fca.sty
LaTeX–macros for Formal Concept Analysis
v3.0

Bernhard Ganter*
Tobias Schlemmer†

2022/09/17

Abstract

Formal Concept Analysis is a field of mathematics based on the theory of ordered sets and complete lattices, with applications to data analysis and knowledge processing. To simplify typesetting of FCA-related text, `fca.sty` provides two environments and some simple text macros. The two environments are

cxt for typesetting small formal contexts as cross-tables, and

diagram for making line diagrams of concept lattices. This environment may be of some interest for other purposes as well, since it can also be used for ordered sets and graphs.

A list of **text macros** is given in Section 3 below.

A recent version of `fca.sty` should be available from

https://github.com/keinstein/latex-fca

Contents

*TU Dresden, Ernst-Schröder-Zentrum für Begriffliche Wissensverarbeitung
†TU Dresden

1 Loading the package

The package `fca` is loaded by adding

```
\usepackage{fca}
```

or

```
\usepackage{tikz}
\usetikzlibrary{fca}
```

to the preamble. The first variant provides all features of this package using PGF graphixs, while the latter adds improved support for Ti*k*Z. Details about the differences are documented in subsection 5.2.

The following package options are supported:

compat, to be used for files which were written for older versions up to Version 2.1 of `fca.sty`. See subsection 7.1 for details.

nocmpat, reverts the effect of `compat`

draft, enables draft mode. See section 6

final, reverts the effect of `draft`

Older versions needed the `newdrawline.sty` package. This is no longer neccessary.

2 Typesetting formal contexts with `cxt`

Formal contexts can be typeset using the `cxt` environment. What this (very simple) environment does can be guessed from an example.

Example 1: A formal context.

Code:

```
\begin{cxt}
  \cxtName{Formula 1}
%
  \att{1.}
  \att{2.}
  \atr{disqualified}
%
  \obj{x..}{Verstappen}
  \obj{.x.}{Hamilton}
  \obj{.xx}{Leclerc}
\end{cxt}
```

Result:

Formula 1	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

A detailed description follows.

2.1 Basic usage of the `cxt` environment

`Env cxt` Environment `cxt` typically consists of the following parts:

`\begin{cxt}` begins typesetting a formal context table.

The commands within a `cxt` environment are

`\cxtName` `{`⟨*text*⟩`}` Define the text for the upper left cell of the table. Optional. The default is no text.

`\att` `{`⟨*text*⟩`}` Give an attribute name. These names are processed in the order in which they are given. Attribute names given after an `\obj` command are ignored.

`\atr` `{`⟨*text*⟩`}` Same as `\att`, but with rotated text.

`\obj` `{`⟨*text*⟩`}{`⟨*text*⟩`}` Give an object's name and its incidence vector, consisting of dots and 'x'es. The incidences come first, for better alignment. The length of each incidence vector must be the number of attributes.

Each instance of `\obj` is directly translated to a row of the `tabular`-environment. It is therefore possible to mix `\obj` commands with usual `tabular`-commands.

`\end{cxt}` Completes typesetting the context table.

`cxt` can handle an arbitrary number of attributes.

2.1.1 Other predefined entries

An incidence vector for an object consists of dots and crosses ("`x`" or "`X`"). The arrow relations may also be noted. Instead of `x` and `.`, type `d` (for "down"), `u` ("up"), or `b` ("both"). In the following example, we also change the colour of these symbols:

Example 2: Some predefined context characters, colour changed.

Code:

```
\begin{cxt}%
\renewcommand{\fcaCxtArrowStyle}{\footnotesize\color{red}}%
\cxtName{Formula 1}
\att{1.}
\att{2.}
\atr{disqualified}
\obj{xbd}{Verstappen}
\obj{uxb}{Hamilton}
\obj{bxx}{Leclerc}
\end{cxt}
```

Result:

Formula 1	1.	2.	disqualified
Verstappen	×	↗↙	↙
Hamilton	↗	×	↗↙
Leclerc	↗↙	×	×

The default for `\fcaCxtArrowStyle` is `\footnotesize`. In the above example we have changed it using `\renewcommand` in order to make the arrows red. The default colour is black.

The digits 0 to 9 may be used as well.

2.2 Advanced usage of the `cxt` environment

2.2.1 Defining context characters

The following letters and numbers are predefined by `fca.sty`.

[context character] . **`.`:** An empty context cell.

[context character] x **`x`:** A cross in the context.

[context character] X **`X`:** Alternative sign for a cross.

[context character] u **`u`:** An up-arrow in the context.

[context character] d **`d`:** A down-arrow in the context.

[context character] b **`b`:** A cell containing both an up- and a down-arrow.

[context character] 0 **`0`:** A zero in a many-valued context.

[context character] 1 **`1`:** A one in a many-valued context.

[context character] 2 **`2`:** Two in a many-valued context.

[context character] 3 **`3`:** Three in a many-valued context.

[context character] 4 **`4`:** Four in a many-valued context.

[context character] 5 **`5`:** Five in a many-valued context.

[context character] 6 **`6`:** Six in a many-valued context.

[context character] 7 **`7`:** Seven in a many-valued context.

[context character] 8 **8:** Eight in a many-valued context.

[context character] 9 **9:** Nine in a many-valued context.

You can define your own markers using `\fcaNewContextChar`. It works like `\newcommand` but defines a single character. For single signs use `\cxtrlap` in order to give it an appropriate size. We give two examples. The first introduces a single new context character for so-called "tight" incidences. These are indicated by a boldface cross (for which the `bm` package was used). The second example is discussed below.

Example 3: (Re-)defining context characters.

Code:

```
\begin{cxt}
\fcaNewContextChar{t}{\cxtrlap{$\bm{\times}$}}
\cxtName{cxt 1}
\att{1.}
\att{2.}
\atr{disqualified}
\obj{t..}{Verstappen}
\obj{.t.}{Hamilton}
\obj{.xt}{Leclerc}
\end{cxt}
\qquad
\begin{cxt}
\fcaNewContextChar{v}{\cxtrlap{$\vee$}}
\fcaProvideContextChar{\wedge}{\cxtrlap{$\wedge$}}
\fcaProvideContextChar{d}{ -- ignored -- }
\fcaRenewContextChar{d}{\cxtrlap{$i$}}
\cxtName{cxt 2}
\att{1.}
\att{2.}
\atr{disqualified}
\obj{x.v}{Verstappen}
\obj{\wedge xb}{Hamilton}
\obj{dxx}{Leclerc}
\end{cxt}
```

Result:

cxt 1	1.	2.	disqualified
Verstappen	$\times$		
Hamilton		$\boldsymbol{\times}$	
Leclerc		$\times$	$\times$

cxt 2	1.	2.	disqualified
Verstappen	$\times$		$\vee$
Hamilton	$\wedge$	$\times$	$\nearrow\!\!\!\!\swarrow$
Leclerc	i	$\times$	$\times$

The second example (with context name "cxt 2") shows different ways to define context characters. First `\fcaNewContextChar` is used so that the symbol `$\vee$` is inserted whenever the letter "v" occurs in the the incidence vector of an object. The next line associates the "token" `\wedge` to the symbol `$\wedge$`, using `\fcaProvideContextChar`, which does the same as `\fcaNewContextChar`, except that no error message is given when the context character was already defined. Instead, the command will then be ignored. If you want to force the redefinition of a context character, you should use `\fcaRenewContextChar`.

There is one more possibility:

`\freeobj` `{`⟨*text*⟩`}` You may define a marker with a single argument that typesets its argument. There also is a macro `\freeobj` that takes a tabular row as argument instead of the usual markers. The following example illustrates these:

Code:

```
\begin{cxt}
\fcaNewContextChar{w}[1]{#1}
\cxtName{Formula 1}
\att{1.}
\att{2.}
\atr{disqualified}
\obj{5bw1}{Verstappen}
\obj{w2xw{77}}{Hamilton}
\freeobj{1&2&3}{Leclerc}
\end{cxt}
```

Result:

Formula 1	1.	2.	disqualified
Verstappen	5	↙↗	1
Hamilton	2	×	77
Leclerc	1	2	3

2.2.2 `cxt` alignment

`cxt` takes an optional alignment parameter, which can be one of `t, c` or `b`. It is passed to the tabular environment (see there for further documentation).

Example 4: Alignment parameter for a `cxt`-environment

Code:

```
\makebox{
\begin{cxt}[t]
  \cxtName{Formula t}
  %
  \att{1.}
  \att{2.}
  \atr{disqualified}
  %
  \obj{x..}{Verstappen}
  \obj{.x.}{Hamilton}
  \obj{.xx}{Leclerc}
\end{cxt}\quad
```

```
\begin{cxt}[c]
  \cxtName{Formula c}
  %
  \att{1.}
  \att{2.}
  \atr{disqualified}
  %
  \obj{x..}{Verstappen}
  \obj{.x.}{Hamilton}
  \obj{.xx}{Leclerc}
\end{cxt}\quad
\begin{cxt}[b]
  \cxtName{Formula b}
  %
  \att{1.}
  \att{2.}
  \atr{disqualified}
  %
  \obj{x..}{Verstappen}
  \obj{.x.}{Hamilton}
  \obj{.xx}{Leclerc}
\end{cxt}
}
```

Result:

Formula t	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

Formula c	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

Formula b	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

2.2.3 Including Burmeister context files

`\cxtinput` `{`⟨*filename.cxt*⟩`}` The package `fca` allows to use context files in Burmeister format (which usually have the `.cxt` file name extension) to be included directly in a LaTeX document. Its usage is as simple as possible.

Example 5: Including a Burmeister context file

Code:

```
\begin{cxt}
\cxtinput{formula1.cxt}
\end{cxt}
```

Result:

Formula 1	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

The name of the context can be overwritten by using `\cxtName` inside the `cxt` environment.

Example 6: Overwriting the context name

Code:

```
\begin{cxt}
\cxtName{Formula 2}
\cxtinput{formula1.cxt}
\end{cxt}
```

Result:

Formula 2	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

To get non-rotated attribute names, redefine the `atr` command as in the following example:

Example 7: Including contexts with unrotated attributes

Code:

```
\makebox{{%
\begin{cxt}[b]
\renewcommand{\atr}{\att}
\cxtinput{formula1.cxt}
\end{cxt}\quad
\begin{cxt}[b]
\cxtName{Formula 2}
\cxtinput{formula1.cxt}
\end{cxt}%
}}
```

Result:

Formula 1	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

Formula 2	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

As an undocumented feature, `\cxtinput` uses the same mechanism to parse context lines as `\obj` does. So specital entries (special symbols or special formatting) can be requested with the help of a simple text editor by changing the corresponding characters in the context matrix of the `.cxt` file.

3 Some macros for text

Abbreviating macros are defined for some strings that occur regularly in FCA texts. See Figure 1 for quick overview.

`\GMI` The formal context (G, M, I).

`\context` The symbol $\mathbb{K}$, a frequently used name for a formal context.

`\context[S]` Other letters, such as $\mathbb{S}$, may also be used.

`\CL` The symbol $\underline{\mathfrak{B}}$ for the concept lattice operator. If $\mathbb{K}$ is a formal context, then $\underline{\mathfrak{B}}(\mathbb{K})$ denotes its concept lattice.

`\BV` same as `\CL`.

`\CLGMI` The concept lattice $\underline{\mathfrak{B}}(G, M, I)$ of the formal context (G, M, I).

`\BVGMI` Same as `\CLGMI`.

`\CGMI` The set $\mathfrak{B}(G, M, I)$ of all formal concepts of the formal context (G, M, I).

`\BGMI` Same as `\CGMI`.

Result	command	German variant
(G, M, I)	`\GMI`	
$\mathbb{K}$	`\context`	
$\mathbb{L}$	`\context[L]`	
$\underline{\mathfrak{B}}$	`\CL`	`\BV`
$\underline{\mathfrak{B}}(G, M, I)$	`\CLGMI`	`\BVGMI`
$\mathfrak{B}(G, M, I)$	`\CGMI`	`\BGMI`
$\mathrm{ext}()$	`\extent{}`	
$\mathrm{int}()$	`\intent{}`	
$\mathrm{Ext}()$	`\extents{}`	
$\mathrm{Int}()$	`\intents{}`	
$(H, N, I \cap H \times N)$	`\HNI`	
I	`\relI`	
$\not\sqcup I$	`\notI`	
$\bigtimes$	`\bigtimes`	
$⧖$	`\Semi`	
$\swarrow$	`\DownArrow`	`\Runterpfeil`
$\nearrow$	`\UpArrow`	`\Hochpfeil`
$\nearrow\!\!\!\swarrow$	`\DoubleArrow`	`\Doppelpfeil`
$\swarrow\!\!\swarrow$	`\DDArrow`	`\DDPfeil`
$\not\swarrow\!\!\swarrow$	`\NDDArrow`	`\NDDPfeil`
Formal Concept Analysis	`\FCA`	
Formale Begriffsanalyse		`\FBA`
Formalen Begriffsanalyse		`\FnBA`

Figure 1: Table of `fca.sty` text macros.

Symbol	command	package required
$\vee$	`\vee`	
$\wedge$	`\wedge`	
$\bigvee$	`\bigvee`	
$\bigwedge$	`\bigwedge`	
$\sqcup$	`\sqcup`	
$\sqcap$	`\sqcap`	
$\bigsqcup$	`\bigsqcup`	
$\bigsqcap$	`\bigsqcap`	stmaryrd

Figure 2: Other symbols that are used in Formal Concept Analysis, and the commands that generate them.

`\extent` The extent $\mathrm{ext}(\mathfrak{c})$ of the formal concept $\mathfrak{c} := (A, B)$ is A.

`\intent` The intent $\mathrm{int}(\mathfrak{c})$ of the formal concept $\mathfrak{c} := (A, B)$ is B.

`\extents` The set $\mathrm{Ext}(\mathbb{K})$ of extents of the formal context $\mathbb{K}$.

`\intents` The set $\mathrm{Int}(\mathbb{K})$ of intents of the formal context $\mathbb{K}$.

`\HNI` The subcontext $(H, N, I \cap H \times N)$.

`\relI` The incidence relation I.

`\notI` The negation $\not{I}$ of the incidence relation.

`\bigtimes` The product symbol $\bigtimes$.

`\DownArrow` The $\swarrow$ of the arrow relations.

`\Runterpfeil` Same as `\DownArrow`.

`\UpArrow` The $\nearrow$ of the arrow relations.

`\Hochpfeil` Same as `\UpArrow`.

`\DoubleArrow` The $\swarrow\!\!\!\nearrow$ of the arrow relations.

`\Doppelpfeil` Same as `\DoubleArrow`.

`\DDArrow` Gives $\swarrow\!\!\!\swarrow$, the symbol for the transitive closure of the arrow relations.

`\DDPfeil` Same as `\DDArrow`.

`\NDDArrow` Gives $\not\swarrow\!\!\!\swarrow$ the symbol for the negation of $\swarrow\!\!\!\swarrow$.

`\NDDPfeil` Same as `\NDDArrow`.

`\Semi` Gives $\between$, the symbol for the semi-product.

`\FCA` Prints "Formal Concept Analysis". In most cases, this command does not eat the space following it (thanks to `\xspace`).

`\FBA, \FnBA` Print "Formale(n) Begriffsanalyse". These commands also use `\xspace` so that blanks are preserved.

Some symbols that are provided by LaTeX are listed in Figure 2.

Here is a sample text:

```
\FCA provides an elegant way to determine the congruence relations of
a complete lattice. The congruence lattice of a doubly founded
concept lattice $\CLGMI$ is isomorphic to $\CL(G,M,\NDDArrow)$.
```

This translates to:

> Formal Concept Analysis provides an elegant way to determine the congruence relations of a complete lattice. The congruence lattice of a doubly founded concept lattice $\underline{\mathfrak{B}}(G, M, I)$ is isomorphic to $\underline{\mathfrak{B}}(G, M, \not\swarrow\!\!\!\swarrow)$.

4 Drawing lattices with `tikzdiagram` and `diagram`

The `tikzdiagram` and `diagram` environment helps typesetting diagrams of concept lattices, but can be used for ordered sets and graphs as well. As of version 3.0, the `diagram` environment uses the *Portable Graphics Format* PGF and, if loaded, its user-friendly syntax layer called Ti*k*Z.

Again we start with a small example of each type:

Example 8: A lattice diagram drawn using Ti*k*Z

Code:

```
\begin{tikzdiagram}
  \Node(1)(2,1)
  \Node(2)(3.5,2)
  \Node(3)(.5,3)
  \Node(4)(3.5,4)
  \Node(5)(2,5)
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Verstappen}
  \rightObjbox(2){Leclerc}
  \rightObjbox(4){Hamilton}
\end{tikzdiagram}
```

Result:

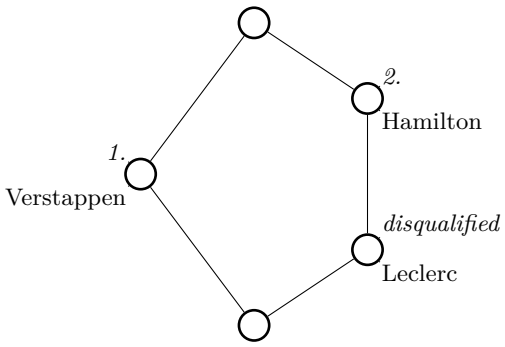
Example 9: A lattice diagram drawn in the classical way with the unit length of 1mm.

Code:

```
\begingroup
\unitlength=1mm.
\begin{diagram}
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge(1)(3)
```

```
    \Edge(2)(4)
    \Edge(3)(5)
    \Edge(4)(5)
    \leftAttbox(3){1.}
    \rightAttbox(2){disqualified}
    \rightAttbox(4){2.}
    \leftObjbox(3){Verstappen}
    \rightObjbox(2){Leclerc}
    \rightObjbox(4){Hamilton}
  \end{diagram}
  \endgroup
```

Result:



.

4.1 Basic usage of the `tikzdiagram` and the `diagram` environment

Env `tikzdiagram`

Env `diagram`

`\begin{tikzdiagram}` begins a Ti*k*Z-picture. This environment is only available if `fca.sty` is loaded via `\usetikzlibrary{fca}`.

`\begin{diagram}` begins a PGF-picture. Unlike the previous versions, width and height of the diagram no longer need to be specified.

`\begin{diagram}{`⟨*width*⟩`}{`⟨*height*⟩`}` The old syntax is still supported and determines the bounding box, which is otherwise calculated automatically.

A major advantage of using PGF and Ti*k*Z-pictures is that one can use the many possibilities offered by PGF or Ti*k*Z, even without knowing much about PGF and Ti*k*Z. The `diagram` environment and its macros allow optional parameters, e.g. for colour, line thickness, etc. These will be described in Section 4.3.2.

When `fca.sty` is loaded via `\usetikzlibrary{fca}`, the `diagram` environment can be placed inside a Ti*k*Z picture. In that case all diagram commands are internally translated into Ti*k*Z lcommands. More details are given in Section 5.2. The `tikzdiagram` environment is a shortcut to place a `diagram` inside a `tikzpicture` environment.

⚠ Unit lengths differ between `diagram` and `tikzdiagram`, more details are given in Section 5.2.

The commands within a `diagram` environment (in their basic form) are

`\Node` `(`⟨*nodename*⟩`)(`⟨*xpos,ypos*⟩`)` Puts a circle at position `(`⟨*xpos,ypos*⟩`)` of the picture. The default diameter of the circles is 4mm. It can be changed (for all circles) with `\fcaCircleSize{}` or using the option `radius`. The coordinates must be numbers (in which the deafault unit is used) or lengths. The node names must be different to each other, but in contrast to versions before 3.0, they do not have to be numbers, although this is recommended for clarity.

The old syntax `\Node{`⟨*nodename*⟩`}{`⟨*xpos*⟩`}{`⟨*ypos*⟩`}` is also supported.

`\Edge` `{`⟨*nodename1*⟩`}{`⟨*nodename2*⟩`}` Puts a line between the two nodes with the given numbers. These must have been declared earlier with a `\Node`–command.

`\leftAttbox` `\leftAttbox(`⟨*nodename*⟩`){`⟨*text*⟩`}`
`\centerAttbox` `\centerAttbox(`⟨*nodename*⟩`){`⟨*text*⟩`}`
`\rightAttbox` `\rightAttbox(`⟨*nodename*⟩`){`⟨*text*⟩`}`
`\leftObjbox` `\leftObjbox(`⟨*nodename*⟩`){`⟨*text*⟩`}`
`\centerObjbox` `\centerObjbox(`⟨*nodename*⟩`){`⟨*text*⟩`}`
`\rightObjbox` `\rightObjbox(`⟨*nodename*⟩`){`⟨*text*⟩`}`

These are used to put text to diagram nodes. The `Attbox`–commands place the text above the corresponding node, the `Objbox` below. Similarly, the text can be placed to the left, be centered, or be placed to the right of the labelled node.

For a better positioning of the label text an optional shift can be specified. For example,

```
\rightObjbox(4)(3,5){Hamilton}
```

moves the object label "Hamilton" at node named "4" by 3 `\unitlength` in x-direction and by 5 `\unitlength` in y-direction.

The old syntax `\rightObjbox{nodename}{xoffset}{yoffset}{labeltext}` is also supported. However, there is a small difference: while in the old syntax the shift `{xoffset}{yoffset}` is understood relative to the node, the shift `(xoffset,yoffset)` is interpreted relative to the diagram. This means that, e.g., ⚠ `\leftAttbox{4}{3}{5}{text}` corresponds to `\leftAttbox(4)(-3,5){text}`.

The `...box` macros have a variable number of arguments. So they cannot work properly if a brace follows them. In that case you should add `\relax` before the grouping brace or `\bgroup`.

`\end{diagram}` This concludes the diagram.

Helpful commands when fine tuning a diagram are discussed in section 6.

4.2 Changing the unit size of the generated diagram

By default, `diagram` und `tikzpicture` use the same units that the underlying graphics environment used when they were first defined. These units can always be overridden by providing a proper unit such as `pt` `in` `mm` or `cm`. Any unit can be used that is known to TEX.

The `tikzdiagram` (or `diagram` inside of `tikzpicture`) uses the default unit length of the `tikzpicture` environment. This is usually 1 cm.

Example 10: A lattice diagram drawn using Ti*k*Z and its default unit length

Code:

```
\begin{tikzdiagram}
  \Node(1)(2,1)
  \Node(2)(3.5,2)
  \Node(3)(.5,3)
  \Node(4)(3.5,4)
  \Node(5)(2,5)
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Verstappen}
  \rightObjbox(2){Leclerc}
  \rightObjbox(4){Hamilton}
\end{tikzdiagram}
```

Result:

The unit length can be easily changed using the `x` and `y` options to `tikzdiagram`

Example 11: A lattice diagram drawn using Ti*k*Z and setting all units to 1 mm

Code:

```
\begin{tikzdiagram}[x=1mm,y=1mm]
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
```

```
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Verstappen}
  \rightObjbox(2){Leclerc}
  \rightObjbox(4){Hamilton}
\end{tikzdiagram}
```

Result:

Len `\unitlength`

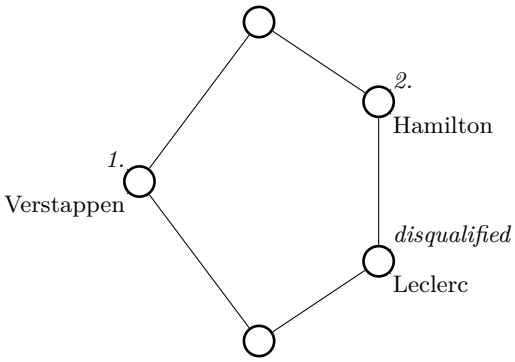
The `diagram` environment uses `\unitlengh` as its default unit length. It's original value is 1 pt so it is often changed at the beginning of the diagram. That is the reason why most examples of `diagram` environments also change `\unitlength`.

