The "alchemist" package

v0.2.0 MIT

A package to render skeletal formulas using CeTZ

Robotechnic
@Robotechnic

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:0.2.0": *
```

I.1 Initializing drawing environment

To start drawing molecules, you first need to initialize 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.8.

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

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.3.

Argument

⟨body⟩ `drawable`

The module to draw or any cetz drawable object.

`#skeletize-config(⟨config⟩: (:)) → function`

Create a `#skeletize` function with the given configuration. The returned function uses this configuration as default.

Argument

⟨config⟩ `dictionary`

Configuration of the drawing environment. See Section I.3.

I.2 Drawing a molecule directly in Cetz

Sometimes, you may want to draw a molecule directly in cetz. To do so, you can use the `#draw-skeleton` function. This function is what is used internally by the `#skeletize` function.

`#draw-skeleton``(`**`{config}`**`: (:), `**`{body}`**`)` → `drawable`

Argument
`{config}` `dictionary`

Configuration of the drawing environment. See Section I.3.

Argument
`{body}` `drawable`

The module to draw or any cetz drawable object.

Argument
`{name}: none`

If a name is provided, the molecule will be placed in a cetz group with this name.

Argument
`{mol-anchor}: none`

Anchor of the group. It is working the same way as the `anchor` argument of the cetz `group` function. The `default` anchor of the molecule is the east anchor of the first atom or the starting point of the first link.

`#draw-skeleton-config``(`**`{config}`**`: (:))` → `function`

Create a `#draw-skeleton` function with the given configuration. The returned function uses this configuration as default.

Argument
`{config}` `dictionary`

Configuration of the drawing environment. See Section I.3.

The usefulness of this function comes when you want to draw multiple molecules in the same cetz environment. See Section II.8.

I.3 Configuration

The 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 a link with no angle defined will be.

Argument

`⟨fragment-margin⟩: 0.2em` `length`

Default space between a molecule and all its attachments (links and lewis formulae elements).

Argument

`⟨fragment-color⟩: none` `color`

Default color of the fragments. It is used to color the atoms in the molecule.

Argument

`⟨fragment-font⟩: none` `str`

Default font of the fragments. It is used to display the atoms in the molecule.

Argument

`⟨link-over-radius⟩: 0.2` `float` | `length`

Default radius around the links used to hide overlapped links.

I.3.1 Link default style

The default values also contain styling arguments for the links. You can specify default `stroke`, `fill`, `dash`, etc, depending on the link type. The default values for each link are in a dictionary named after the link name.

single

Argument	Default value
stroke	luma(0%)

double

Argument	Default value
gap	0.25em
offset	"center"
offset-coeff	0.85
stroke	luma(0%)

triple

Argument	Default value
gap	0.25em
stroke	luma(0%)

filled-cram

Argument	Default value
stroke	none
fill	luma(0%)
base-length	0.8em

dashed-cram

Argument	Default value
stroke	0.05em + luma(0%)
dash-gap	0.3em
base-length	0.8em
tip-length	0.1em

lewis

Argument	Default value
angle	0deg
radius	0.2em

lewis-single

Argument	Default value
stroke	luma(0%)
fill	luma(0%)
radius	0.1em
gap	0.25em
offset	"top"

lewis-double

Argument	Default value
stroke	luma(0%)
fill	luma(0%)
radius	0.1em
gap	0.25em

lewis-line

Argument	Default value
stroke	luma(0%)
length	0.7em

lewis-rectangle

Argument	Default value
stroke	0.08em + luma(0%)
fill	luma(100%)
height	0.7em
width	0.3em

lewis-charge

Argument	Default value
stroke	luma(0%)
charge	equation(block: false, body: [+])

I.4 Available commands

```
#fragment(
  (mol),
  (name): none,
  (links): (:),
  (lewis): (),
  (vertical): false,
  (ignore-charge): false,
  (colors): none
) → drawable
```

I.4.1 Fragment function

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

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

H_2O

```
#skeletize({
  fragment("H^A_EF^5_4")
})
```

$$\mathrm{H_E^AF_4^5}$$

It is possible to use an equation as a fragment. In this case, the splitting of the equation uses the same rules as in the string case. However, you get some advantages over the string version:

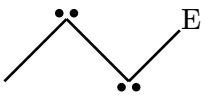
- You can use parentheses to group elements together.
- You have no restriction about what you can put in exponent or indice.

```
#skeletize({
  fragment($C(C H_3)_3$)
})
```

$$\mathrm{C(CH_3)_3}$$

Empty fragments are also possible. They can be useful to draw charges and lewis structures without drawing carbons. In this case, the fragment is invisible and won't take any space in the drawing. However, it can still be linked to other fragments and hooks and it can still have lewis structures. See Section I.4.8 for more details about lewis structures.

```
#skeletize(config: (debug: false), {
  single(angle:1)
  fragment("", lewis: (
    lewis-double(angle: 90deg),
  ))
  single(angle: -1)
  fragment("", lewis: (
    lewis-double(angle: -90deg),
  ))
  single(angle:1)
  fragment("E")
})
```



Argument

⟨mol⟩ `str` | `equation`

The string representing the fragment or an equation of the fragment

Argument

⟨name⟩: none `content`

The name of the fragment. It is used as the cetz name of the fragment and to link other fragments to it.

Argument

⟨links⟩: (:) `dictionary`

The links between this fragment and previous fragments or hooks. The key is the name of the fragment or hook and the value is the link function. See Section I.4.7.

Note that the atom-sep and angle arguments are ignored

Argument

⟨lewis⟩: () `list`

The list of lewis structures to draw around the fragments. See Section I.4.8

Argument

⟨vertical⟩: false `bool`

If true, the fragment is drawn vertically

```
#skeletize({
  fragment("ABCD", vertical: true)
})
```

A
B
C
D

Argument

⟨ignore-charge⟩: false `bool`

If true, charges of the fragment are excluded from the link connections. This is useful when you want to have the center of your connection to the text without having the charge in the way.

