`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 for details.

nocmpat, reverts the effect of `compat`

draft, enables draft mode. See

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:

5

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.

8: Eight in a many-valued context.

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			$\times$	
Leclerc			$\times$	$\times$

cxt 2		1.	2.	disqualified
Verstappen		$\times$		$\vee$
Hamilton		$\wedge$	$\times$	$\nearrow$
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 b		1.	2.	disqualified
Verstappen		×		
Hamilton			×	
Leclerc			×	×

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

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

2.2.3 Including Burmeister context files

\cxtinput

{⟨*filname.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 special 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`.

11

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	
$\underline{I}_\square$	\notI	
$\times$	\bigtimes	
$\boxtimes$	\Semi	
$\swarrow$	\DownArrow	\Runterpfeil
$\nearrow$	\UpArrow	\Hochpfeil
$\nearrow$	\DoubleArrow	\Doppelpfeil
$\nwarrow$	\DDArrow	\DDPfeil
$\nwarrow$	\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 $\swarrow$ of the arrow relations.

\Doppelpfeil   Same as **\DoubleArrow**.

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

\DDPfeil   Same as **\DDArrow**.

\NDDArrow   Gives $\nearrow\!\!\!\!\!\nearrow$ the symbol for the negation of $\nearrow\!\!\!\nearrow$.

\NDDPfeil   Same as **\NDDArrow**.

\Semi   Gives $\mathbb{X}$, 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 $\underline{\mathfrak{B}}(G, M, \nearrow\!\!\!\!\!\nearrow)$.

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.
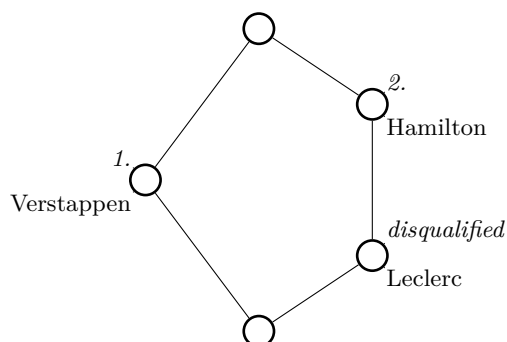
Again we start with a small example of each type:

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:

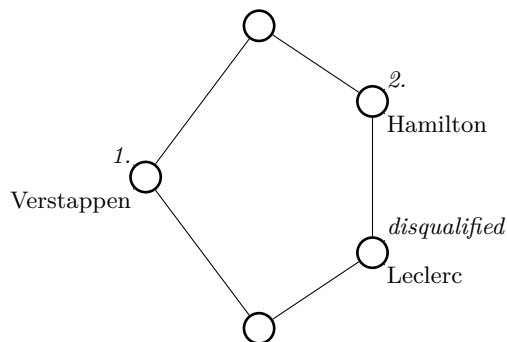**Example 9: A lattice diagram drawn in the classical way with the unit length of 1mm.**

Code:

14

```
\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 TikZ-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 TikZ-pictures is that one can use the many possibilities offered by PGF or TikZ, even without knowing much about PGF and TikZ. 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 TikZ picture. In that case all diagram commands are internally translated into TikZ 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.
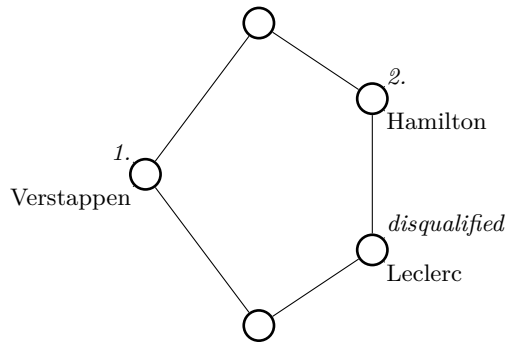
The `tikzdiagram` (or `diagram` inside of `tikzpicture`) uses the default unit length of the `tikzpicture` environment. This is usually 1 cm.

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 $1\,\mathrm{mm}$

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}
```
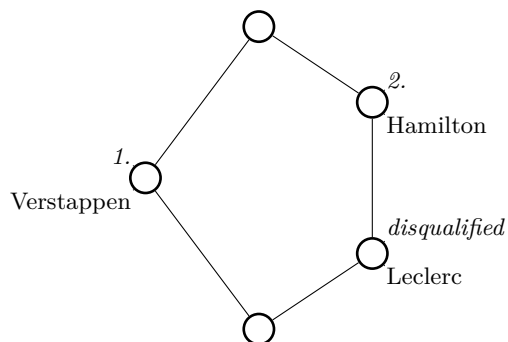
Result:

18

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:
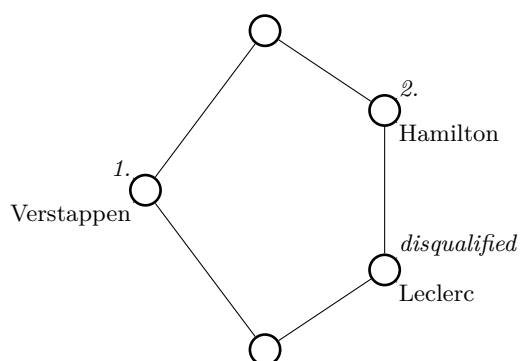
Changing the unit length can make the units more handier.