Example 12: A lattice diagram drawn in the classical way with the default unit length of 1.0pt.

Code:

```
\begin{diagram}
  \Node(1)(60,30)
  \Node(2)(105,60)
  \Node(3)(15,90)
  \Node(4)(105,120)
  \Node(5)(60,150)
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Verstappen}
  \rightObjbox(2){Leclerc}
  \rightObjbox(4){Hamilton}
\end{diagram}
```

Result:



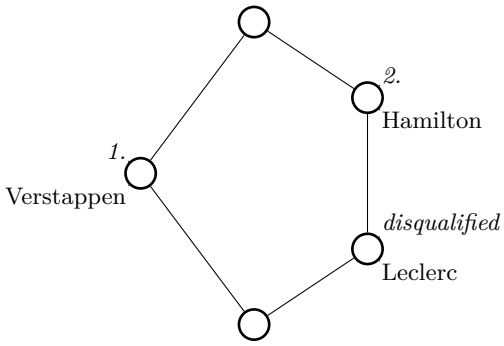
Changing the unit length can make the units more handier.

Example 13: A lattice diagram drawn in the classical way with the unit length of 1mm.

Code:

```
\begingroup
\unitlength=1mm.
\begin{diagram}
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Verstappen}
  \rightObjbox(2){Leclerc}
  \rightObjbox(4){Hamilton}
\end{diagram}
\endgroup
```

Result:



.

4.3 Changes to the style of the graphics

The connection with PGF and Ti*k*Z results in many design tools, some of which will be presented in Section 5. Before that we show how to change colour and other graphics parameters in simple diagrams.

4.3.1 Optional parameters

All macros of the `diagram` environment, except for `\end{diagram}`, allow optional parameters. These can be specified within square brackets. For example,

```
\Node[draw=red, fill=blue](4)(20,10)
```

specifies that a red circle, filled with blue, is to be drawn at position (20,10), representing the node with nodenumber 4.

The impatient reader may infer some of the possibilities from Example 10. For details see Subsection 4.3.2.

4.3.2 Fine tuning concept nodes

In Version 2.1 and before some macros allow to change certain parameters of the diagrams.

For compatibility reasons these still do work in diagrams that use bare PGF (i.e., that are not inside the argument to the `\tikz` macro or one of the environments `tikzpicture` or `tikzdiagram`).

`\fcaCircleSize`

The diameter of the circles that represent the concept nodes can be changed using `\fcaCircleSize{`⟨*size*⟩`}` The parameter ⟨*size*⟩ can be a dimension or a plain number. If the unit is omitted the current default unit of the diagram (`\unitsize` or Ti*k*Z coordinates) is used. The default is 4 mm. The old behaviour can be restored by adding `\fcaCircleSize{4}` after loading the package `fca`.

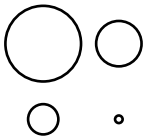
The circle size can be changed for single nodes by specifying the radius as an optional parameter (e.g., `[radius=3mm]`).

Example 14: Usage of `\fcaCircleSize`

Code:

```
\begingroup\fcaCircleSize{1}
\begin{diagram}
  \unitlength=1mm
  \Node(3)(20,10)
  \fcaCircleSize{10}
  \Node(1)(10,20)
  \CircleSize{4}
  \Node(2)(10,10)
  \Node[radius=3mm](4)(20,20)
\end{diagram}
\begin{tikzdiagram}
  \Node(3)(2,1)
  \CircleSize{.2}
  \Node(2)(1,1)
  \fcaCircleSize{.15}
  \Node(1)(1,2)
  \Node[radius=3mm](4)(2,2)
\end{tikzdiagram}
\endgroup
```

Result:



`\fcaNodeColor`

The color with which the concept nodes are filled can be changed with `\fcaNodeColor{`⟨*color*⟩`}`. The default is *white*. It can be changed for single nodes by using `fill=`*color* as an option.

`\NodeColor`

`\NodeColor{`⟨*color*⟩`}` is an alias to `\fcaNodeColor` inside the `diagram` environment.

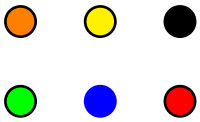
Example 15: Usage of `\NodeColor`, `fill`, `color`, `node/color`

Code:

```
\begingroup
\fcaNodeColor{yellow}
\begin{diagram}
  \Node(3)(60,60)
  \NodeColor{green}
  \Node(2)(30,30)
  \Node[fill=orange](1)(30,60)
  \Node[color=blue](4)(60,30)
  \Node[node/color=red](5)(90,30)
```

```
  \fcaNodeColor{black}
  \Node(6)(90,60)
\end{diagram}
\begin{tikzdiagram}
  \Node(3)(2,2)
  \NodeColor{green}
  \Node(2)(1,1)
  \Node[fill=orange](1)(1,2)
  \Node[color=blue](4)(2,1)
  \Node[node/color=red](5)(3,1)
  \fcaNodeColor{black}
  \Node(6)(3,2)
\end{tikzdiagram}
\endgroup
```

Result:



`\fcaNodeThickness`

The width of the annulus representing the circumferential line of a concept node can be changed using `\fcaNodeThickness{`⟨*thickness*⟩`}`. The default is 1.2 pt. Can be changed for single nodes by specifying the line width as an optional parameter (e.g., `[line width=3pt]`).

`\NodeThickness`

`\NodeThickness{`⟨*thickness*⟩`}` is an alias to `\fcaNodeThickness` inside a `diagram` environment.

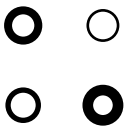
Example 16: Usage of `\NodeThickness` and `line width`

Code:

```
\begingroup
\fcaNodeThickness{1pt}
\begin{diagram}
  \Node(3)(60,60)
  \NodeThickness{2pt}
  \Node(2)(30,30)
  \Node[line width=3pt](1)(30,60)
  \fcaNodeThickness{4pt}
  \Node(4)(60,30)
\end{diagram}
\begin{tikzdiagram}
  \Node(3)(2,2)
  \NodeThickness{2pt}
  \Node(2)(1,1)
  \Node[line width=3pt](1)(1,2)
  \fcaNodeThickness{4pt}
  \Node(4)(2,1)
```

```
\end{tikzdiagram}
\endgroup
```

Result:



4.4 Modifying edges

`\fcaEdgeThickness`

`\fcaEdgeThickness{`⟨*thickness*⟩`}`. The default is .8pt. It can be changed for all or for single edges by specifying `line width` as an optional parameter.

`\EdgeThickness`

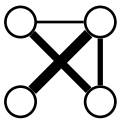
`\EdgeThickness{`⟨*thickness*⟩`}` is an alias to `\fcaEdgeThickness` inside a `diagram` environment.

Example 17: Usage of `\EdgeThickness` and `line width`

Code:

```
\begingroup
\fcaEdgeThickness{1pt}
\begin{diagram}
  \Node(1)(60,60)
  \Node(2)(30,60)
  \Node(3)(60,30)
  \Node(4)(30,30)
  \Edge(1)(2)
  \EdgeThickness{2pt}
  \Edge(1)(3)
  \fcaEdgeThickness{3pt}
  \Edge(2)(3)
  \Edge[line width=4pt](1)(4)
\end{diagram}
\begin{tikzdiagram}
  \Node(1)(2,2)
  \Node(2)(1,2)
  \Node(3)(2,1)
  \Node(4)(1,1)
  \Edge(1)(2)
  \EdgeThickness{2pt}
  \Edge(1)(3)
  \fcaEdgeThickness{3pt}
  \Edge(2)(3)
  \Edge[line width=4pt](1)(4)
\end{tikzdiagram}
\endgroup
```

Result:



4.5 Labels

Some of the nodes in a concept lattice diagram have text labels. Typically, object and attibute concepts are labeled with the associated object or attibute names. This is done using the `\**box(){}`-macros, which were introduced in Subsection 4.1. Font size and font colour of such labels can be changed, and the label boxes as well. Usually the label text is put into a one-line box (i.e. hbox), the width of which is automatically detected. To allow for line breaks, the parameter `text width` can be used. It affects the type and size of the label boxes.

When `text width` is set to a length, then the text of the label is put in a `\parbox`. It can be broken into several lines using `\\`. The width of the `\parbox` is the value of the `text width` parameter. It is unset by default.

Setting `text width` to a length can either be done as an optional parameter to the diagram, such as

```
\begin{diagram}[text width=6mm]
```

or globally (for all diagrams) by

```
\fcaset{text width=6mm}
```

or for single labels only, again as an optional parameter.

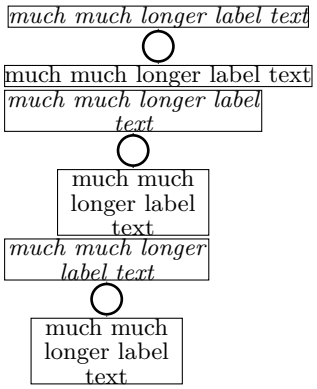
Example 18: Concept node labels

Code:

```
\begin{diagram}
  \Node(0)(0,0)
  \centerAttbox[draw=black](0){much much longer label text}
  \centerObjbox[draw=black](0){much much longer label text}
\end{diagram}
\begin{tikzdiagram}
  \Node(0)(0,0)
  \centerAttbox[draw=black](0){much much longer label text}
  \centerObjbox[draw=black](0){much much longer label text}
\end{tikzdiagram}\\
\begin{diagram}[text width=3.4cm]
  \Node(0)(0,0)
  \centerAttbox[draw](0){much much longer label text}
  \centerObjbox[draw,text width=2cm](0){much much longer label text}
```

```
\end{diagram}
\begin{tikzdiagram}[text width=3.4cm]
  \Node(0)(0,0)
  \centerAttbox[draw](0){much much longer label text}
  \centerObjbox[draw,text width=2cm](0){much much longer label text}
\end{tikzdiagram}\\
\begingroup
\fcaset{text width=2.7cm}
\begin{diagram}
  \Node(0)(0,0)
  \centerAttbox[draw](0){much much longer label text}
  \centerObjbox[draw,text width=2cm](0){much much longer label text}
\end{diagram}
\begin{tikzdiagram}
  \Node(0)(0,0)
  \centerAttbox[draw](0){much much longer label text}
  \centerObjbox[draw,text width=2cm](0){much much longer label text}
\end{tikzdiagram}
\endgroup
```

Result:



Opt `draw`

Opt `fill`

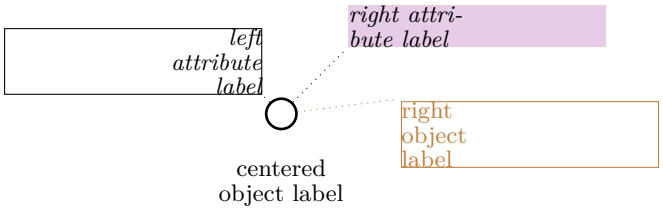
Label boxes may be framed with the `draw` option, which may also be used to set a color for the text and the frame. Their background color can be set with `fill`.

Example 19: Concept node labels

Code:

```
\begin{diagram}[text width=3.4cm]
  \Node(0)(20,10)
  \leftAttbox[draw](0)(-1,1){left\\ attribute\\ label}
  \rightAttbox[fill=red!50!blue!20](0)(10,10){right
  attri-\\ bute label}
  \rightObjbox[draw=brown](0)(20,5){right\\ object\\ label}
  {\fcaNoDots\centerObjbox(0)(0,-5){centered\\ object label}}
\end{diagram}
```

Result:



Opt `font`

Font settings of the object and attribute label can be given with the option `font`. For compatibility reasons with prior versions of `fca.sty` Its default value is either `\ObjecktLabelStyle` or `\AttributeLabelStyle`. They can be changed using `\renewcommand`. Similarly, The default value of `\ObjecktLabelStyle` is `\fcaObjectLabelStyle` and `\AttributeLabelStyle` defaults to `\fcaAttributeLabelStyle`. So either of them can be changed in order to modify the appearence of the labels.

`\ObjectLabelStyle`
`\AttributeLabelStyle`

`\fcaObjectLabelStyle`

The macro `\fcaObjectLabelStyle` used to define the font of object labels *outside* of `diagram` environments. Default: `\small\baselineskip1em\rmfamily\upshape`

`\fcaAttributeLabelStyle`

In the same way `\fcaAttributeLabelStyle` used to define the font of object labels *outside* of `diagram` environments. Default: `\small\baselineskip1em\rmfamily\itshape`.

The four macros `\ObjectLabelStyle`, `\AttributeLabelStyle`, `\fcaObjectLabelStyle` and `\fcaAttributeLabelStyle` are defined only inside of the `diagram` environment.

Care must be taken when different approaches are mixed. Setting the option `font` on an attribute label will remove the call to `\AttributeLabelStyle`. So also `\fcaAttributeLabelStyle` will not be called anymore. Object labels will behave similarely.

Therefore, it is recommended to modify only the `font` option and leave the macros for only for compatibility with older diagrams.

Example 20: Changing the concept label font

Code:

```
\begingroup
\begin{diagram}
  \renewcommand\fcaObjectLabelStyle{\huge}%
  \renewcommand\fcaAttributeLabelStyle{\tiny}%
  \Node(0)(0,0)
  \leftAttbox(0){tiny}
  \leftObjbox(0){huge}
  \renewcommand\ObjectLabelStyle{\tiny}%
```

```
    \renewcommand\AttributeLabelStyle{\huge}%
    \rightAttbox(0){tiny}
    \rightObjbox(0){huge}
    \centerAttbox[font=\bfseries](0){bf}
    \centerObjbox[font=\itshape](0){it}
  \end{diagram}
  \begin{tikzdiagram}
    \renewcommand\fcaObjectLabelStyle{\huge}%
    \renewcommand\fcaAttributeLabelStyle{\tiny}%
    \Node(0)(0,0)
    \leftAttbox(0){tiny}
    \leftObjbox(0){huge}
    \renewcommand\ObjectLabelStyle{\tiny}%
    \renewcommand\AttributeLabelStyle{\huge}%
    \rightAttbox(0){tiny}
    \rightObjbox(0){huge}
    \centerAttbox[font=\bfseries](0){bf}
    \centerObjbox[font=\itshape](0){it}
  \end{tikzdiagram}
  \endgroup
```

Result:

4.6 Configuring the help lines for labels

By default each label box is connected to the respective concept node by a dotted line. This can be switched off:

`\fcaNoDots` Causes no dotted line to be drawn from the concept node to the label boxes.

`\NoDots` is an alias to `\fcaNoDots` inside a `diagram` environment.

These macros can be focussed to single instances, using braces. For example,

```
\relax{\fcaNoDots\centerObjbox(node){labeltext}}
```

⚠ generates a single centered object label without a dotted line.

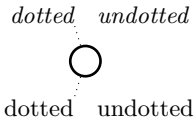
The `...box` macros have a variable number of arguments. So they cannot work properly if a brace follows them. In that case you should add `\relax` before the grouping brace or `\bgroup`.

Example 21: Usage of `\NoDots` and `\fcaNoDots`

Code:

```
\begingroup
\fcaEdgeThickness{1pt}
\begin{diagram}
  \Node(1)(30,60)
  \leftObjbox[shift={(0,-10)}](1){dotted}\relax
  {\NoDots\rightObjbox[shift={(0,-10)}](1){undotted}}
  \leftAttbox[shift={(0,10)}](1){dotted}\relax
  {\fcaNoDots\rightAttbox[shift={(0,10)}](1){undotted}}
\end{diagram}
\begin{tikzdiagram}
  \Node(1)(1,2)
  \leftObjbox[shift={(0,-1)}](1){dotted}\relax
  {\NoDots\rightObjbox[shift={(0,-1)}](1){undotted}}
  \leftAttbox[shift={(0,1)}](1){dotted}\relax
  {\fcaNoDots\rightAttbox[shift={(0,1)}](1){undotted}}
\end{tikzdiagram}
\endgroup
```

Result:



5 Advanced usage of the `diagram` environment

5.1 Make your own diagram style

With some knowledge about PGF and Ti*k*Z you can define your own style for concept lattice diagrams. For multiple use such definitions can be outsourced using the `\fcaset{}` macro.

We demonstrate this with an example where we define and then use a style called conexp style. It is based on Serhiy Yevtushenko's `Concept Explorer`, in which object and attribute concepts are color-coded, by a black lower semicircle and a blue upper semicircle, respectively. The version presented in this example does not behave well when the radius of concept nodes is changed.

Example 22: Defining a `diagram` style

Code:

```
\fcaset{conexp style/.style={%
    every attributes/.append style={
      label concept/.append style={
        shape=semicircle,
        fill=blue,
        anchor=south,
```

```
      outer sep=0pt,
      minimum height=2\unitlength,
      label/name suffix=attribute concept,
      label/at=center,
      solid
    }
  },
  every objects/.append style={
    label concept/.append style={
      shape=semicircle,
      fill=black,
      anchor=south,
      outer sep=0pt,
      minimum height=2\unitlength,
      label/name suffix=attribute concept,
      label/at=center,
      solid,
      rotate=180
    }
    }}}
```

Result:

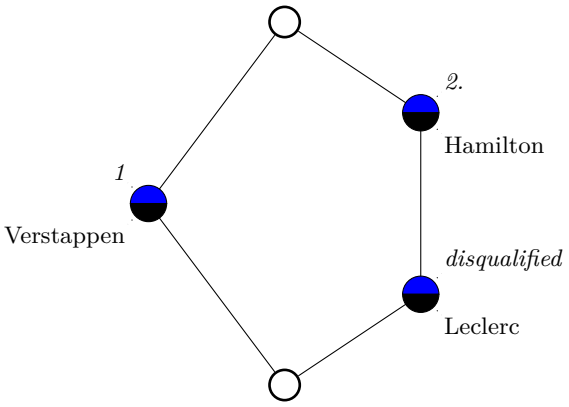
To apply the defined style, use its name as an optional argument of `\begin{diagram}`, or, if it shall be applied to single nodes only, as an optional argument for the respective node declarations.

Example 23: Using a `diagram` style

Code:

```
{\unitlength 1.2mm
\begin{diagram}[conexp style]
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3)(-1,1){1}
  \rightAttbox(2)(1,1){disqualified}
  \rightAttbox(4)(1,1){2.}
  \leftObjbox(3)(-1,-1){Verstappen}
  \rightObjbox(2)(1,-1){Leclerc}
  \rightObjbox(4)(1,-1){Hamilton}
\end{diagram}
}
```

Result:



5.2 `fca.sty`, PGF, and Ti*k*Z

`fca.sty` requires PGF, the *Portable Graphics Format*, but not necessarily its syntax layer Ti*k*Z. That means that PGF will be loaded automatically with `fca.sty`, but Ti*k*Z is not. When you do not need any of Ti*k*Z's special features, you may slightly increase the processing speed by loading `fca.sty` without Ti*k*Z. Since Ti*k*Z is built on PGF, you can get from PGF whatever you get from Ti*k*Z, though often with difficulty.

Most users will load both `fca.sty` and Ti*k*Z. In that case, and provided that you want to use Ti*k*Z-features, you should also put `\usetikzlibrary{fca}` in the preamble.

Even when both Ti*k*Z and `fca.sty` are loaded, there are still two possibilities. The `diagram` environment may be used inside or outside a `tikzpicture` (or a `\tikz` command). Table 1 gives an impression of the differences.

To make sure that you are using `diagram` inside a `tikzpicture`, you may write `\begin{tikzdiagram}` ... `\end{tikzdiagram}` instead of `\begin{diagram}` ... `\end{diagram}`.

You can put your `diagram` into a `pgfpicture` or a `tikzpicture`. It will get a new "scope" in order to prevent bleeding of options into later graphics operations. Putting more than one diagram into one `pgf-` or `tikzpicture` may require giving each of these diagrams its own namespace by setting the `namespace=`⟨*name*⟩ option for each `diagram` environment to a different ⟨*name*⟩.

5.3 Which environment to choose?

There are several options how to start and end a `diagram` environment. Table 1 gives a short overview of TEXnical differences between them.

[1] options for `\pgfusepath` are limited to those from `fca.sty`

	`diagram` environment with `\usepackage{fca}`	`diagram` environment with `\usetikzlibrary{fca}` outside of `\tikz` or `tikzpicture` environments	`diagram` environment with `\usetikzlibrary{fca}` inside `\tikz` or a `tikzpicture` environment, or `tikzdiagram`
Main command in `\Node`	`\pgfnode`	`\pgfnode`	`\node`
Main command(s) in `\Edge`	`\pgfmoveto`, `\pgflineto`, `\pgfusepath`	`\pgfmoveto`, `\pgflineto`, `\pgfusepath`	`\draw(...) edge (...);`
Available options	all `/fca/...`	all `/fca/...` and some `/tikz/...`[1]	all `/fca/...` and all `/tikz/...`
Macros from Section 4.3.2	working	working	not supported
Compatibility with version 2.1 and below	yes	yes	partial
Initial unit lengths	(`\unitlength`, `\unitlength`)	(`\unitlength`, `\unitlength`)	Current TikZ transformation matrix for coordinates and `\unitlength` for other sizes (line width, node radius, etc.)

Table 1: Comparison of the different diagram environments

You can put your `diagram` into a `pgfpicture` or a `tikzpicture`. In that case the diagram is drawn in a new scope in order to prevent bleeding of options into later graphics operations. This option also allows to put several diagrams into one picture. You can keep each of the diagrams in its own namespace by setting the `namespace=`⟨*name*⟩ option for each `diagram` environment to a different ⟨*name*⟩.

When you don't need any of Ti*k*Z's special features, you can slightly increase the processing speed when you load `fca.sty` directly instead through Ti*k*Z.

When Ti*k*Z is unsed elsewhere in your document (or if you need some of Ti*k*Z special features in your diagram), you have still the choice whether `diagram` should access PGF directly or via Ti*k*Z. Direct access to PGF is probably faster and provides more compatibility to prior versions of `fca.sty`. When Ti*k*Z is loaded, implementation uses some Ti*k*Z internals, so that some improvements of Ti*k*Z are also accessible in the when PGF is directly accessed.

When put into a Ti*k*Z graphics a `diagram` acesses PGF through the Ti*k*Z frontend layer. The environment `tikzdiagram` behaves in the same way. This provides even deeper integration with Ti*k*Z graphics. As a drawback the macros from Section 4.3.2 do not work reliably anymore. So you must use options and styles for fine tunging the diagram.

⚠ Due to the way Ti*k*Z is internally implemented, a `diagram` or a `pgfpicture` environment inside the argument to `\tikz` or inside a `tikzpicture` environment is always processd as Ti*k*Z graphics, independent from whether the graphics has been interrupted or not.

6 Draft mode

In order to assist authors in the creation of diagrams some debugging features have been added to the `fca.sty`. This functions are controlled by package optinos.

Opt `draft`

Opt `final`

enables the draft mode. disables the draft mode.

In draft mode the macros `\Numbers` and `\NoNumbers` can be used to show the PGF labels on top of the nodes in the diagram. In prior versions these labels had to be numbers, hence the name.

⚠ The draft mode must be disabled during final typesetting as it may have a bad impact on the readability of the typeset document.

Opt `/fca/node/numbers`

`\Numbers`

The option `/fca/node/numbers` or the cammand `\Numbers` put the node labels into the nodes. While working on a diagram it can be helpful to have a picture with numbered nodes.

