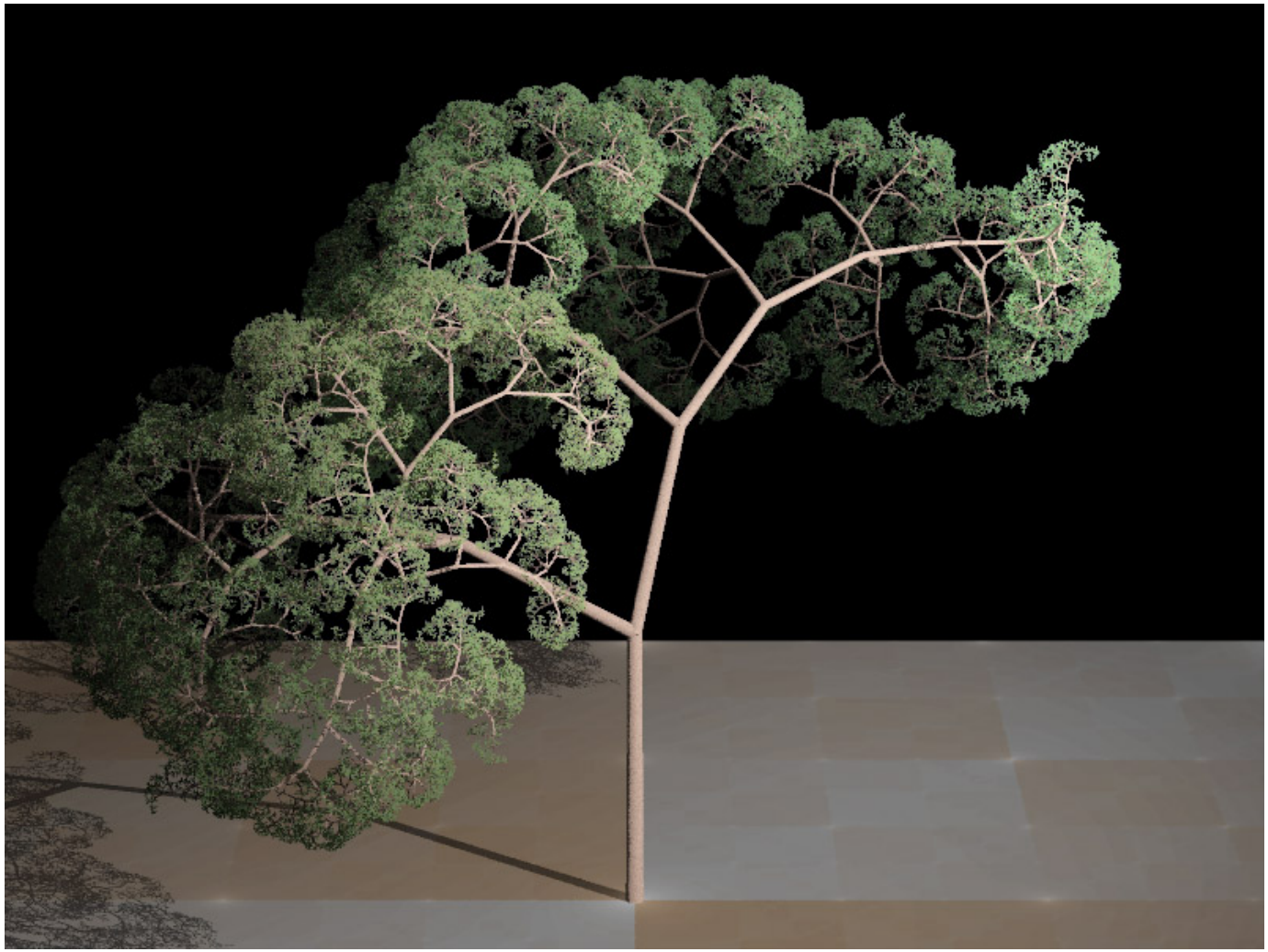
The Cloud Code

Example

```
\node[align=center,draw,shading=ball,text=white,
    cloud callout,cloud puffs=17,cloud puff arc=140,
    callout pointer segments=3,anchor=pointer,
    callout relative pointer={(200:2cm)},aspect=2.5]
    at (current page.center)
    { I wouldn't mind having\\these fancy clouds! };
```

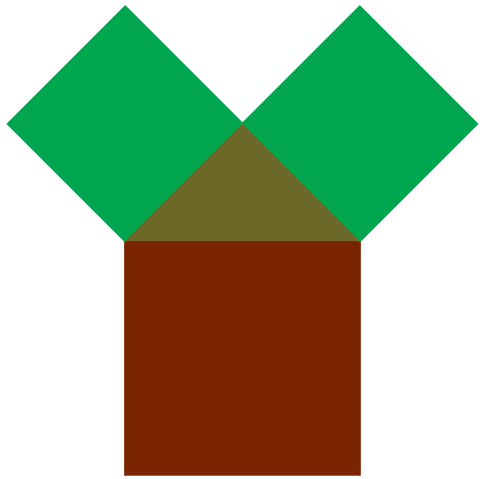
`current page.center` – *absolute* coordinate on the page