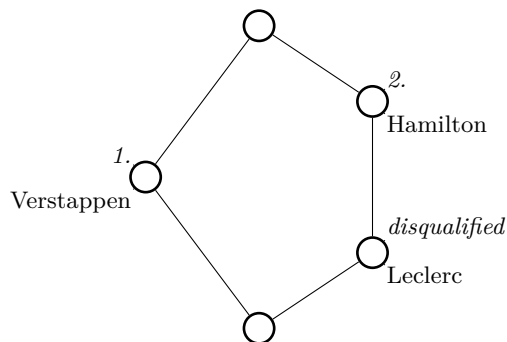**Example 13: A lattice diagram drawn in the classical way with the unit length of 1mm.**

Code:

19

```
\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.

Example 14: A `diagram` with some optional Ti*k*Z parameters

Code:

```
{\unitlength 1.2mm
\definecolor{darkgreen}{rgb}{0.05,0.5,0.}
\begin{diagram}
 \Node[draw=red, fill=green,line width=.5mm](1)(20,10)
 \Node[fill=black, radius=1.5mm](2)(35,20)
 \Node[opacity=0.3,fill=red](3)(5,30)
 \Node[fill=darkgreen](4)(35,40)
 \Node[/tikz/rectangle](5)(20,50)
 {\color{red}\Edge(1)(2)}
 \Edge[draw=blue,dotted,line width=1.5pt](1)(3)
 \Edge(2)(4)
 \Edge[draw=green,line width=1mm](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:
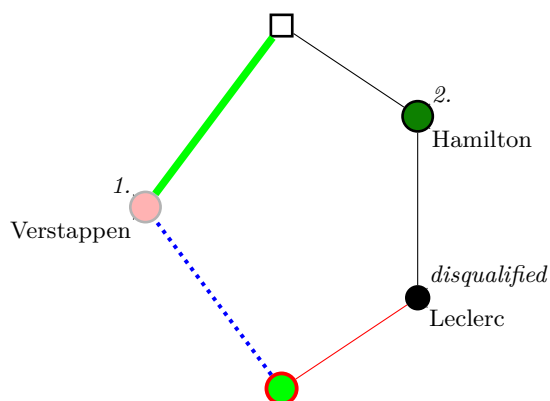
4.3.2 Fine tuning concept nodes

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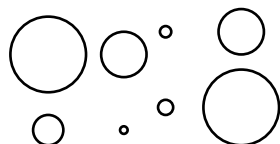
The circle size can be changed for single nodes by specifying the radius as an optional parameter (e.g., [radius=3mm]).

Example 15: 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.
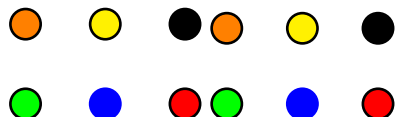
\NodeColor   \NodeColor{⟨*color*⟩} is an alias to \fcaNodeColor inside the diagram environment.

Example 16: 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 17: Usage of \NodeThickness and line width

Code:

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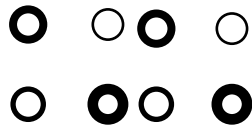
```
    \Node(2)(30,30)
    \Node[line width=3pt](1)(30,60)
    \fcaNodeThickness{4pt}
    \Node(4)(60,30)
\end{diagram}
\begin{tikzdiagram}
    \Node(3)(2,2)
    \NodeThickness{2pt}
    \Node(2)(1,1)
    \Node[line width=3pt](1)(1,2)
    \fcaNodeThickness{4pt}
    \Node(4)(2,1)
\end{tikzdiagram}
\endgroup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

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 18: 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)
```
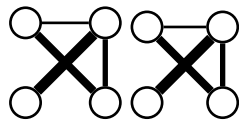
```
    \Node(2)(1,2)
    \Node(3)(2,1)
    \Node(4)(1,1)
    \Edge(1)(2)
    \EdgeThickness{2pt}
    \Edge(1)(3)
    \fcaEdgeThickness{3pt}
    \Edge(2)(3)
    \Edge[line width=4pt](1)(4)
  \end{tikzdiagram}
  \endgroup
```