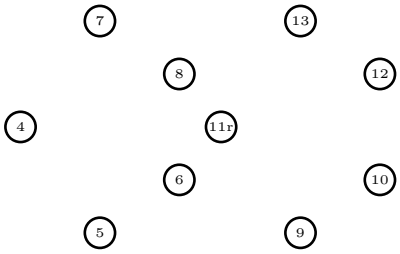
Example 24: Plotting labels inside of nodes

Code:

```
{\unitlength .7mm
\begin{diagram}
```

```
    \Numbers
    \Node(5)(20,10)
    \Node(6)(35,20)
    \Node(4)(5,30)
    \Node(8)(35,40)
    \Node(7)(20,50)
  \end{diagram}
  \begin{diagram}[node/numbers]
    \Node(9)(20,10)
    \Node(10)(35,20)
    \Node(11r)(5,30)
    \Node(12)(35,40)
    \Node(13)(20,50)
  \end{diagram}}
```

Result:



`\Numbers` should be used only to aid the development of diagrams. Thus it is active only in the draft mode. It is activated with the option `draft`.

We recommend to remove the `\Numbers`-command when the diagram is ready.

`\NoNumbers`

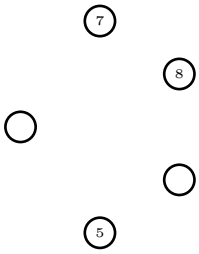
When the current or following nodes shall be excluded from showing their internal node label the option `/fca/node/numbers` can be set to `false`. This can be achieved also with the command `\NoNumbers`.

Example 25: `\Numbers` and `\NoNumbers`

Code:

```
  {\unitlength .7mm
  \begin{diagram}
    \Numbers
    \Node(5)(20,10)
    \Node[node/numbers=false](6)(35,20)
    \NoNumbers
    \Node(4)(5,30)
    \Node[node/numbers](8)(35,40)
    \Numbers
    \Node(7)(20,50)
  \end{diagram}}
```

Result:



7 About the current version of `fca.sty`

The original version of fca.sty had been written by Bernhard Ganter, mainly for his personal use. With the further development of TEX, especially the introduction of Ti*k*Z, this package became obsolete. Then, in 2022, Tobias Schlemmer, on behalf of the Ernst Schröder Center, set up the completely revised version that is now available.

7.1 Compatibility with earlier versions

In order to reduce incompatibilities with other packages, since version 2.2 of `fca.sty` nearly all macros of `fca.sty` belong to a so called namespace. That means most of them start with `\fca...`. Only within `cxt` and `diagram` environments this rule is relaxed.

Opt `compat`

There is a package option `compat` that also defines the old names which have been used before Version 2.2 of `fca.sty`.

⚠ The current implementation of `compat` should be only used to compile unmodified LATEX code written before Version 2.2. It maps the new macros to use the old ones. As soon as the new macros are redefined, the old ones are ignored. So make sure to replace all occurrences of the old macros by the corresponding new ones at once.

Opt `nocompat`

This option reverts the effect of `compat`.

7.2 Other incompatibilities

The default circle size in diagrams has been changed to 4 mm instead of `4\unitlength`. The old behaviour can be restored by adding `\fcaCircleSize{4}` just after loading the package `FC`.

7.3 Error messages

Package error messages are not yet implemented.

7.4 A caveat for future implementations

It is discouraged to change `\unitlength` inside the `diagram` environment. Currently `\unitlength` is used inside in the `diagram` environment. However, since PGF already comes with support for coordinate and canvas transformations, a future version might completely rely on them and abandon the usage of `\unitlength`. Then, `\unitlength` may be used only to initially set up the coordinate system and be ignored later.

For the same reason, it is recommended to use only Euclidean coordinates for diagrams. Other coordinate systems should work, too. However, in that case the output may change with future versions of this package. In particular the algorithm for the calculation of the node radius may change, and changes to `\unitlength` inside the environment will have no effect.

The newly added option to encapsulate a `diagram` environment inside a `tikzpicture` or `pgfpicture` already goes into that direction. In this usage the `\unitlength` register is set to the sum of the horizontal unit lengths of the x and the y coordinates as provided by the surrounding environment.

8 Some demonstrations

This example needs Ti*k*Z

Figure 3: Elements of a `diagram` environment.

```
\begin{tikzdiagram}[
    conexp style,
    every concept/.append style={%
      radius=0.2cm
    },
    /tikz/documentation/.style={
      color=red,
      font=\tiny,
      outer sep=0pt,
      inner sep=0pt,
      anchor=east,
      ->
    }
  ]
  \Node(top)(0,2)
  \Node(join)(0,1)
  \Node(left)(-1,0)
  \Node(right)(1,0)
  \Node(meet)(0,-1)
  \Node(bottom)(0,-2)
  \Edge(join)(left)%
```

```
\Edge(join)(top)
\Edge(left)(meet)
\Edge(join)(right)
\Edge(right)(meet)
\Edge(meet)(bottom)
\centerAttbox(top){top Attribute}
\leftAttbox(left){left Attribute}
\rightAttbox(right){right Attribute}
\centerObjbox(bottom){bottom object}
\leftObjbox(left){left object}
\rightObjbox(right){right object}
%   documentation
\draw[documentation,<-] (fca node join) -- +(6,0.25)
  node[anchor=south west] (doku)
  {\textbackslash Node(join)(0,-1)};
%     Other nodes are right aligned to the first docu node
  \draw[documentation]
  (fca node top attributes center -| doku.east)
  node(centerAttBox){%
  \textbackslash centerAttBox(join)\{top Attribute\}%
} (centerAttBox) -- (fca node top attributes center);
  \draw[documentation]
  (fca node left attributes left -| doku.east) + (0,0.5)
  node(leftAttBox){%
  \textbackslash leftAttBox(join)\{left Attribute\}%
} (leftAttBox.west) -- (fca node left attributes left.north east);
\draw[documentation]
  (fca node right attributes right -| doku.east)
  node(rightAttBox){\textbackslash rightAttBox(join)\{right Attribute\}}
  (rightAttBox) -- (fca node right attributes right);
\draw[documentation]
  (fca node left objects left -| doku.east) + (0,-0.5)
  node(leftObjBox){\textbackslash leftObjBox(join)\{left Object\}}
  (leftObjBox.west) -- (fca node left objects left.south east);
\draw[documentation]
  (fca node right objects right -| doku.east)
  node(rightObjBox){\textbackslash rightObjBox(join)\{right Object\}}
  (rightObjBox) -- (fca node right objects right);
\draw[documentation]
  (fca node bottom objects center -| doku.east)
  node(centerObjBox){%
    \textbackslash centerObjBox(meet)\{bottom Object\}}
  (centerObjBox) -- (fca node bottom objects center) ;
\coordinate (edgepoint) at ($(bottom)!0.5!(meet)$);
\draw[documentation] (edgepoint -| doku.east)
  node(centerObjBox){%
    \textbackslash Edge(meet)(bottom)}
  (centerObjBox) -- (edgepoint) ;
\end{tikzdiagram}
```

The same diagram can be typeset using PGF:

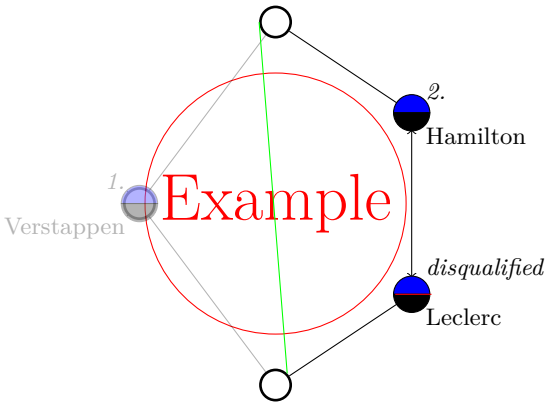
Example 26: A pgf enhanced diagram

Code:

```
\begin{diagram}[
  every attributes/.append style={
    label concept/.append style={
      shape=semicircle,
      fill=blue,
      anchor=south,
      minimum height=2\unitlength,
      label/name suffix=attribute concept,
      label/at=center,
      solid
    }
  },
  every objects/.append style={
    label concept/.append style={
      shape=semicircle,
      fill=black,
      anchor=south,
      minimum height=2\unitlength,
      label/name suffix=attribute concept,
      label/at=center,
      solid,
      rotate=180
    }
    }]
  \begin{pgfscope}
    \color{red}
    \pgftransformshift{\pgfpoint{20\unitlength}{30\unitlength}}%
    \pgfnode{circle}{center}{\Huge Example}{example
    text}{\pgfusepath{stroke}}%
  \end{pgfscope}
  \Node(1)(20,10)
  \Node(2)(35,20)
  \begin{pgfscope}
    \pgfsetstrokeopacity{0.3}
    \pgfsetfillopacity{0.3}
    \Node(3)(5,30)
  \end{pgfscope}
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \begin{pgfscope}
    \pgfsetstrokeopacity{0.3}
    \Edge(1)(3)
  \end{pgfscope}
```

```
    \begin{pgfscope}
      \pgfsetarrows{<->}
      \Edge(2)(4)
    \end{pgfscope}
    \begin{pgfscope}
      \pgfsetstrokeopacity{0.3}
      \Edge(3)(5)
    \end{pgfscope}
    \Edge(4)(5)
    \begin{pgfscope}
      \pgfsetstrokeopacity{0.3}
      \pgfsetfillopacity{0.3}
      \leftAttbox(3){1.}
    \end{pgfscope}
    \rightAttbox(2){disqualified}
    \rightAttbox(4){2.}
    \begin{pgfscope}
      \pgfsetstrokeopacity{0.3}
      \pgfsetfillopacity{0.3}
      \leftObjbox(3){Verstappen}
  \end{pgfscope}
    \rightObjbox(2){Leclerc}
    \rightObjbox(4){Hamilton}
    \begin{pgfscope}%
      \edef\tempa{%
      \noexpand\pgftransformshift{%
      \noexpand\pgfpointanchor{\pgfkeysvalueof{/fca/name prefix}2\pgfkeysvalueof{/fca/
    }%
      \tempa
      \pgfmoveto{\pgfpoint{0pt}{0pt}}%
      \pgflineto{\pgfpoint{4\unitlength}{0pt}}
      \pgfsetstrokecolor{red}
      \pgfusepath{stroke}
    \end{pgfscope}%
    {
      \pgfmoveto{\pgfpointanchor{fca node 5}{west}}
      \pgflineto{\pgfpointanchor{fca node 1}{north east}}
      \pgfsetstrokecolor{green}
      \pgfusepath{stroke}
    }
  \end{diagram}
```

Result:



The old syntax is also supported:

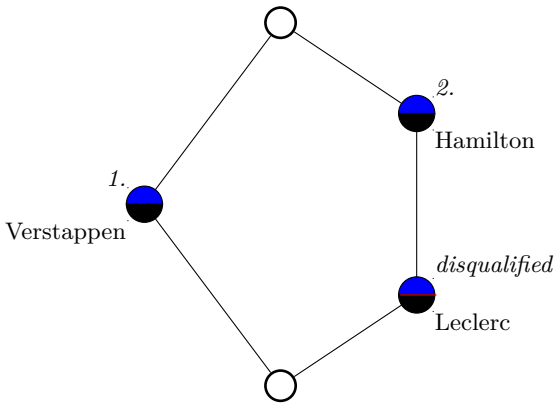
Example 27: A diagram

Code:

```
\begin{diagram}[
  every attributes/.append style={
    label concept/.append style={
      shape=semicircle,
      fill=blue,
      anchor=south,
      minimum height=2\unitlength,
      label/name suffix=attribute concept,
      solid
    }
  },
  every objects/.append style={
    label concept/.append style={
      shape=semicircle,
      fill=black,
      anchor=south,
      minimum height=2\unitlength,
      label/name suffix=attribute concept,
      solid,
      rotate=180
    }
  }]{40}{55}
  \Node{1}{20}{10}
  \Node{2}{35}{20}
  \Node{3}{5}{30}
  \Node{4}{35}{40}
  \Node{5}{20}{50}
  \Edge{1}{2}
  \Edge{1}{3}
  \Edge{2}{4}
```

```
    \Edge{3}{5}
    \Edge{4}{5}
    \Numbers
    \leftAttbox{3}{2}{2}{1.}
    \rightAttbox{2}{2}{2}{disqualified}
    \rightAttbox{4}{2}{2}{2.}
    \leftObjbox{3}{2}{2}{Verstappen}
    \rightObjbox{2}{2}{2}{Leclerc}
    \rightObjbox{4}{2}{2}{Hamilton}
    \begin{pgfscope}%
      \edef\tempa{%
      \noexpand\pgftransformshift{%
      \noexpand\pgfpointanchor{\pgfkeysvalueof{/fca/name prefix}2\pgfkeysvalueof{/fca/
    }%
      \tempa
      \pgfmoveto{\pgfpoint{0pt}{0pt}}%
      \pgflineto{\pgfpoint{4\unitlength}{0pt}}
      \pgfsetstrokecolor{red}
      \pgfusepath{stroke}
    \end{pgfscope}%
  \end{diagram}
```

Result:



9 The Code

This is file 'fca.sty' : LaTeX macros for Formal Concept Analysis

This program is provided under the terms of the LaTeX Project Public License distributed from CTAN archives in directory macros/latex/base/lppl.txt.

This package contains two environments, called cxt and diagram, for typesetting formal contexts and order diagrams, and a few macros for frequently used symbols in FCA.

9.1 Package options for fca.sty

Opt `compat`

Opt `nocompat`

`\iffca@compat@macros`

.

Activation of ancient parts that don't have a proper prefix.

```
\newif\iffca@compat@macros
\fca@compat@macrosfalse
\DeclareOption{compat}{%
  \fca@compat@macrostrue
}
\DeclareOption{nocompat}{%
  \fca@compat@macrosfalse
}
```

Opt `draft`

`\iffca@draft`

. Enable draft mode. Some features are available only in draft mode as the should not be used in final diagrams.

```
\newif\iffca@draft
\fca@draftfalse
\DeclareOption{draft}{%
  \fca@drafttrue
}
\DeclareOption{final}{%
  \fca@draftfalse
}
```

Evaluate the packages options.

```
\ProcessOptions\relax
```

9.2 Loading other packages and general helpers

We use `pgf` for graphics either direct or – when Ti*k*Z is loaded indirect via that package. For semicircles its geometric shapes library is used. `amssymb` is used for spacial symbols, `graphics` for `\rotatebox` and similar macros. The package `color` provides colrouring features. `ifthen` is used for some of the boolean variables.

```
\RequirePackage{pgf}
\usepgflibrary{shapes.geometric}
\RequirePackage{amssymb,graphics,color,ifthen,xspace}%%
\@ifundefined{AfterPackage}{%
  \RequirePackage{afterpackage}%
}{}%
\AfterPackage{tikz}{%
  \usetikzlibrary{fca}%
}%
```

`\fca@parselength` `{`⟨*register*⟩`}{`⟨*expression*⟩`}{`⟨*default unit*⟩`}` Evaluates ⟨*expression*⟩. If it has no unit the ⟨*default unit*⟩ is use instead. The result is stored in ⟨*register*⟩.

```
\newcommand*\fca@parselength[3]{%
  \pgfmathparse{#2}%
  \ifpgfmathunitsdeclared
    #1=\pgfmathresult pt\relax
  \else
    #1=\pgfmathresult #3\relax%
  \fi
}
```

9.3 The context environment `cxt`

9.3.1 Some configurations

`\fca@cxt@Kreuz` `\fca@cxt@Punkt` `\fcaCxtArrowStyle` `\cxtArrowStyle` `\fca@cxt@down` `\fca@cxt@up` `\fca@cxt@both` First we define some symbols that are used in formal contexts.

```
\newcommand{\fca@cxt@Kreuz}{$\times$}%
\newcommand{\fca@cxt@Punkt}{}%
\iffca@compat@macros
  \newcommand{\cxtArrowStyle}{\footnotesize}
  \def\fcaCxtArrowStyle{\cxtArrowStyle}
\else
  \newcommand{\fcaCxtArrowStyle}{\footnotesize}
\fi
\newcommand{\fca@cxt@down}{\fcaCxtArrowStyle$\Runterpfeil$}%
\newcommand{\fca@cxt@up}{\fcaCxtArrowStyle$\Hochpfeil$}%
\newcommand{\fca@cxt@both}{\fcaCxtArrowStyle$\Doppelpfeil$}%
```

Ctr `fca@cxt@mAnz` Count the number of attributes in the current context.

```
\newcounter{fca@cxt@mAnz}%
```

Bool `fca@cxt@ttributes` Record whether we ar still in the attributes section or whether we have already started the object section of a formal context.

```
\newboolean{fca@cxt@ttributes}%
```

Len `\fca@cxt@nameraise` Helper length for vertical alignment of the context and attribute names. Helper

Len `\fca@cxt@ttnameheight`

length for vertical alignment of the context and attribute names.

```
\newlength{\fca@cxt@nameraise}%
\newlength{\fca@cxt@ttnameheight}%
```

\adjcxt@name Adjusts the the vertical alignment of the attributes.

```
\newcommand{\adjcxt@name}{%
  \ifthenelse{\fca@cxt@nameraise<\fca@cxt@ttnameheight}%
  {\setlength{\fca@cxt@nameraise}{\fca@cxt@ttnameheight}}{}}%
```

\alignBottom \fcaCxtAlignBottom Align the current context to the bottom. Inside the `cxt` environment the macro `\fcaCxtAlignBottom` is available in its short form `\alignBottom`.

```
\newcommand{\fcaCxtAlignBottom}{\def\fca@cxt@align{b}}
```

\alignCenter \fcaCxtAlignCenter Align the current context to the top. Inside the `cxt` environment the macro `\fcaCxtAlignCenter` is available in its short form `\alignCenter`.

```
\newcommand{\fcaCxtAlignCenter}{\def\fca@cxt@align{t}}
```

\alignTop \fcaCxtAlignTop Align the current context to the top. Inside the `cxt` environment the macro `\fcaCxtAlignTop` is available in its short form `\alignTop`.

```
\newcommand{\fcaCxtAlignTop}{\def\fca@cxt@align{t}}
```

9.3.2 The main structure of a formal context

Typically a context is created in the following way:

1. The environment `cxt` is opened. This sets up the basic configuration. An empty name is constructed.
2. Attributes are added to the context. They are added as tokens to `\fca@cxt@tabtop`.
3. When the control arrives at the first call to `\obj`, the `tabular` environment is opened and `\att` and `\atr` are disabled.

\cxtName \fcaCxtName Set the name of the current formal context. If used outside of a `cxt` environment it sets the name for all following contexts. Inside a `cxt` environment `\fcaCxtName` can be accessed also with the shorter name `\cxtName`.

```
\newcommand{\fcaCxtName}[1]{%
  \def\fca@cxtn@me{%
    \multicolumn{1}{|c||}{%
      \settoheight{\fca@cxt@ttnameheight}{#1}%
```

```
      \addtolength{\fca@cxt@nameraise}{-1\fca@cxt@ttnameheight}%
      \raisebox{.5\fca@cxt@nameraise}{#1}%
    }%
  }%
  \ignorespaces
}%
\iffca@compat@macros
  \newcommand{\cxtName}{\fcaCxtName}
\fi
```

`\fca@cxt@att` {⟨*name*⟩} Implementation for `\att`. The corresponding alias is set up during `\begin`{⟨*cxt*⟩}.

`\att` {⟨*name*⟩}

For each attribute the user must provide us a name either with `\att` or with `\atr`. Both save the prvided name as heading. For horizontally oriented attribute names (typically very short ones) the user should use `\att`.

```
\newcommand{\fca@cxt@att}[1]{%
  \ifthenelse{\boolean{fca@cxt@ttributes}}{%
    \settoheight{\fca@cxt@ttnameheight}{#1}\adjcxt@name%
    \expandafter\def\expandafter\fca@cxt@tabtop\expandafter{%
      \fca@cxt@tabtop&#1}%
    \stepcounter{fca@cxt@mAnz}%
  }{%
    \PackageWarning{fca}{Attribute following object in
      cxt-environment%
      has been ignored}{}}%
  \ignorespaces }%
```

`\fca@cxt@atr` {⟨*name*⟩} Implementation for `\atr`. The corresponding alias is set up during `\begin`{⟨*cxt*⟩}.

`\atr` {⟨*name*⟩} The macro `\atr` is available only in the `cxt` environment. it calls `\att` with its name rotated by 90 degrees, so that the name is typeset vertically.

```
\newcommand{\fca@cxt@atr}[1]{\att{\rotatebox{90}{#1~~}}}%
```

`\fca@cxt@obj` {⟨*crosses*⟩}{⟨*objectname*⟩} Implementation for `\obj`. The corresponding alias is set up during `\begin`{⟨*cxt*⟩}.

`\obj` {⟨*crosses*⟩}{⟨*objectname*⟩} This macro typesets an object line of a formal context inside the `cxt` environment. The second argument {⟨*objectname*⟩} is the name of the objects. The first argument {⟨*crosses*⟩} is a line of tokens. Each token represents the contents of one cell in the context table. Typically tokens contain spaces, arrows, crosses. But they can be defined to represent other material as well. Even multi character tokens are possible. However these are not documented.

```
\newcommand{\fca@cxt@obj}[2]{%
```

```
  \fca@cxt@tabdef
  #2\strut
  \fca@cxt@Line{#1}%
  \\
}%
```

`\fca@cxt@freeobj` {⟨*columns*⟩}{⟨*name*⟩} Implementation for `\freeobj`. The corresponding alias is set up during `\begin{`⟨*cxt*⟩`}`.

`\freeobj` {⟨*columns*⟩}{⟨*name*⟩} This macro allows to typeset any material in the incidence area of the context. The second argument is typeset in the name column of the context, while the first one occures inside the incidence area of the context. The different fields in the first argument must be separated as usual by `&`.

```
\newcommand{\fca@cxt@freeobj}[2]{%acrocod}
  \fca@cxt@tabdef%
  #2&#1\\\hline }%
```

`\fca@cxt@tabtop` Material that is typeset above a context line.

At the beginning this macro contains the table header. Later it is used to typeset the lines between the objects.

The Table heading for the attributes will be filled by `\att` and `\atr` during the attribute section of the formal context. The first `\obj` will use it.

```
\def\fca@cxt@tabtop{}
```

`\fca@cxt@tabdef` `\fca@cxt@tabdef@@` Expand `\fca@cxt@align` as argument to `\fca@cxt@tabdef@`.

At the begin of a `cxt` environment `\fca@cxt@tabdef` set to be an alias of `\fca@cxt@tabdef@@`. As soon as `\fca@cxt@tabdef` is executed, this macro is set to `\relax`.

It is executed at the end of the cxt environment so that we can typeset contexts without any objects.

```
\def\fca@cxt@tabdef@@{%
  \expandafter\fca@cxt@tabdef@\fca@cxt@align
  \fca@cxtn@me%&%
  \fca@cxt@tabtop\strut\\\hline\hline
}
```

`\fca@cxt@tabdef@` {⟨*alignment*⟩} Do the work of `\fca@cxt@tabdef`: Open the tabular environment. The parameter {⟨*alignment*⟩} will be used to set the vertiacal alignment of the context. See the documentation of the `tabular` environment for further documentation.

Note: We must do all definitions outside of the tabular environment.

```
\def\fca@cxt@tabdef@#1{%
  \def\fca@cxt@tabdef{\hline}%
  \tabcolsep0.5ex\relax%
  \begin{tabular}[#1]{|l||*{\value{fca@cxt@mAnz}}{c|}}%
    \hline%
    }%
```

Env `cxt` [⟨*alignment*⟩]

The `cxt` environment. During setup a set of macros are defined, and a new group is opened. The alignment argument is saved. During the the first call to `\obj` a tabular environment will be opened by calling `\fca@cxt@tabdef`. At the end of the environment the tabular environment is closed.

Here we test empty contexts. They should work, but don't need to be documented:

Example 28: A formal context without objects.

