alchemist

v0.1.0 MIT

A package to render skeletal formulas using cetz

Robotechnic <@Robotechnic>

https://github.com/Robotechnic/alchemist

Alchemist is a package used to draw chemical structures with skeletal formulas using Cetz. It is heavily inspired by the Chemfig package for LaTeX. This package is meant to be easy to use and customizable. It can also be used alongside the cetz package to draw more complex structures.

Table of contents

Part I. Usage

To start using Alchemist, just import the package in your document:

```
#import "@preview/alchemist:0.1.0": *
```

I.1. Initializing drawing environment

To start drawing molecules, you first need to initialise the drawing environment. This is done by calling the `#skeletize()` function.

```
#skeletize({
  ...
})
```

The main argument is a block of code that contains the drawing instructions. The block can also contain any cetz code to draw more complex structures, see Section II.7.

```
#skeletize(⟨debug⟩: false, ⟨background⟩: none, ⟨config⟩: (:), ⟨body⟩)
```

Argument

`⟨debug⟩` `bool`

Display bounding boxes of the objects in the drawing environment.

Argument

`⟨background⟩` `color` | `none`

Background color of the drawing environment

Argument

`⟨config⟩` `dictionary`

Configuration of the drawing environment. See Section I.2.

Argument

`⟨body⟩` `drawable`

The module to draw or any cetz drawable object.

I.2. Configuration

Th configuration dictionary that you can pass to skeletize defines a set of default values for a lot of parameters in alchemist.

Argument

`⟨atom-sep⟩: 3em` `length`

It defines the distance between each atom center. It is overridden by the `atom-sep` argument of link

Argument
⟨angle-increment⟩: 45deg `angle`

It defines the angle added by each increment of the `angle` argument of link

Argument
⟨base-angle⟩: 0deg `angle`

Default angle at which the link with no angle defined will be.

I.3. Available commands

I.3.1. Molecule function

#molecule(⟨name⟩: none, ⟨links⟩: "(:)", ⟨mol⟩) ⟶ drawable

Build a molecule group based on mol Each molecule is represented as an optional count followed by a molecule name starting by a capital letter followed by an optional indice

```
#skeletize({
  molecule("H_2O")
})
```

H_2O

```
#skeletize({
  molecule("2Aa_7 3Bb")
})
```

$2\mathrm{Aa}_7 3\mathrm{Bb}$

Argument
⟨name⟩: none `str`

The name of the molecule. It is used as the cetz name of the molecule and to link other molecules to it.

Argument
⟨links⟩: "(:)" `dictionary`

The links between this molecule and the previous ones. The key is the name of the molecule and the value is the link you want to draw between the two molecules.

Note that the antom-sep and angle arguments are ignored

Argument
⟨mol⟩ `str`

The string representing the molecule

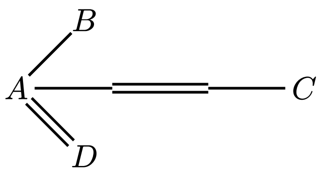
I.3.2. Branch and cycles

#branch(..⟨args⟩)

Create a branch from the current molecule, the first element of the branch has to be a link.

You can specify an angle argument like for links. This angle will be then used as the `base-angle` for the branch.

```
#skeletize({
  molecule("A")
  branch({
    single(angle:1)
    molecule("B")
  })
  branch({
    double(angle: -1)
    molecule("D")
  })
  single()
  double()
  single()
  molecule("C")
})
```

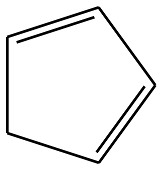


`#cycle(..⟨args⟩)`

Create a regular cycle of molecules You can specify an angle argument like for links. This angle will be then the angle of the first link of the cycle.

The argument `align` can be used to force align the cycle according to the relative angle of the previous link.

```
#skeletize({
  cycle(5, {
    single()
    double()
    single()
    double()
    single()
  })
})
```



I.3.3. Link functions

I.3.3.1. Common arguments

Links functions are used to draw links between molecules. They all have the same base arguments but can be customized with additional arguments.