Result:

4.5 Labels

Some of the nodes in a concept lattice diagram have text labels. Typically, object and attitude concepts are labeled with the associated object or attitude 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 19: 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:
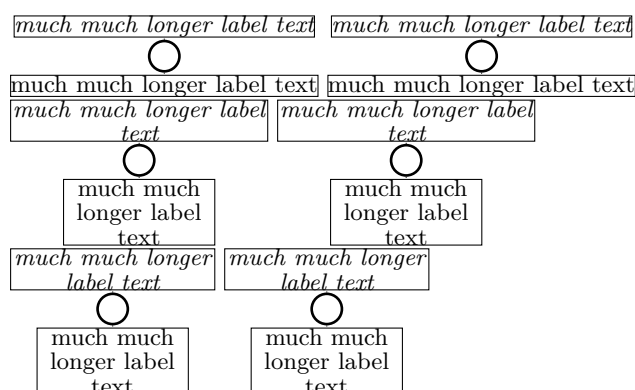
| Opt | draw | Label boxes may be framed with the draw option, which may also be used to set |
| Opt | fill | a color for the text and the frame. Their background color can be set with fill. |

Example 20: 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:
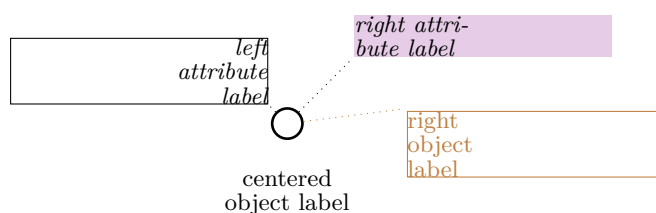
Opt font

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

`\ObjectLabelStyle`
`\AttributeLabelStyle`

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.

`\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 similarly.

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

Example 21: 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 22: 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:

---
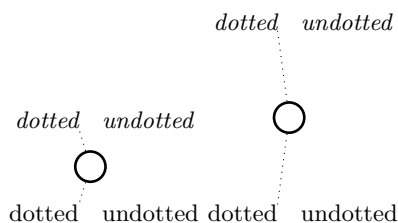
5 Advanced usage of the `diagram` environment

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 23: 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 24: Using a `diagram` style

Code:

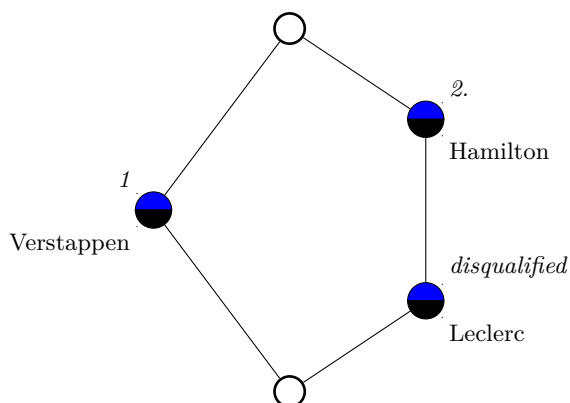```
{\unitlength 1.2mm
\begin{diagram}[conexp style]
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
```

```
  \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}`.

	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`. 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*⟩.

PGF and Ti*k*Z may be used inside a `diagram` environment, but with some caution. PGF uses canvas transformations, and these can collide with the `diagram` commands. To avoid such collisions, PGF or Ti*k*Z command sequences should be encapsulated. The *pgfscope* environment is used for this purpose. Example 25 illustrates this. The same diagram is drawn as in our earlier examples (e.g., Example 9), but is decorated using PGF. First a red circle with center $(20, 10)$\unitlength is drawn around the word "Example". Then a green line is drawn from the north east of Node 1 to the west of Node 5, and finally a blue line bisecting Node 2.

Example 25: Using pgf inside a diagram environment

Code:

---

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

```
\begin{diagram}
  \begin{pgfscope}
    \color{red}
    \pgftransformshift{\pgfpoint{20\unitlength}{30\unitlength}}%
    \pgfnode{circle}{center}{\Large Example}{example
    text}{\pgfusepath{stroke}}%
  \end{pgfscope}
  \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}
  \begin{pgfscope}
    \edef\tempa{%
    \noexpand\pgftransformshift{%
    \noexpand\pgfpointanchor{\pgfkeysvalueof{/fca/name%
    prefix}2\pgfkeysvalueof{/fca/name suffix}}{west}}%
  }%
    \tempa
    \pgfmoveto{\pgfpoint{0pt}{0pt}}%
    \pgflineto{\pgfpoint{4\unitlength}{0pt}}%
    \pgfsetstrokecolor{blue}%
    \pgfusepath{stroke}%
  \end{pgfscope}%
  {
    \pgfmoveto{\pgfpointanchor{fca node 5}{west}}
    \pgflineto{\pgfpointanchor{fca node 1}{north east}}
    \pgfsetstrokecolor{green}
    \pgfusepath{stroke}
  }
  \end{diagram}
```