Code:

```
\begin{cxt}
  \cxtName{Formula 1}
%
  \att{1.}
  \att{2.}
  \atr{disqualified}
%
\end{cxt}
```

Result:

Formula 1	1.	2.	disqualified

Example 29: A formal context without attributes.

Code:

```
\begin{cxt}
  \cxtName{Formula 1}
%
  \obj{}{Verstappen}
  \obj{}{Hamilton}
  \obj{}{Leclerc}
\end{cxt}
```

Result:

Formula 1
Verstappen
Hamilton
Leclerc

Example 30: An empty named context.

Code:

```
\begin{cxt}
  \cxtName{Formula 0}
\end{cxt}
```

Result:

Formula 0

Example 31: An empty `cxt` environment.

Code:

```
\begin{cxt}
\end{cxt}
```

Result:

```
    \newenvironment{cxt}[1][t]{%
    \begingroup
    \fca@cxt@resetDefaults
    \def\fca@cxt@align{#1}%
    \ignorespaces
    }{%
    \fca@cxt@tabdef% open the tabular in case there are no objects
  \end{tabular}%
  \endgroup }%
```

`\fca@cxt@resetDefaults` Initialize macros an registered that are used in a formal context. This macro is called at `\begin{cxt}`.

```
\newcommand{\fca@cxt@resetDefaults}{%
  \setlength{\fca@cxt@nameraise}{0pt}%
  \setlength{\fca@cxt@ttnameheight}{0pt}%
```

```
  \setcounter{fca@cxt@mAnz}{0}%
  \setboolean{fca@cxt@ttributes}{true}%
  \let\cxtName\fcaCxtName
  \let\alignBottom\fcaCxtAlignBottom
  \let\alignCenter\fcaCxtAlignCenter
  \let\alignTop   \fcaCxtAlignTop
  \let\att\fca@cxt@att
  \let\atr\fca@cxt@atr
  \let\obj\fca@cxt@obj
  \let\freeobj\fca@cxt@freeobj
  \let\cxtphantom\fca@cxt@phantom
  \let\cxtrlap\fca@cxt@rlap
  \let\fca@cxtn@me\@empty%
  \let\fca@cxt@tabdef\fca@cxt@tabdef@@
}%
```

9.4 Cross table contents

9.4.1 Defining the characters in a context.

Formal contexts usually contain crosses and – when we are looking for irreducible elements – arrows. Many-valued contexts can contain arbitrary content. This section describes the macros that used to typeset a single cell. This incluedes the symbols that can be used in a context line as well as the macros that allow to define them.

It is necessary that very symbol in the context lines must be defined with the macros from this sections. Only this ensures that the parser gets restarted whenever a character is executed.

`\fca@cxt@phantom` `\cxtphantom` This macro creates the horizontal space that would have been taken by a cross in the context. It is used to properly position other signs in the table without modifying the spacing.

```
\def\fca@cxt@phantom{\phantom{\fca@cxt@Kreuz}}%
```

`\fca@cxt@rlap` `{`⟨*content*⟩`}`

`\cxtrlap` `{`⟨*content*⟩`}` The argument ⟨*content*⟩ will be typeset centered in a cell that has the same size as an ordinary cross. `\cxtrlap` is only available in the `cxt` environment.

```
\def\fca@cxt@rlap#1{%
  \settowidth\@tempdima{\cxtphantom}%
  \makebox[\@tempdima][c]{\hss #1\hss}%
}
```

`\fca@cxt@M@kechar@newcommand` Put a starred `\newcommand*` into one single token. We will use it when we define a single character.

```
\def\fca@cxt@M@kechar@newcommand{%
  \newcommand*%
}
```

\fca@cxt@Makechar@newcommand {⟨*letter*⟩} Defines a macro for the ⟨*letter*⟩ to be used as a single token in the `cxt` environment. It is equivalent to `\newcommand*{letter}` and can take all additional arguments that `\newcommand` takes.

Note: the macro is not restricted to letters. It can also use command names (that are converted to strings)

```
\def\fca@cxt@Makechar@newcommand#1{%
  \expandafter \fca@cxt@M@kechar@newcommand \csname cxt@char@\string#1 \endcsname
}
```

\fcaNewContextChar {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩}

\fcaNewContextChar@ {⟨*character*⟩}{⟨*definition*⟩}

\fcaNewContextChar@@ {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩}

\fcaNewContextCh@r {⟨*character*⟩}[⟨*arguments*⟩]{⟨*definition*⟩}

\fcaNewContextCh@r@ {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩} The macro `\fcaNewContextChar` can be used to define a new character for usage in the `cxt` environment in the `\obj` macro. Internally it calls `\newcommand*` with the given arguments.

The remaining macros `\fcaNewContextChar@`, `\fcaNewContextChar@@`, `\fcaNewContextCh@r` and `\fcaNewContextCh@r@` are used to pass the right arguments to `\newcommand*`.

```
\def\fcaNewContextCh@r@#1[#2][#3]#4{%
  \fca@cxt@Makechar@newcommand{#1}[{#2}][{#3}]{#4\fca@cxt@read@line}%
}%
\def\fcaNewContextCh@r#1[#2]#3{%
  \fca@cxt@Makechar@newcommand{#1}[{#2}]{#3\fca@cxt@read@line}%
}%
\def\fcaNewContextChar@@#1[#2]{%
  \@ifnextchar[{\fcaNewContextCh@r@{#1}[{#2}]}{\fcaNewContextCh@r{#1}[{#2}]}%
}%
\def\fcaNewContextChar@#1#2{%
  \fca@cxt@Makechar@newcommand{#1}{#2\fca@cxt@read@line}%
}%
\def\fcaNewContextChar#1{%
  \@ifundefined{cxt@char@\string#1 }{}{%
    \PackageError{fca}{The character ‘\string#1’ is already defined.}%
    \expandafter\let
    \csname cxt@char@\string#1 \endcsname\@undefined
  }%
```

```
  \@ifnextchar[{\fcaNewContextChar@@{#1}}{\fcaNewContextChar@{#1}}%
}
```

a@cxt@M@kechar@providecommand Put a starred `\providecommand*` into one single token. We will use it when we define a single character.

```
\def\fca@cxt@M@kechar@providecommand{%
  \providecommand*%
}
```

a@cxt@Makechar@providecommand {⟨*letter*⟩} Defines a macro for the ⟨*letter*⟩ to be used as a single token in the `cxt` environment. It is equivalent to `\providecommand*letter` and can take all additional arguments that `\providecommand` takes.

Note: the macro is not restricted to letters. It can also use command names (that are converted to strings)

```
\def\fca@cxt@Makechar@providecommand#1{%
  \expandafter \fca@cxt@M@kechar@providecommand \csname cxt@char@\string#1 \endcsname
}
```

\fcaProvideContextChar {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩}

\fcaProvideContextChar@ {⟨*character*⟩}{⟨*definition*⟩}

\fcaProvideContextChar@@ {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩}

\fcaProvideContextCh@r {⟨*character*⟩}[⟨*arguments*⟩]{⟨*definition*⟩}

\fcaProvideContextCh@r@ {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩} The macro `\fcaProvideContextChar` can be used to define a provide character for usage in the `cxt` environment in the `\obj` macro. Internally it calls `\providecommand*` with the given arguments.

The remaining macros `\fcaProvideContextChar@`, `\fcaProvideContextChar@@`, `\fcaProvideContextCh@r` and `\fcaProvideContextCh@r@` are used to pass the right arguments to `\providecommand*`.

```
\def\fcaProvideContextCh@r@#1[#2][#3]#4{%
  \fca@cxt@Makechar@providecommand{#1}[{#2}][{#3}]{#4\fca@cxt@read@line}%
}%
\def\fcaProvideContextCh@r#1[#2]#3{%
  \fca@cxt@Makechar@providecommand{#1}[{#2}]{#3\fca@cxt@read@line}%
}%
\def\fcaProvideContextChar@@#1[#2]{%
  \@ifnextchar[{\fcaProvideContextCh@r@{#1}[{#2}]}{\fcaProvideContextCh@r{#1}[{#2}]}%
}%
\def\fcaProvideContextChar@#1#2{%
  \fca@cxt@Makechar@providecommand{#1}{#2\fca@cxt@read@line}%
}%
```

```
\def\fcaProvideContextChar#1{%
  \@ifnextchar[{\fcaProvideContextChar@@{#1}}{\fcaProvideContextChar@{#1}}%
}
```

`fca@cxt@M@kechar@renewcommand` Put a starred `\renewcommand*` into one single token. We will use it when we define a single character.

```
\def\fca@cxt@M@kechar@renewcommand{%
  \renewcommand*%
}
```

`fca@cxt@Makechar@renewcommand` [⟨*letter*⟩] Defines a macro for the ⟨*letter*⟩ to be used as a single token in the `cxt` environment. It is equivalent to `\renewcommand*letter` and can take all additional arguments that `\renewcommand` takes.

Note: the macro is not restricted to letters. It can also use command names (that are converted to strings)

```
\def\fca@cxt@Makechar@renewcommand#1{%
  \expandafter \fca@cxt@M@kechar@renewcommand \csname cxt@char@\string#1 \endcsname
}
```

`\fcaRenewContextChar` {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩}

`\fcaRenewContextChar@` {⟨*character*⟩}{⟨*definition*⟩}

`\fcaRenewContextChar@@` {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩}

`\fcaRenewContextCh@r` {⟨*character*⟩}[⟨*arguments*⟩]{⟨*definition*⟩}

`\fcaRenewContextCh@r@` {⟨*character*⟩}[⟨*arguments*⟩][⟨*default*⟩]{⟨*definition*⟩} The macro `\fcaRenewContextChar` can be used to define a renew character for usage in the `cxt` environment in the `\obj` macro. Internally it calls `\renewcommand*` with the given arguments.

The remaining macros `\fcaRenewContextChar@`, `\fcaRenewContextChar@@`, `\fcaRenewContextCh@r` and `\fcaRenewContextCh@r@` are used to pass the right arguments to `\renewcommand*`.

```
\def\fcaRenewContextCh@r@#1[#2][#3]#4{%
  \fca@cxt@Makechar@renewcommand{#1}[{#2}][{#3}]{#4\fca@cxt@read@line}%
}%
\def\fcaRenewContextCh@r#1[#2]#3{%
  \fca@cxt@Makechar@renewcommand{#1}[{#2}]{#3\fca@cxt@read@line}%
}%
\def\fcaRenewContextChar@@#1[#2]{%
  \@ifnextchar[{\fcaRenewContextCh@r@{#1}[{#2}]}{\fcaRenewContextCh@r{#1}[{#2}]}%
}%
\def\fcaRenewContextChar@#1#2{%
  \fca@cxt@Makechar@renewcommand{#1}{#2\fca@cxt@read@line}%
```

```
}%
\def\fcaRenewContextChar#1{%
  \@ifundefined{cxt@char@\string#1 }{%
    \PackageError{fca}{The character ‘\string#1’ is undefined.^^J
      It must have been defined in order to be redefined.}%
    \fcaNewContextChar{#1}{}%
  }{}%
  \@ifnextchar[{\fcaRenewContextChar@@{#1}}{\fcaRenewContextChar@{#1}}%
}
```

9.4.2 Reading context lines

`\fca@cxt@stop` This macro does nothing. It is used for checking emptiness when a context line is parsed.

```
\def\fca@cxt@stop{}%
```

`\fca@cxt@executechar` `{`⟨*character*⟩`}` This macro executes the command sequence associated to the meaning of a character in a context line. In order to process a whole context line the macro is appended to each context character definition. This gives the code the meaning of an unfolded `\@for` loop.

If the token is `\fca@cxt@stop`, the loop ends.

```
\def\fca@cxt@executechar#1{%
  &\@ifundefined{cxt@char@\string#1 }{%
    \PackageWarning{fca}{Undefinded character \string#1 \space in the context}%
    \let\fca@cxt@tmp=\fca@cxt@aPunkt%
  }{%
    \expandafter\let\expandafter\fca@cxt@tmp \csname
    cxt@char@\string#1 \endcsname }%
  \fca@cxt@tmp }
```

`\fca@cxt@read@line` `{`⟨*character*⟩`}` Start processing a context line. We look only at the first character. If it is `\fca@cxt@stop`, then the line is empty and nothing is to do. Otherwise we execute the character macro. The definition of the character ensures that this starts a loop until eventually `\fca@cxt@stop` is reached.

```
\def\fca@cxt@read@line#1{%
  \ifx#1\fca@cxt@stop \let\fca@cxt@zeile@excecutechar\@gobble \else
  \let\fca@cxt@zeile@excecutechar\fca@cxt@executechar \fi
  \fca@cxt@zeile@excecutechar{#1}%
}%
% \end{macro}
%
% \begin{macro}{\fca@cxt@Line}\marg{\line}
%   Process a whole crosstable line. We add \cs{fca@cxt@stop} at the end and start the loop
% \begin{macrocode}
\def\fca@cxt@Line#1{%
  \fca@cxt@read@line#1\fca@cxt@stop }%
```

9.4.3 The context characters

Finally, we can define the predefined characters that can be used inside a formal context:

[context character] . **.:** An empty context cell.

[context character] x **x:** A cross in the context.

[context character] X **X:** Alternative sign for a cross.

[context character] u **u:** An up-arrow in the context.

[context character] d **d:** A down-arrow in the context.

[context character] b **b:** A cell containing both an up- and a down-arrow.

[context character] 0 **0:** A zero in a many-valued context.

[context character] 1 **1:** A one in a many-valued context.

[context character] 2 **2:** Two in a many-valued context.

[context character] 3 **3:** Three in a many-valued context.

[context character] 4 **4:** Four in a many-valued context.

[context character] 5 **5:** Five in a many-valued context.

[context character] 6 **6:** Six in a many-valued context.

[context character] 7 **7:** Seven in a many-valued context.

[context character] 8 **8:** Eight in a many-valued context.

[context character] 9 **9:** Nine in a many-valued context.

```
\fcaNewContextChar .{\cxtphantom}
\fcaNewContextChar x{\fca@cxt@Kreuz}
\fcaNewContextChar X{\fca@cxt@Kreuz}
\fcaNewContextChar u{\cxtrlap{\fca@cxt@up}}
\fcaNewContextChar d{\cxtrlap{\fca@cxt@down}}
\fcaNewContextChar b{\cxtrlap{\fca@cxt@both}}
\@for\tmp:= 0,1,2,3,4,5,6,7,8,9\do{
  \edef\@tmp{
    \noexpand\fcaNewContextChar\tmp{%
      \noexpand\cxtrlap\tmp
    }%
  }%
  \@tmp
}
```

end of cxt environment definition

9.5 Reading Burmeister context files

LaTeX macros for Formal Concept Analysis input of Burmeister format contexts

This package defines the macro `\cxtinput`, which can input a context file in Burmeister format

Usage:

```
\begin{cxt}                                                    %
\cxtAlignBottom                                                %
\end{cxt}                                                      %
```

Known bugs: • The end of the .cxt file is not correctly detected. You will get the error message: Runaway argument? ! File ended while scanning use of `\fca@cxt@input@getline`.

TODO: • Make everything configurable

used counters

Ctr `fca@cxt@input@obj`

Ctr `fca@cxt@input@attr`

Ctr `fca@cxt@input@line`

fca@cxt@input@contextlines

This counter is used to store the number of objects in a `.cxt` file. This counter is used to store the number of attributes in a `.cxt` file. This counter is used to count the input lines in a formal context. In this register the content of a `cxt` environment is collected before the environment is actually inserted into the LaTeX stream.

```
\newcount\fca@cxt@input@obj
\newcount\fca@cxt@input@attr
\newcount\fca@cxt@input@line
\newtoks\fca@cxt@input@contextlines
\fca@cxt@input@line0
\fca@cxt@input@contextlines{}%
```

`\oarg`

file name The end user macro. It inclues the context from ⟨*file name*⟩. The context must be stored in Burmeister format.

```
%
\newcommand\cxtinput[1]{%
%  \begingroup
  \fca@cxt@input@contextlines{}%
  \fca@cxt@input@save@nl@active%
  \fca@cxt@input@make@nl@active%
  \fca@cxt@input@input{#1}%
  \fca@cxt@input@restore@nl%
  \xdef\fca@cxt@input@tempa{\the\fca@cxt@input@contextlines}%
  %\aftergroup
  \fca@cxt@input@tempa%
%  \endgroup%
}
```

`\fca@cxt@input@newline`

Macro holding the command for the next line

```
\def\fca@cxt@input@newline{}
```

```
\def\fca@cxt@input@head{%
 \fca@cxt@input@getline\fca@cxt@input@check@B
}
```

Check the “B” at the beginning of the file

```
\def\fca@cxt@input@check@B#1{%
  \def\tempa{B}\def\tempb{#1}%
  \ifx\tempa\tempb
  \typeout{Burmeister format detected}%
  \else
    \fca@cxt@input@error{No Burmeister format detected}{The \string\cxtinput macro can input
  \fi
  \def\fca@cxt@input@newline{%
    \fca@cxt@input@getline{\fca@cxt@input@read@cxtname}%
  }%
}
```

Check for an empty line and continue with command #2 afterwards

```
\def\fca@cxt@input@match@mptyline#1#2{%
  \edef\tempa{#2}%
  \ifx\tempa\@empty
  \else
    \fca@cxt@input@error{Error in Burmeister format.}{At the current position an empty line
  \fi
  \def\fca@cxt@input@newline{%
    \fca@cxt@input@getline{#1}%
  }%
}
```

Read the context name

```
\def\fca@cxt@input@read@cxtname#1{%
  \ifx\fca@cxtn@me\@empty
    \fcaCxtName{#1}%
  \fi
  \def\fca@cxt@input@newline{%
    \fca@cxt@input@getline{\fca@cxt@input@readobjcount}%
  }%
}
```

Read number of objects from the file

```
\def\fca@cxt@input@readobjcount#1{%
 \fca@cxt@input@obj=#1\relax
 \def\fca@cxt@input@newline{%
   \fca@cxt@input@getline\fca@cxt@input@readattrcount%
 }%
}
```

Read number of attributes

```
\def\fca@cxt@input@readattrcount#1{%
  \def\fca@cxt@input@newline{%
    \fca@cxt@input@getline{\fca@cxt@input@match@mptyline\fca@cxt@input@readobjects}%
  }%
  \fca@cxt@input@attr=#1\relax
}
```

initializes the reading of the object names

```
\def\fca@cxt@input@readobjects{%
  \ifnum\fca@cxt@input@obj>0\relax
    \def\fca@cxt@input@newline{%
      \fca@cxt@input@getline\fca@cxt@input@readobjname
   }%
   \@tempcnta=1\relax
   \let\tempa\fca@cxt@input@readobjname%
 \else
    \let\tempa\fca@cxt@input@readattributes%
  \fi
 \tempa
}
```

Read the object names

```
\def\fca@cxt@input@readobjname#1{%
   \expandafter\def\csname cxt@input@objname@\the\@tempcnta\endcsname{#1}%
   \ifnum\@tempcnta < \fca@cxt@input@obj
    \advance\@tempcnta by 1\relax
 \else
    \def\fca@cxt@input@newline{%
      \fca@cxt@input@getline\fca@cxt@input@readattributes%
    }%
  \fi
}
```

Initialize reading of attibute names

```
\def\fca@cxt@input@readattributes{%
  \ifnum\fca@cxt@input@attr>0\relax
    \def\fca@cxt@input@newline{%
      \fca@cxt@input@getline\fca@cxt@input@readattrname
    }%
    \@tempcnta=1\relax
    \def\tempa{\fca@cxt@input@readattrname}%
  \else
    \def\tempa{\fca@cxt@input@readcontext}%
  \fi
  \tempa
}
```

Read the attribute names and store `\atr` macros for each attribute

```
\def\fca@cxt@input@readattrname#1{%
  \fca@cxt@input@appendtotok{#1}\atr%
  \ifnum\@tempcnta < \fca@cxt@input@attr
    \advance\@tempcnta by 1\relax
  \else
    \def\fca@cxt@input@newline{%
      \fca@cxt@input@getline\fca@cxt@input@readcontext
    }%
  \fi
}
```

initalize reading of the cross table

```
\def\fca@cxt@input@readcontext{%
  \ifnum\fca@cxt@input@obj>0\relax
    \def\fca@cxt@input@newline{%
      \fca@cxt@input@getline
      \fca@cxt@input@readcontextline
    }%
    \@tempcnta=1\relax
    \def\tempa{\fca@cxt@input@readcontextline}%
  \else
    \def\tempa{}%
  \fi
  \tempa
}
```

Read cross table and store `\obj` macros for each object

```
\def\fca@cxt@input@readcontextline#1{%
  \expandafter\expandafter\expandafter\fca@cxt@input@appendtotok
  \expandafter\expandafter\expandafter{%
    \csname cxt@input@objname@\the\@tempcnta\endcsname}%
  {\obj{#1}}%
  \ifnum\@tempcnta < \fca@cxt@input@obj
    \advance\@tempcnta by 1\relax
  \else
    \let\fca@cxt@input@newline\relax
    \let\fca@cxt@input@endoffile\relax
  \fi
}
```

add some stuff to the token register needed to have some tool, which can be used with `\expandafter`

```
\def\fca@cxt@input@appendtotok#1#2{%
  \expandafter\fca@cxt@input@contextlines\expandafter{%
    \the\fca@cxt@input@contextlines
    #2{#1}%
  }%
}
```

```

\def\fca@cxt@input@error#1#2{%
  \PackageError{fca}{At line \the\fca@cxt@input@line : #1}{#2}%
  \def\fca@cxt@input@newline{}%
}%
```

Some end of file mark

```
\def\fca@cxt@input@endoffile{%
  \fca@cxt@input@error{unexpected end of file}{The context file is somehow
    inconsistent.\MessageBreak The last lines of it seem to be lost.}%
}
```

macro for usage with `\ifx`

```
\def\fca@cxt@input@@endoffile{\fca@cxt@input@endoffile}
```

swich catcode of newline to runtime mode

```
\begingroup%
\catcode`\^^M\active%
```

Define a macro to save the catcode. Define a macro to set the catcode.

```
\gdef\fca@cxt@input@make@nl@active{%
  \catcode`\^^M\active%
% \let\fca@cxt@input@oldcr^^M%
% \def^^M{\fca@cxt@input@newline}%
}%

\gdef\fca@cxt@input@save@nl@active{%
  \chardef\fca@cxt@input@catcode@nl=\catcode`\^^M%
% \let\fca@cxt@input@oldcr^^M%
% \def^^M{\fca@cxt@input@newline}%
}%
```

Define a macro to set the catcode.

```
\gdef\fca@cxt@input@restore@nl{%
  \catcode`\^^M\fca@cxt@input@catcode@nl\relax%
% \let\fca@cxt@input@oldcr^^M%
% \def^^M{\fca@cxt@input@newline}%
}%

\fca@cxt@input@make@nl@active%
```

reads a line from the context file.

```
\long\gdef\fca@cxt@input@getline #1#2^^M{%
  \advance\fca@cxt@input@line by 1\relax%
  \def\tempa{#2}%
```

```
  \ifx\tempa\fca@cxt@input@@endoffile%
    \tempa%
  \fi%
  #1{#2}%
  \fca@cxt@input@newline%
}%
```

read the inputfile and use its content as argument for `\fca@cxt@input@head`

```
\gdef\fca@cxt@input@input#1{%
  \expandafter\fca@cxt@input@head\@@input #1 %
  \fca@cxt@input@endoffile%
}
```

restore newline catcode

```
\endgroup%
```

10 Environment `diagram` for making diagrams of ordered sets, graphs and concept lattices

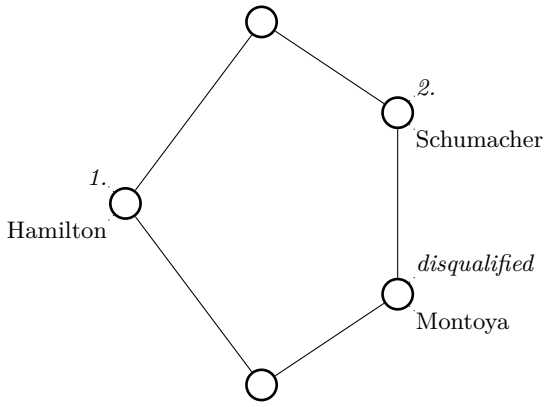
To obtain a diagram for the concept lattice of the formal context above, try this:

Example 32: A lattice diagram

Code:

```
{\unitlength 1.2mm
\begin{diagram}{40}{55}
  \Node{1}{20}{10}
  \Node{2}{35}{20}
  \Node{3}{5}{30}
  \Node{4}{35}{40}
  \Node{5}{20}{50}
  \Edge{1}{2}
  \Edge{1}{3}
  \Edge{2}{4}
  \Edge{3}{5}
  \Edge{4}{5}
  \leftAttbox{3}{2}{2}{1.}
  \rightAttbox{2}{2}{2}{disqualified}
  \rightAttbox{4}{2}{2}{2.}
  \leftObjbox{3}{2}{2}{Hamilton}
  \rightObjbox{2}{2}{2}{Montoya}
  \rightObjbox{4}{2}{2}{Schumacher}
\end{diagram}}
```

Result:



The syntax of the commands is

```
\begin{diagram}{width}{height}                                          %
                                                                        %
  \Node{number}{xcoordinate}{ycoordinate} % (nodenames from 0 to 50)      %
                                                                        %
  \Edge{nodename1}{nodename2}                                          %
                                                                        %
  \leftAttbox{nodename}{xoffset}{yoffset}{text1 \\ text2 \\ ... }         %
                                                                        %
  similarly: \rightAttbox, \centerAttbox,                               %
  \leftObjbox,  \centerObjbox,                                          %
  \rightObjbox.                                                         %
                                                                        %
\end{diagram}
```

The circle size can be changed with the `\fcaCircleSize` command. The value must be a positive integer, which will be mutiplied by `\unitlength`. The default is

`\fcaCircleSize{4}`.