Argument

`⟨angle⟩: 0` `int`

Multiplier of the `angle-increment` argument of the drawing environment. The final angle is relative to the abscissa axis.

Argument

`⟨relative⟩: none` `angle`

Relative angle to the previous link. This argument override all other angle arguments.

Argument

⟨absolute⟩: none — angle

Absolute angle of the link. This argument override `angle` argument.

Argument

⟨antom-sep⟩: 3em — length

Distance between the two connected atom of the link. Default to the `atom-sep` entry of the configuration dictionary.

Argument

⟨from⟩ — int

Index of the molecule in the group to start the link from. By default, it is computed depending on the angle of the link.

Argument

⟨to⟩ — int

Index of the molecule in the group to end the link to. By default, it is computed depending on the angle of the link.

I.3.3.2. Links

#build-link(⟨draw-function⟩) ⟶ function

Create a link function that is then used to draw a link between two points

Argument

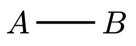
⟨draw-function⟩ — function

The function that will be used to draw the link. It should takes three arguments: the length of the link, the context, and a dictionary of named arguments that can be used to configure the links

#single

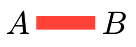
Draw a single line between two molecules

```
#skeletize({
  molecule("A")
  single()
  molecule("B")
})
```



It is possible to change the color and width of the line with the `stroke` argument

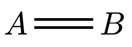
```
#skeletize({
  molecule("A")
  single(stroke: red + 5pt)
  molecule("B")
})
```



#double

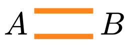
Draw a double line between two molecules

```
#skeletize({
  molecule("A")
  double()
  molecule("B")
})
```



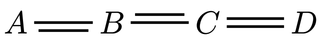
It is possible to change the color and width of the line with the `stroke` argument and the gap between the two lines with the `gap` argument

```
#skeletize({
  molecule("A")
  double(
    stroke: orange + 2pt,
    gap: .8em
  )
  molecule("B")
})
```



This link also supports an `offset` argument that can be set to `left`, `right` or `center`. It allows to make either the let side, right side or the center of the double line to be aligned with the link point.

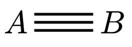
```
#skeletize({
  molecule("A")
  double(offset: "right")
  molecule("B")
  double(offset: "left")
  molecule("C")
  double(offset: "center")
  molecule("D")
})
```



#triple

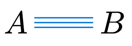
Draw a triple line between two molecules

```
#skeletize({
  molecule("A")
  triple()
  molecule("B")
})
```



It is possible to change the color and width of the line with the `stroke` argument and the gap between the three lines with the `gap` argument

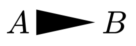
```
#skeletize({
  molecule("A")
  triple(
    stroke: blue + .5pt,
    gap: .15em
  )
  molecule("B")
})
```



#cram-filled-right

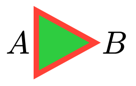
Draw a filled cram between two molecules with the arrow pointing to the right

```
#skeletize({
  molecule("A")
  cram-filled-right()
  molecule("B")
})
```



It is possible to change the stroke and fill color of the arrow with the `stroke` and `fill` arguments. You can also change the base length of the arrow with the `base-length` argument

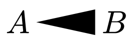
```
#skeletize({
  molecule("A")
  cram-filled-right(
    stroke: red + 2pt,
    fill: green,
    base-length: 2em
  )
  molecule("B")
})
```



#cram-filled-left

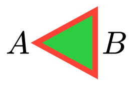
Draw a filled cram between two molecules with the arrow pointing to the left

```
#skeletize({
  molecule("A")
  cram-filled-left()
  molecule("B")
})
```



It is possible to change the stroke and fill color of the arrow with the `stroke` and `fill` arguments. You can also change the base length of the arrow with the `base-length` argument

```
#skeletize({
  molecule("A")
  cram-filled-left(
    stroke: red + 2pt,
    fill: green,
    base-length: 2em
  )
  molecule("B")
})
```