Trees

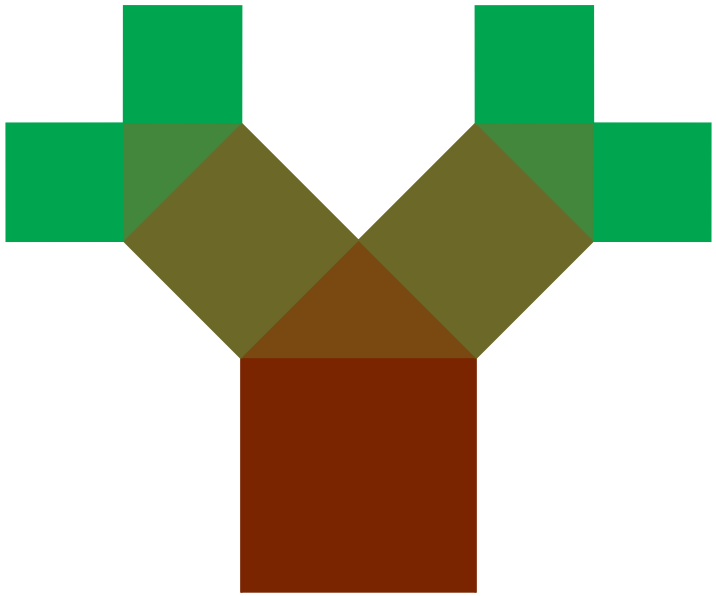


Trees

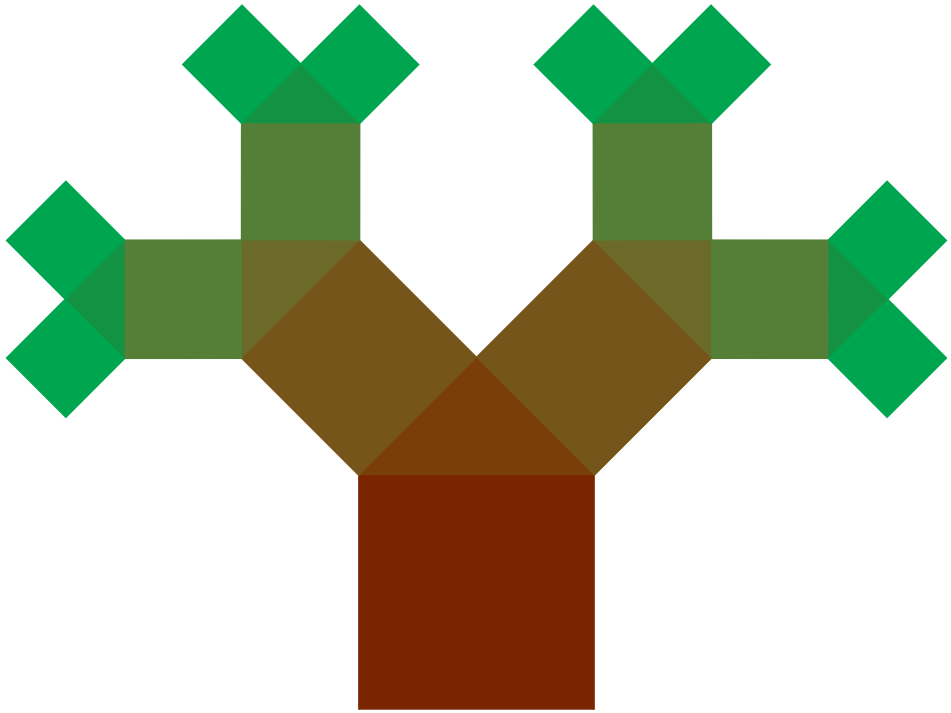
Trees



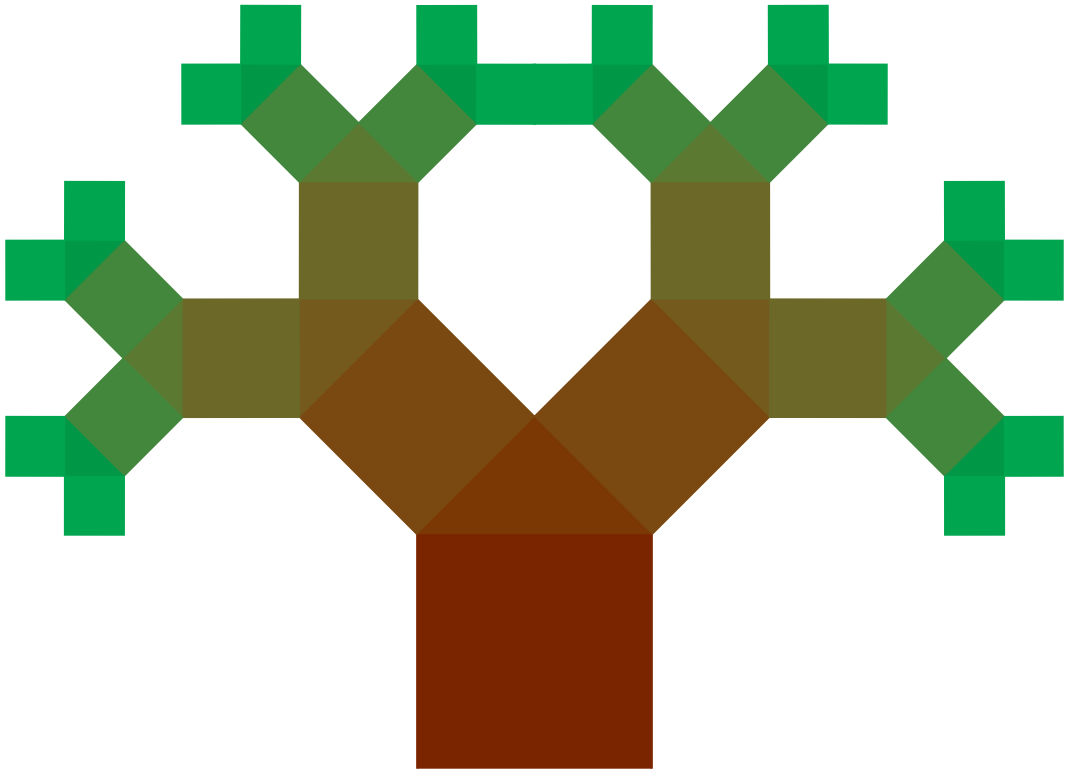
Trees



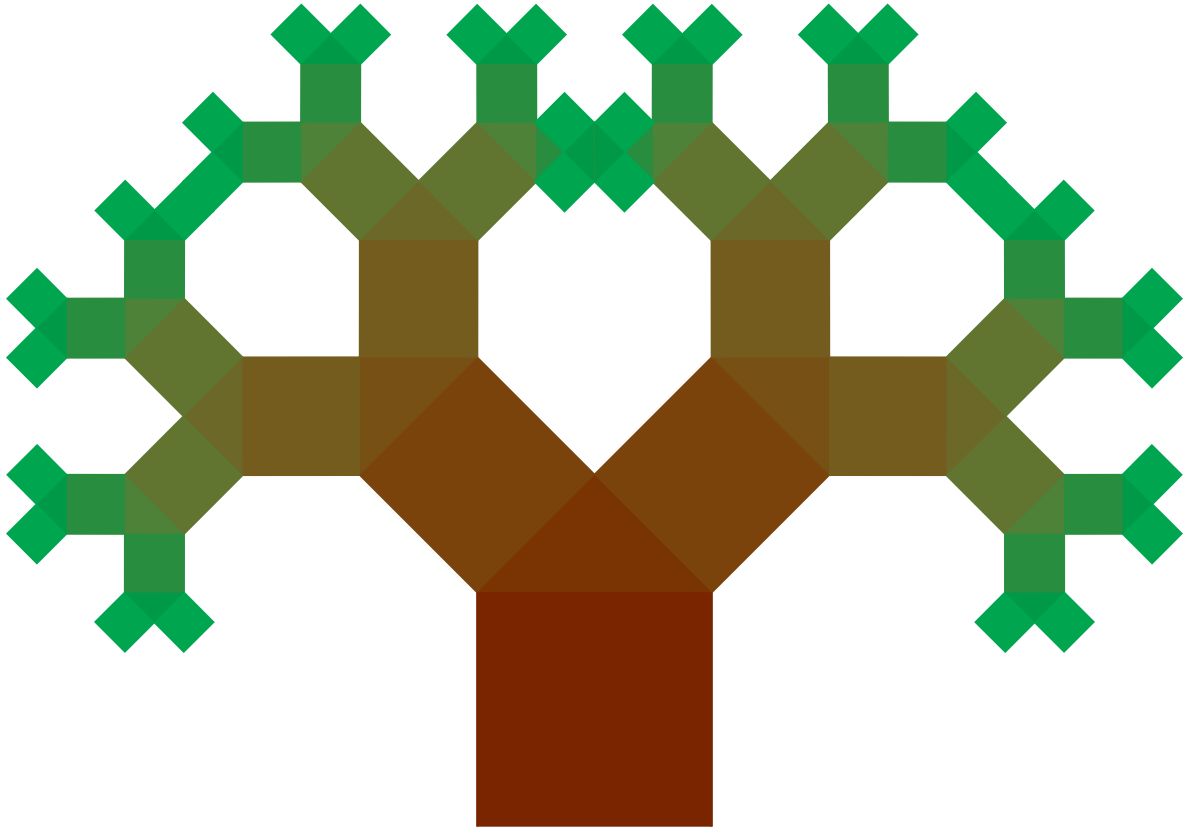
Trees



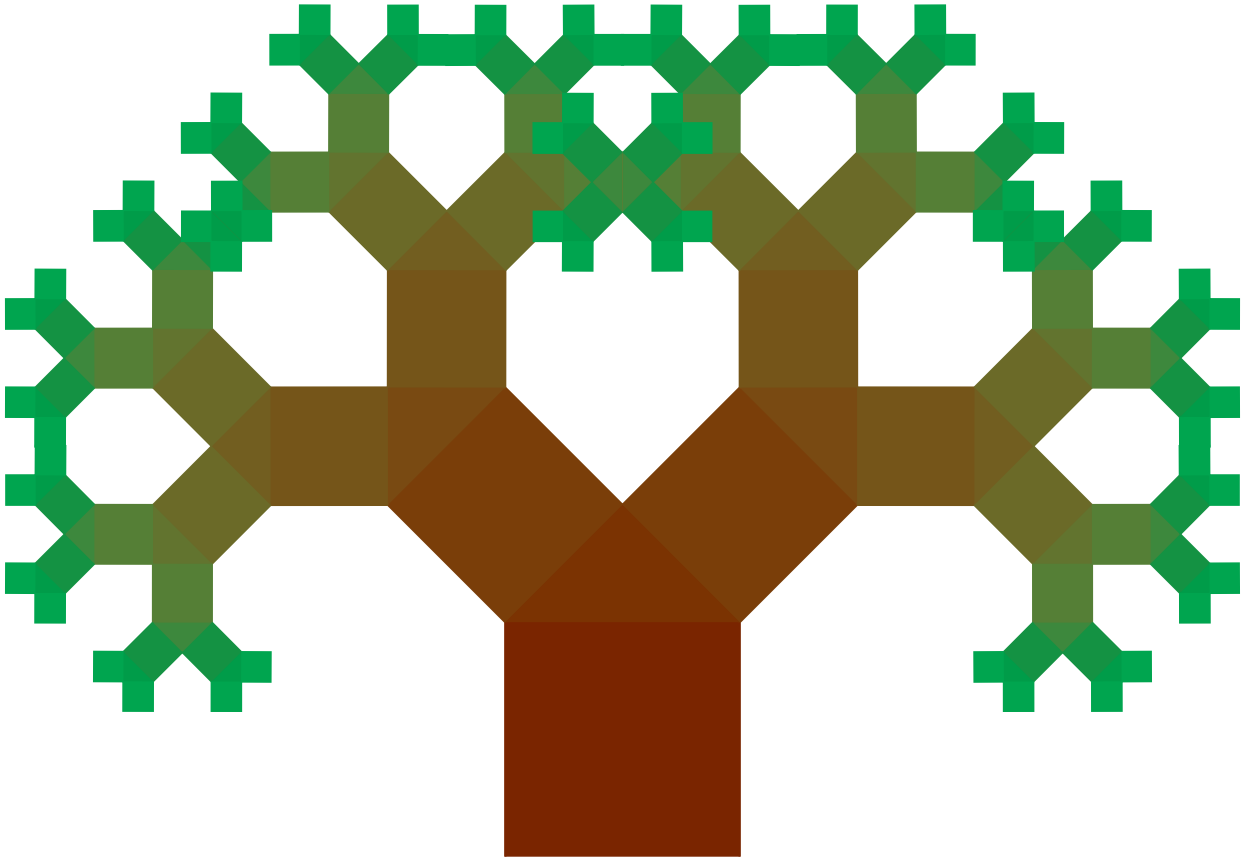
Trees



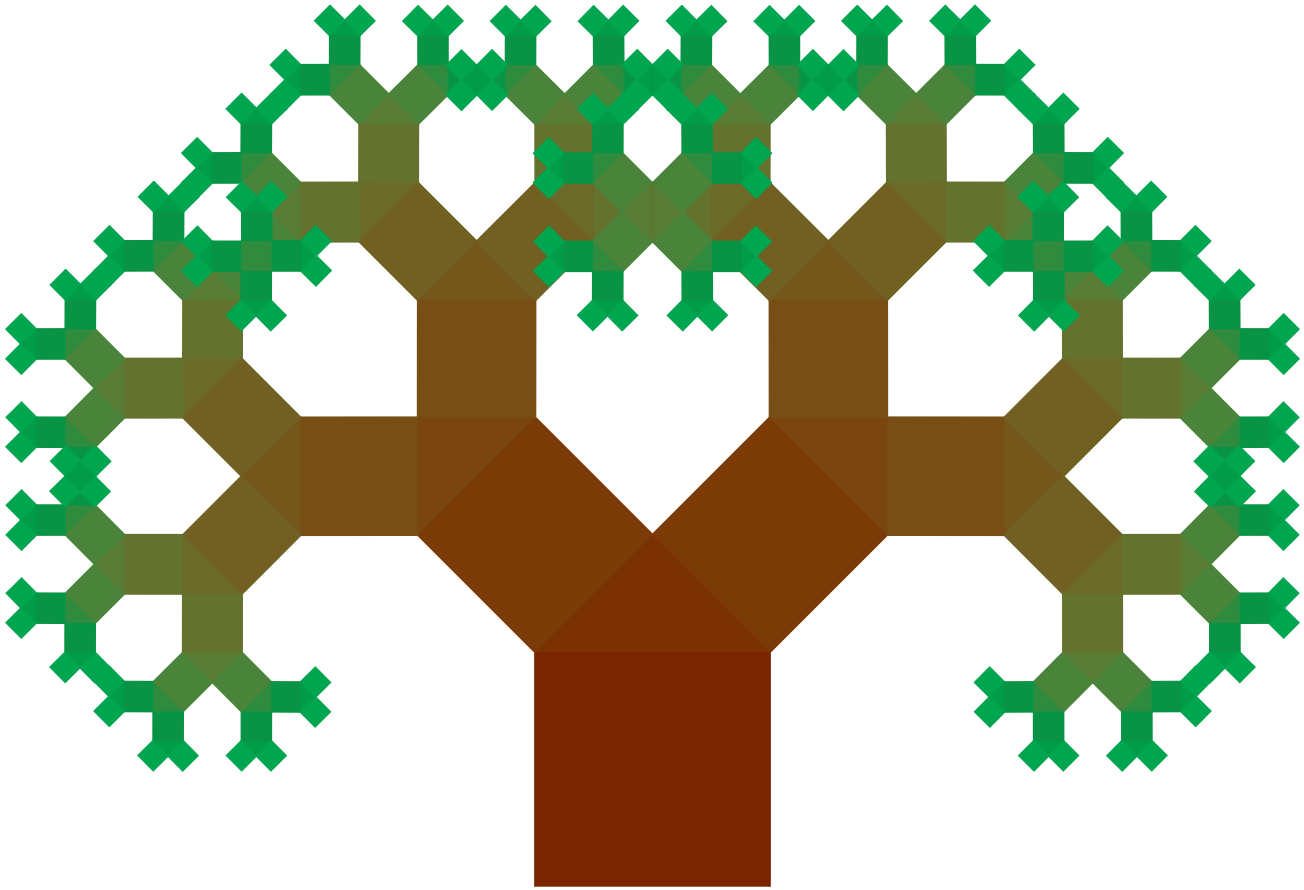
Trees



Trees



Trees

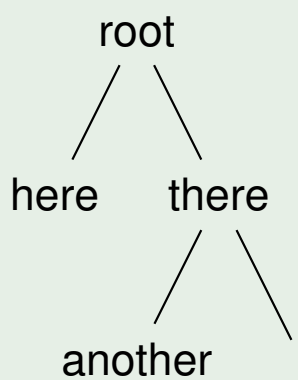


Trees – More Like This

Defining Trees

- `child{}` in a node definition creates a child
- use `node` and `child` iterativeley to create a tree

Example




```
\node {\footnotesize root}
  child {
    node {\footnotesize here}}
  child {node {\footnotesize there}
    child {
      node {\footnotesize another}}
    child {
      node {}
};
```

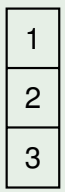
Split nodes

The Rectangle Split Shape