```
#skeletize({
  fragment("A")
  single(relative: 90deg)
  fragment("B^-", ignore-charge: true)
})
```

Argument

⟨colors⟩: none — color | list

The color of the fragment. If a list is provided, it colors each group of the fragment with the corresponding color from right to left. If the number of colors is less than the number of groups, the last color is used for the remaining groups. If the number of colors is greater than the number of groups, the extra colors are ignored.

```
#skeletize({
  fragment("ABCD", colors: (red, green, blue))
  single()
  fragment("EFGH", colors: (orange))
})
```

#hook(⟨name⟩) → drawable

I.4.2 Hooks

Create a hook in the fragment. It allows to connect links to the place where the hook is. Hooks are placed at the end of links or at the beginning of the fragment.

Argument

⟨name⟩ — str

The name of the hook

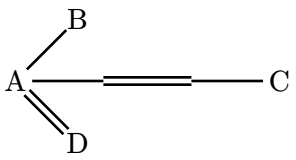
#branch(⟨body⟩, ..⟨args⟩) → drawable

I.4.3 Branch and cycles

Create a branch from the current fragment, 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({
  fragment("A")
  branch({
    single(angle:1)
    fragment("B")
  })
  branch({
    double(angle: -1)
    fragment("D")
  })
  single()
  double()
  single()
  fragment("C")
})
```



Argument

⟨body⟩ drawable

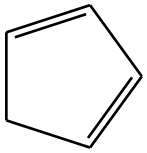
the body of the branch. It must start with a link.

#cycle(⟨faces⟩, ⟨body⟩, ..⟨args⟩) → drawable

Create a regular cycle of fragments 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()
  })
})
```



Argument

⟨faces⟩ `int`

the number of faces of the cycle

Argument

⟨body⟩ `drawable`

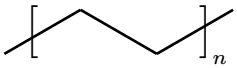
the body of the cycle. It must start and end with a fragment or a link.

```
#parenthesis(
  ⟨body⟩,
  ⟨l⟩: "(",
  ⟨r⟩: ")",
  ⟨align⟩: true,
  ⟨resonance⟩: false,
  ⟨height⟩: none,
  ⟨yoffset⟩: none,
  ⟨xoffset⟩: none,
  ⟨right⟩: none,
  ⟨tr⟩: none,
  ⟨br⟩: none
) → drawable
```

I.4.4 Parenthesis

Encapsulate a drawable between two parenthesis. The left parenthesis is placed at the left of the first element of the body and by default the right parenthesis is placed at the right of the last element of the body.

```
#skeletize(
  config: (
    angle-increment: 30deg
  ), {
  parenthesis(
    l:"[", r:"]",
    br: $n$, {
    single(angle: 1)
    single(angle: -1)
    single(angle: 1)
  })
})
```



For more examples, see Section II.10

Argument

⟨body⟩ drawable

the body of the parenthesis. It must start and end with a fragment or a link.

Argument

⟨l⟩: "(" str

the left parenthesis

Argument

⟨r⟩: ")" str

the right parenthesis

Argument

⟨align⟩: true true

if true, the parenthesis will have the same y position. They will also get sized and aligned according to the body height. If false, they are not aligned and the height argument must be specified.

Argument

⟨resonance⟩: false bool

if true, the parenthesis will be drawn in resonance mode. This means that the left and right parenthesis will be placed outside the molecule. Also, the parenthesis will be separated from the previous and next molecule. This can be true only if the parenthesis is the first element of the skeletal formula or if the previous element is an operator. See Section II.6 for more details.

Argument

⟨height⟩: none — float | length

the height of the parenthesis. If align is true, this argument is optional.

Argument

⟨yoffset⟩: none — float | length | list

the vertical offset of parenthesis. You can also provide a tuple for left and right parenthesis

Argument

⟨xoffset⟩: none — float | length | list

the horizontal offset of parenthesis. You can also provide a tuple for left and right parenthesis

Argument

⟨right⟩: none — str

Sometime, it is not convenient to place the right parenthesis at the end of the body. In this case, you can specify the name of the fragment or link where the right parenthesis should be placed. It is especially useful when the body end by a cycle. See Section II.10.11

Argument

⟨tr⟩: none — content

the exponent content of the right parenthesis

Argument

⟨br⟩: none — content

the indice content of the right parenthesis

`#operator(⟨op⟩, ⟨name⟩: none, ⟨margin⟩: 1em)` → drawable

I.4.5 Operator

Create an operator between two fragments. Creating an operator "resets" the placement of the next fragment. This allows adding multiple molecules in the same skeletal formula. Without this, the next fragment would be placed at the end of the previous one. An important point is that you can't use previous hooks to link two molecules separated by an operator. This element is used in resonance structures (Section II.6) and in some cases to put multiple molecules in the same skeletal formula (as you can set op to none).

Argument

⟨op⟩ — content | str | none

The operator content. It can be a string or content. A none value won't display anything.