#cram-hollow-right

Draw a hollow cram between two molecules with the arrow pointing to the right It is a shorthand for `cram-filled-right(fill: none)`

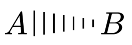
#cram-hollow-left

Draw a hollow cram between two molecules with the arrow pointing to the left It is a shorthand for `cram-filled-left(fill: none)`

#cram-dashed-right

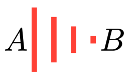
Draw a dashed cram between two molecules with the arrow pointing to the right

```
#skeletize({
  molecule("A")
  cram-dashed-right()
  molecule("B")
})
```



It is possible to change the stroke of the lines in the arrow with the `stroke` argument. You can also change the base length of the arrow with the `base-length` argument and distance between the dashes with the `dash-gap` argument

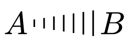
```
#skeletize({
  molecule("A")
  cram-dashed-right(
    stroke: red + 2pt,
    base-length: 2em,
    dash-gap: .5em
  )
  molecule("B")
})
```



#cram-dashed-left

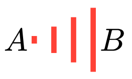
Draw a dashed cram between two molecules with the arrow pointing to the left

```
#skeletize({
  molecule("A")
  cram-dashed-left()
  molecule("B")
})
```



It is possible to change the stroke of the lines in the arrow with the `stroke` argument. You can also change the base length of the arrow with the `base-length` argument and distance between the dashes with the `dash-gap` argument

```
#skeletize({
  molecule("A")
  cram-dashed-left(
    stroke: red + 2pt,
    base-length: 2em,
    dash-gap: .5em
  )
  molecule("B")
})
```



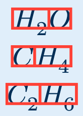
Part II.
Drawing molecules

II.1. Atoms

In alchemist, the name of the function `#molecule()` is used to create a group of atom but here it is a little bit abusive as it do not necessarily represent real molecules. An atom is in our case something of the form: optional number + capital letter + optional lowercase letter + optional _ number. For the ones interested here is the regex used: `^ *([0-9]*[A-Z][a-z]*)(_[0-9]+)?`.

For instance, H_2O is a molecule of the atoms H_2 and O. If we look at the bounding boxes of the molecules, we can see that separation.

$$H_2O$$
$$CH_4$$
$$C_2H_6$$



This separation does not have any impact on the drawing of the molecules but it will be useful when we will draw more complex structures.

II.2. Links

There are already som links available with the package (see Section I.3.3) and you can create your own links with the `#build-link()` function but they all share the same base arguments used to control their behaviors.

II.2.1. Atom separation

Each atom is separated by a distance defined by the `atom-sep` argument of the drawing environment. This distance can be overridden by the `atom-sep` argument of the link. It defines the distance between the center of the two connected atoms.

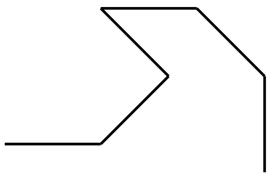
The behavior is not well defined yet.

II.2.2. Angle

There are three ways to define the angle of a link: using the `angle` argument, the `relative` argument, or the `absolute` argument.

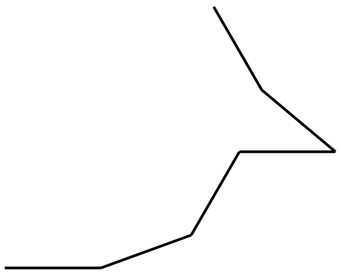
The argument `angle` is a multiplier of the `angle-increment` argument.

```
#skeletize({
  single()
  single(angle:1)
  single(angle:3)
  single()
  single(angle:7)
  single(angle:6)
})
```