A helpful command when fine tuning a diagram is

`\Numbers`.

You may wish to permanently adjust the following values to your personal preferences. They can also be changed inside each diagram environment using `\renewcommand`.

```
\newcommand{\fca@notikz@Defaults}{% Do not change this line!  %
```

```
  \newcommand{\fcaObjectLabelStyle}{%
    \small\baselineskip1em\rmfamily\upshape%
  }% %
  \newcommand{\fcaAttributeLabelStyle}{%
    \small\baselineskip1em\rmfamily\itshape
  }% %
  \newcommand{\fcaLabelBoxWidth}{40mm}%
  \let\Node\fca@node
  \let\Edge\fca@edge
  \iffca@compat@macros
  \else
    \def\ObjectLabelStyle{\fcaObjectLabelStyle}%
    \def\AttributeLabelStyle{\fcaAttributeLabelStyle}%
    \let\LabelBoxWidth\fcaLabelBoxWidth
    \let\EdgeThickness\fcaEdgeThickness
    \let\NodeThickness\fcaNodeThickness
    \let\Numbers\fcaNumbers
    \let\NoNumbers\fcaNoNumbers
    \let\CircleSize\fcaCircleSize
    \let\NodeColor\fcaNodeColor
    \let\ColorNode\fcaColorNode
    \let\NoDots\fcaNoDots
    \let\Dots\fcaDots
  \fi
  \let\leftAttbox\fca@leftAttbox
  \let\centerAttbox\fca@centerAttbox
  \let\rightAttbox\fca@rightAttbox
  \let\leftObjbox\fca@leftObjbox
  \let\centerObjbox\fca@centerObjbox
  \let\rightObjbox\fca@rightObjbox
}% %
\let\fca@Defaults\fca@notikz@Defaults
\newboolean{fca@connectors}\setboolean{fca@connectors}{true}% %
\iffca@compat@macros
  \newcommand{\diagramXoffset}{0}% %
  \newcommand{\diagramYoffset}{0}% %
  \newcommand{\fcaDiagramXoffset}{\diagramXoffset}% %
  \newcommand{\fcaDiagramYoffset}{\diagramYoffset}% %
\else
  \newcommand{\fcaDiagramXoffset}{0}% %
  \newcommand{\fcaDiagramYoffset}{0}% %
\fi
\newcommand*{\fca@xunitlength}{\unitlength}%
\newcommand*{\fca@yunitlength}{\unitlength}%
\newcommand*{\fca@edge@thickness}{.8pt}% %
\newcommand*{\fca@node@thickness}{1pt}% %
\newcommand*{\fca@transform}{}%
\newcommand*{\fca@options}{}%
\newcommand*{\fca@defaultoptions}{}%
\newcommand*{\fca@usepath}{}%
\newcommand*{\fca@node@number@prefix}{\pgfkeysvalueof{/fca/namespace}\space node\space}%
\newcommand*{\fca@node@number@suffix}{\space number}%
\newcommand*{\fca@label@edge@width}{\@wholewidth}%
\newcommand*{\fca@label@at}{center}% anchor of node
```

```
\newcommand*{\fca@label@shift@x}{0pt}%
\newcommand*{\fca@label@shift@y}{0pt}%
\newcommand*{\fca@label@shift@x@sign}{}
\newcommand*{\fca@label@shift@y@sign}{}
\newcommand*{\fca@label@type}{attributes}%
\newcommand*{\fca@label@position}{right}%

\newcommand*\fca@none{none}
\newcommand*\fca@firstofthree[3]{#1}
\newcommand*\fca@secondofthree[3]{#2}
\newcommand*\fca@thirdofthree[3]{#3}
\newcommand*\fca@testoption[1]{%
  \def\fca@tempa{#1}%
  \ifx\empty\fca@tempa
    \let\fca@tempb\fca@thirdofthree
  \else
    \def\fca@tempb{\pgfkeysnovalue}%
    \ifx\fca@tempa\fca@tempb
      \let\fca@tempb\fca@thirdofthree
    \else
      \ifx\fca@tempa\fca@none
        \let\fca@tempb\fca@secondofthree
      \else
        \let\fca@tempb\fca@firstofthree
      \fi
    \fi
  \fi
  \fca@tempb
}
```

Sometimes (e.g. for large contexts) it is not possible to arrange all nodes in such a way that the edges do not cross any nodes. In such case its often worse not to print any lattice than to live with some compromises. I such situations we must divide the lattice into several layers. Fortunately, PGF and Ti*k*Z support to assign certain graphical objects into layers. This allows us, to keep the order of defining nodes before the edges, while the nodes are drawn on top of the edges (see below).

Please, do not confuse implementation layers (Ti*k*Z, PGF Basic Layer and drivers) with graphical layers. Graphical layers can be thought of slide overlays stacked on top of each other. You can draw on one layer, use the positions in another layer for drawing objects and return to the original layer to add further content.

The formal definition and documentation of layers can be found in the Ti*k*Z and PGF manual at https://tikz.dev or in your local tex installation calling either texdoctk or – on the command line using the command "`texdoc pgfmanual`" in the section "IX The Basic Layer" → "Layered Graphics".

`\fcaLayer` `{`⟨*fca entity*⟩`}{`⟨*layer*⟩`}`

This macro assigns an entity to an existing PGF layer. The layer can be accessed in two ways: Ether by using the environment `pgfonlayer` with the argument ⟨*layer*⟩ or by putting the material between the two macros `\fca@`⟨*entity*⟩`layer`

and `\fca@end`⟨*entity*⟩`layer`. As the at sign shows, the latter form is intended for internal use in LaTeX packages.

```
\newcommand*\fcaLayer[2]{%
  \def\@tempa{main}%
  \edef\@tempb{#2}%
  \ifx\@tempb\@empty\let\@tempb\@tempa\fi
  \ifx\@tempa\@tempb
    \def\@tempa{}%
    \def\@tempb{}%
  \else
    \def\@tempa{%
      \begin{pgfonlayer}{#2}%
      }
    \def\@tempb{%
      \end{pgfonlayer}%
    }
  \fi
  \expandafter\let\csname fca@#1layer\endcsname\@tempa
  \expandafter\let\csname fca@end#1layer\endcsname\@tempb
}
```

`\fca@labelslayer`
`\fca@endlabelslayer`
`\fca@labelconceptslayer`
`\fca@endlabelconceptslayer`

Since objects and attributes can be collected on two different layers, and the same is true for the object and attribute overlays of concepts, we use two meta layers to select the correct layer depending on the type of the current label. However, help lines (so called connectors) are always on the same layer for both object labels and attribute labels.

```
\newcommand*{\fca@labelslayer}{\csname fca@\fca@label@type layer\endcsname}
\newcommand*{\fca@endlabelslayer}{\csname fca@end\fca@label@type layer\endcsname}
\newcommand*{\fca@labelconceptslayer}{\csname fca@\fca@label@type conceptlayer\endcsname}
\newcommand*{\fca@endlabelconceptslayer}{\csname fca@end\fca@label@type conceptlayer\endcsna
```

`\fcaNewLayer` `{`⟨*fca entity*⟩`}{`⟨*layer*⟩`}`

This macro is similar to `\fcaLayer`, but it creates the PGF layer ⟨*layer*⟩ before assinging it to the macros.

```
\newcommand*{\fcaNewLayer}[2]{%
  \pgfdeclarelayer{#2}%
  \fcaLayer{#1}{#2}%
}
```

`\fcaDeclareLayers` creates a standard set of layers. The layers are assigned and used in the following way:

nodes main

nodenames fca node numbers

attributes fca attributes

objects fca objects

attributesconcept fca attribute concepts

objectsconcept fca object concepts

edges fca edges

connectors fca connectors

fca above nodes

fca below nodes

```
\newcommand*{\fcaDeclareLayers}{%
  \fcaLayer{nodes}{main}
  \fcaNewLayer{nodenames}{\pgfkeysvalueof{/fca/namespace} node numbers}
  \fcaNewLayer{attributes}{\pgfkeysvalueof{/fca/namespace} attributes}
  \fcaNewLayer{objects}{\pgfkeysvalueof{/fca/namespace} objects}
  \fcaNewLayer{attributesconcept}{%
    \pgfkeysvalueof{/fca/namespace} attribute concepts}
  \fcaNewLayer{objectsconcept}{%
    \pgfkeysvalueof{/fca/namespace} object concepts}
  \fcaNewLayer{edges}{\pgfkeysvalueof{/fca/namespace} edges}
  \fcaNewLayer{connectors}{\pgfkeysvalueof{/fca/namespace} connectors}
  \pgfdeclarelayer{\pgfkeysvalueof{/fca/namespace} above nodes}
  \pgfdeclarelayer{\pgfkeysvalueof{/fca/namespace} below nodes}
}
```

`\fcaSetLayers` defines the order of the layers used in a diagram. The order can changed using `\pgfsetlayers`, if necessary. Note: All used layers should occur in this list. Otherwise they will be ignored by PGF.

```
\newcommand*{\fcaSetLayers}{%
  \pgfsetlayers{%
    \pgfkeysvalueof{/fca/namespace} edges,%
    \pgfkeysvalueof{/fca/namespace} connectors,%
    \pgfkeysvalueof{/fca/namespace} below nodes,%
    main,%
    \pgfkeysvalueof{/fca/namespace} above nodes,%
    \pgfkeysvalueof{/fca/namespace} node numbers,%
    \pgfkeysvalueof{/fca/namespace} attribute concepts,%
    \pgfkeysvalueof{/fca/namespace} object concepts,%
    \pgfkeysvalueof{/fca/namespace} attributes,%
    \pgfkeysvalueof{/fca/namespace} objects%
  }%
}
```

`\fcaNoLayers` assigns all layers to the current layer. This means the graphic objects are drawn one on top of the other in the order they appear in the source code. This is the default behaviour of the `fca` packages.

```
\newcommand*\fcaNoLayers{%
```

```
  \fcaLayer{nodes}{main}%
  \fcaLayer{nodenames}{main}%
  \fcaLayer{attributes}{main}%
  \fcaLayer{objects}{main}%
  \fcaLayer{attributesconcept}{main}%
  \fcaLayer{objectsconcept}{main}%
  \fcaLayer{edges}{main}%
  \fcaLayer{connectors}{main}%
}
\fcaNoLayers
```

instructs a diagram to collect the different parts of a diagram into layers. By default later parts are drawn above prior parts. So edges are drawn above nodes. In simple lattices this is not a problem, but for more complex lattice diagrams it may not be completely possible to draw all edges between nodes and labels.

```
\newcommand*{\fcaLayers}{%
  \fcaDeclareLayers
  \fcaSetLayers
}
```

Now, two tests (one for PGF and Ti*k*Z):

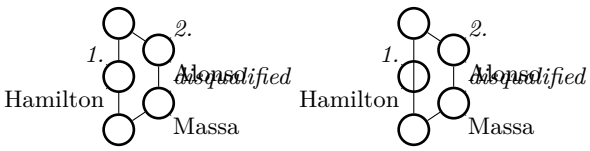
Example 33: A diagram with layers (left) and flat (right)

Code:

```
\begin{diagram}
  \fcaLayers
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(20,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge(1)(5)
  \Edge(2)(4)
  \Edge(4)(5)
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Hamilton}
  \rightObjbox(2){Massa}
  \rightObjbox(4){Alonso}
  \end{diagram}
\begin{diagram}
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(20,30)
  \Node(4)(35,40)
```

```
\Node(5)(20,50)
\Edge(1)(2)
\Edge(1)(5)
\Edge(2)(4)
\Edge(4)(5)
\leftAttbox(3){1.}
\rightAttbox(2){disqualified}
\rightAttbox(4){2.}
\leftObjbox(3){Hamilton}
\rightObjbox(2){Massa}
\rightObjbox(4){Alonso}
\end{diagram}
```

Result:




```
% We allow to select the label type and positioning independently.
% This function executes the right positioning settings.
\newcommand\fca@dolabelposition{%
  \pgfkeysgetvalue{/fca/label/type}{\@tempa}%
  \pgfkeysgetvalue{/fca/label/position}{\@tempb}%
  \ifx\pgfnovalue\@tempa
  \else
    \ifx\pgfnovalue\@tempb
    \else
      \pgfkeysalso{/fca/label/\@tempa{} position=\@tempb}%
    \fi
  \fi
}

\newcommand*\fcaNewLabelType{%
  \@ifstar{\fca@NewLabelType}{\fca@NewL@belType}%\fistar
}
\newcommand*\fca@NewLabelType[1]{%
  \pgfqkeys{/fca/label}{%
    type/#1/.style={type=#1},
    #1 position/.is choice,
  }%
}
\newcommand*\fca@NewL@belType[1]{%
  \fca@NewLabelType{#1}%
  \pgfqkeys{/fca/label}{%
    #1/.style={type=#1}%
  }%
}
\newcommand*\fcaNewLabelPosition{%
  \@ifstar{\fca@NewLabelPosition}{\fca@NewL@belPosition}%\fistar
}
\newcommand*\fca@NewLabelPosition[1]{%
```

```
  \pgfqkeys{/fca/label}{%
    position/#1/.style={position=#1},
  }%
}
\newcommand*\fca@NewL@belPosition[1]{%
  \fca@NewLabelPosition{#1}%
  \pgfqkeys{/fca/label}{%
    #1/.style={position=#1}%
  }%
}

```

Changing the default values

10.1 Generate some parameters which may be shared with Ti*k*Z

```
\newcommand*\fca@generate@tikz@parameter[3]{%
  \pgfqkeys{/fca/#1}{.initial={#2},.value required}
}
\newcommand*\fca@generate@tikz@parameters{%
  \@for \fca@tmp:=%
  {font}{}{\tikz@textfont},%
  {node font}{}{\tikz@node@textfont},%
  {text opacity}{}{\tikz@textopacity},%
  {text width}{\pgfkeysnovalue}{\tikz@text@width},%
  {text height}{}{\tikz@text@height},
  {text depth}{}{\tikz@text@depth},
  {text action}{}{\tikz@text@action},
  {options}{}{\tikz@options},%\newcommand*{\fca@node@}{white}%
  {anchor}{}{\tikz@anchor},
  {shape}{}{\tikz@shape}
  \do {
    \expandafter\fca@generate@tikz@parameter\fca@tmp
  }
}
\fca@generate@tikz@parameters
```

10.2 set up what we need from tikz

```
\RequirePackage{fca}
\let\fca@tikz@without@library\relax
\def\fca@generate@tikz@parameter#1#2#3{%
  \pgfkeysdef{/fca/#1}{\pgfkeys{/tikz/#1=##1}}
  \pgfkeyssetvalue{/fca/#1}{#3}%
}
```

10.3 Backports and bugfixes for Ti*k*Z

```
\@ifpackagelater{tikz}{2020/09/28}{}{%
  \PackageWarning{fca}{Your TikZ version does not fully support fca
    name spaces.^^J
```

```
    I'm trying to fix that..., trying to apply^^J
    patch 88951be592b558b94b14a97aaffe9df6c1ce61ee from TikZ}
  \def\tikz@calc@anchor#1.#2\tikz@stop{%
    % Check if a shape with name prefix exists, otherwise try the global name
    % without prefix.
    \ifcsname pgf@sh@ns@\tikz@pp@name{#1}\endcsname%
      \pgfpointanchor{\tikz@pp@name{#1}}{#2}%
    \else
      \pgfpointanchor{#1}{#2}%
    \fi
  }%
\def\tikz@subpicture@handle@#1{
  \pgfkeys{/tikz/pics/.cd,#1}%
  \tikz@node@transformations%
  \let\tikz@transform=\relax%
  \let\tikz@picmode\tikz@mode%
  \tikzset{name prefix ../.style/.expanded={/tikz/name prefix=\pgfkeysvalueof{/tikz/name pre
  \ifx\tikz@fig@name\pgfutil@empty\else%
    \tikzset{name prefix/.expanded=\tikz@fig@name}%
  \fi%
  \pgfkeysvalueof{/tikz/pics/setup code}%
  \pgfkeysgetvalue{/tikz/pics/code}{\tikz@pic@code}
  \ifx\tikz@pic@code\pgfutil@empty\else%
  \setbox\tikz@whichbox=\hbox\bgroup%
    \unhbox\tikz@whichbox%
      \hbox\bgroup
      \bgroup%
      \pgfinterruptpath%
      \pgfscope%
      \tikz@options%
      \setbox\tikz@figbox=\box\pgfutil@voidb@x%
      \setbox\tikz@figbox@bg=\box\pgfutil@voidb@x%
      \tikz@atbegin@scope%
      \scope[every pic/.try]%
      \tikz@pic@code%
      \endscope%
      \tikz@atend@scope%
      \endpgfscope%
      \endpgfinterruptpath%
      \egroup
      \egroup%
      \egroup%
    \fi%
    \pgfkeysgetvalue{/tikz/pics/foreground code}{\tikz@pic@code}
    \ifx\tikz@pic@code\pgfutil@empty\else%
      \setbox\tikz@figbox=\hbox\bgroup%
      \unhbox\tikz@figbox%
      \hbox\bgroup
      \bgroup%
      \pgfinterruptpath%
      \pgfscope%
      \tikz@options%
      \setbox\tikz@figbox=\box\pgfutil@voidb@x%
      \setbox\tikz@figbox@bg=\box\pgfutil@voidb@x%
```

```
      \tikz@atbegin@scope%
      \scope[every front pic/.try]%
      \tikz@pic@code%
      \endscope%
      \tikz@atend@scope%
      \endpgfscope%
      \endpgfinterruptpath%
      \egroup
      \egroup%
      \egroup%
    \fi%
    \pgfkeysgetvalue{/tikz/pics/background code}{\tikz@pic@code}
    \ifx\tikz@pic@code\pgfutil@empty\else%
      \setbox\tikz@figbox@bg=\hbox\bgroup%
      \unhbox\tikz@figbox@bg%
      \hbox\bgroup
      \bgroup%
      \pgfinterruptpath%
      \pgfscope%
      \tikz@options%
      \setbox\tikz@figbox=\box\pgfutil@voidb@x%
      \setbox\tikz@figbox@bg=\box\pgfutil@voidb@x%
      \tikz@atbegin@scope%
      \scope[every behind pic/.try]%
      \tikz@pic@code%
      \endscope%
      \tikz@atend@scope%
      \endpgfscope%
      \endpgfinterruptpath%
      \egroup
      \egroup%
      \egroup%
    \fi%
    \tikz@node@finish%
  }%
\def\tikz@parse@node#1(#2){%
  \pgfutil@in@.{#2}% Ok, flag this
  \ifpgfutil@in@
    \tikz@calc@anchor#2\tikz@stop%
  \else%
    \tikz@calc@anchor#2.center\tikz@stop% to be on the save side, in
                                % case iftikz@shapeborder is ignored...
    \ifcsname pgf@sh@ns@\tikz@pp@name{#2}\endcsname
      \expandafter\ifx\csname pgf@sh@ns@\tikz@pp@name{#2}\endcsname\tikz@coordinate@text%
      \else
        \tikz@shapebordertrue%
        \def\tikz@shapeborder@name{\tikz@pp@name{#2}}%
      \fi%
    \else\ifcsname pgf@sh@ns@#2\endcsname
      \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\tikz@coordinate@text%
      \else
        \tikz@shapebordertrue%
        \def\tikz@shapeborder@name{#2}%
      \fi%
```

```
    \fi\fi
  \fi%
  \edef\tikz@marshal{\noexpand#1{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}}%
  \tikz@marshal%
}%
 }%
```

Options that are directly forwarded to Ti*k*Z If they are not documented somewhere else, their implementation needs `fca.sty` fröm the Ti*k*Zlibrary `fca`.

Opt `shape` — **`shape`:** shape of the node

Opt `text ragged` — **`text ragged`:** node text is ragged right with hyphenation

Opt `text badly ragged` — **`text badly ragged`:** node text is ragged right nearly without hyphenation (normal LaTeX mode)

Opt `text ragged left` — **`text ragged left`:** node text is ragged left

Opt `text badly ragged left` — **`text badly ragged left`:** node text is ragged left nearly without hyphenation (normal LaTeX mode)

Opt `text justified` — **`text justified`:** The text is spread to fit the border on both sides of the text box (typically with hyphanation)

Opt `text centered` — **`text centered`:** The node text is horizontally centered

Opt `text badly centered` — **`text badly centered`:** node text is centered horizontally nearly without hyphenation (normal LaTeX mode)

Opt `even odd rule` — **`even odd rule`:** self overlapping paths are filled so that every border is an outside border (an even number of borders in each direction means outside and an odd number means inside.

Opt `nonzero rule` — **`nonzero rule`:** The default rule of Ti*k*Z for filling paths. The algorithm is described in the Ti*k*Z documentation (see `\exdoc pgfmanual.pdf`).

Opt `fill opacity` — **`fill opacity`:** The opacity of the filled copy of the path or node

Opt `opacity` — **`opacity`:** General option of the opacity

Opt `blend mode` — **`blend mode`:** This option defines, how thransparency changes the colors of semi-transparent objects. See the Ti*k*Z documentation for further details.