```
#skeletize({
 fragment("A")
operator($->$, margin: 1em)
fragment("B")
})
```

A → B

See Section II.6 for more examples.

Argument

⟨name⟩: none — str

The name of the operator.

Argument

⟨margin⟩: 1em — float | length

The margin between the operator and previous/next molecule.

#hide(⟨body⟩, ⟨bounds⟩: true) → drawable

I.4.6 Hiding part of the molecule

This element allows hiding part of the molecule. It can be used to hide the part of the molecule that is not relevant for the current discussion. It can also be used to create some animation effects by hiding and showing different parts of the molecule. Note that the hidden part is still present in the drawing and can be linked to other fragments. This means that you can hide a part of the molecule and still link it to other fragments or hooks. The hidden part is also still present in the cetz record, which means that you can still use it in the cetz drawing. The only thing that is hidden is the drawing of the hidden part.

Argument

⟨body⟩ — drawable

The body to hide. It can be any drawable.

Argument

⟨bounds⟩: true — bool

If true, the hidden part keeps the same bounding box as if it was not hidden. If false, the hidden part doesn't take any space in the drawing.

I.4.7 Link functions

Common arguments

Link functions are used to draw links between fragments. 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

`⟨atom-sep⟩: 3em` `length`

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

Argument

`⟨from⟩` `int`

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

Argument

`⟨links⟩` `dictionary`

Dictionary of links to other fragments or hooks. The key is the name of the fragment or the hook and the value is the link function.

Argument

`⟨over⟩` `dictionary` | `str`

If the link overlaps other links and is drawn over them, this argument can be used to specify that you want to hide the overlapped links. There are three possible values:

- `str` : The name of the link to overlap
- `dictionary` : A dictionary containing the keys:
  - `name`: The name of the link to overlap
  - `length`: The length of the overlap mask
  - `radius`: The distance from the link center to the edge of the mask
- `array` : An array of the two above.

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 four arguments: the length of the link, the alchemist context, the cetz context, and a dictionary of named arguments that can be used to configure the links

#single

Draw a single line between two fragments

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

A —— B

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

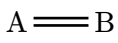
```
#skeletize({
  fragment("A")
  single(stroke: red + 5pt)
  fragment("B")
})
```

A ▬ B

#double

Draw a double line between two fragments

```
#skeletize({
  fragment("A")
  double()
  fragment("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

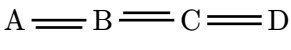
```
#skeletize({
  fragment("A")
  double(
    stroke: orange + 2pt,
    gap: .8em
  )
  fragment("B")
})
```

It is also possible to only change the color and width of the lines seperately with the `stroke-left` and `stroke-right` arguments.

```
#skeletize({
  double(stroke: 2pt, stroke-right: red, stroke-left: (dash: "dashed"))
})
```

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.

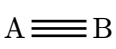
```
#skeletize({
  fragment("A")
  double(offset: "right")
  fragment("B")
  double(offset: "left")
  fragment("C")
  double(offset: "center")
  fragment("D")
})
```



#triple

Draw a triple line between two fragments

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

It is also possible to only change the color and width of the lines seperately with the `stroke-left`, `stroke-center` and `stroke-right` arguments.

```
#skeletize({
   triple(stroke: 2pt, stroke-left: red, stroke-center: green, stroke-
right: blue)
})
```

#cram-filled-right

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

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

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

A  B

#cram-filled-left

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

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

#cram-hollow-right

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

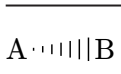
#cram-dashed-right

```
#skeletize({
  fragment("A")
  cram-dashed-right(
    stroke: red + 2pt,
    base-length: 2em,
    tip-length: 1em,
    dash-gap: .5em
  )
  fragment("B")
})
```

#cram-dashed-left

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

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

#plus-link

Draw a plus sign between two fragments

```
#skeletize({
  fragment("A")
  plus-link()
  fragment("B")
})
```

A + B

You can change the filling, size and stroke of the glyph with the `fill`, `size` and `stoke` arguments. The default values are the parent `text` parameters.

I.4.8 Lewis structures

All the lewis elements have two common arguments to control their position:

Argument

`⟨angle⟩: 0deg` `angle`

Angle of the lewis element relative to the abscissa axis.

Argument

`⟨fragment-margin⟩: 0.2em` `length`

Space between the lewis element and the fragment.

`#build-lewis(⟨draw-function⟩)` → `function`

Create a lewis function that is then used to draw a lewis formulae element around the fragment

Argument

`⟨draw-function⟩` `function`

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

`#lewis-single`

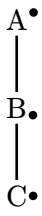
draw a sigle electron around the fragment

It is possible to change the distance from the center of the electron with the `gap` argument.

The position of the electron is set by the `offset` argument. Available values are:
- “top”: the electron is placed above the fragment center line
- “bottom”: the electron is placed below the fragment center line
- “center”: the electron is placed at the fragment center line

It is also possible to change the `radius`, `stroke` and `fill` arguments

```
#skeletize({
  fragment("A", lewis:(
    lewis-single(offset: "top"),
  ))
  single(angle:-2)
  fragment("B", lewis:(
    lewis-single(offset: "bottom"),
  ))
  single(angle:-2)
  fragment("C", lewis:(
    lewis-single(offset: "center"),
  ))
})
```



#lewis-double

Draw a pair of electron around the fragment

It is possible to change the distance from the center of the electron with the `gap` argument. It is also possible to change the `radius`, `stroke` and `fill` arguments

```
#skeletize({
  fragment("A", lewis:(
    lewis-double(),
    lewis-double(angle: 90deg),
    lewis-double(angle: 180deg),
    lewis-double(angle: -90deg)
  ))
})
```

#lewis-line

Draw a pair of electron liked by a single line

It is possible to change the length of the line with the `lenght` argument. It is also possible to change the `stroke` agument

```
#skeletize({
  fragment("B", lewis:(
    lewis-line(angle: 45deg),
    lewis-line(angle: 135deg),
    lewis-line(angle: -45deg),
    lewis-line(angle: -135deg)
  ))
})
```

#lewis-rectangle

Draw a rectangle to denote a lone pair of electrons

It is possible to change the height and width of the rectangle with the `height` and `width` arguments. It is also possible to change the `fill` and `stroke` arguments

```
#skeletize({
  fragment("C", lewis:(
    lewis-rectangle(),
    lewis-rectangle(angle: 180deg)
  ))
})
```

C

#lewis-charge

Draw a positive charge around the fragment

```
#skeletize({
  fragment("A", lewis:(
    lewis-charge(charge: $+$, angle: 45deg),
  ))
  single(angle:-2)
  fragment("B", lewis:(
    lewis-charge(charge: $-$, angle: 45deg),
  ))
})
```

Part II
Drawing molecules

II.1 Atoms

In alchemist, the name of the function `#fragment` is used to create a group of atom in a molecule. A fragment is in our case something of the form: optional number + capital letter + optional lowercase letter optionally followed by a charge, an exponent or a subscript.