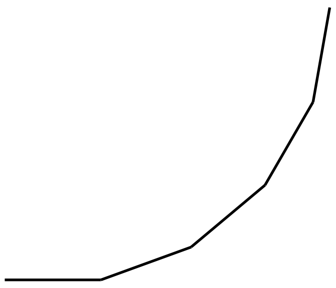
Changing the `angle-increment` argument of the drawing environment will change the angle of the links.

```
#skeletize(config:(angle-
increment:20deg),{
  single()
  single(angle:1)
  single(angle:3)
  single()
  single(angle:7)
  single(angle:6)
})
```



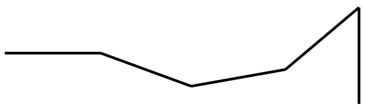
The argument `relative` allows you to define the angle of the link relative to the previous link.

```
#skeletize({
  single()
  single(relative:20deg)
  single(relative:20deg)
  single(relative:20deg)
  single(relative:20deg)
})
```



The argument `absolute` allows you to define the angle of the link relative to the abscissa axis.

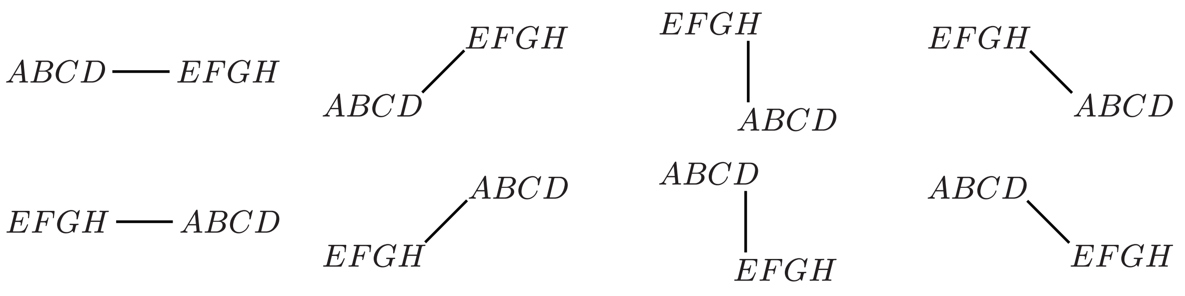
```
#skeletize({
  single()
  single(absolute:-20deg)
  single(absolute:10deg)
  single(absolute:40deg)
  single(absolute:-90deg)
})
```



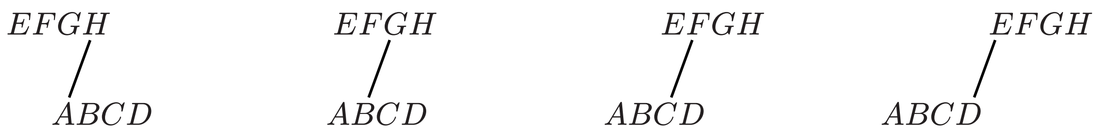
II.2.3. Starting and ending points

By default, the starting and ending points of the links are computed depending on the angle of the link. You can override this behavior by using the `from` and `to` arguments.

If the angle is in $]-90\text{ deg}; 90\text{ deg}]$, the starting point is the last atom of the previous molecule and the ending point is the first atom of the next molecule. If the angle is in $]90\text{ deg}; 270\text{ deg}]$, the starting point is the first atom of the previous molecule and the ending point is the last atom of the next molecule.



If you choose to override the starting and ending points, you can use the `from` and `to` arguments. The only constraint is that the index must be in the range $[0, n - 1]$ where n is the number of atoms in the molecule.



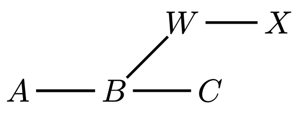
The fact that you can chose any index for the `from` and `to` arguments can lead to some weird results. Alchemist can't check if he result is beautiful or not.

II.3. Branches

Drawing linear molecules is nice but being able to draw molecule with branches is even better. To do so, you can use the `#branch()` function.