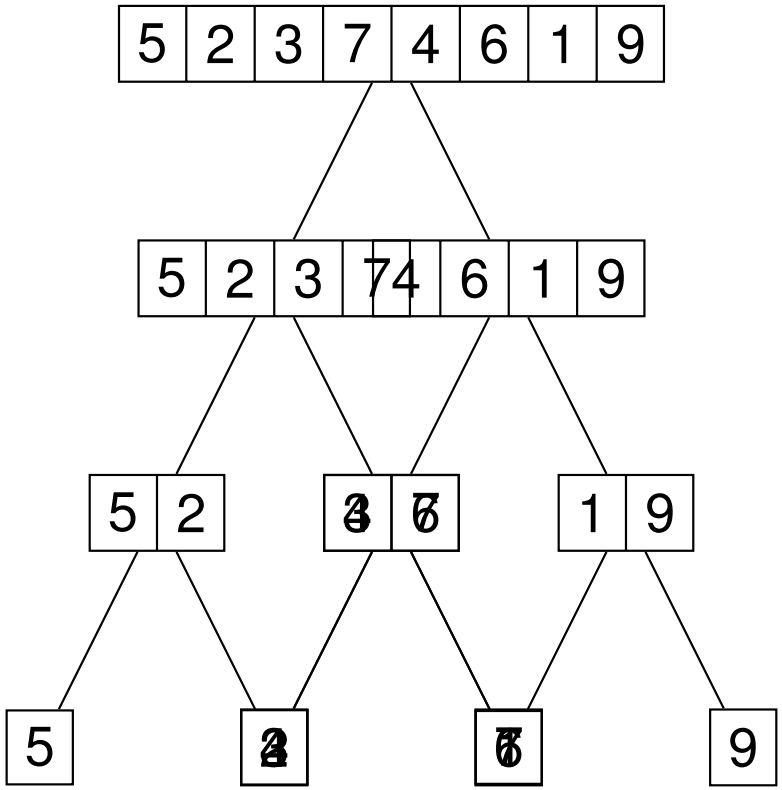
- allows to put more than one text into nodes
- need to include tikzlibrary `shapes.multipart`
- the style is `rectangle split`
- this gives a vertically split. For horizontally split add `rectangle split horizontal`

Example

```
\node[rectangle split,rectangle split parts=3,draw]
{1\nodepart{two}2\nodepart{three}3};
```



Recursive Sorting



Recursive Sorting

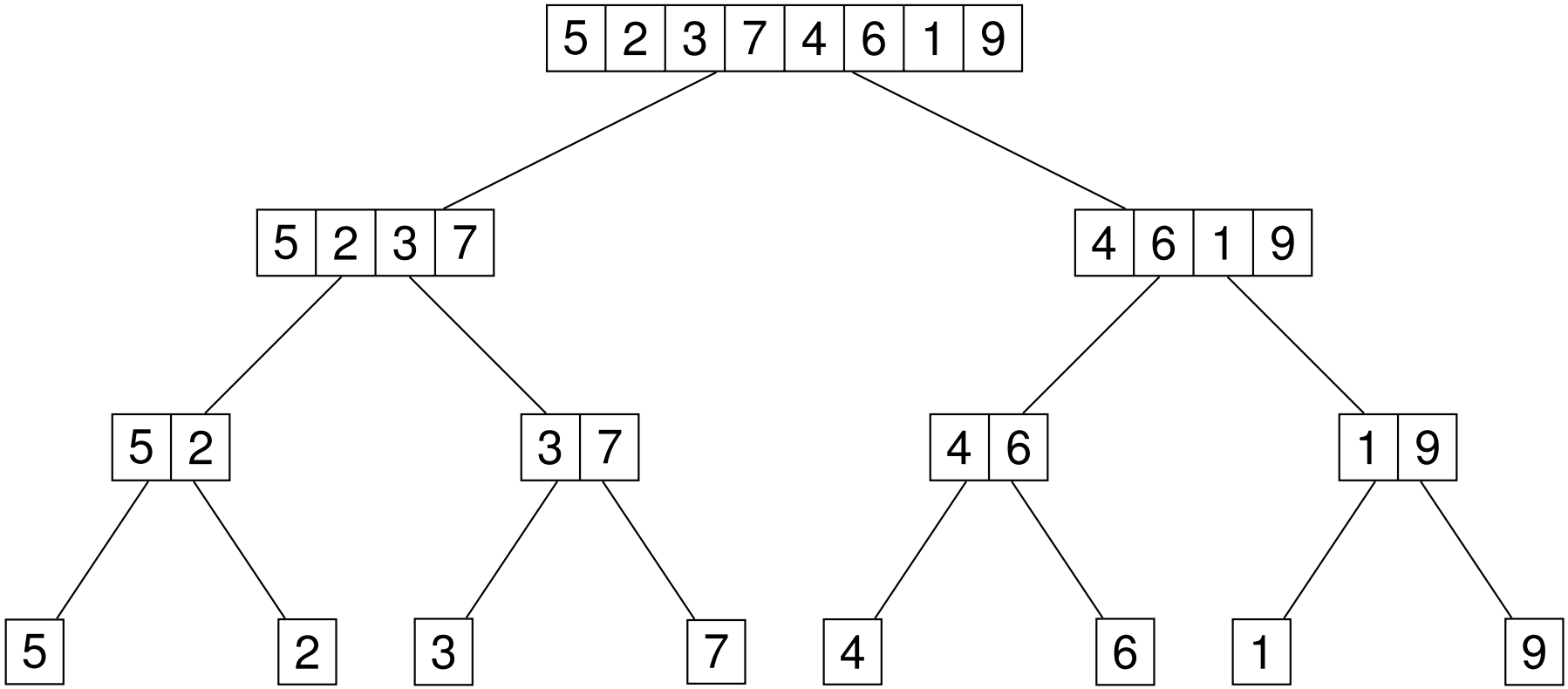
- unfortunately children overlap
- can be solved via `sibling distance`
- a style for each level of the tree

Example

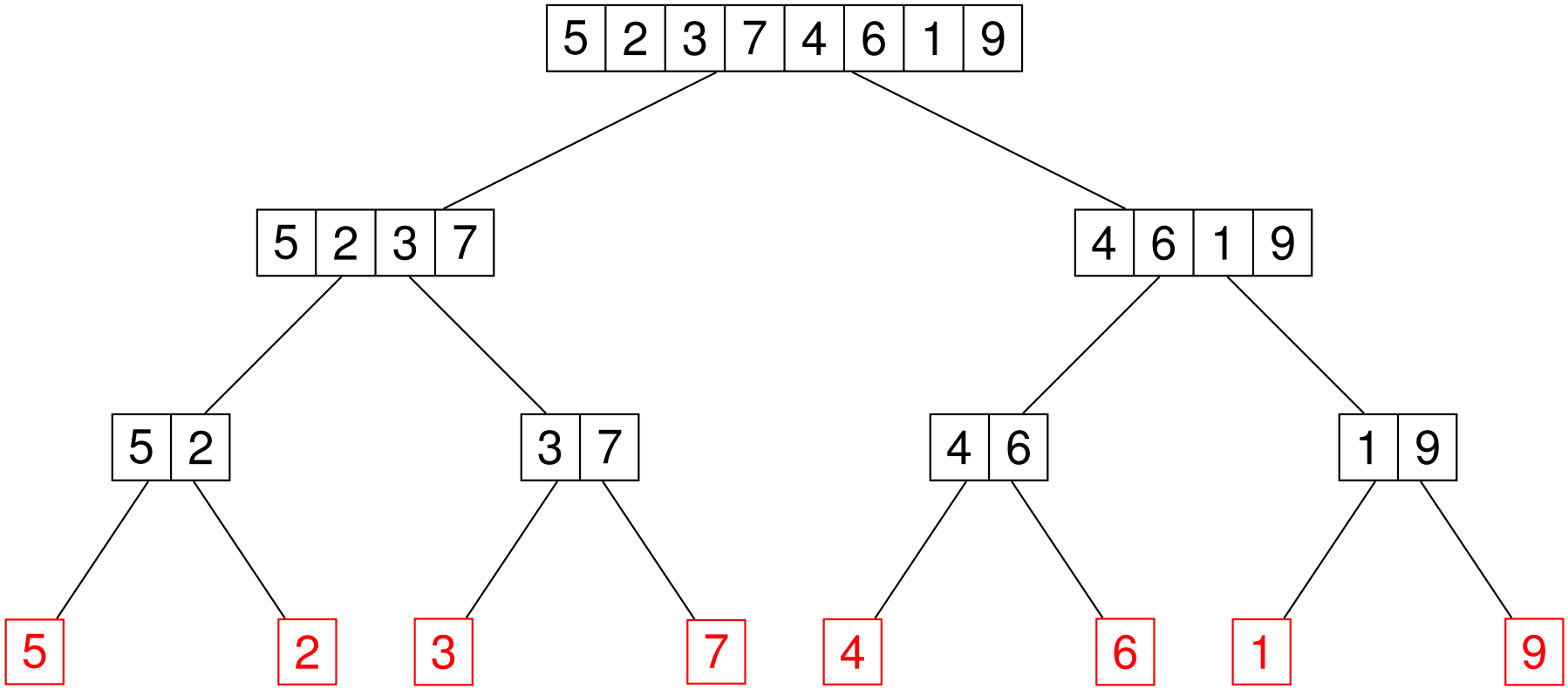
```
\begin{tikzpicture}[
level 1/.style={sibling distance=60mm},
level 2/.style={sibling distance=30mm},
level 3/.style={sibling distance=20mm}]
\end{tikzpicture}
```