> 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 
> R'^2 

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 some links available with the package (see Section I.4.7) and you can create your own links with the `#build-link` function but they all share the same base arguments used to control their behavior.

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.

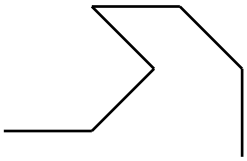
The behavior is not well defined yet.

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.

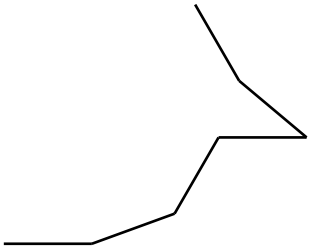
The argument `angle` is a multiplier of the `angle-increment` argument.

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



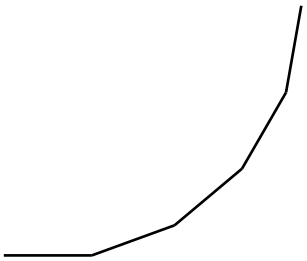
Changing the `angle-increment` argument of the drawing environment will change the angle of the links.

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



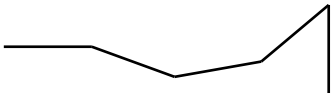
The argument `relative` allows you to define the angle of the link relative to the previous link.

```
#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.

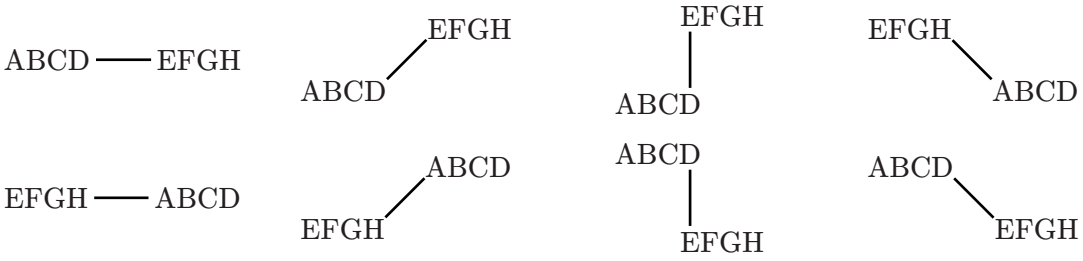
```
#skeletize({
  single()
  single(absolute:-20deg)
  single(absolute:10deg)
  single(absolute:40deg)
  single(absolute:-90deg)
})
```



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



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 fragment.

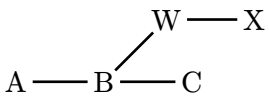
> The fact that you can choose any index for the `from` and `to` arguments can lead to some weird results. Alchemist can't check if the result is beautiful or not.

II.3 Branches

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

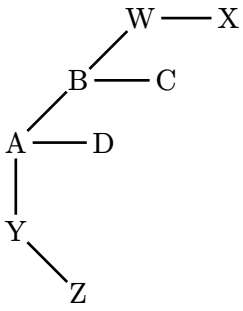
The principle is simple. When you draw normal molecules, each time an element is added, the attachment point is moved accordingly to the added object. Drawing a branch is a way to tell alchemist that you want the attachment point to say the same for the others elements outside the branch. The only constraint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

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



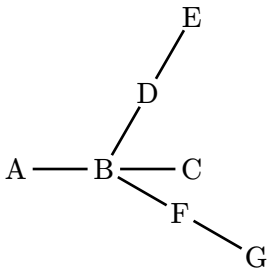
It is of course possible to have nested branches or branches with the same starting point.

```
#skeletize({
  fragment("A")
  branch({
    single(angle:1)
    fragment("B")
    branch({
      single(angle:1)
      fragment("W")
      single()
      fragment("X")
    })
    single()
    fragment("C")
  })
  branch({
    single(angle:-2)
    fragment("Y")
    single(angle:-1)
    fragment("Z")
  })
  single()
  fragment("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({
  fragment("A")
  single()
  fragment("B")
  branch(relative:60deg,{
    single()
    fragment("D")
    single()
    fragment("E")
  })
  branch(relative:-30deg,{
    single()
    fragment("F")
    single()
    fragment("G")
  })
  single()
  fragment("C")
})
```

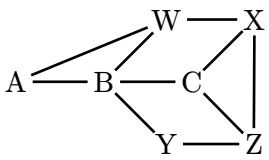


II.4 Link distant atoms

II.4.1 Basic usage

From then on, the only way to link atoms is to use link functions and put them one after the other. This doesn't allow doing cycles or linking atoms that are not next to each other in the code. The way alchemist handles this is with the `links` and `name` arguments of the `#fragment` function.

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

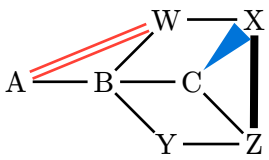


In this example, we can see that the molecules are linked to the molecules defined before with the `name` argument. Note that you can't link to a molecule that is defined after the current one because the name is not defined yet. It's a limitation of the current implementation.

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



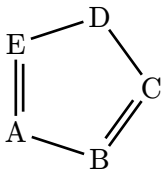
II.5 Cycles

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.

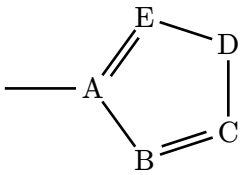
The default behavior if the angle is 0 deg is to be placed in a way that the last link is vertical.

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



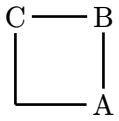
If the angle is not $0\,\mathrm{deg}$ or if the `align` argument is set, the cycle will be drawn in relation with the relative angle of the last link.