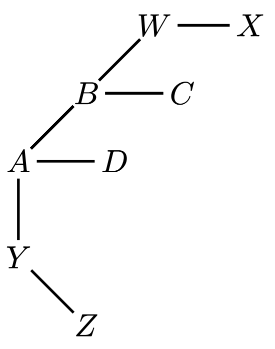
The principle is simple. When you draw normal molecules, each time an element is added, the attachement point is moved accordingly to the added object. Drawing a branch is a way to tell alchemist that you want the attachement point to say the same for the others elements outside the branch. The only constraint is that the branch must start with a link.

```
#skeletize({
  molecule("A")
  single()
  molecule("B")
  branch({
    single(angle:1)
    molecule("W")
    single()
    molecule("X")
  })
  single()
  molecule("C")
})
```



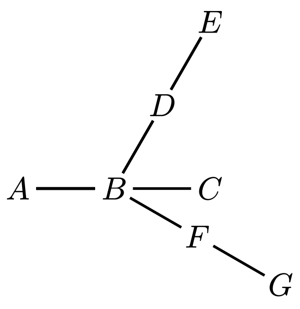
It is of course possible to have nested branches or branches with the same starting point.

```
#skeletize({
  molecule("A")
  branch({
    single(angle:1)
    molecule("B")
    branch({
      single(angle:1)
      molecule("W")
      single()
      molecule("X")
    })
    single()
    molecule("C")
  })
  branch({
    single(angle:-2)
    molecule("Y")
    single(angle:-1)
    molecule("Z")
  })
  single()
  molecule("D")
})
```



You can also specify an angle argument like for links. This angle will be then used as the `base-angle` for the branch. It means that all the links with no angle defined will be drawn with this angle.

```
#skeletize({
  molecule("A")
  single()
  molecule("B")
  branch(relative:60deg,{
    single()
    molecule("D")
    single()
    molecule("E")
  })
  branch(relative:-30deg,{
    single()
    molecule("F")
    single()
    molecule("G")
  })
  single()
  molecule("C")
})
```

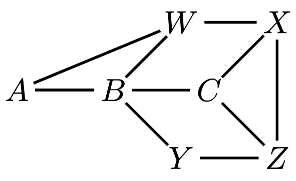


II.4. Link distant atoms

II.4.1. Basic usage

From then, the only way to link atoms is to use links functions and putting them one after the other. This doesn't allow to do cycles or to link atoms that are not next to each other in the code. The way alchemist handle this is with the `links` and `name` arguments of the `#molecule()` function.

```
    #skeletize({
    molecule(name: "A", "A")
    single()
    molecule("B")
    branch({
      single(angle: 1)
      molecule(
        "W",
        links: (
          "A": single(),
        ),
      )
      single()
      molecule(name: "X", "X")
    })
    branch({
      single(angle: -1)
      molecule("Y")
      single()
      molecule(
        name: "Z",
        "Z",
        links: (
          "X": single(),
        ),
      )
    })
    single()
    molecule(
      "C",
      links: (
        "X": single(),
        "Z": single(),
      ),
    )
  })
```

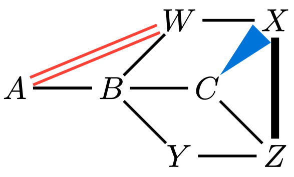


In this example, we can see that the molecules are linked to the molecules defined before with the `name` argument. Note that you can't link to a molecule that is defined after the current one because the name is not defined yet. It's a limitation of the current implementation.

II.4.2. Customizing links

If you look at the previous example, you can see that the links used in the `links` argument are functions. This is because you can still customize the links as you want. The only thing that is not taken into account are the `length` and `angle` arguments. It means that you can change color, `from` and `to` arguments, etc.

```
#skeletize({
  molecule(name: "A", "A")
  single()
  molecule("B")
  branch({
    single(angle: 1)
    molecule(
      "W",
      links: (
        "A": double(stroke: red),
      ),
    )
    single()
    molecule(name: "X", "X")
  })
  branch({
    single(angle: -1)
    molecule("Y")
    single()
    molecule(
      name: "Z",
      "Z",
      links: (
          "X": single(stroke: black
+ 3pt),
      ),
    )
  })
  single()
  molecule(
    "C",
    links: (
        "X": cram-filled-left(fill:
blue),
      "Z": single(),
    ),
  )
})
```



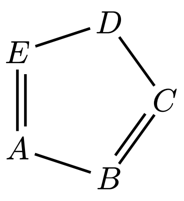
II.5. Cycles

II.5.1. Basic usage

Using branches and `links` arguments, you can draw cycles. However, depending on the number of faces, the angle calculation is fastidious. To help you with that, you can use the `#cycle()` function.