➡ All siblings on level 1 will have a distance of 60*mm*

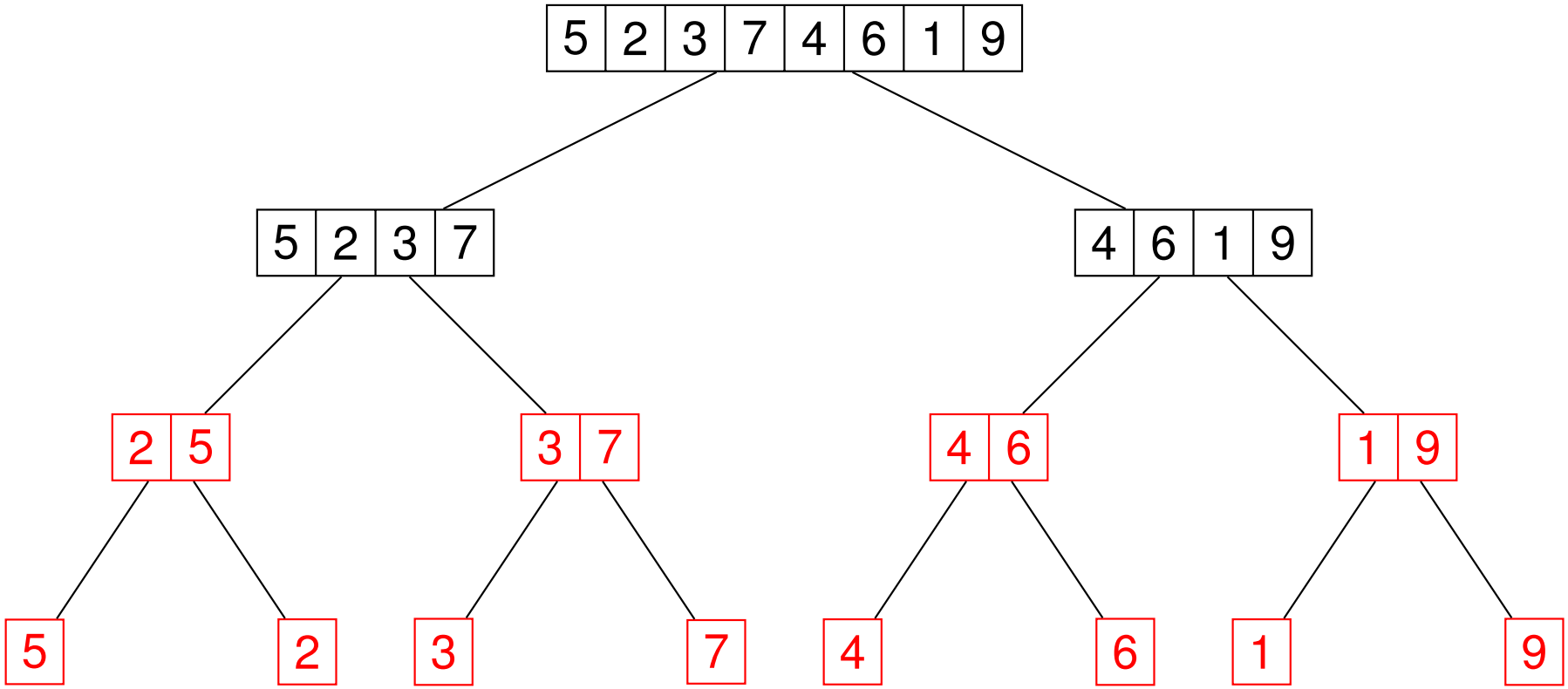
Recursive Sorting



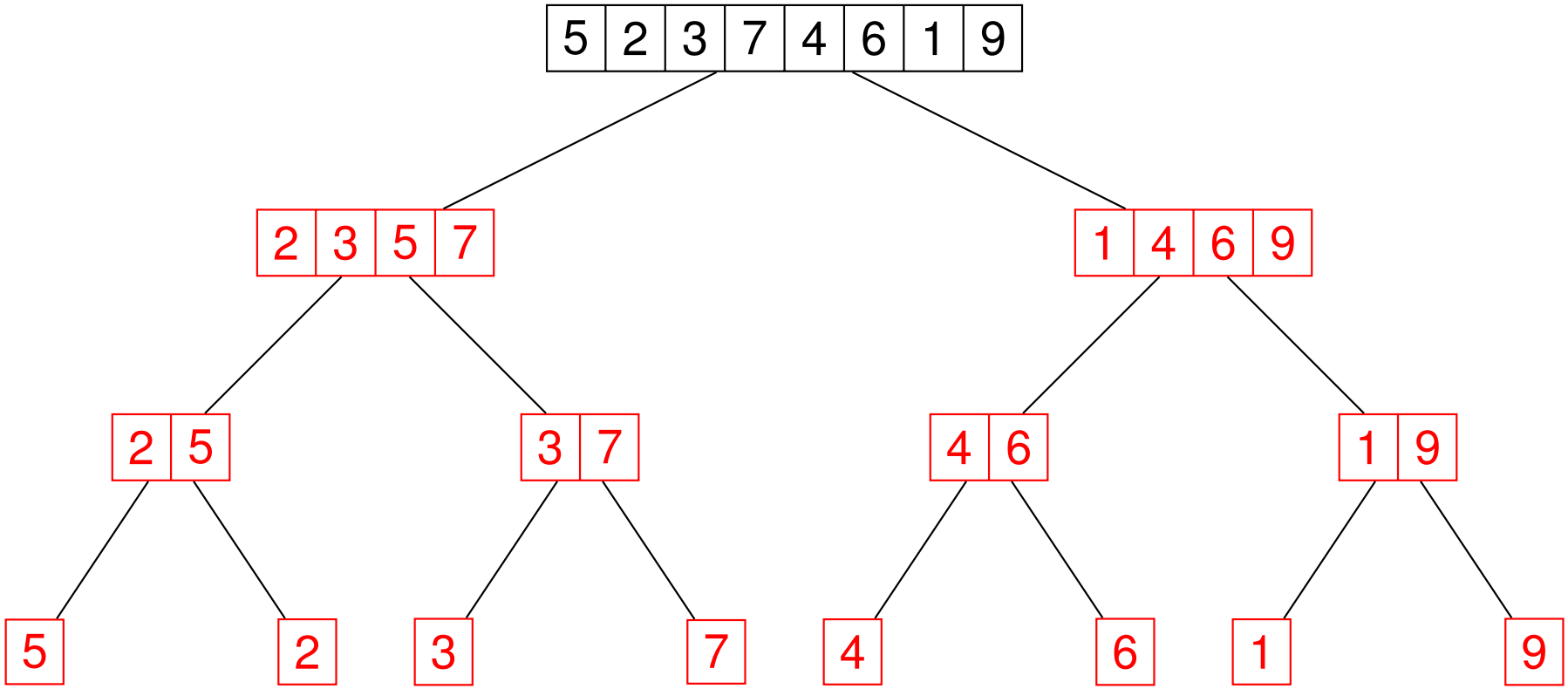
Recursive Sorting



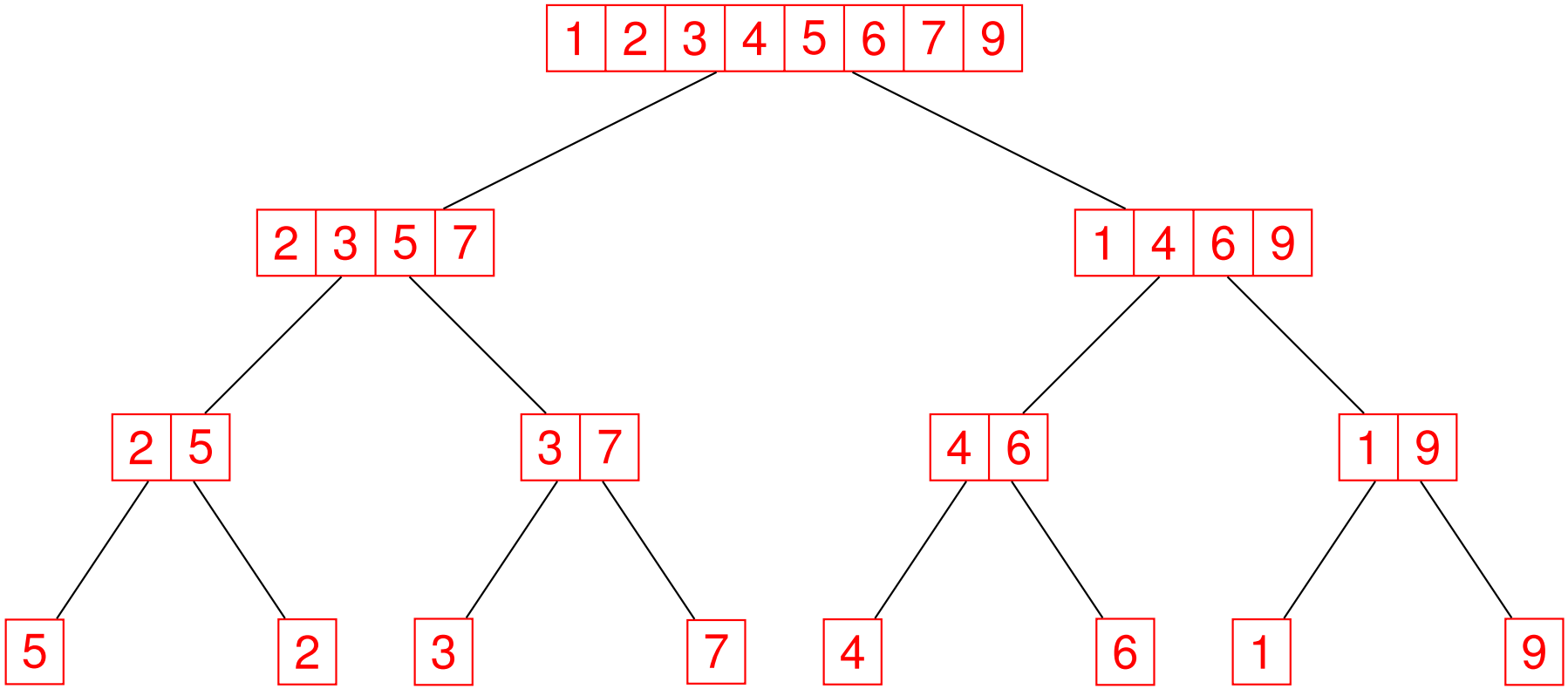
Recursive Sorting



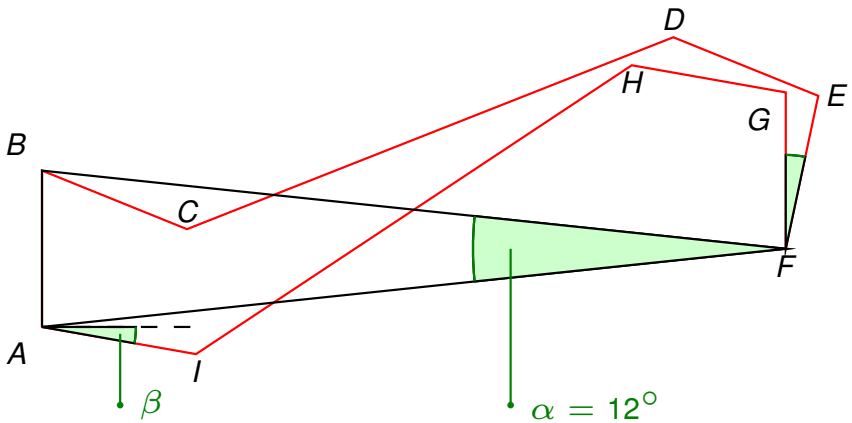
Recursive Sorting



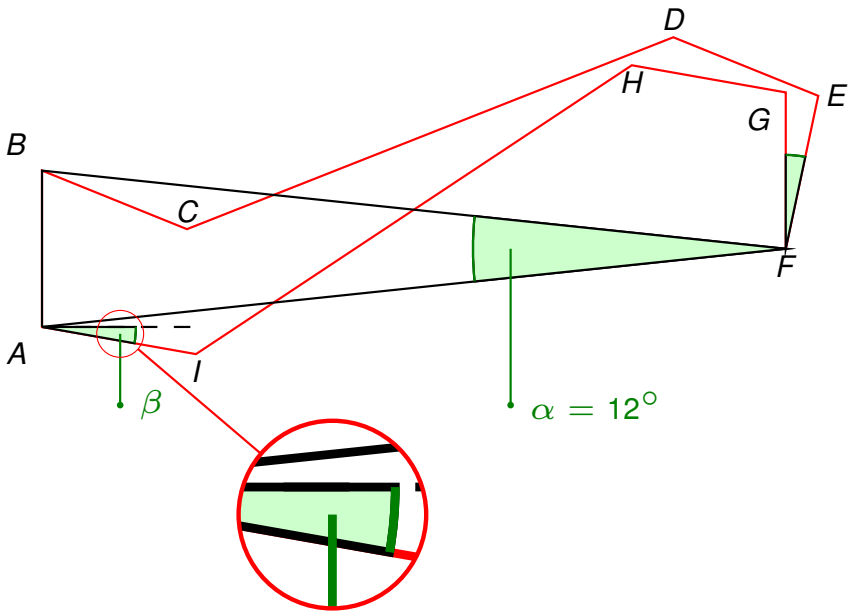
Recursive Sorting



Intersections

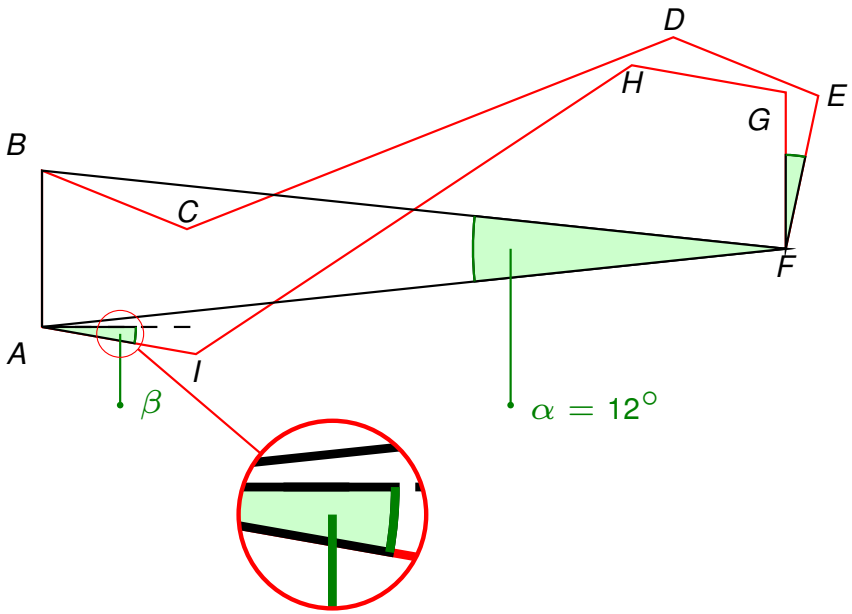


Intersections



Nice detail:
End point of helping line is in the middle of the angle.

Intersections



Nice detail:
End point of helping line is in the middle of the angle.

The good thing:
Ti*k*Z can automatically compute these intersection points.
Paths can be arbitrary, not only line segments!

Intersections cont.

How to compute intersections

1. include the TikZ library `intersections`
2. name paths using the option `[name path=pname]`
   Hint: invisible paths can be drawn using `\path`
3. compute intersections as new path:
   `\path [name intersections={of=pname1 and pname2}];`

➡ new intersection points are now available at
`(intersection-1)`, `(intersection-2)` etc.

Intersections cont.

How to compute intersections

1. include the TikZ library `intersections`