```
#skeletize({
  single()
  fragment("A")
  cycle(5, align: true, {
    single()
    fragment("B")
    double()
    fragment("C")
    single()
    fragment("D")
    single()
    fragment("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.

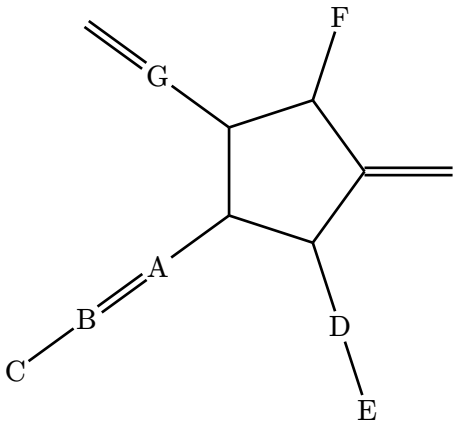
```
#skeletize({
  cycle(4,{
    single()
    fragment("A")
    single()
    fragment("B")
    single()
    fragment("C")
    single()
    fragment("D")
  })
})
```



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.

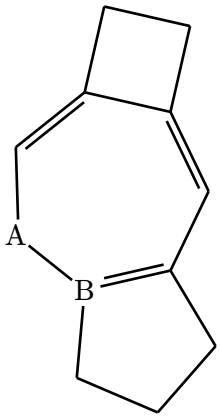
```
#skeletize({
  cycle(5,{
    branch({
      single()
      fragment("A")
      double()
      fragment("B")
      single()
      fragment("C")
    })
    single()
    branch({
      single()
      fragment("D")
      single()
      fragment("E")
    })
    single()
    branch({
      double()
  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

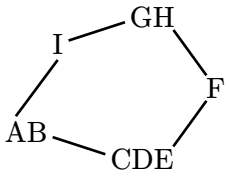
```
#skeletize({
  fragment("A")
  cycle(7,{
    single()
    fragment("B")
    cycle(5,{
      single()
      single()
      single()
      single()
    })
    double()
    single()
    double()
    cycle(4,{
      single()
      single()
      single()
    })
    single()
    double()
    single()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



II.5.4 Issues with atom groups

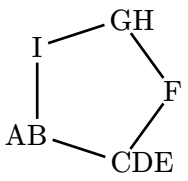
Cycles by default have an issue with atom groups with multiples atoms. The links are not well placed for the cycle to be drawn correctly.

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

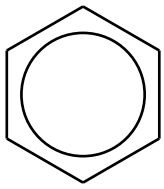
Argument

`(radius): none` `float`

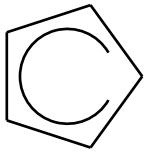
Radius of the arc in percentage of the smallest distance between two opposite atoms in the cycle. By default, it is set to 0.7 for cycle with more than 4 faces and 0.5 for cycle with 4 or 3 faces.

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()
  })
})
```



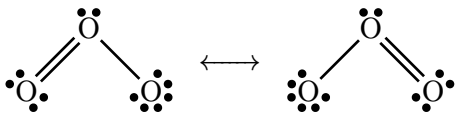
```
#skeletize({
   cycle(4, arc:(start: 0deg, delta:
270deg, stroke: (paint: black, dash:
"dashed")), {
    single()
    single()
    single()
    single()
  })
})
```

II.6 Resonance structures

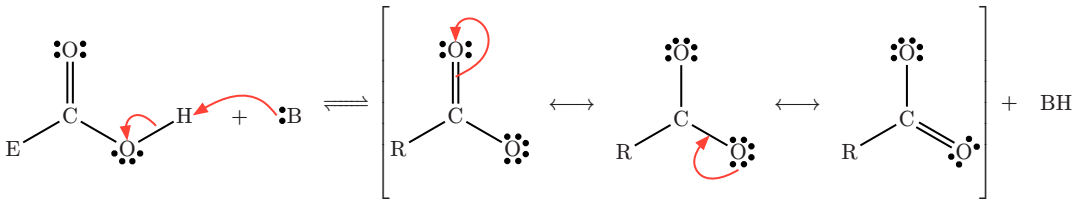
Chemfig allows you to draw resonance formulae. To use this feature, the package provides the `#operator` function and the `resonance` argument of the `#parenthesis` function. With these two things, you can draw formulae with molecules.

For instance, with the `#operator` function, you can draw the resonance formulae of ozone:

```
#skeletize({
  fragment("O", lewis: (
    lewis-double(angle: 135deg),
    lewis-double(angle: -45deg),
  ))
  double(angle: 1)
  fragment("O", lewis: (
    lewis-double(angle: 90deg),
  ))
  single(angle: -1)
  fragment("O", lewis: (
    lewis-double(angle: 45deg),
    lewis-double(angle: -45deg),
    lewis-double(angle: -135deg),
  ))

  operator(math.stretch(sym.arrow.r.l,
size: 2em))

    fragment("O", lewis: (
      lewis-double(angle: 135deg),
      lewis-double(angle: -135deg),
      lewis-double(angle: -45deg),
    ))
    single(angle: 1)
    fragment("O", lewis: (
      lewis-double(angle: 90deg),
    ))
    double(angle: -1)
    fragment("O", lewis: (
      lewis-double(angle: 45deg),
      lewis-double(angle: -135deg),
    ))