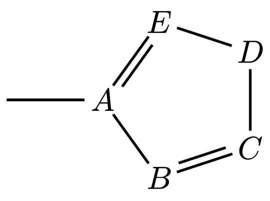
The default behavior if the angle is 0 deg is to be placed in a way that the last link is vertical.

```
#skeletize({
  molecule("A")
  cycle(5, {
    single()
    molecule("B")
    double()
    molecule("C")
    single()
    molecule("D")
    single()
    molecule("E")
    double()
  })
})
```



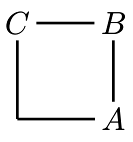
If the angle is not 0 deg or if the `align` argument is set, the cycle will be drawn in relation with the relative angle of the last link.

```
#skeletize({
  single()
  molecule("A")
  cycle(5, align: true, {
    single()
    molecule("B")
    double()
    molecule("C")
    single()
    molecule("D")
    single()
    molecule("E")
    double()
  })
})
```



A cycle must start by a link and if there is more links than the number of faces, the excess links will be ignored. Nevertheless, it is possible to have less links than the number of faces.

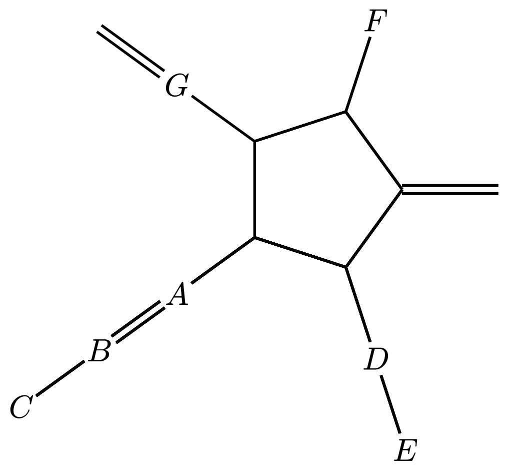
```
#skeletize({
  cycle(4,{
    single()
    molecule("A")
    single()
    molecule("B")
    single()
    molecule("C")
    single()
    molecule("D")
  })
})
```



II.5.2. Branches in cycles

It is possible to add branches in cycles. You can add a branch at any point of the cycle. The default angle of the branch will be set in a way that it is the bisector of the two links that are next to the branch.

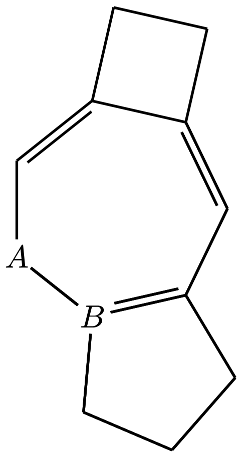
```
#skeletize({
  cycle(5,{
    branch({
      single()
      molecule("A")
      double()
      molecule("B")
      single()
      molecule("C")
    })
    single()
    branch({
      single()
      molecule("D")
      single()
      molecule("E")
    })
    single()
    branch({
      double()
    })
    single()
    branch({
      single()
      molecule("F")
    })
    single()
    branch({
      single()
      molecule("G")
      double()
    })
    single()
    single()
    single()
    single()
  })
})
```



II.5.3. Cycles imbrication

Like branches, you can add cycles in cycles. By default the cycle will be placed in a way that the two cycles share a common link.