2. name paths using the option `[name path=pname]`
   Hint: invisible paths can be drawn using `\path`
3. compute intersections as new path:
   `\path [name intersections={of=pname1 and pname2}];`

➡ new intersection points are now available at `(intersection-1)`, `(intersection-2)` etc.

```
\path[name path=helpPath] (helpPoint) -- (labelPoint);
\path[name path=ai] (B) -- (F);
\path [name intersections={of=helpPath and ai}];
\coordinate (inters) at ($ (intersection-1)!0.5!(helpPoint) $
```

Some Styles to Define a Graph and Algorithm Visualization

Requirements

- need styles for nodes and edges
- styles should change with the algorithm state

➡ good idea to nest styles!

Some Styles to Define a Graph and Algorithm Visualization

Requirements

- need styles for nodes and edges
- styles should change with the algorithm state

➡ good idea to nest styles!

```
\tikzstyle{vertex}=[draw,circle,fill=Gray,minimum size=20pt]
\tikzstyle{selected vertex} = [vertex, fill=Maroon]
\tikzstyle{edge} = [draw,thick,-]
\tikzstyle{weight} = [font=\small]
\tikzstyle{selected edge} = [draw,line width=5pt,-,Green]
\tikzstyle{ignored edge} = [draw,line width=5pt,-,Salmon]
```