Opt `color` — **`color`:** Sets stroke and fill color (border and interior) to the same color

Opt `rotate` — **`rotate`:** rotate some path

Opt `solid` — **`solid`:** draw lines solid

Opt `dotted` — **`dotted`:** draw lines dotted

Opt `loosely dotted` **loosely dotted:** draw lines dotted with more space between the dots

Opt `densely dotted` **densely dotted:** draw lines dotted with less space between the dots

```
\fca@generate@tikz@parameters
\@for \@tempa:=%
shape,
text ragged,%
text badly ragged,%
text ragged left,%
text badly ragged left,%
text justified,%
text centered,%
text badly centered,%
align,%
even odd rule,%
nonzero rule,%
fill opacity,%
opacity,%
blend mode,%
color,%
rotate,%
solid,%
dotted,%
loosely dotted,%
densely dotted,%
line width,%
x,%
y%
\do {
  \edef\@tempb{%
    \noexpand\pgfkeysdef{/fca/\@tempa}{\noexpand\pgfqkeys{/tikz}{\@tempa={##1}}}%
  }%
  \@tempb
}
\pgfkeysdef{/fca/options}{\tikz@addoption{#1}}
\pgfkeysdef{/fca/stroke}{%
  \pgfqkeys{/tikz}{draw={#1}}%
  \fca@testoption{#1}{%
    \fca@append\fca@usepath{stroke,}%
  }{}{%
    \fca@append\fca@usepath{stroke,}%
  }%
}
\pgfkeysdef{/fca/fill}{%
  \pgfqkeys{/tikz}{fill={#1}}%
  \fca@testoption{#1}{%
    \fca@append\fca@usepath{fill,}%
  }{}{%
    \fca@append\fca@usepath{fill,}%
  }%
}
\def\fca@options{\tikz@options}%
\def\fca@defaultoptions{\tikz@options}%
```

```
\def\fca@transform{\tikz@transform}%
\@ifundefined{tikz@transform}{\let\tikz@transform\empty}{}
\let\fca@ifintikz\@secondoftwo
\tikzaddtikzonlycommandshortcutlet\fca@ifintikz\@firstoftwo
\pgfqkeys{/fca/label}{%
  attributes left/.append style={
    /tikz/.cd,
    align=flush right,
    /fca/.cd
  },
  attributes center/.append style={%
    /tikz/.cd,
    align=flush center,
    /fca/.cd%
  },
  attributes right/.append style={
    /tikz/.cd,
    align=flush left,
    /fca/.cd
  },
  objects left/.append style={
    /tikz/.cd,
    align=flush right,
    /fca/.cd
  },
  objects center/.append style={
    /tikz/.cd,
    align=flush center,
    /fca/.cd
  },
  objects right/.append style={
    /tikz/.cd,
    align=flush left,
    /fca/.cd
  }
}
```

```
\let\fca@notikz@parse@paren@vector\fca@parse@paren@vector
\newcommand\fca@tikz@parse@paren@vector[3]{%
  \def\tikz@coordinate@caller{%
    \fca@notikz@parse@paren@vector{#1}{#2}{#3}%
  }%
  \tikz@scan@one@point \tikz@@coordinate@at@math
}

\newcommand\fca@tikz@oldnode[4]{%
  \begin{pgfscope}%
    \fca@nodeslayer
    \let\tikz@text@width\pgfutil@empty
    \path (#3,#4) node[
      anchor=center,
      shape=circle,
      line width=\fca@node@thickness,
      /fca/.cd,
```

```
      /fca/every node/.try,
      /fca/every concept/.try,
      #1]
      (#2) {} ;%
      \fca@endnodeslayer
    \iffca@draft
      \ifthenelse{\boolean{fca@CircledNumbers}}{%
        \fca@nodenameslayer
        \path[overlay] (#2)
        node[
        shape=circle,
        anchor=center,
        /fca/.cd,
        /fca/every node/.try,
        /fca/every concept/.try,
        #1
        /fca/.cd,
        /fca/node number,
        draw=none]%
        (#2 number)
        {#2};%
        \fca@endnodenameslayer
      }{}%
    \fi
  \end{pgfscope}%
  \ignorespaces
}

\newcommand{\fca@tikz@oldedge}[3]{%
  \fca@edgeslayer
  \path[draw,/fca/.cd,every edge/.try,#1] (#2) edge (#3);%
  \fca@endedgeslayer
  \ignorespaces
}

\newcommand\fca@tikz@labelBox@label[1]{%
  \typeout{fca@tikz@labelBox@label: |attributes\space \fca@label@position|}%
  \fca@labelslayer
  \path[/fca/.cd,every node/.try,every label/.try,#1]
  (\fca@temp@node@name.\fca@label@at)
  ++(\fca@label@shift@x,\fca@label@shift@y)
  node[/fca/.cd,every node/.try,/tikz/.cd,shape=rectangle,/fca/.cd,every label/.try,#1] (%
  \pgfkeysvalueof{/fca/label/name prefix}%
  \pgfkeysvalueof{/fca/node}%
  \pgfkeysvalueof{/fca/label/name suffix}%
  ){%
    \pgfkeysvalueof{/fca/node contents}%
  };%
  \fca@endlabelslayer
}

\newcommand\fca@tikz@labelBox@connector[1]{%
  \typeout{fca@tikz@labelBox@connector}%
  \begin{scope}
```

```
    \fca@connectorslayer
    \path[draw,/fca/.cd,#1,every label edge/.try,/tikz/.cd]
    (\pgfkeysvalueof{/fca/node}) --
    (%
    \pgfkeysvalueof{/fca/label/name prefix}%
    \pgfkeysvalueof{/fca/node}%
    \pgfkeysvalueof{/fca/label/name suffix}%
    .\pgfkeysvalueof{/fca/anchor}) ;
    \fca@endconnectorslayer
  \end{scope}%
}

\newcommand\fca@tikz@labelBox@concept[1]{%
  \typeout{fca@tikz@labelBox@concept}%
  \fca@labelconceptslayer
  % \edef\fca@tempa{
  \path
  (\fca@temp@node@name.\fca@label@at)%
  node[draw,
  /fca/.cd,
  shape=coordinate,
  #1,
  every label concept/.try,
  label concept/.try]
  (\pgfkeysvalueof{/fca/label/name prefix}%
  \pgfkeysvalueof{/fca/node}%
  \pgfkeysvalueof{/fca/label/name suffix}%
  ){};
  % \pgfsetstrokecolor{\fca@node@color}%
  % \pgfnode{}{\pgfkeysvalueof{/fca/anchor}}{}{}%
  \fca@endlabelconceptslayer
}

\newcommand\fca@tikz@labelBox[1]{%
  \begin{scope}%
    \fcaset{%
      draw/.forward to=/tikz/draw,
      every node/.try,every label/.try,#1}%
    \edef\@tempa{\pgfkeysvalueof{/fca/pgfnode}}%
    \xdef\fca@temp@node@name{%
      \expandafter\fca@remove@anchor\expandafter{\@tempa}%
    }%
    \fca@tikz@labelBox@label{#1}%
    \iffca@connectors
      \fca@tikz@labelBox@connector{#1}%
    \fi
    \begin{scope}
      \fcaset{shape=coordinate,#1}%
      \fcaset{%
        every label concept/.try}%
      \fcaset{%
        label concept/.try
      }%
      \def\@tempa{coordinate}%
```

```
      \ifx\@tempa\tikz@shape
      \else
        \fca@tikz@labelBox@concept{#1}%
      \fi
    \end{scope}
  \end{scope}%
  \typeout{end fca@tikz@labelBox}%
  \ignorespaces
}
\newcommand\fca@tikz@startdiagram[1][]{%
  \fca@notikz@startdiagram%
  \fca@tikz@diagram@
}
\newcommand\fca@tikz@Defaults{
  \fca@notikz@Defaults
  \pgfkeysgetvalue{/tikz/name prefix}{\fca@tikz@origprefix}%
  \pgfkeysgetvalue{/fca/name prefix}{\@tempb}%
  \let\@tempa\fca@tikz@origprefix
  \fca@concat\@tempa\@tempb%
  \pgfqkeys{/tikz}{%
    name prefix/.expand once=\@tempa
  }%
  \pgfkeysgetvalue{/fca/name suffix}{\@tempa}%
  \pgfkeysgetvalue{/tikz/name suffix}{\@fca@tikz@origsuffix}%
  \fca@concat\@tempa\@fca@tikz@origsuffix%
  \pgfqkeys{/tikz}{%
    name suffix/.expand once=\@tempa
  }%
  \pgfkeyssetvalue{/fca/name prefix}{%
    \pgfkeysvalueof{/tikz/name prefix}%
  }%
  \pgfkeyssetvalue{/fca/name suffix}{%
    \pgfkeysvalueof{/tikz/name suffix}%
  }%
}
\tikzaddtikzonlycommandshortcutlet\fca@oldnode\fca@tikz@oldnode
\tikzaddtikzonlycommandshortcutlet\fca@oldedge\fca@tikz@oldedge
\tikzaddtikzonlycommandshortcutlet\fca@@labelBox\fca@tikz@labelBox
\tikzaddtikzonlycommandshortcutlet\fca@parse@paren@vector\fca@tikz@parse@paren@vector
\tikzaddtikzonlycommandshortcutlet\fca@Defaults\fca@tikz@Defaults
```

`\fcaset` `{`⟨*key value list*⟩`}` Sets the options for the following operations. Possible options are listed below. The beginning `/fca/` can be omitted as it is provieded by the macro `\fcaset`. `\fcaset{`⟨*key value list*⟩`}` is expanded to `\pgfqkeys/fca{`⟨*key value list*⟩`}`.

This macro is similar to `\pgfkeys`, except that it sets `/fca` as default path. This is a powerful macro that cannot be described here in full detail. The full documentation can be found in the PGF/TikZ Manual in `pgfmanual.pdf` of your PGF documentation.

```
\newcommand*\fcaset{\pgfqkeys{/fca}}
```

Options can be set globally outside

Opt `/fca/font` — **`/fca/font:`** font selection macros used inside nodes (forwarded to `/tikz/font` when loaded as TikZ library).

Opt `/fca/node font` — **`/fca/node font:`** The font selection macros that are used during node size calculation (forwarded to `/tikz/node font` when loaded as TikZ library).

Opt `/fca/text opacity` — **`/fca/text opacity:`** Opacity of the text in nodes (forwarded to `/tikz/text opacity` when loaded as TikZ library).

Opt `/fca/text width` — **`/fca/text width:`** Width of the text in nodes (allows multiline nodes e.g. multiline labels) (forwarded to `/tikz/text width` when loaded as TikZ library).

Opt `/fca/text height` — **`/fca/text height:`** height of the text box of PGF nodes (forwarded to `/tikz/text height` when loaded as TikZ library).

Opt `/fca/text depth` — **`/fca/text depth:`** depth of the text box of PGF nodes (forwarded to `/tikz/text depth` when loaded as TikZ library).

Opt `/fca/text action` — **`/fca/text action:` Undocumented extension to Ti*k*Z** Stores the alignment setup macros Ti*k*Z nodes.

Opt `/fca/options` — **`/fca/options:` Undocumented extension to Ti*k*Z** direct access to `\tikz@options`.

Opt `/fca/anchor` — **`/fca/anchor:`** specifies the anchor of a node to be used for placement (forwarded to `/tikz/anchor` when loaded as TikZ library).

Opt `/fca/shape` — **`/fca/shape:`** the shape of a node (concept or label) (forwarded to `/tikz/shape` when loaded as TikZ library).

Opt `/fca/connector` — **`/fca/connector:`** Draw a connector line between the node and the label. Values are `true` and `false`.

Opt `/fca/namespace` — **`/fca/namespace:`** Namespace used in name prefixes and layer names, Default: fca, **value required**

Opt `/fca/name prefix` — **`/fca/name prefix:`** Pattern to be added before node names. This can be interesting when a diagram is inside a `tikzpicture` or `pgfpicture` environment in order to address the diagram nodes. Default: `\pgfkeysvalueof{/fca/namespace}\space node\space`, **value required**

Opt `/fca/name suffix` — **`/fca/name suffix:`** Default: empty, **value required**

Opt `/fca/every node/.style` — **`/fca/every node/.style:`** Style to be executed at the beginning of every node (both labels as well as concepts). Default: empty

Opt `/fca/every concept/.style` — **`/fca/every concept/.style:`** Style to be executed at every concept node (when calling `\Node`). Default:

```
{%
  radius = 2mm,
  fill=white,
  draw=black
},
```

Opt /fca/every attributes/.style

/fca/every attributes/.style: Style that is executed whenever an attribute label is typeset. Default:

```
{%
  font=\small\baselineskip1em\rmfamily\itshape
}%
```

Opt /fca/every objects/.style

/fca/every objects/.style: Style that is executed whenever an object label is placed in the diagram. Defaul:

```
{%
  font=\small\baselineskip1em\rmfamily\upshape
}
```

Opt /fca/every edge/.style

/fca/every edge/.style: Style that is executed whenever an edge betwee two concepts is drawn using `\Edge`. Default: empty.

Opt /fca/every label edge/.style

/fca/every label edge/.style: Style that is drawn whenever a connector between a concept and one of its labels is drawn. Default:

```
{
  dotted, draw=black
}
```

Opt /fca/font

/fca/font: Font to be used inside labels. This is the actually set font. Default: `\small\baselineskip1em\rmfamily`

⚠ the font is applied after the node size has been calculated. The font size for the size calculations must be set using the option `node font`.

Opt /fca/shape

/fca/shape: Shape of the nodes. Default: circle,

Opt /fca/minimum width

/fca/minimum width: Set the minimum width of a node or label. This is an alias of `/pgf/minimum width`.

Opt /fca/minimum height

/fca/minimum height: set the minimum height of a node or label. This is an alias of /pgf/minimum height.

Opt /fca/minimum size

/fca/minimum size: Set both `minimum width` and `minimum height`. This is an alias of /pgf/minimum size.

Opt `/fca/inner xsep` **`/fca/inner xsep:`** Minimum horizontal distance between node content and node border (same for labels). This is an alias of /pgf/inner xsep.

Opt `/fca/inner ysep` **`/fca/inner ysep:`** Minimum vertical distance between node content and node border (same for labels). This is an alias of /pgf/inner ysep.

Opt `/fca/inner sep` **`/fca/inner sep:`** Sets both `inner xsep` and `inner ysep`. This is an alias of /pgf/inner sep.

Opt `/fca/outer xsep` **`/fca/outer xsep:`** Minimum horizontal distance between node border and the next elements. This is an alias of /pgf/outer xsep.

Opt `/fca/outer ysep` **`/fca/outer ysep:`** Minimum vertical distance between node border and the next elements. This is an alias of /pgf/outer ysep.

Opt `/fca/outer sep` **`/fca/outer sep:`** Sets both `inner xsep` and `inner ysep`. This is an alias of /pgf/outer sep.

Opt `/fca/radius` **`/fca/radius:`** Set the radius of circled nodes. If the unit is omitted the sum of the horizontal coordinate of the first two unit vectors is used.

Opt `/fca/anchor` **`/fca/anchor:`** Determines which anchor should be represented by the given coordinates. **value required**

Opt `/fca/node number/.style` **`/fca/node number/.style:`** Style of the node names (traditionally numbers) of the diagram when `\fcaNumbers` is used. Default:

```
{
  node font=\tiny,
  font=\tiny
}
```

Opt `/fca/color` **`/fca/color:`** Change the color of the things to be drawn

Opt `/fca/stroke` **`/fca/stroke:`** Draw the line in the current object. If a value is given it is interpreted to be the line colour.

Opt `/fca/draw` **`/fca/draw:`** This is an alias of `/fca/stroke`.

Opt `/fca/fill` **`/fca/fill:`** Fill the current object with the colour given as value or the default fill colour.

Opt `/fca/text` **`/fca/text:`** The colour that is used for texts in nodes. Default: `pgfstrokecolor`, that is the line colour. **value required**

Opt `/fca/label type` **`/fca/label type:`** This is an alias of `/fca/label/type`.

Opt `/fca/label position` **`/fca/label position:`** This is an alias of `/fca/label/position`.

Opt `/fca/pgfnode` **`/fca/pgfnode:`** Pattern used to construct node names (if Ti*k*Z is not loaded).

value required Default:

```
\pgfkeysvalueof{/fca/name prefix}\pgfkeysvalueof{/fca/node}\pgfkeysvalueof{/fca/name suff
```

Opt `/fca/node` — **`/fca/node:`** Name of the node where the label should be attached to. Default: , **value required**

Opt `/fca/node contents` — **`/fca/node contents:`** Contents of the new node to be drawn. This is mainly used for labels. Default: , **value required**

Opt `/fca/shift/x` — **`/fca/shift/x:` value required**

Opt `/fca/shift/y` — **`/fca/shift/y:` value required**

Opt `/fca/shift={(⟨x,y⟩)}` — **`/fca/shift={(⟨x,y⟩)}:`** Offset values for shifting labels from their default position. Usually the offset is added to the coordinates where the anchor would be. Prior to version 3.0 `fca.sty` used distences to shift the nodes. Both Schemes are valid and have their benefits. The current implementation follows the scheme of Ti*k*Z.

⚠ changes in 3.0

For compatibility reasons the sign of the shift parameter is changed when the old syntax for labels is used.

Opt `/fca/shift/x/signv2.1` — **`/fca/shift/x/signv2.1:` value required**

Opt `/fca/shift/y/signv2.1` — **`/fca/shift/y/signv2.1:` value required** These parameters define how the distance values in the old syntax should be interpreted by LaTeX. You should change this parameter only, if you really know what you are doing. Otherwise you could easily mess up the diagram.

Opt `/fca/rotate` — **`/fca/rotate:`** rotate the current object

Opt `/fca/solid` — **`/fca/solid:`** Draw lines as solid lines

Opt `/fca/dotted` — **`/fca/dotted:`** Draw lines as dotted lines

Opt `/fca/loosely dotted` — **`/fca/loosely dotted:`** Draw lines as dotted lines with more space between the dots

Opt `/fca/densely dotted` — **`/fca/densely dotted:`** Draw lines as dotted lines with less space between the dots

Opt `/fca/every label/.style` — **`/fca/every label/.style:`** A style that describes how object and attribute labels should be decorated. Default:

```
{
  text=pgfstrokecolor,
  /pgf/outer sep=0pt,
  /pgf/inner sep=0pt,
}
```

Opt `/fca/node/line width` **`/fca/node/line width:`** line width of the nodes **value required**

Opt `/fca/node/radius` **`/fca/node/radius:`** Node radius

Opt `/fca/node/layer` **`/fca/node/layer:`** Layer where the node shall be put in.

Opt `/fca/node/numbers` **`/fca/node/numbers:`** ⟨*true*⟩ or ⟨*false*⟩, defaults to ⟨*true*⟩, if given without value. Draw PGF node names of the concepts in the diagram.

Opt `/fca/node/number prefix` **`/fca/node/number prefix: value required`** Prefix to be added to the node name for the name of the node that contains the node name

Opt `/fca/node/number suffix` **`/fca/node/number suffix: value required`** Prefix to be added to the node name for the name of the node that contains the node name The printed node name is a node itself. And every node in PGF must have a unique name. This is constructed by appending this prefix to the already existing prefix and prepending this suffix to the already existing suffix.

⚠ Confused?

⚠ TODO!

Or are prefix and suffix simply replaced?

Opt `/fca/label/name prefix` **`/fca/label/name prefix:`** Default: `\pgfkeysvalueof{/fca/namespace}\space node\space`, **value required**

Opt `/fca/label/name suffix` **`/fca/label/name suffix:`** Default: `\space\fca@label@type\space\fca@label@position`, **value required** Prefix and suffix to be added to the node name for the name of the label

⚠ TODO!

Or are prefix and suffix simply replaced?

Opt `/fca/label/fill` **`/fca/label/fill: value required`**, fill the label with the given colour,

Opt `/fca/label/text width` **`/fca/label/text width:`** , **value required** Width of the text of the node. This option enables multiline labels.

Opt `/fca/label/node font` **`/fca/label/node font: value required`**, Font macros to be applied to the label content before the size of the node is calculated.

Opt `/fca/label/edge width` **`/fca/label/edge width: value required`**, thickness of the edge of the label node (typically the border size)

Opt `/fca/label/at` **`/fca/label/at: value required`** Information for anchoring labels on nodes look up what is really done

⚠ TODO!!!

Opt `/fca/label/type` **`/fca/label/type:`** Defines the type of the given label. Every subobject of `/fca/label/type` can be chose as label type. The predefined options are `attributes` and `objects`.

New types can be defined by adding it to this path. In that case also a styles named `/fca/label/`⟨*type*⟩ ⟨*position*⟩ should be defined that do the actual formatting of the new label type. These types are immediately executed whenever a label type or a label position is changed (see below).

Opt `/fca/label/type/attributes/.style` **`/fca/label/type/attributes/.style:`** Makes the label type `attributes` available and defines styles for all attribute labels.

Opt /fca/label/type/objects/.style **/fca/label/type/objects/.style:** Makes the label type `objects` available and defines styles for all object labels.

Opt /fca/label/position **/fca/label/position:** Defines the positon of the given label. Every subobject of `/fca/label/positon` can be chose as position. The predefined options are `left`, `center` and `right`.

Opt /fca/label/position/left/.style **/fca/label/position/left/.style:** Makes the label position `left` available and defines styles for all left labels.

Opt /fca/label/position/center/.style **/fca/label/position/center/.style:** Makes the label position `center` available and defines styles for all centered labels.

Opt /fca/label/position/right/.style **/fca/label/position/right/.style:** Makes the label position `right` available and defines styles for all right labels.

Opt /fca/label/attributes left/.style **/fca/label/attributes left/.style:** Describes how attribute labels on the left hand side of the concept should be formatted. By default this affects placement and text alignment.

Opt /fca/label/attributes center.style **/fca/label/attributes center.style:** Describes how attribute labels that are horizontally centered to the concept should be formatted. By default this affects placement and text alignment.

Opt /fca/label/attributes left/.style **/fca/label/attributes left/.style:** Describes how attribute labels on the right hand side of the concept should be formatted. By default this affects placement and text alignment.

Opt /fca/label/objects left/.style **/fca/label/objects left/.style:** Describes how object labels on the left hand side of the concept should be formatted. By default this affects placement and text alignment.

Opt /fca/label/objects center.style **/fca/label/objects center.style:** Describes how object labels that are horizontally centered to the concept should be formatted. By default this affects placement and text alignment.