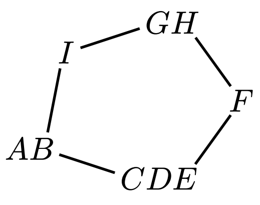
```
#skeletize({
  molecule("A")
  cycle(7,{
    single()
    molecule("B")
    cycle(5,{
      single()
      single()
      single()
      single()
    })
    double()
    single()
    double()
    cycle(4,{
      single()
      single()
      single()
    })
    single()
    double()
    single()
  })
})
```



II.5.4. Issues with atom groups

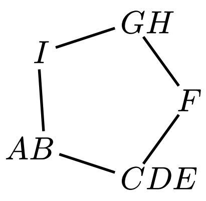
Cycles by default have an issue with atom groups with multiples atoms. The links are not well placed for the cycle to be drawn correctly.

```
#skeletize({
  molecule("AB")
  cycle(5,{
    single()
    molecule("CDE")
    single()
    molecule("F")
    single()
    molecule("GH")
    single()
    molecule("I")
    single()
  })
})
```



To fix that, you have to use the `from` and `to` arguments of the links to specify the starting and ending points of the links.

```
#skeletize({
  molecule("AB")
  cycle(5,{
    single(from: 1, to: 0)
    molecule("CDE")
    single(from: 0)
    molecule("F")
    single(to: 0)
    molecule("GH")
    single(from: 0)
    molecule("I")
    single(to: 1)
  })
})
```



II.5.5. Arcs

It is possible to draw arcs in cycles. The `arc` argument is a dictionary with the following entries:

Argument
`⟨start⟩: 0deg` `angle`

Angle at which the arc starts.

Argument
`⟨end⟩: 360deg` `angle`

Angle at which the arc ends.

Argument
`⟨delta⟩: none` `angle`

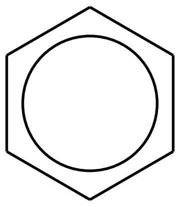
Angle of the arc in degrees.

Argument
`⟨radius⟩: none` `float`

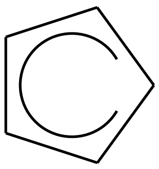
Radius of the arc in percentage of the smallest distance between two opposite atoms in the cycle. By default, it is set to 0.7 for cycle with more than 4 faces and 0.5 for cycle with 4 or 3 faces.

Any styling argument of the cetz `arc` function can be used.

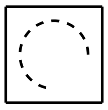
```
#skeletize({
  cycle(6, arc:(:), {
    single()
    single()
    single()
    single()
    single()
    single()
  })
})
```



```
#skeletize({
   cycle(5, arc:(start: 30deg, end:
330deg), {
     single()
     single()
     single()
     single()
     single()
  })
})
```



```
#skeletize({
  cycle(4, arc:(start: 0deg, delta:
270deg, stroke: (paint: black, dash:
"dashed")), {
     single()
     single()
     single()
     single()
  })
})
```



II.6. Custom links

Using the `#build-link()` function, you can create your own links. The function passed as argument to `#build-link()` must takes three arguments:

- The length of the link
- The cetz context of the drawing environment
- A dictionary of named arguments that can be used to configure the links

You can then draw anything you want using the cetz functions. For instance, here is the code for the `single` link:

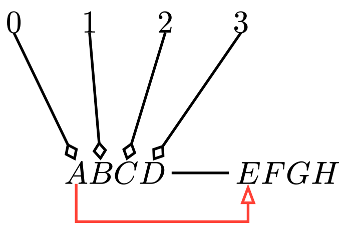
```
#let single = build-link((length, _, args) => {
  import cetz.draw: *
  line((0, 0), (length, 0), stroke: args.at("stroke", default: black))
})
```

II.7. Integration with cetz

II.7.1. Molecules

If you name your molecules with the `name` argument, you can use them in cetz code. The name of the molecule is the name of the cetz object. Accessing to atoms is done by using the anchors numbered by the index of the atom in the molecule.