})
```



For parenthesis, you can use them like usual parenthesis but with the `resonance` argument set to `true`. If the resonance argument is not set, the parenthesis will be drawn as usual. Furthermore, operators are not allowed in normal parenthesis while they are allowed in resonance parenthesis.



This example is from the test suite of alchemist. It is too long to be displayed in the manual but you can find it in the git repository in the folder `tests/resonance/test.typ` or directly in the manual source file. It was made by @Kusaanko[1].

II.7 Custom links

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

- The length of the link
- The alchemist context
- 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, ctx, _, args) => {
  import cetz.draw: *
  line((0, 0), (length, 0), stroke: args.at("stroke", default:
  ctx.config.single.stroke))
})
```

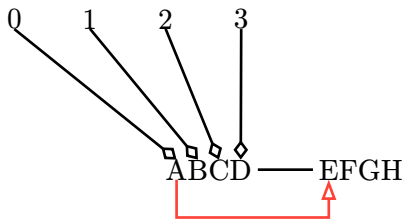
II.8 Integration with Cetz

II.8.1 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.

[1] https://github.com/Kusaanko

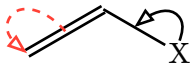
```
#skeletize({
  import cetz.draw: *
  fragment("ABCD", name: "A")
  single()
  fragment("EFGH", name: "B")
  line(
    "A.0.south",
    (rel: (0, -0.5)),
    (to: "B.0.south", rel: (0, -0.5)),
    "B.0.south",
    stroke: red,
    mark: (end: ">"),
  )
  for i in range(0, 4) {
    content((-2 + i,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



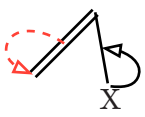
II.8.2 Links

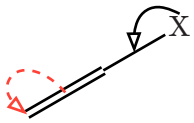
If you name your links with the `name` argument, you can use them in cetz code. The name of the link is the name of the cetz object. It exposes the same anchors as the `line` function of cetz.

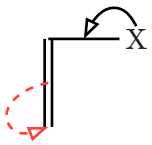
```
#skeletize({
  import cetz.draw: *
  double(absolute: 30deg, name: "l1")
  single(absolute: -30deg, name: "l2")
  fragment("X", name: "X")
  hobby(
    "l1.50%",
    ("l1.start", 0.5, 90deg, "l1.end"),
    "l1.start",
    stroke: (paint: red, dash: "dashed"),
    mark: (end: ">"),
  )
  hobby(
    (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.



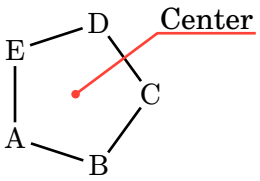




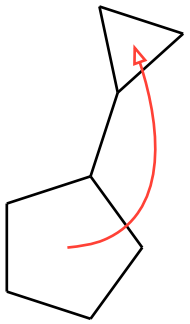
II.8.3 Cycles centers

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: *
  fragment("A")
  cycle(
    5,
    name: "cycle",
    {
      single()
      fragment("B")
      single()
      fragment("C")
      single()
      fragment("D")
      single()
      fragment("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,
  )
})
```



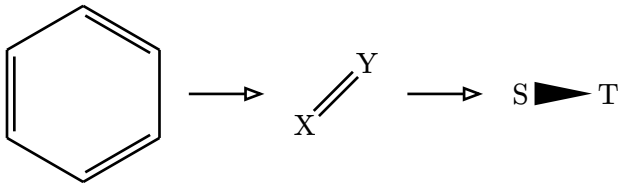
```
#skeletize({
  import cetz.draw: *
  cycle(5, name: "c1", {
    single()
    single()
    single()
    branch({
      single()
      cycle(3, name: "c2", {
        single()
        single()
        single()
      })
    })
    single()
    single()
  })
  hobby(
    "c1",
    ("c1", 0.5, -60deg,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



II.8.4 Multiple molecules

Alchemist allows you to draw multiple molecules in the same cetz environment. This is useful when you want to draw things like reactions.

```
#cetz.canvas({
  import cetz.draw: *
  draw-skeleton(name: "mol1", {
    cycle(6, {
      single()
      double()
      single()
      double()
      single()
      double()
    })
  })
  line((to: "mol1.east", rel: (1em, 0)), (rel: (1, 0)), mark: (end: ">"))
  set-origin((rel: (1em, 0)))
  draw-skeleton(name: "mol2", mol-anchor: "west", {
      fragment("X")
      double(angle: 1)
      fragment("Y")
    })
  line((to: "mol2.east", rel: (1em, 0)), (rel: (1, 0)), mark: (end: ">"))
  set-origin((rel: (1em, 0)))
  draw-skeleton(name: "mol3", {
    fragment("S")
    cram-filled-right()
    fragment("T")
  })
})
```



II.9 Integration with Touying

II.9.1 Simple animation

Using `touying-reducer` Alchemist can be used with Touying presentations. This provides compatibility with the `pause` animation.

```
#import "@preview/touying:0.6.3": *
#import "@preview/alchemist:0.1.10": *

#import themes.metropolis: *

#let skeletize = touying-reducer.with(reduce: skeletize, cover: hide)


#show: metropolis-theme.with(aspect-ratio: "16-9")

#slide[
  #skeletize({
    fragment("A")
    (pause,)
    single()
    fragment("B")
  })
]
```

II.9.2 only and uncover

We can also use `only` and `uncover` but this require a bit of technique:

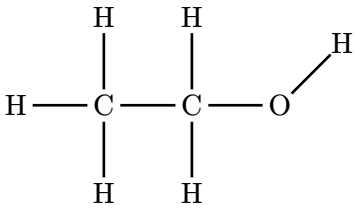
```
#import "@preview/touying:0.6.3": *
#import "@preview/alchemist:0.1.10": *

#import themes.metropolis: *

#let skeletize = touying-reducer.with(reduce: skeletize, cover: hide)

#show: metropolis-theme.with(aspect-ratio: "16-9")

#slide(repeat: 3, self => {
  skeletize({
    let self = utils.merge-dicts(self, config-methods(cover:
    utils.method-wrapper(hide)))
    let (uncover, only, alternatives) = utils.methods(self)
    fragment("A")
    (only(1, {
      double()
      fragment("B")
    }),)
    (only(2, {
       single()
       fragment("C")
    }),)
  })
]
```

II.10 Examples

The following examples are the same ones as in the Chemfig documentation. They are here for two purposes: To show you how to draw the same structures with Alchemist and to show you how to use the package.