➡ Style „selected vertex“ based on „vertex“, but changes the fill color

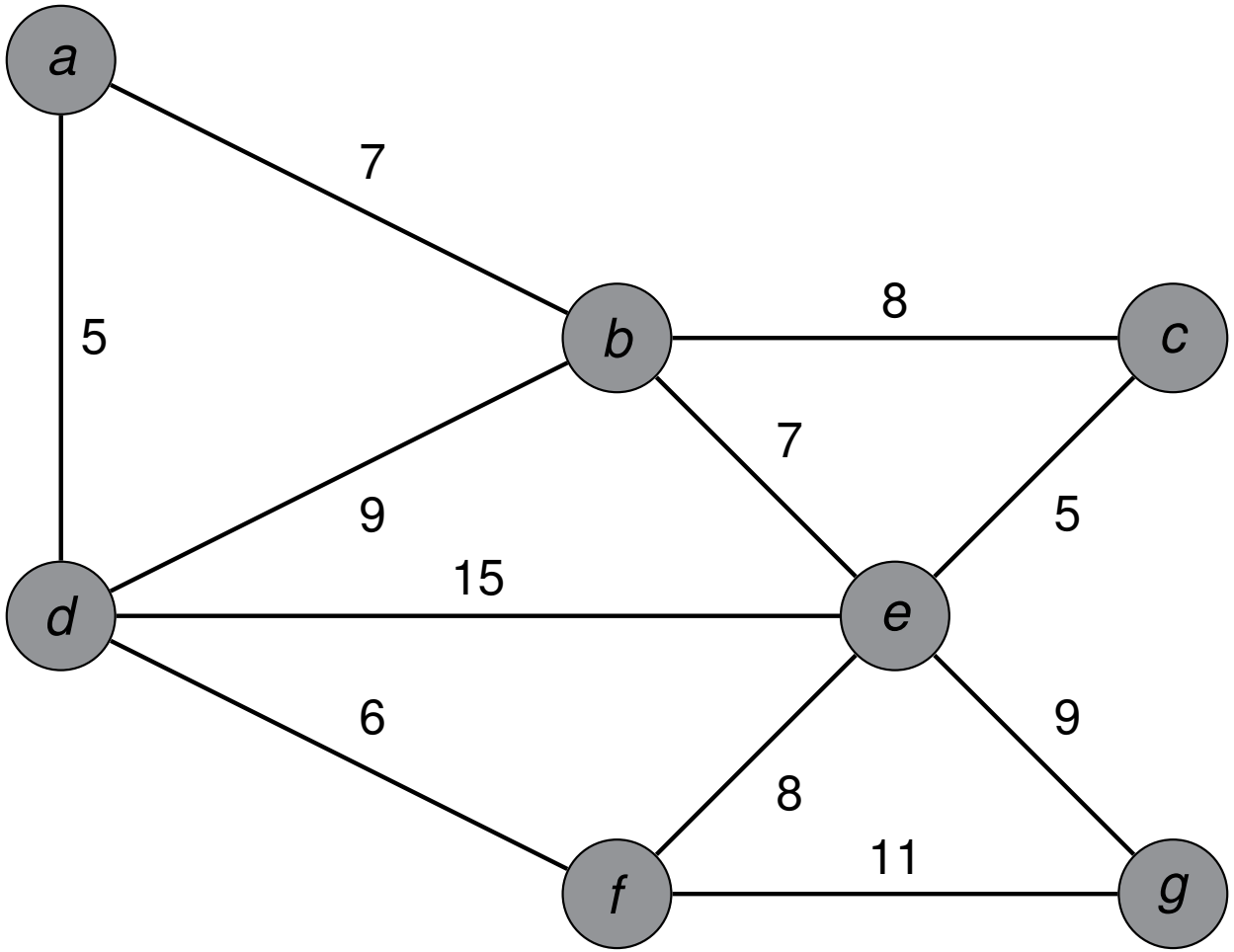
Defining a Graph in Four Lines...

Using all our so far gained knowledge, we can say:

```
\foreach \pos/\name in {{(0,2)/a}, {(2,1)/b}, {(4,1)/c},
          {(0,0)/d}, {(3,0)/e}, {(2,-1)/f}, {(4,-1)/g}}
  \node[vertex] (\name) at \pos {$\name$};
\foreach \source/\dest /\weight in {b/a/7,c/b/8,d/a/5,d/b/9,
                e/b/7,e/c/5,e/d/15,f/d/6,f/e/8,g/e/9,g/f/11}
  \path[edge] (\source) -- node[weight] {$\weight$} (\dest);
```

Using: styles, node names and foreach with tuples

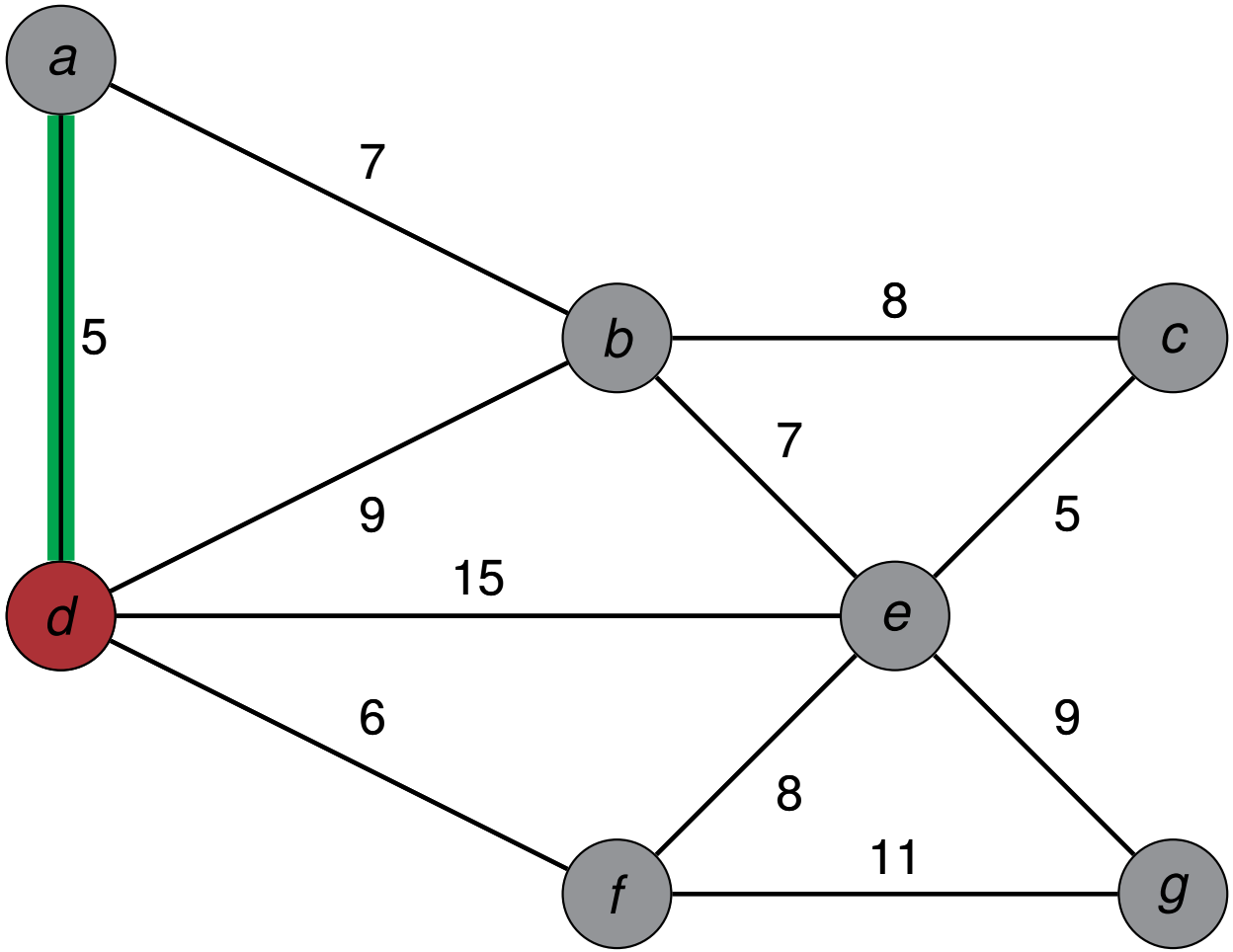
Defining a Graph in Four Lines...



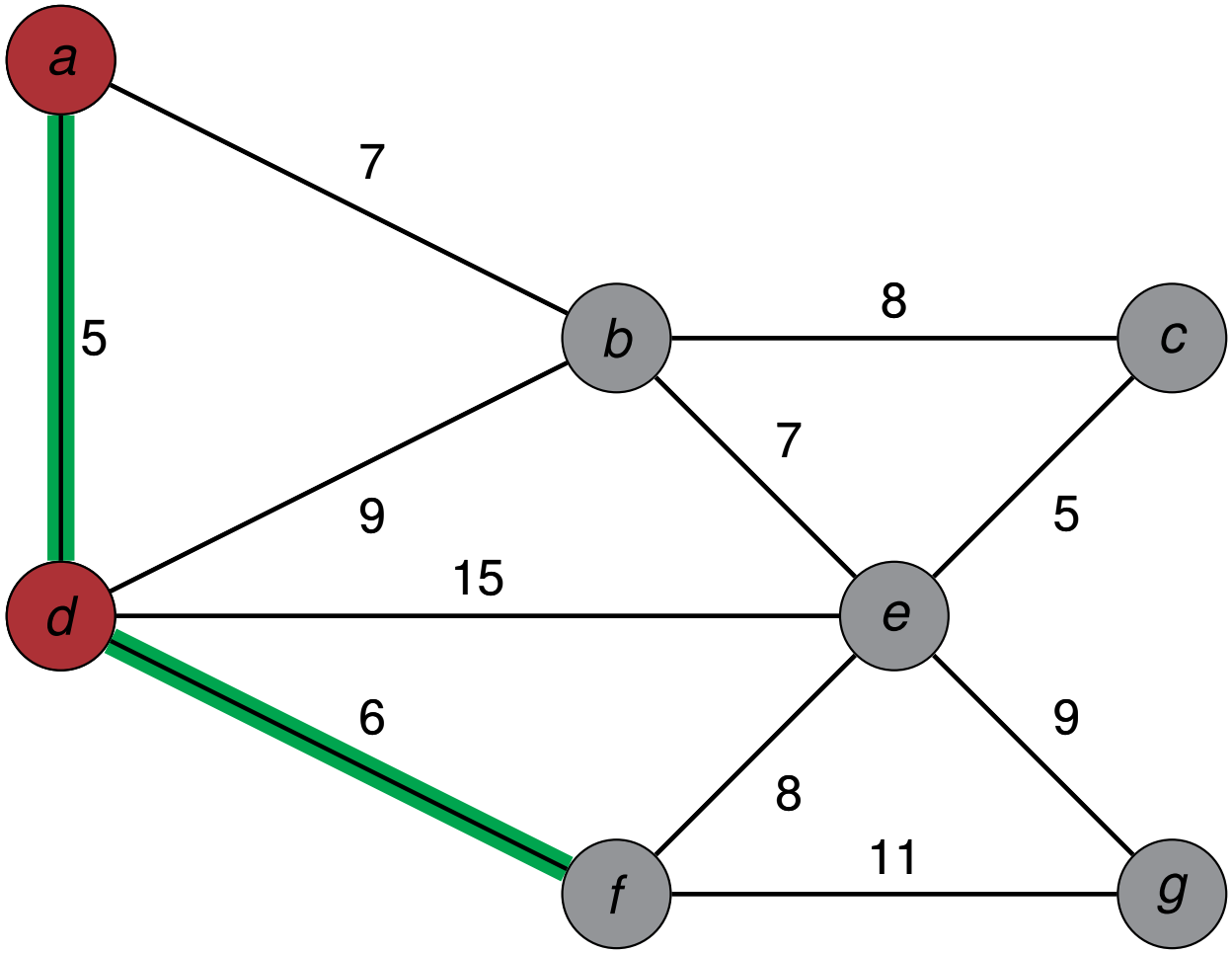
... and an Algorithm in Another Six

```
\foreach \vertex / \slide in {d/1,a/2,f/3,b/4,e/5,c/6,g/7}
  \path<\slide-> node[selected vertex] at (\vertex) {$\vertex
\foreach \source / \dest in {d/a,d/f,a/b,b/e,e/c,e/g}
  \path<+->[selected edge] (\source) -- (\dest);
\foreach \source / \dest / \slide in
        {d/b/4,d/e/5,e/f/5,b/c/6,f/g/7}
  \path<\slide->[ignored edge] (\source) -- (\dest);
```