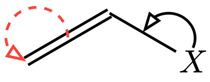
```
#skeletize({
  import cetz.draw: *
  molecule("ABCD", name: "A")
  single()
  molecule("EFGH", name: "B")
  line(
    "A.0.south",
    (rel: (0, -0.5)),
    (to: "B.0.south", rel: (0, -0.5)),
    "B.0.south",
    stroke: red,
    mark: (end: ">"),
  )
  for i in range(0, 4) {
    content((-2 + i, 2), $#i$, name: "label-" + str(i))
    line(
      (name: "label-" + str(i), anchor: "south"),
      (name: "A", anchor: (str(i), "north")),
      mark: (end: "<>"),
    )
  }
})
```



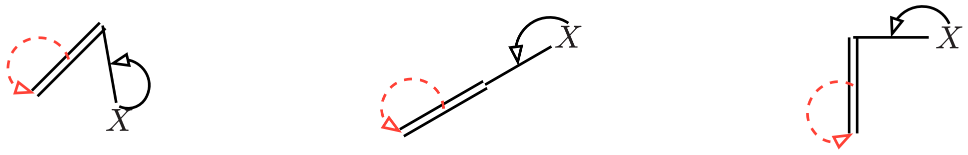
II.7.2. Links

If you name your links with the `name` argument, you can use them in cetz code. The name of the link is the name of the cetz object. It exposes the same anchors as the `line` function of cetz.

```
#skeletize({
  import cetz.draw: *
  double(absolute: 30deg, name: "l1")
  single(absolute: -30deg, name: "l2")
  molecule("X", name: "X")
  hobby(
    "l1.50%",
    ("l1.start", 0.5, 90deg, "l1.end"),
    "l1.start",
    stroke: (paint: red, dash: "dashed"),
    mark: (end: ">"),
  )
  hobby(
    (to: "X.north", rel: (0, 1pt)),
    ("l2.end", 0.4, -90deg, "l2.start"),
    "l2.50%",
    mark: (end: ">"),
  )
})
```



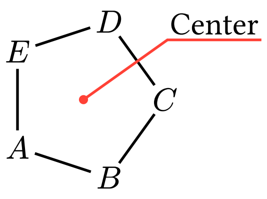
Here, all the used coordinates for the arrows are computed using relative coordinates. It means that if you change the position of the links, the arrows will be placed accordingly without any modification.



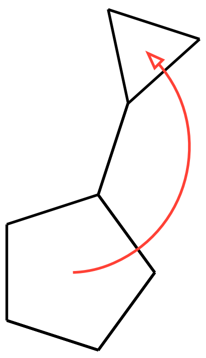
II.7.3. Cycles centers

The cycles centers can be accessed using the name of the cycle. If you name a cycle, an anchor will be placed at the center of the cycle. If the cycle is incomplete, the missing vertex will be approximated based on the last link and the `atom-sep` value. This will in most cases place the center correctly.

```
#skeletize({
  import cetz.draw: *
  molecule("A")
  cycle(
    5,
    name: "cycle",
    {
      single()
      molecule("B")
      single()
      molecule("C")
      single()
      molecule("D")
      single()
      molecule("E")
      single()
    },
  )
  content(
    (to: "cycle", rel: (angle: 30deg, radius: 2)),
    "Center",
    name: "label",
  )
  line(
    "cycle",
    (to: "label.west", rel: (-1pt, -.5em)),
    (to: "label.east", rel: (1pt, -.5em)),
    stroke: red,
  )
  circle(
    "cycle",
    radius: .1em,
    fill: red,
    stroke: red,
  )
})
```



```
#skeletize({
  import cetz.draw: *
  cycle(5, name: "c1", {
    single()
    single()
    single()
    branch({
      single()
      cycle(3, name: "c2", {
        single()
        single()
        single()
      })
    })
    single()
    single()
  })
  hobby(
    "c1",
    ("c1", 0.5, -60deg, "c2"),
    "c2",
    stroke: red,
    mark: (end: ">"),
  )
})
```



Part III.
Index