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.4.

```
#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 argu- │
│   ment 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({                        $H_2O$
      molecule("H_2O")
    })
```

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

---- 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`
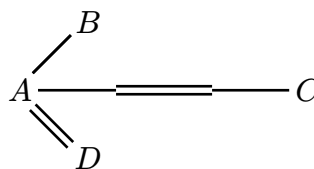
The string representing the molecule

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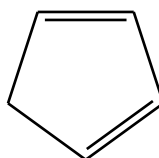
```
#skeletize({
  molecule("A")
  branch({
    single(angle:1)
    molecule("B")
  })
  branch({
    double(angle: -1)
    molecule("D")
  })
  single()
  double()
  single()
  molecule("C")
})
```

#cycle(..⟨args⟩)

Create a regular cycle of molecules

```
#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

4

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({                          A ——— B
  molecule("A")
  single()
  molecule("B")
})
```

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

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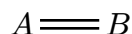
#**double**

Draw a double line between two molecules

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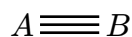
```
#skeletize({                          A═══B
  molecule("A")
  double(
    stroke: orange + 2pt,
    gap: .8em
  )
  molecule("B")
})
```

#**triple**

Draw a triple line between two molecules

```
#skeletize({                          A≡══B
  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({                          A═══B
  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({                          A►B
  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")
})
```

$A \blacktriangleright B$

#**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")
})
```

$A \blacktriangleleft B$

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")
})
```

$A \blacktriangleleft B$

#**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")
})
```

$A\|\|\|\|\|\|\|B$

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")
})
```

$A \| \ | \ | \cdot B$

#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")
})
```

$A\, \cdots \| \| B$

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")
})
```

$A \cdot \ | \ | B$

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.
>
> $$H_2O \qquad H_2O$$
> $$CH_4 \qquad CH_4$$
> $$C_2H_6 \qquad C_2H_6$$
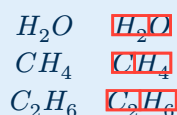
This separation does not have any impact on the drawing of the molecules but it will be useful when we will draw more complex structures.

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.
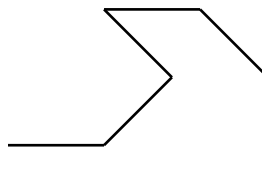
The behavior is not well defined yet.

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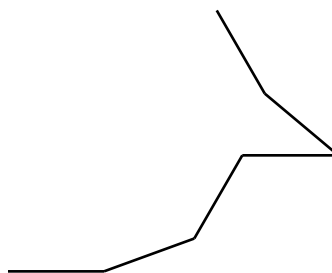
The argument `relative` allows you to define the angle of the link relative to the previous link.

```
    #skeletize({
      single()
      single(relative:20deg)
      single(relative:20deg)
      single(relative:20deg)
      single(relative:20deg)
    })
```
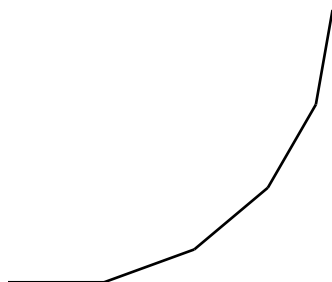
The argument `absolute` allows you to define the angle of the link relative to the abscissa axis.
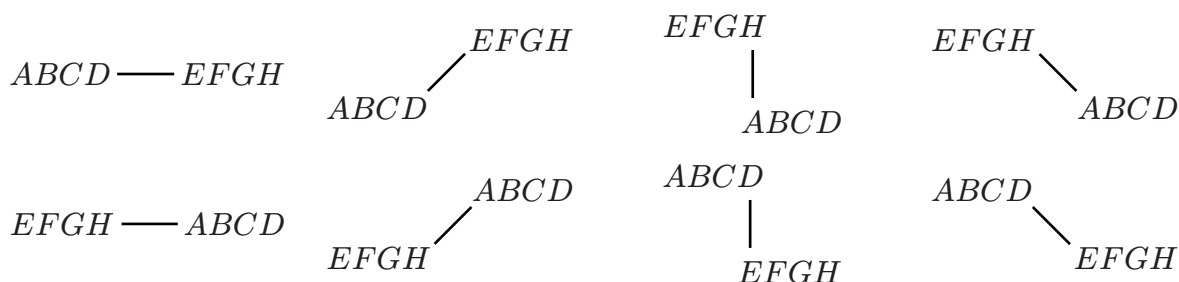
```
    #skeletize({
      single()
      single(absolute:-20deg)
      single(absolute:10deg)
      single(absolute:40deg)
      single(absolute:-90deg)
    })
```
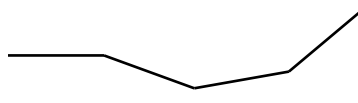
II.2.3. Starting and ending points

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

$$ABCD \longrightarrow EFGH$$

$$\begin{array}{c} EFGH \\ / \\ ABCD \end{array}$$

$$\begin{array}{c} EFGH \\ | \\ ABCD \end{array}$$

$$\begin{array}{c} EFGH \\ \diagdown \\ ABCD \end{array}$$

$$EFGH \longrightarrow ABCD$$

$$\begin{array}{c} ABCD \\ / \\ EFGH \end{array}$$

$$\begin{array}{c} ABCD \\ | \\ EFGH \end{array}$$

$$\begin{array}{c} ABCD \\ \diagdown \\ EFGH \end{array}$$

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.

$$\begin{array}{c} EFGH \\ / \\ ABCD \end{array} \qquad \begin{array}{c} EFGH \\ / \\ ABCD \end{array} \qquad \begin{array}{c} EFGH \\ / \\ ABCD \end{array} \qquad \begin{array}{c} EFGH \\ / \\ ABCD \end{array}$$

> 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")
})
```

$$\begin{array}{c} W \longrightarrow X \\ / \\ A \longrightarrow B \longrightarrow C \end{array}$$
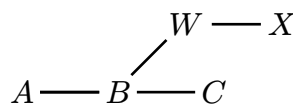
It is of course possible to have nested branches or branches with the same starting point.

II.4. Integration with cetz

Part III.
Index