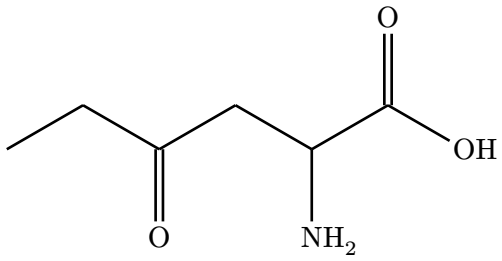
II.10.1 Ethanol

```
#skeletize({
    fragment("H")
    single()
    fragment("C")
    branch({
        single(angle:2)
        fragment("H")
    })
    branch({
        single(angle:-2)
        fragment("H")
    })
    single()
    fragment("C")
    branch({
        single(angle:-2)
        fragment("H")
    })
    branch({
        single(angle:2)
        fragment("H")
    })
    branch({
        single()
        fragment("O")
        single(angle: 1)
        fragment("H")
    })
})
```



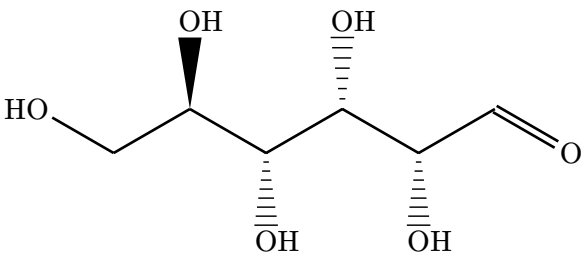
II.10.2 2-Amino-4-oxohexanoic acid

```
#skeletize(
  config: (angle-increment: 30deg),
  {
  single(angle:1)
  single(angle:-1)
  branch({
    double(angle:-3)
    fragment("O")
  })
  single(angle:1)
  single(angle:-1)
  branch({
    single(angle:-3)
    fragment("NH_2")
  })
  single(angle:1)
  branch({
    double(angle:3)
    fragment("O")
  })
  single(angle:-1)
  fragment("OH")
})
```



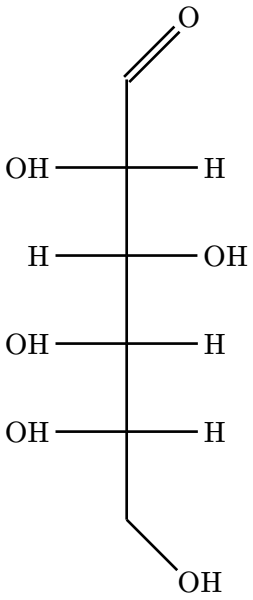
II.10.3 D-Glucose

```
#skeletize(
  config: (angle-increment: 30deg),
  {
  fragment("HO")
  single(angle:-1)
  single(angle:1)
  branch({
    cram-filled-left(angle: 3)
    fragment("OH")
  })
  single(angle:-1)
  branch({
    cram-dashed-left(angle: -3)
    fragment("OH")
  })
  single(angle:1)
  branch({
    cram-dashed-left(angle: 3)
    fragment("OH")
  })
  single(angle:-1)
  branch({
    cram-dashed-left(angle: -3)
    fragment("OH")
  })
  single(angle:1)
  branch({
    double(angle: -1)
    fragment("O")
  })
})
```



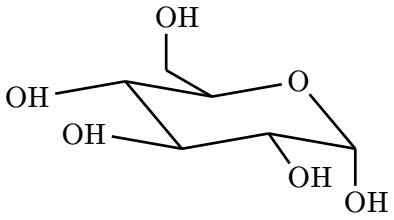
II.10.4 Fisher projection

```
#let fish-left = {
  single()
  branch({
    single(angle:4)
    fragment("H")
  })
  branch({
    single(angle:0)
    fragment("OH")
  })
}
#let fish-right = {
  single()
  branch({
    single(angle:4)
    fragment("OH")
  })
  branch({
    single(angle:0)
    fragment("H")
  })
}
#skeletize(
  config: (base-angle: 90deg),
  {
  fragment("OH")
  single(angle:3)
  fish-right
  fish-right
  fish-left
  fish-right
  single()
  double(angle: 1)
  fragment("O")
})
```



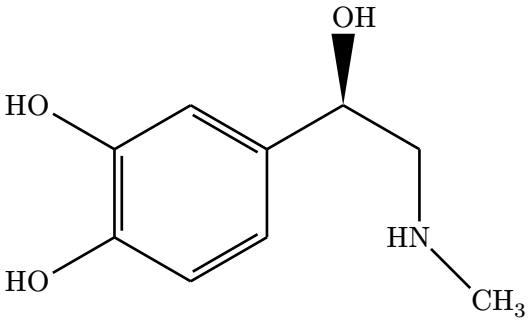
II.10.5 α-D-glucose

```
#skeletize({
  hook("start")
  branch({
    single(absolute: 190deg)
    fragment("OH")
  })
  single(absolute: -50deg)
  branch({
    single(absolute: 170deg)
    fragment("OH")
  })
  single(absolute: 10deg)
  branch({
    single(
      absolute: -55deg,
      atom-sep: 0.7
    )
    fragment("OH")
  })
  single(absolute: -10deg)
  branch({
    single(angle: -2, atom-sep: 0.7)
    fragment("OH")
  })
  single(absolute: 130deg)
  fragment("O")
    single(absolute:  190deg,  links:
("start": single()))
  branch({
    single(
      absolute: 150deg,
      atom-sep: 0.7
    )
    single(angle: 2, atom-sep: 0.7)
    fragment("OH")
  })
})
```