Opt /fca/label/objects left/.style **/fca/label/objects left/.style:** Describes how object labels on the right hand side of the concept should be formatted. By default this affects placement and text alignment.

```

\newcommand\fca@append[2]{%
  \expandafter\def\expandafter#1\expandafter{#1#2}%
}
\newcommand\fca@concat[2]{%
  \expandafter\fca@append\expandafter#1\expandafter{#2}%
}

\newcommand\fca@node@fill{white}

\fcaset{%
  connector/.is if=fca@connectors,
```

```
namespace/.initial=fca,
namespace/.value required,
name prefix/.initial=\pgfkeysvalueof{/fca/namespace}\space node\space,
name prefiv/.value required,
name suffix/.initial={},
name suffix/.value required,
every node/.style = {},
every concept/.style = {
  radius = 2mm,
  fill=\fca@node@fill,
  draw=black
},
every attributes/.style = {
  font=\AttributeLabelStyle
},%\newcommand*{\fca@node@fill}{white}%%\newcommand*{\fca@node@fill}{white}%
every objects/.style = {
  font=\ObjectLabelStyle
},
every edge/.style = {},
every label edge/.style = { dotted, draw=black },
font=\small\baselineskip1em\rmfamily,
shape/.initial=circle,
minimum width/.forward to=/pgf/minimum width,
minimum height/.forward to=/pgf/minimum height,
minimum size/.forward to=/pgf/minimum size,
inner xsep/.forward to=/pgf/inner xsep,
inner ysep/.forward to=/pgf/inner ysep,
inner sep/.forward to=/pgf/inner sep,
outer xsep/.forward to=/pgf/outer xsep,
outer ysep/.forward to=/pgf/outer ysep,
outer sep/.forward to=/pgf/outer sep,
radius/.code={
  \@tempdimb\dimexpr\pgf@xx+\pgf@yx\relax
  \fca@parselength\@tempdima{#1}{\@tempdimb}%
  \@tempdima=.707106781186547524400844362104 84\@tempdima
  \edef\@tempa{\noexpand\pgfkeysalso{/pgf/inner sep=\the\@tempdima}}%
  \@tempa
},
% anchor/.store in=\fca@label@anchor,
% anchor/.value required,
node number/.style = { node font=\tiny, font=\tiny },
color/.code={\fca@append\fca@options{\color{#1}}},
line width/.code={%
  \pgfmathsetlength\pgflinewidth{#1}%
  \fca@append\fca@options{\pgfsetlinewidth{#1}}%
},
stroke/.code={%
  \fca@testoption{#1}{%
    \fca@append\fca@options{%
      \pgfsetstrokecolor{#1}%
    }%
    \fca@append\fca@usepath{stroke,}%
  }{%
    \fca@append\fca@options{%
```

```
      \let\pgf@up@stroke\pgfutil@empty
    }%
  }{%
    \fca@append\fca@usepath{stroke,}%
  }%
},
draw/.forward to=/fca/stroke,
fill/.code={%
  \fca@testoption{#1}{%
    \fca@append\fca@options{%
      \pgfsetfillcolor{#1}%
    }%
    \fca@append\fca@usepath{fill,}%
  }{%
    \fca@append\fca@options{%
      \let\pgf@up@fill\pgfutil@empty
    }%
  }{%
    \fca@append\fca@usepath{fill,}%
  }%
},
text/.initial=pgfstrokecolor,
text/.value required,
label type/.forward to=/fca/label/type,
label position/.forward to=/fca/label/position,
pgfnode/.initial={%
  \pgfkeysvalueof{/fca/name prefix}%
  \pgfkeysvalueof{/fca/node}%
  \pgfkeysvalueof{/fca/name suffix}%
},
pgfnode/.value required,
node/.initial={},
node/.value required,
node contents/.initial={},
node contents/.value required,
shift/x/.store in=\fca@label@shift@x,
shift/x/.value required,
shift/y/.store in=\fca@label@shift@y,
shift/y/.value required,
shift/.value required,
shift/.style args={(#1,#2)}{%
  /fca/shift/x=#1,
  /fca/shift/y=#2
},%
shift/x/signv2.1/.store in=\fca@label@shift@x@sign,
shift/x/signv2.1/.value required,
shift/y/signv2.1/.store in=\fca@label@shift@y@sign,
shift/y/signv2.1/.value required,
rotate/.code={%
  \fca@append\fca@transform{%
    \pgftransformrotate{#1}%
  }%
},
solid/.code={%
```

```
    \fca@append\fca@options{%
      \pgfsetdash{}{0pt}%
    }%
  },%
  dotted/.code={%
    \fca@append\fca@options{%
      \pgfsetdash{{\pgflinewidth}{2pt}}{0pt}%
    }%
  },%
  loosely dotted/.code={%
    \fca@append\fca@options{%
      \pgfsetdash{{\pgflinewidth}{4pt}}{0pt}%
    }%
  },%
  densely dotted/.code={%
    \fca@append\fca@options{%
      \pgfsetdash{{\pgflinewidth}{4pt}}{0pt}%
    }%
  }%
}
\fcaset{%
  every label/.style = {
    text=pgfstrokecolor,
    /pgf/outer sep=0pt,
    /pgf/inner sep=0pt,
  }
}

\pgfqkeys{/fca/node}{%
  line width/.forward to=/fca/line width
  line width/.value required,
  radius/.style={
    /fca/radius=#1
  },
  color/.forward to=/fca/fill,
  layer/.code=\fcalayer{node},%
  numbers/.is if=fca@CircledNumbers,
  numbers/.default=true,
  number prefix/.store in=\fca@node@number@prefix,
  number prefix/.value required,
  number suffix/.store in=\fca@node@number@suffix,
  number suffix/.value required,
}

% Line thickness in standard \LaTeX{}:
% thin lines are .4pt
% thick lines are 0.8pt

% Dotted: \pgfsetdash{{\pgflinewidth}{2pt}}

\pgfqkeys{/fca/label}{%
  name prefix/.initial=\pgfkeysvalueof{/fca/namespace}\space node\space,
  name prefix/.value required,
  name suffix/.initial=\space\fca@label@type\space\fca@label@position,
```

```
name suffix/.value required
fill/.store in=\fca@node@fill,
fill/.value required,
text width/.store in=\fca@label@text@width,
text width/.value required,
node font/.initial={},
node font/.value required,
edge width/.store in=\fca@label@edge@width,
edge width/.value required,
at/.store in=\fca@label@at,
at/.value required,
type/.store in=\fca@label@type,
type/.is choice,
type/.append code={%
  \def\fca@label@type{#1}% the choice does not store the value
  \pgfqkeysalso{/fca}{every #1/.try}%
  \pgfqkeysalso{/fca/label}{\fca@label@type\space\fca@label@position}%
},
type/attributes/.style={},
type/objects/.style={},
position/.store in=\fca@label@position,
position/.is choice,
position/.append code={%
  \def\fca@label@position{#1}% the choice does not store the value
  \pgfqkeysalso{/fca/label}{\fca@label@type\space\fca@label@position}%
},
position/left/.style={},
position/right/.style={},
position/center/.style={},
%
attributes left/.style={
  at=north west,
  /fca/anchor=south east,
  /fca/shift={(-1pt,1pt)},
  /fca/shift/x/signv2.1=-,
  /fca/shift/y/signv2.1={},
  /fca/font/.append=\raggedleft
},
attributes center/.style={
  at=north,
  /fca/anchor=south,
  /fca/shift={(0,1pt)},
  /fca/shift/x/signv2.1={},
  /fca/shift/y/signv2.1={},
  /fca/font/.append=\centering
},
attributes right/.style={
  at=north east,
  /fca/anchor=south west,
  /fca/shift={(1pt,1pt)},
  /fca/shift/x/signv2.1={},
  /fca/shift/y/signv2.1={},
  /fca/font/.append=\raggedright
},
```

```
  objects left/.style={
    at=south west,
    /fca/anchor=north east,
    /fca/shift={(-1pt,-1pt)},
    /fca/shift/x/signv2.1=-,
    /fca/shift/y/signv2.1=-,
    /fca/font/.append=\raggedleft
  },
  objects center/.style={
    at=south,
    /fca/anchor=north,
    /fca/shift={(0,-1pt)},
    /fca/shift/x/signv2.1={},
    /fca/shift/y/signv2.1=-,
    /fca/font/.append=\centering
  },
  objects right/.style={
    at=south east,
    /fca/anchor=north west,
    /fca/shift={(1pt,-1pt)},
    /fca/shift/x/signv2.1={},
    /fca/shift/y/signv2.1=-,
    /fca/font/.append=\raggedright
  }
}

\newcommand*\fca@tikz@without@library{%
  \PackageWarning{fca}{FCA TikZ integration is not activated.^^J
    You should consider \tikzlibrary{fca} instead of
    \string\usepackage{fca}
  }%
}

\AtBeginDocument{%
  \@ifundefined{tikz@color}{}{\fca@tikz@without@library}
}


\newcommand*{\fcaNodeThickness}[1]{%
  \def\fca@node@thickness{#1}%
  \ignorespaces
}% %
\newcommand*{\fcaEdgeThickness}[1]{%
  \fcaset{every edge/.append style={%
      line width={#1}%
    }%
  }%
  \ignorespaces
}
\newcommand*{\fcaNodeColor}[1]{%
  \def\fca@node@fill{#1}%
  \ignorespaces
}%
\iffca@draft
```

```
  \newcommand*{\fcaNumbers}{%
    \setboolean{fca@CircledNumbers}{true}%
  }%
\else
  \newcommand*{\fcaNumbers}{%
    \setboolean{fca@CircledNumbers}{false}%
  }
\fi
\newcommand*{\fcaNoNumbers}{%
  \setboolean{fca@CircledNumbers}{false}%
}%
\newcommand*{\fcaCircleSize}[1]{%
  \fcaset{every concept/.append style={radius=(#1)*0.5\unitlength}}%
  \ignorespaces
}%
\iffca@compat@macros
  \def\fcaObjectLabelStyle{\fcaObjectLabelStyle}%
  \def\fcaAttributeLabelStyle{\AttributeLabelStyle}%
  \def\fcaLabelBoxWidth{\LabelBoxWidth}%
  \def\EdgeThickness{\fcaEdgeThickness}%
  \def\NodeThickness{\fcaNodeThickness}%
  \def\NodeColor{\fcaNodeColor}%
  \def\Numbers{\fcaNumbers}%
  \def\NoNumbers{\fcaNoNumbers}
  \def\CircleSize{\fcaCircleSize}%
  \def\NoDots{\fcaNoDots}
  \def\Dots{\fcaDots}
\fi

\newcounter{fca@minNode}%
\newcounter{fca@maxNode}%
\newcounter{fca@runNode}%
\newboolean{fca@CircledNumbers}%
\newcounter{fca@CircleDiameter}%
\newcounter{fca@AuxCounter}%
\newcounter{fca@BuxCounter}%
\def\fca@adjNode#1{%
  \ifthenelse{#1<\value{fca@minNode}}{\setcounter{fca@minNode}{#1}}{}%
  \ifthenelse{\value{fca@maxNode}<#1}{\setcounter{fca@maxNode}{#1}}{}}%
\newcommand\fca@typeset@node@content[1]{%
  \pgfkeysgetvalue{/fca/text width}\@tempa
  \edef\@tempb{\@tempa}%
  \expandafter\fca@testoption\expandafter{\@tempb}{%
    \def\fca@node@end@content{\end{minipage}}%
    \fca@parselength\@tempdima{\fca@tempa}\fca@xunitlength
    \begin{minipage}{\@tempdima}%
  }{%
    \let\fca@node@end@content\egroup
    \makebox\bgroup%
  }{%
    \let\fca@node@end@content\egroup
    \makebox\bgroup
  }%
  \pgfkeysvalueof{/fca/font}%
```

```
  \color{\pgfkeysvalueof{/fca/text}}%
  #1%
  \fca@node@end@content%
}
\newcommand\fca@node[1][]{%
  \@ifnextchar\bgroup{\fca@oldnode{#1}}{%
      \@ifnextchar({\fca@pictnode{#1}}{\fca@tikznode{#1}}%)
    }%
% \egroup
}
\newcommand\fca@pictnode[1]{%
  \fca@parse@parenlabel{\fca@pictn@de{#1}}%
}
\newcommand\fca@pictn@de[2]{%
  \fca@parse@paren@vector{%
    \edef\fca@tempa{{\the\@tempdima}{\the\@tempdimb}}%
    \def\fca@tempb{\fca@oldnode{#1}{#2}}%
    \expandafter\fca@tempb\fca@tempa
  }\@tempdima\@tempdimb
}
\newcommand\fca@oldnode[4]{%
  \begin{pgfscope}%
    \fcaset{shape=circle,
      line width=\fca@node@thickness,
      color/.forward to=/fca/fill}%
    \fcaset{every node/.try}%
    \fcaset{every concept/.try}%
    \fcaset{#1}%
    \fca@nodeslayer
    \pgfkeysvalueof{/fca/node font}%
    \fca@parselength\@tempdima{#3}\unitlength
    \fca@parselength\@tempdimb{#4}\unitlength
    \pgftransformshift{\pgfpoint{\@tempdima}{\@tempdimb}}%
    \fca@options
    % \pgfsetstrokecolor{\fca@node@color}%
    \pgfnode{\pgfkeysvalueof{/fca/shape}}{center}{}{%
      \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name suffix}%
    }{\expandafter\pgfusepath\expandafter{\fca@usepath}}%
    \fca@endnodeslayer
    \ifthenelse{\boolean{fca@CircledNumbers}}{%
      \begin{pgfscope}
        \fca@nodenameslayer
        \pgf@relevantforpicturesizefalse
        \def\fca@usepath{}%
        \fcaset{node number}%
        \fca@options \pgfnode{\pgfkeysvalueof{/fca/shape}}{center}{%
          \fca@typeset@node@content {%
            \centering \pgfkeysvalueof{/fca/font}%
            #2%
          }%
        }{%
          \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name
            suffix} number%
        }{\expandafter\pgfusepath\expandafter{\fca@usepath}}%
```

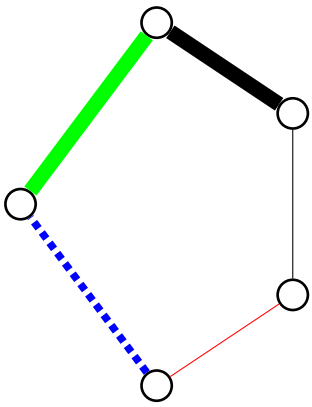
```
        \fca@endnodenameslayer
      \end{pgfscope}
    }{}%
  \end{pgfscope}%
  \ignorespaces
}
\newcommand{\fca@edge}[1][]{%
  \@ifnextchar\bgroup{\fca@oldedge{#1}}{\fca@newedge{#1}}%
% \egroup
}
\newcommand\fca@newedge[1]{%
  \fca@parse@parenlabel{\fca@new@dge{#1}}%
}
\newcommand\fca@new@dge[2]{%
  \fca@parse@parenlabel{\fca@oldedge{#1}{#2}}%
}
```

Example 34: Testing lines

Code:

```
{\unitlength 1.2mm
\definecolor{darkgreen}{rgb}{0.05,0.5,0.}
\begin{diagram}
 \Node(1)(20,10)
 \Node(2)(35,20)
 \Node(3)(5,30)
 \Node(4)(35,40)
 \Node(5)(20,50)
 {\color{red}\Edge(1)(2)}
 \Edge[draw=blue,dotted,line width=1mm](1)(3)
 \Edge(2)(4)
 \EdgeThickness{2mm}
 \Edge[draw=green](3)(5)
 \Edge(4)(5)
\end{diagram}}
```

Result:



```
\newcommand{\fca@oldedge}[3]{%
  \begin{pgfscope}%
    \fca@edgeslayer
    \fcaset{every edge/.try,#1}%
    \fca@options
    \fca@transform
    %\pgfsetlinewidth{\pgfkeysgetvalue{/fca/line width}}%
    \pgfpathmoveto{%
      \pgfpointshapeborder{%
        \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name suffix}%%
      }{%
        \pgfpointanchor{%
          \pgfkeysvalueof{/fca/name prefix}#3\pgfkeysvalueof{/fca/name suffix}%%
        }{center}%
      }%
    }%
    \pgfpathlineto{%
      \pgfpointshapeborder{%
        \pgfkeysvalueof{/fca/name prefix}#3\pgfkeysvalueof{/fca/name suffix}%%
      }{%
        \pgfpointanchor{%
        \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name suffix}%%
        }{center}%
      }%
    }%
    \pgfusepath{stroke}%
    \fca@endedgeslayer
  \end{pgfscope}%
  \ignorespaces
}
%
\def\fca@changeaValue#1#2#3{\setcounter{fca@AuxCounter}{#1}%
  \addtocounter{fca@AuxCounter}{#2}\def#3{\value{fca@AuxCounter}}}%
\def\fca@changebValue#1#2#3{\setcounter{fca@BuxCounter}{#1}%
  \addtocounter{fca@BuxCounter}{#2}\def#3{\value{fca@BuxCounter}}}%
\newcommand{\fcaNoDots}{\setboolean{fca@connectors}{false}}
\newcommand{\fcaDots}{\setboolean{fca@connectors}{true}}
```

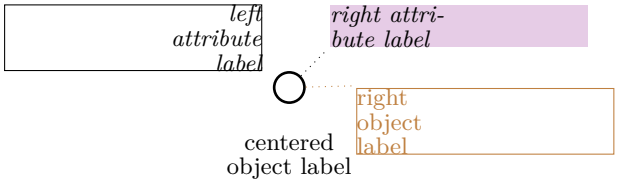
Example 35: Testing nodots

Code:

```
\begin{diagram}[text width=3.4cm]
  \Node(0)(20,10)
  \NoDots
  \leftAttbox[draw](0)(-5,1){left\\ attribute\\ label}
  \Dots
  \rightAttbox[fill=red!50!blue!20](0)(10,10){right
  attri-\\ bute label}
  \fcaNoDots\centerObjbox(0)(0,-10){centered\\ object label}
  \fcaDots
```

```
    \rightObjbox[draw=brown](0)(20,5){right\\ object\\ label}
  \end{diagram}
```

Result:




```
\def\fca@parse@parenlabel#1(#2){#1{#2}}
\def\fca@parse@paren@vector#1#2#3(#4,#5){%
  \fca@parse@brace@vector{#1}{#2}{#3}{#4}{#5}%
}
```

`\fca@parse@label` {⟨*true material*⟩}{⟨*else material*⟩}(⟨*label*⟩)

`\fca@parse@label` {⟨*true material*⟩}{⟨*else material*⟩}{⟨*label*⟩}

`\fca@parse@label` {⟨*true material*⟩}{⟨*else material*⟩} The argument ⟨*label*⟩ is parsed if it is given. In that case it is fed to ⟨*true path*⟩ as ordinary argument. If it is not given, the code in ⟨*else material*⟩ is executed.

```
\newcommand*\fca@parse@label[2]{%
  \@ifnextchar\bgroup{%\egroup
    #1%
  }{%
    \@ifnextchar(%)
    {%
      \fca@parse@parenlabel{#1}%
    }{%
      #2%
    }%
  }%
}
\newcommand*\fca@parse@vector[4]{%
  \@ifnextchar(%)
  {%
    \fca@parse@paren@vector{#1}{#3}{#4}%
  }{%
    \@ifnextchar\bgroup{%\egroup
      \fca@parse@brace@vector{#1}{#3}{#4}%
    }{#2}%
  }%
}
\newcommand*\fca@parse@brace@vector[5]{%
  \fca@parselength#2{#4}\fca@xunitlength
  \fca@parselength#3{#5}\fca@yunitlength
  #1%
}
```

`\fca@labelBox` [⟨*options*⟩]

`\fca@labelBox` [⟨*options*⟩]{⟨*label*⟩}

`\fca@labelBox` [⟨*options*⟩]{⟨*label*⟩}{⟨*content*⟩}

`\fca@labelBox` [⟨*options*⟩]{⟨*label*⟩}{⟨*xoffset*⟩}{⟨*yoffset*⟩}{⟨*content*⟩}

`\fca@labelBox` [⟨*options*⟩]{⟨*label*⟩}(⟨*xoffset,yoffset*⟩){⟨*content*⟩}

`\fca@labelBox` [⟨*options*⟩](⟨*label*⟩){⟨*content*⟩}

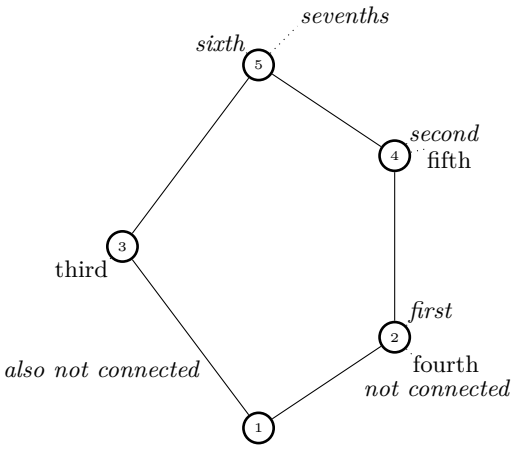
`\fca@labelBox` [⟨*options*⟩](⟨*label*⟩)(⟨*xoffset,yoffset*⟩){⟨*content*⟩}

Example 36: Syntax test for labels

Code:

```
  {\unitlength 1.2mm\relax
  \fcadocDraft
  \begin{diagram}{40}{55}
    \Numbers
    \Node{1}{20}{10}
    \Node{2}{35}{20}
    \Node{3}{5}{30}
    \Node{4}{35}{40}
    \Node{5}{20}{50}
    \Edge{1}{2}
    \Edge{1}{3}
    \Edge{2}{4}
    \Edge{3}{5}
    \Edge{4}{5}
    \fcaNoDots
    \leftAttbox(1)(-5,4){also not connected}
    \fcaDots
    \rightAttbox[node=2,node contents={first}]
    \rightAttbox[node contents=second]{4}
    \leftObjbox{3}{third}
    \rightObjbox{2}{2}{2}{fourth}
    \rightObjbox{4}(2,2){fifth}
    \leftAttbox(5){sixth}
    \rightAttbox(5)(3,3){sevenths}
    \rightAttbox[connector=false](1)(10,2){not connected}
  \end{diagram}
}
```

Result:




```
\newcommand*\fca@labelBox[1][]{%
  \fca@parse@label{\fca@l@belBox{#1}}{\fca@@labelBox{#1}}%
}
\newcommand*\fca@l@belBox[2]{%
  \@ifnextchar\bgroup{%
    \fca@lab@lBox{#1}{#2}%
  }{%
    \@ifnextchar({%)
      \fca@parse@vector{%
        \def\fca@tempa{outer sep=1pt,#1,node=#2,shift=}%
        \edef\fca@tempb{{(\the\@tempdima,\the\@tempdimb)}}%
        \fca@concat\fca@tempa\fca@tempb%
        \@ifnextchar\bgroup{%
          \expandafter\fca@l@b@lBox\expandafter{\fca@tempa}%
        }{%
          \expandafter\fca@@labelBox\expandafter{\fca@tempa}%
        }%
      }{}{\@tempdima}{\@tempdimb}%
    }{%
      \fca@@labelBox{#1,node={#2}}%
    }%
  }%
}
\newcommand*\fca@lab@lBox[3]{%
  \@ifnextchar\bgroup{%
    \fca@parse@vector{%
      \@ifnextchar\bgroup{%
        \fca@oldlabelBox{#1,node=#2}{\the\@tempdima}{\the\@tempdimb}%
      }{%
        \fca@oldlabelBox{#1,node=#2}{\the\@tempdima}{\the\@tempdimb}{}%
      }%
    }{}{\@tempdima}{\@tempdimb}{#3}%
  }{%
    \fca@@labelBox{#1,node={#2},node contents={#3}}%
  }%
}
```

```
\newcommand*\fca@l@b@lBox[2]{%
  \fca@@labelBox{#1,node contents=#2}
}
\newcommand*\fca@label@shift@sign@{}%
\expandafter\newcommand\expandafter*\csname fca@label@shift@sign@+\endcsname[1]{+(#1)}
\expandafter\newcommand\expandafter*\csname fca@label@shift@sign@-\endcsname[1]{-(#1)}
\newcommand*\fca@addshiftsign[1]{%
  \expandafter\fca@@ddshiftsign\expandafter{\csname
  fca@label@shift@#1@sign\endcsname}
}
\newcommand*\fca@@ddshiftsign[1]{%
  \csname fca@label@shift@sign@#1\endcsname
}

\newcommand\fca@oldlabelBox[4]{%
  \def\fca@tempa{%
    #1,
    label/at=center,
    shift=
  }%
  \edef\fca@tempb{%
    {({\noexpand\fca@addshiftsign x{#2}},{\noexpand\fca@addshiftsign y{#3}})}%
  }%
  \expandafter\expandafter\expandafter\fca@@labelBox
  \expandafter\expandafter\expandafter{%
    \expandafter\fca@tempa
    \fca@tempb,
    node contents={#4}%
  }%
}
```

`\fca@remove@anchor` `{`⟨*point*⟩`}` This macro removes the anchor part of a node name.

```
\newcommand\fca@remove@anchor[1]{%
  \fca@remove@anch@r#1.\fca@remove@anchor%
}
```

`\fca@remove@anch@r`

`\fca@prepare@node` `{`⟨*point*⟩`}{`⟨*direction*⟩`}{`⟨*prefix*⟩`}{`⟨*suffix*⟩`}{`⟨*node*⟩`}`

This macro is used to find the correct anchor point and the point a vector from another node will likely point to. This follows the following logic: If `{`⟨*node*⟩`}` is not passed in braces and contains a dot (`.`), then `{`⟨*point*⟩`}` will contain a macro that crates a point pointing to some other `\pgfpoint` on the border of `{`⟨*prefix*⟩`}{`⟨*node*⟩`}{`⟨*suffix*⟩`}`. And `{`⟨*direction*⟩`}` will be the `{`⟨*default anchor*⟩`}` of that node. Otherwise we assume that an anchor is given and we should draw lines directly from that anchor. So both macros `{`⟨*point*⟩`}` and `{`⟨*direction*⟩`}` will be defined to return the same point `\pgfpointanchor{{`⟨*prefix*⟩`}{`⟨*node name*⟩`}{`⟨*suffix*⟩`}}{{`⟨*node anchor*⟩`}}`.

```
\newcommand\fca@prepare@node[5]{%
```

```
  \def\@tempa{\fca@split@node@{#1}{#2}{#3}{#4}}%
  \edef\@tempb{#5.center.}%
  \fca@concat\@tempa{\@tempb\fca@split@node@}%
  \@tempa
}
```

`\fca@split@node@` {⟨*point*⟩}{⟨*direction*⟩}{⟨*prefix*⟩}{⟨*suffix*⟩}{⟨*node*⟩}.{⟨*default anchor*⟩}.`\fca@split@node@`

This macro is called by `\fca@prepare@node`. It should not be called directly.