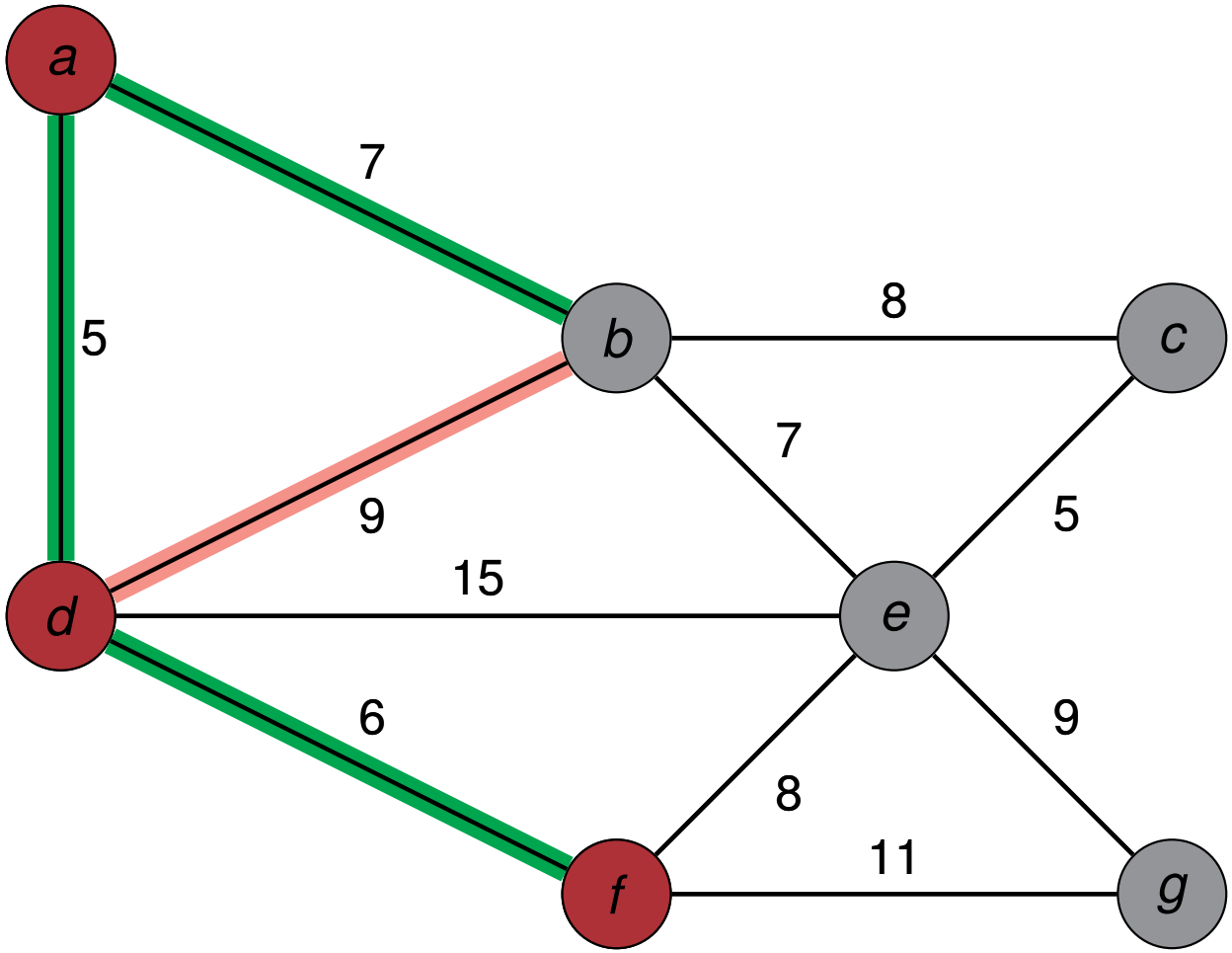
... and an Algorithm in Another Six



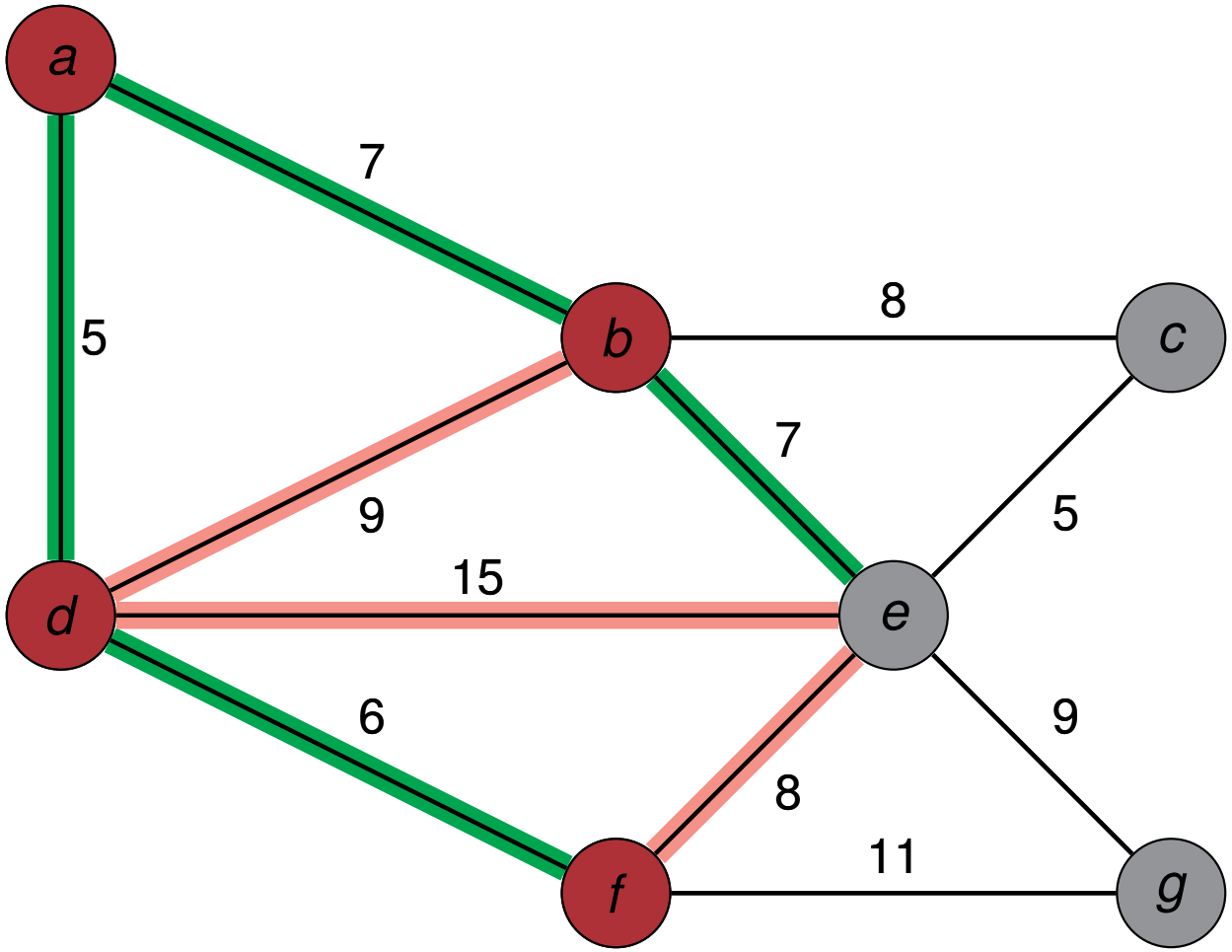
... and an Algorithm in Another Six



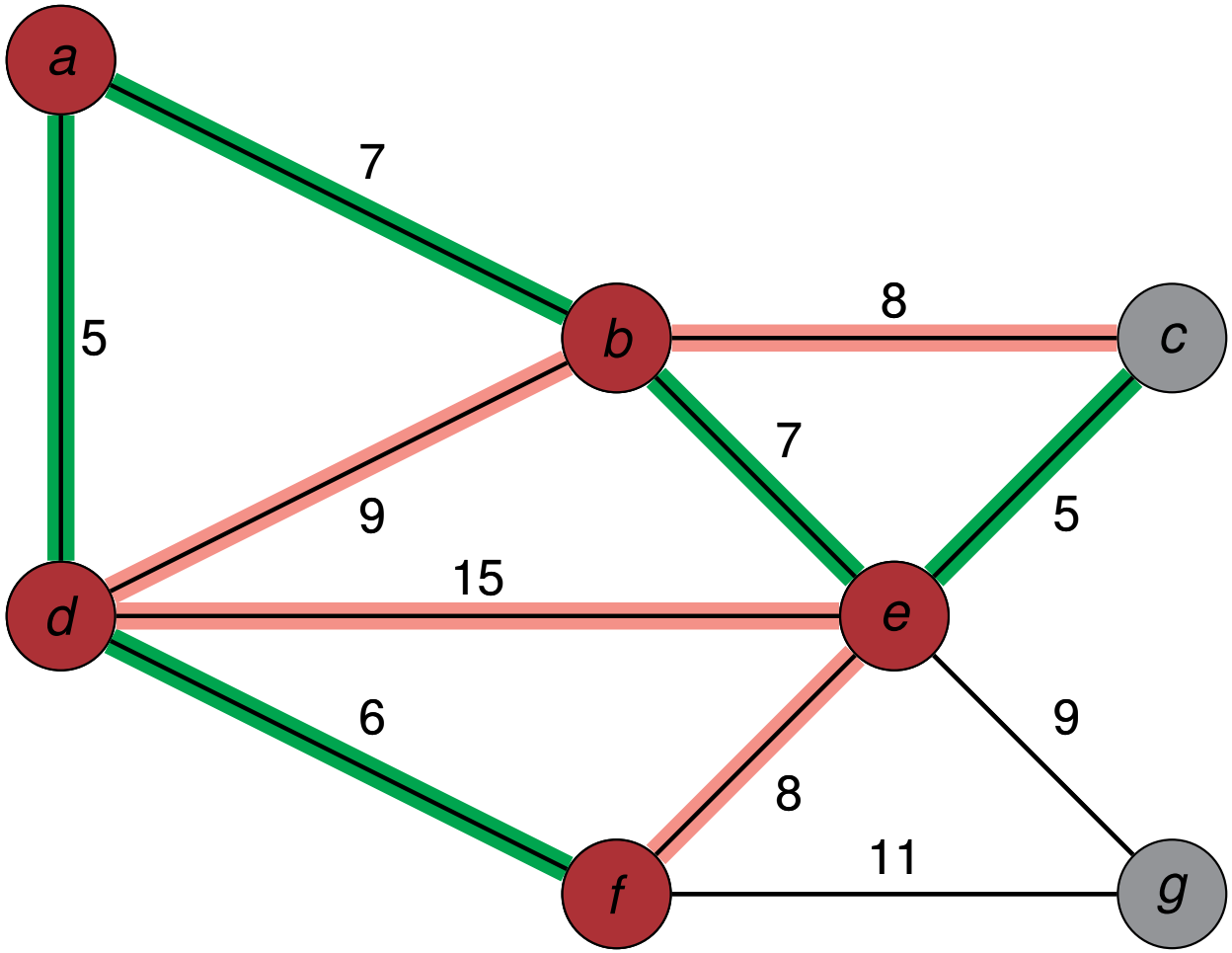
... and an Algorithm in Another Six



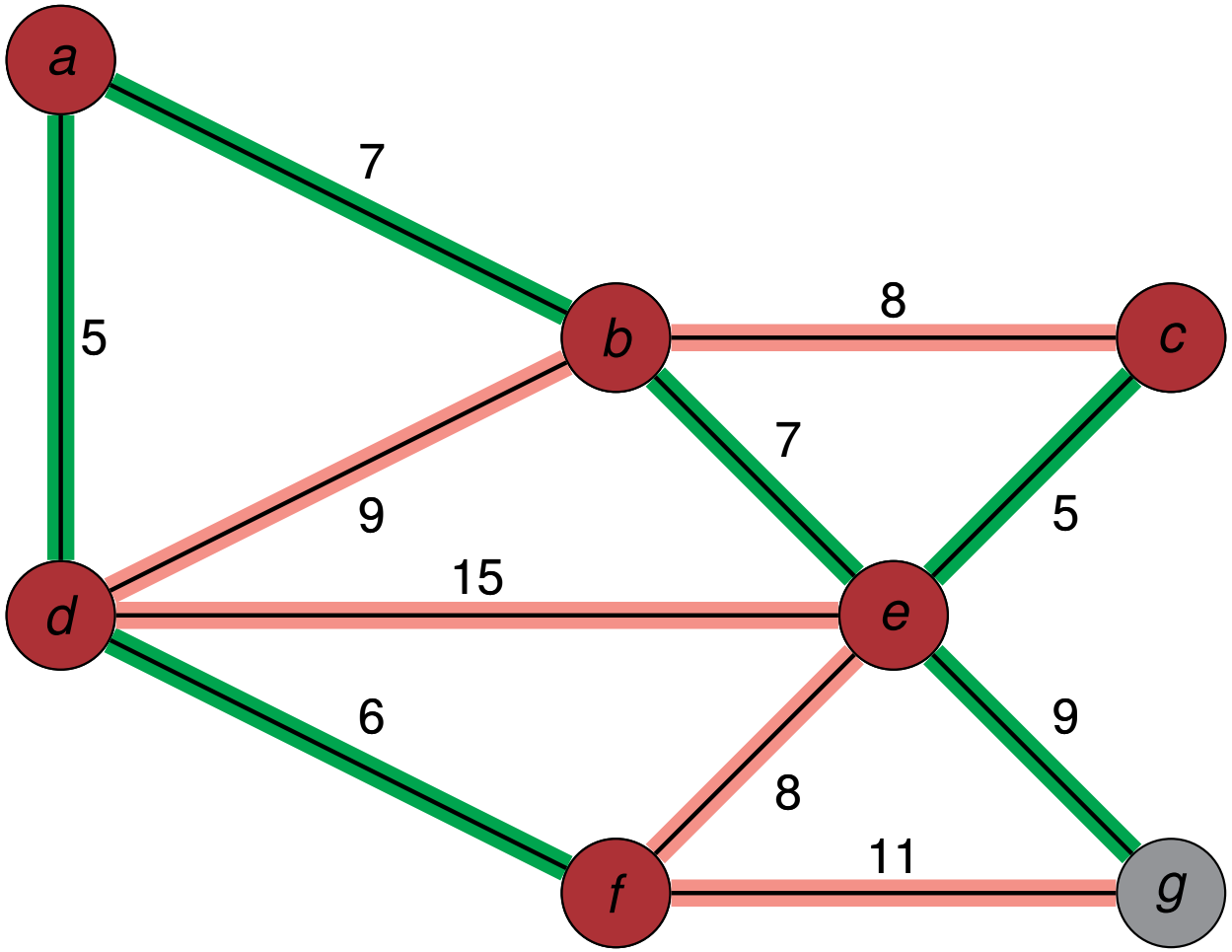
... and an Algorithm in Another Six



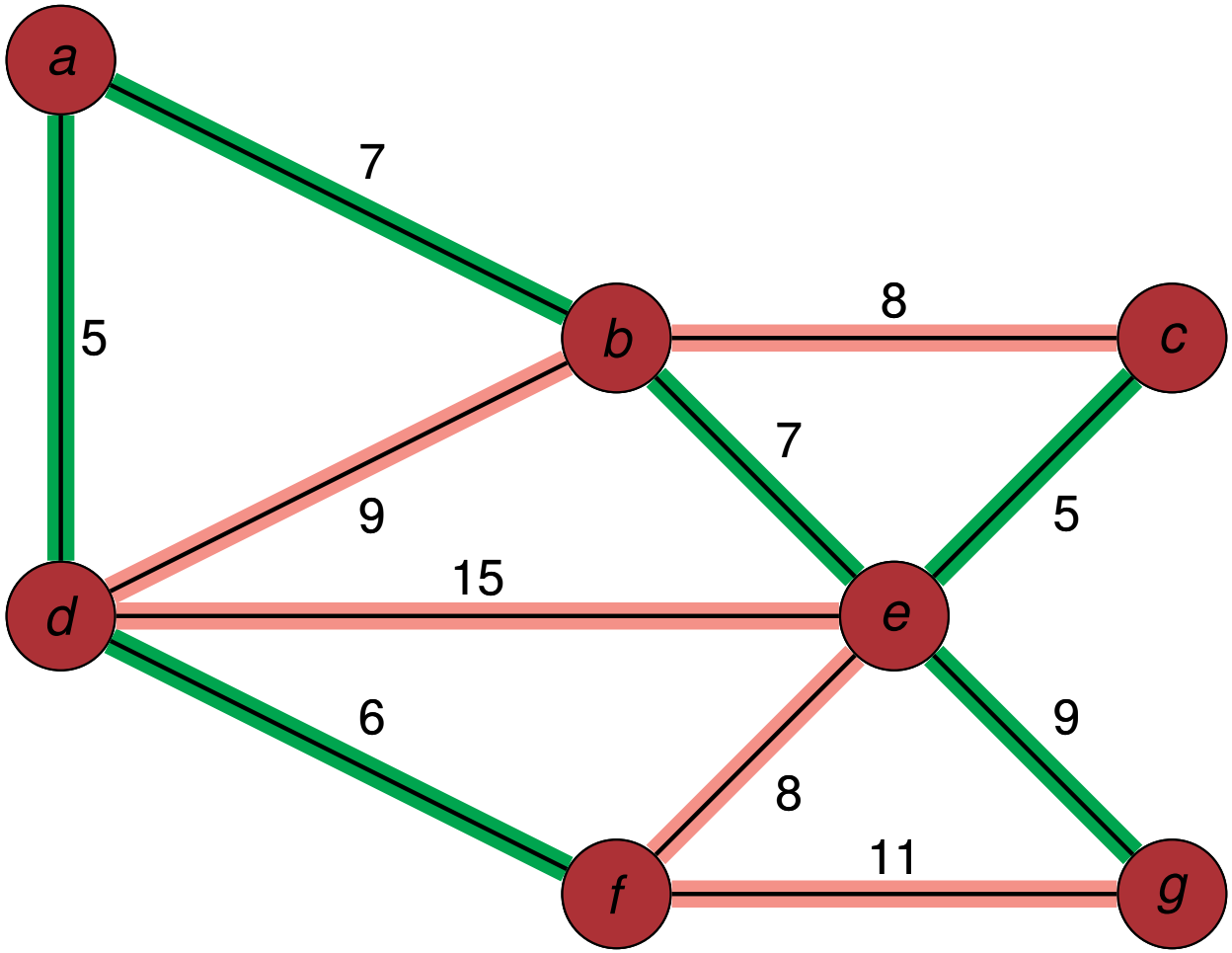
... and an Algorithm in Another Six



... and an Algorithm in Another Six

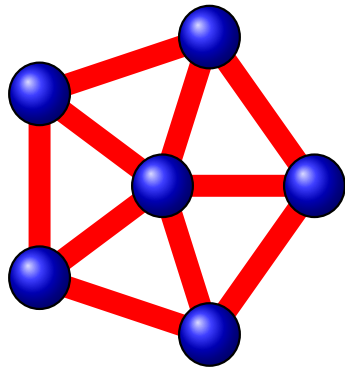


... and an Algorithm in Another Six



Outlook

- TikZ really can do a lot of stuff
- much more can be done using some of the many packages
- for example object oriented programming
- worth reading a bit in the manual



Thank you!