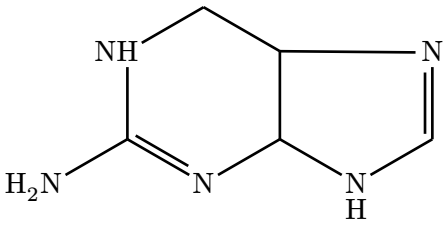
II.10.6 Adrenaline

```
#skeletize({
  cycle(6, {
    branch({
      single()
      fragment("HO")
    })
    single()
    double()
    cycle(6,{
      single(stroke:transparent)
      single(
        stroke:transparent,
        to: 1
      )
      fragment("HN")
      branch({
        single(angle:-1)
        fragment("CH_3")
      })
      single(from:1)
      single()
      branch({
        cram-filled-left(angle: 2)
        fragment("OH")
      })
      single()
    })
    single()
    double()
    single()
    branch({
      single()
      fragment("HO")
    })
    double()
  })
})
```



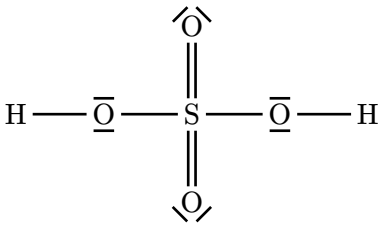
II.10.7 Guanine

```
#skeletize({
  cycle(6, {
    branch({
      single()
      fragment("H_2N")
    })
    double()
    fragment("N")
    single()
    cycle(6, {
      single()
      fragment("NH", vertical: true)
      single()
      double()
      fragment("N", links: (
        "N-horizon": single()
      ))
    })
    single()
    hook("N-horizon")
    single()
    single()
    fragment("NH")
    single(from: 1)
  })
})
```



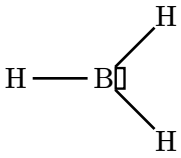
II.10.8 Sulfuric Acid

```
#skeletize({
  fragment("H")
  single()
  fragment("O", lewis: (
    lewis-line(angle: 90deg),
    lewis-line(angle: -90deg)
  ))
  single()
  fragment("S")
  let do(sign) = {
    double()
    fragment("O", lewis: (
      lewis-line(angle: sign * 45deg),
      lewis-line(angle: sign * 135deg)
    ))
  }
  branch(angle: 2, do(1))
  branch(angle: -2, do(-1))
  single()
  fragment("O", lewis: (
    lewis-line(angle: 90deg),
    lewis-line(angle: -90deg)
  ))
  single()
  fragment("H")
})
```



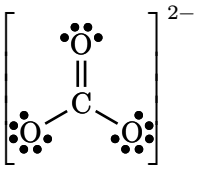
II.10.9 BH_3

```
#skeletize({
  fragment("H")
  single()
  fragment("B", lewis: (
     lewis-rectangle(fragment-margin:
5pt),
  ))
  branch(angle:1, {
    single()
    fragment("H")
  })
  branch(angle:-1, {
    single()
    fragment("H")
  })
})
```



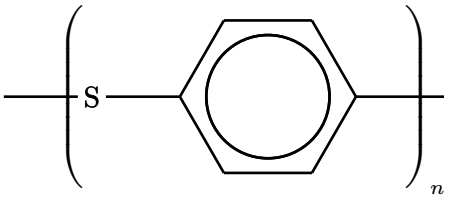
II.10.10 Carbonate ion

```
#skeletize(
  config: (
    atom-sep: 2em,
  ),
  {
    parenthesis(
      l: "[",
      r: "]",
      tr: $2-$,
      xoffset: .05,
      {
        fragment(
          "O",
          lewis: (
            lewis-double(angle: 135),
            lewis-double(angle: -45),
           lewis-double(angle: -135),
          ),
        )

        single(relative: 30deg)
        fragment("C")
        branch(
          angle: 2,
          {
            double()
            fragment(
              "O",
              lewis: (
               lewis-double(angle: 45),
              lewis-double(angle: 135),
              ),
            )
          },
        )
        single(absolute: -30deg)
        fragment(
          "O",
          lewis: (
            lewis-double(angle: 45),
            lewis-double(angle: -45),
           lewis-double(angle: -135),
          ),
        )
      },
    )
  },
)
```



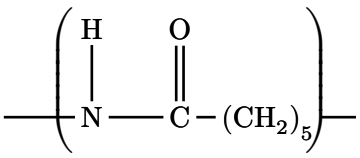
II.10.11 Polphenyl sulfide

```
#skeletize({
  single()
  parenthesis(
    br: $n$,
    right: "end",
    {
      fragment("S")
      single()
      cycle(
        6,
        align: true,
        arc: (:),
        {
          for i in range(3) {
            single()
          }
          branch(single(name: "end"))
          for i in range(3) {
            single()
          }
        },
      )
    },
  )
})
```



II.10.12 Nylon 6

```
#skeletize({
  parenthesis(xoffset: (.4, -.15), {
    single()
    fragment("N")
    branch(angle: 2, {
      single()
      fragment("H")
    })
    single()
    fragment("C")
    branch(angle: 2, {
      double()
      fragment("O")
    })
    single()
    fragment($(C H_2)_5$)
    single()
  })
})
```



Part III

Index