This macro is used to find the correct anchor point and the point a vector from another node will likely point to. This follows the following logic: If {⟨*node*⟩} is not passed in braces and contains a dot (`.`), then {⟨*point*⟩} will contain a macro that crates a point pointing to some other `\pgfpoint` on the border of {⟨*prefix*⟩}{⟨*node*⟩}{⟨*suffix*⟩}. And {⟨*direction*⟩} will be the {⟨*default anchor*⟩} of that node. Otherwise we assume that an anchor is given and we should draw lines directly from that anchor. So both macros {⟨*point*⟩} and {⟨*direction*⟩} will be defined to return the same point `\pgfpointanchor`{{⟨*prefix*⟩}{⟨*node name*⟩}{⟨*suffix*⟩}}{{⟨*node anchor*⟩}}.

```
\def\fca@split@node@#1#2#3#4#5.#6.#7\fca@split@node@{%
  \def\@tempa{#7}%
  \ifx\@tempa\empty\relax
    \def#1##1{\pgfpointshapeborder{#3#5#4}{##1}}%
  \else
    \def#1##1{\pgfpointanchor{#3#5#4}{#6}}%
  \fi
  \def#2{%
    \pgfpointanchor{#3#5#4}{#6}%
  }%
}
```

```
\def\fca@def@expanded@node#1#2#3#4{%
  \edef\@tempa{\noexpand\def\noexpand#1\noexpand\fca@node@full@node{#2}{#3}#4..\fca@node@ful
  \@tempa
}
```

`\fca@node@anchor` {⟨*node*⟩}.{⟨*anchor*⟩}`\fca@node@anchor` Create a pgfpoint from a string of the form `node.anchor` as in Ti*k*Z. The anchor must be given, no default can be defined.

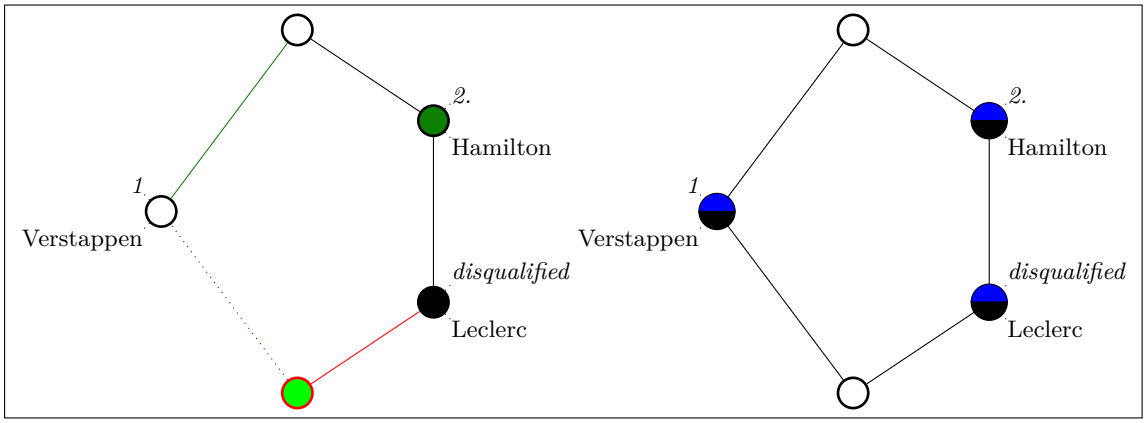
```
\def\fca@node@anchor#1.#2\fca@node@anchor{%
  \pgfpointanchor{#1}{#2}%
}
```

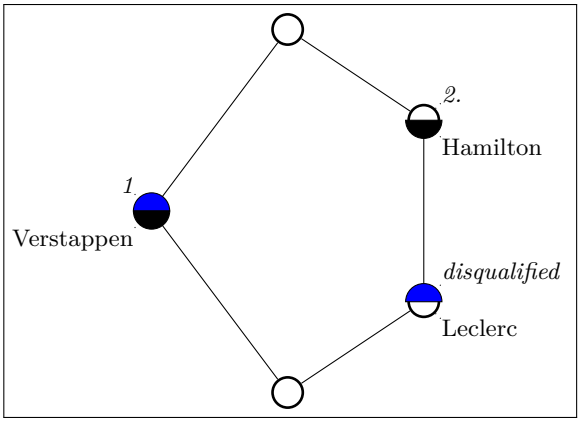
`\fca@node@border` {⟨*node1*⟩},{⟨*node2*⟩}.{⟨*anchor2*⟩}.{⟨*garbage*⟩}`\fca@node@border` This macro finds the `\pgfpoint` on the border of {⟨*node1*⟩} that goes towards {⟨*node2*⟩}.{⟨*anchor2*⟩}. The argument {⟨*garbage*⟩} is thrown away. This allows to select a default anchor in case no anchor is given for the {⟨*node2*⟩}.

```
\def\fca@node@border#1,#2.#3.#4\fca@node@border{%
```

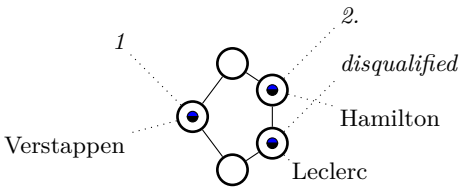
```
  \pgfpointshapeborder{#1}{\pgfpointanchor{#2}{#3}}%
}

\newcommand\fca@calculate@edgepoint[3]{%
  \edef\@tempa{\noexpand\pfgutil@in@{.}{#2}}%
  \@tempa
  \ifpfgutil@in@
    \def#1{%
      \fca@node@anchor#2\fca@node@anchor
    }%
  \else
    \def#1{%
      \fca@node@border#2,#3.center.\fca@node@border
    }%
  \fi
}
```





Using the style only for special nodes

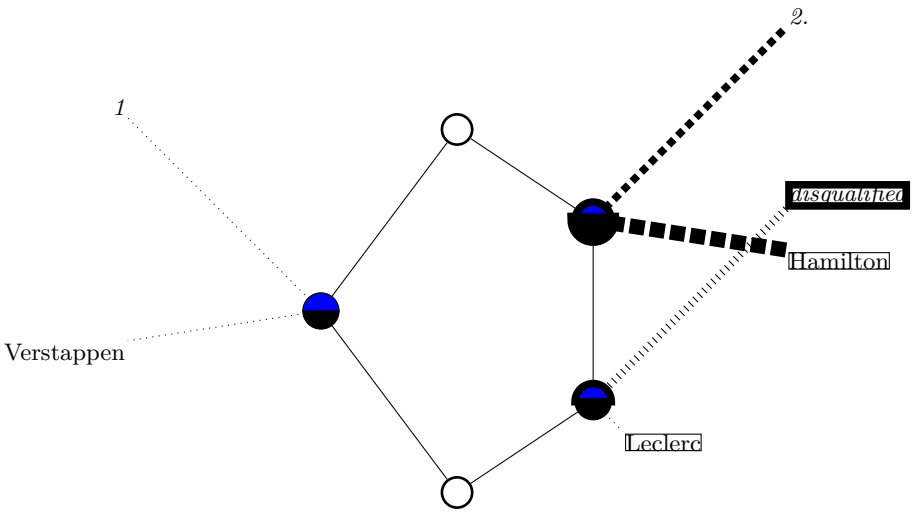


Example 37: testing label lines

Code:

```
\unitlength=1.2mm
\begin{diagram}[conexpstyle]
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  \leftAttbox(3)(-20,20){1}
  \rightAttbox[draw,line width=1mm](2)(20,20){disqualified}
  \rightAttbox[label/edge width=1mm](4)(20,20){2.}
  \leftObjbox(3)(-20,-2){Verstappen}
  \rightObjbox[draw](2)(2,-2){Leclerc}
  \fcaset{label/edge width=2mm}
  \rightObjbox[draw](4)(20,-2){Hamilton}
\end{diagram}
```

Result:



Notice the dependency between the connector lines and the border of the attribute and object concepts. These are the reason why the macros from

```
\newcommand\fca@@labelBox[1]{%
  \begin{pgfscope}%
    \fcaset{every node/.try,
      node font=\pgfkeysvalueof{/fca/label/node font},
```

```
    every label/.try,#1}%
  \fca@labelslayer
  \pgfkeysvalueof{/fca/node font}%
  \fca@options
  \edef\@tempa{\pgfkeysvalueof{/fca/pgfnode}}%
  \xdef\fca@temp@node@name{%
    \expandafter\fca@remove@anchor\expandafter{\@tempa}%
  }%
  \pgftransformshift{%
    \pgfpointanchor{%
      \fca@temp@node@name
    }{\fca@label@at}%
  }%
  \fca@parselength\@tempdima{\fca@label@shift@x}\unitlength
  \fca@parselength\@tempdimb{\fca@label@shift@y}\unitlength
  \pgftransformshift{\pgfpoint{\@tempdima}{\@tempdimb}}%
  \fca@transform
  \pgfnode{rectangle}{\pgfkeysvalueof{/fca/anchor}}{%
    \fca@typeset@node@content
    {%
      \pgfkeysvalueof{/fca/node contents}%
    }%
  }{%
    \pgfkeysvalueof{/fca/label/name prefix}%
    \pgfkeysvalueof{/fca/node}%
    \pgfkeysvalueof{/fca/label/name suffix}%
  }{%
    \expandafter\pgfusepath\expandafter{\fca@usepath}%
  }%
  \fca@endlabelslayer
  % pin edge
  \iffca@connectors
    \fca@connectorslayer
    \pgftransformreset
    \let\fca@options\fca@defaultoptions
    \fcaset{%
      line width=\fca@label@edge@width
    }
    \fcaset{every label edge/.try}%
    \fcaset{#1}%
    \fca@options
    \fca@transform
    %\pgfsetlinewidth{\pgfkeysgetvalue{/fca/line width}}%
    \fca@prepare@node\fca@temp@start@point\fca@temp@start@point@{}{}{\pgfkeysvalueof{/fca/
    \fca@prepare@node\fca@temp@end@point\fca@temp@end@point@
    {\pgfkeysvalueof{/fca/label/name prefix}}%
    {\pgfkeysvalueof{/fca/label/name suffix}}%
    {\pgfkeysvalueof{/fca/node}.\pgfkeysvalueof{/fca/anchor}}%
    \pgfpathmoveto{\fca@temp@start@point\fca@temp@end@point@}%
    \pgfpathlineto{\fca@temp@end@point\fca@temp@start@point@}%
    \pgfusepath{stroke}%
    \fca@endconnectorslayer
  \fi
  % overlay concept
```

```
    \pgftransformreset
    \let\fca@options\fca@defaultoptions
    \pgfsetdash{}{0pt}%
    \fcaset{shape=coordinate,#1}%
    \fcaset{every label concept/.try}%
    \fcaset{label concept/.try}%
    \ifx\pgfkeysvalueof{/fca/shape}\pgfkeysnovalue
    \else
      \fca@labelconceptslayer
      \pgftransformshift{%
        \pgfpointanchor{%
          \fca@temp@node@name
        }{\fca@label@at}%
      }%
      \fca@transform
      \fca@options
      % \pgfsetstrokecolor{\fca@node@color}%
      \pgfnode{\pgfkeysvalueof{/fca/shape}}{\pgfkeysvalueof{/fca/anchor}}{}{%
        \pgfkeysvalueof{/fca/label/name prefix}%
        \pgfkeysvalueof{/fca/node}%
        \pgfkeysvalueof{/fca/label/name suffix}%
      }{\expandafter\pgfusepath\expandafter{\fca@usepath}}%
      \fca@endlabelconceptslayer
    \fi
  \end{pgfscope}%
}


\newcommand\fca@leftAttbox[1][]{%
  \fca@labelBox[label type=attributes,label position=left,#1]
}
\newcommand\fca@centerAttbox[1][]{%
  \fca@labelBox[label type=attributes,label position=center,#1]
}
\newcommand\fca@rightAttbox[1][]{%
  \fca@labelBox[label type=attributes,label position=right,#1]
}
\newcommand\fca@leftObjbox[1][]{%
  \fca@labelBox[label type=objects,label position=left,#1]
}
\newcommand\fca@centerObjbox[1][]{%
  \fca@labelBox[label type=objects,label position=center,#1]
}
\newcommand\fca@rightObjbox[1][]{%
  \fca@labelBox[label type=objects,label position=right,#1]
}
%
\def\fca@ResetDefaults{\setcounter{fca@minNode}{60}%
  \setcounter{fca@maxNode}{0}%
  \setcounter{fca@CircleDiameter}{4}%
  \setboolean{fca@CircledNumbers}{false}%
  \fca@Defaults}%
%
\def\fcaColorNode#1{{%
```

```
    \PackageWarning{fca}{The \string\ColorNode\space macro is
      deprecated.^^J
      Use `\string\Node[fill=#1]` instead}
    \linethickness{\fca@node@thickness}%
    \color{#1}{\circle*{\value{fca@CircleDiameter}}}%
    \color{black}{\circle{\value{fca@CircleDiameter}}}}}%
%
\def\fca@circle{{%
    \linethickness{\fca@node@thickness}%
    \color{\fca@node@fill}{\circle*{\value{fca@CircleDiameter}}}%
    \color{black}{\circle{\value{fca@CircleDiameter}}}}}%
%
\def\fca@DrawCircles{\setcounter{fca@runNode}{\value{fca@minNode}}%
  \stepcounter{fca@maxNode}%
  \whiledo{\value{fca@runNode}<\value{fca@maxNode}}%
  {\fca@getNode{\value{fca@runNode}}%
    \put(\fca@x,\fca@y){\fca@circle}%
    \ifthenelse{\boolean{fca@CircledNumbers}}%
    {\put(\fca@x,\fca@y){\makebox(0,0){\tiny\arabic{fca@runNode}}}}{}%
    \stepcounter{fca@runNode}}}%
%

\newcommand*\fca@setb@undingbox{%
}%
\newcommand*\fca@ignoreboundingbox{}%

% set bounding box using brace notation
\newcommand*\fca@olddiagram[2]{%
  \fca@parselength\@tempdima{#1}\unitlength
  \edef\fca@right@border{\the\@tempdima}%
  \fca@parselength\@tempdimb{#2}\unitlength
  \edef\fca@top@border{\the\@tempdimb}%
  \let\fca@boundingbox\fca@setboundingbox
  \fca@parselength\@tempdima{-(\fcaDiagramXoffset)}\unitlength%
  \edef\fca@left@border{\the\@tempdima}%
  \fca@parselength\@tempdima{-(\fcaDiagramYoffset)}\unitlength%
  \edef\fca@bottom@border{\the\@tempdima}%
  \fca@startdiagram
  \fca@setboundingbox{\fca@left@border}{\fca@bottom@border}
  {\fca@right@border}{\fca@top@border}%
}
\newcommand*\fca@diagram{%
  \@ifnextchar(\fca@diagr@m\fca@di@gram
% )
}
% set bounding box using parentheses
\newcommand*\fca@diagr@m{%(#1,#2)
  \PackageError{fca}{not implemented}{not implemented}%
}
% set bounding box using defaults
\newcommand*\fca@di@gram{%
  \fca@startdiagram
}
\newcommand*\fca@startdiagram{%
```

```
  \fca@begindiagram
  \fca@Defaults
}
\newcommand*\fca@setboundingbox[4]{%
  \pgfpathrectangle{\pgfpoint{#1}{#2}}{\pgfpoint{#3}{#4}}%
  \pgfusepath{use as bounding box}%
}
% \todo{Lattice diagrams inside pgf pictures should be drawn inside
% their own scope}
\newenvironment{diagram}[1][]
{%
  \ifpgfpicture
    \def\fca@begindiagram{%
      \begin{pgfscope}%
        \fcaset{#1}%
        \unitlength=\dimexpr\pgf@xx+\pgf@yx\relax
        \def\fca@enddiagram{%
          \end{pgfscope}%
        }%
    }%
  \else
    \def\fca@begindiagram{%
      \begin{pgfpicture}%
        \fcaset{#1}%
        \def\fca@enddiagram{\end{pgfpicture}}%
    }%
  \fi
  \noindent
  \@ifnextchar\bgroup\fca@olddiagram\fca@diagram% This line must be
                                % executed before \begin{pgfpicture}
}%
{%
  %\fca@DrawCircles
  \fca@enddiagram%
}%
```

Env `tikzdiagram` This environment places a diagram in a Ti*k*Z picture.

```
\newenvironment{tikzdiagram}[1][]{%
  \def\fca@begindiagram{%
    \begin{tikzpicture}%
      \fcaset{#1}%
      \unitlength=\dimexpr\pgf@xx+\pgf@yx\relax
    }%
    \noindent
    \@ifnextchar\bgroup\fca@olddiagram\fca@diagram% This line must be
    % executed before \begin{pgfpicture}
  }{%
  \end{tikzpicture}%
}
```

Opt `/fca/copy unitlength` This option copys the value of `\unitlength` to the `x` and `y` coordinates of Ti*k*Z.

```
\fcaset{copy unitlength/.style={%
    x=\unitlength,
    y=\unitlength,
  }
}
```

End of diagram environment definition.

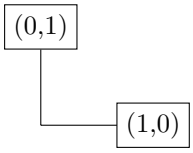
10.4 Some simple macros for FCA texts

```
\providecommand{\GMI}{(G,M,\relI)}
\newcommand{\context}[1][K]{\ensuremath{\mathbb{#1}}}
\providecommand{\'}{\ensuremath{\sp\prime}}% derivation operator
\providecommand{\extent}[1]{\textrm{ext}(#1)}
\providecommand{\intent}[1]{\textrm{int}(#1)}
\providecommand{\extents}[1]{\textrm{Ext}(#1)}
\providecommand{\intents}[1]{\textrm{Int}(#1)}
\providecommand{\BV}{\underline{{\mathfrak B}}}
\providecommand{\CL}{\BV}
\providecommand{\BGMI}{{\mathfrak B} (G,M,\relI)}
\providecommand{\CGMI}{\BGMI}
\providecommand{\BVGMI}{\BV (G,M,\relI)}
\providecommand{\CLGMI}{\BVGMI}
\providecommand{\HNI}{(H,N,\relI \cap\; H{\times}N)}
\providecommand{\relI}{\mathrel{I}}
\providecommand{\notI}{\mathrel{\mbox{\rlap{\char'40}%
      {\it I}\hspace*{-0.09em}\raisebox{.27ex}{\char'40}}}}%
%
\providecommand{\bigtimes}{\mathop{%
    \mathchoice{\raisebox{-2pt}{\huge$\times$}}{\mbox{\LARGE$\times$}}%
    {\raisebox{0pt}{\Large$\times$}}{\times}}\displaylimits}%
%
\providecommand{\Runterpfeil}{\mathrel{\swarrow}}
\providecommand{\DownArrow}{\Runterpfeil}
\providecommand{\Hochpfeil}{\mathrel{\nearrow}}
\providecommand{\UpArrow}{\Hochpfeil}
\providecommand{\IRunterpfeil}{\mathrel{\searrow}}
\providecommand{\IDownArrow}{\IRunterpfeil}
\providecommand{\IHochpfeil}{\mathrel{\nwarrow}}
\providecommand{\IUpArrow}{\IHochpfeil}
\providecommand{\Doppelpfeil}{\mathrel{\!\!\rlap{$\;\nearrow$}\swarrow}}
\providecommand{\DoubleArrow}{\Doppelpfeil}
%
\newcommand{\DDPfeil}{\mathrel{\mathchoice%
    {\mbox{$\displaystyle\swarrow\hspace{-.7em}\swarrow$}}
    {\mbox{$\textstyle\swarrow\hspace{-.7em}\swarrow$}}
    {\mbox{$\scriptstyle\swarrow\hspace{-.5em}\swarrow$}}
    {\mbox{$\scriptscriptstyle\swarrow\hspace{-.35em}\swarrow$}}}}
%
\newcommand{\NDDPfeil}{\mathrel{\mathchoice%
    {\mbox{$\displaystyle\swarrow\hspace{-.7em}\swarrow%
        \hspace{-1.2em}\backslash\hspace{.4em}$}}
    {\mbox{$\textstyle\swarrow\hspace{-.7em}\swarrow
```

```
        \hspace{-1.1em}\backslash\hspace{.4em}$}}
    {\mbox{$\scriptstyle\swarrow\hspace{-.5em}\swarrow
        \hspace{-.6em}\backslash\hspace{.2em}$}}
    {\mbox{$\scriptscriptstyle\swarrow\hspace{-.35em}\swarrow
        \hspace{-.5em}\backslash\hspace{.1em}$}}}}
%
\providecommand{\DPfeil}{\DDPfeil}
\providecommand{\NDPfeil}{\NDDPfeil}
\newcommand{\DDArrow}{\DDPfeil}
\newcommand{\NDDArrow}{\NDDPfeil}
\providecommand{\Semi}{\mathrel{\mbox{\tiny\rlap{\raisebox{2.0ex}%
        {$\bigtriangledown$}}\raisebox{-0.0ex}{$\bigtriangleup$}}}}
\def\ovee{\mbox{\small$\mathrel{\hspace{.35em}\raisebox{-1pt}%
      {$\smash{\vee}$}\hspace*{-.835em}\bigcirc\hspace{.2em}}$}}
\def\owedge{\mbox{\small$\mathrel{\hspace{.35em}\smash{\wedge}%
      \hspace*{-.835em}\bigcirc\hspace{.2em}}$}}
%
\providecommand{\ptimes}[1]{\mathrel{\stackrel{#1}{\times}}}
%
\providecommand{\FCA}{Formal Concept Analysis\xspace}
\providecommand{\FBA}{Formale Begriffsanalyse\xspace}
\providecommand{\FnBA}{Formalen Begriffsanalyse\xspace}
```

End of fca.sty style file definitions

11 The PGF Coordinate system



12 Formal contexts for demonstrating `\cxtinput`

The following formal context file is saved as `formula1.dtx` in the documentation folder.

```
B
Formula 1
3
3

Verstappen
Hamilton
Leclerc
1.
2.
disqualified
X..
```

```
.X.
.XX

B

3
3

Verstappen
Hamilton
Leclerc
1.
2.
disqualified
X..
.X.
.XX
```