Result:
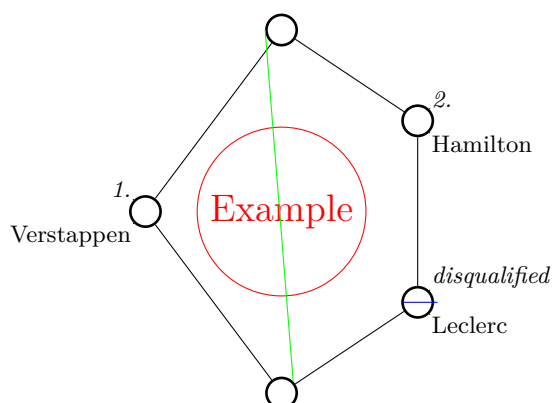
5.3 Which environment to choose?

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

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 26: 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:
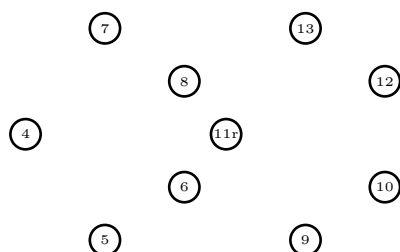
`\Numbers` should be used only to aid the development of diagrams. Thus it is active only in the draft mode. It is activated with the option `draft`.

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 27: \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:

---
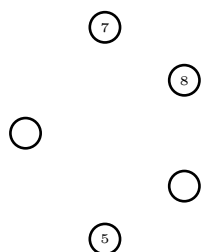
7   About the current version of `fca.sty`

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 TikZ, 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.

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.

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

Example 28: Changes of the bounding box with big numbers

```
\fbox{\unitlength .7mm
\begin{diagram}
  \Numbers
  \Node[node number/.style={font=\huge\color{red},node font=\huge,
  fill=orange},opacity=0.5]{5}{20}{10}
  \Node[font=\color{red}]{6}{35}{20}
  \Node{4}{5}{30}
  \Node{8}{35}{40}
  \Node{7}{20}{50}
\end{diagram}}
\fbox{\unitlength .7mm
\begin{diagram}
  \Node{5}{20}{10}
  \Node{6}{35}{20}
  \Node{4}{5}{30}
  \Node{8}{35}{40}
  \Node{7}{20}{50}
\end{diagram}}
```

Result:

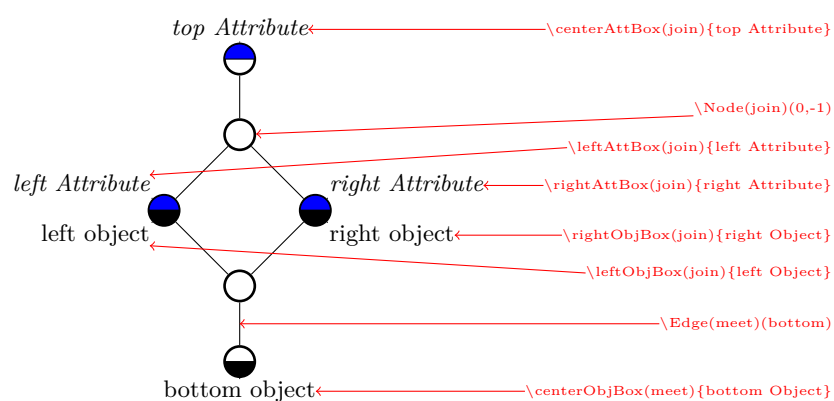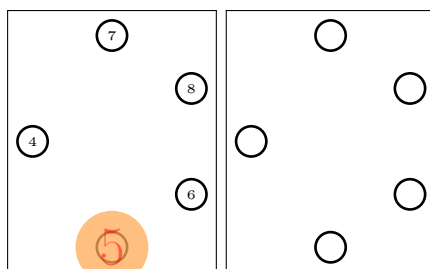

Figure 3: Elements of a `diagram` environment.

```
\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}
```

Example 29: A Ti*k*Zdiagram

Code:

```
\begin{tikzpicture}[x=0.1cm, y=0.1cm]
%      x and y should have the same size, here
  \node[red,shape=circle,anchor=center,draw=red] (example
  text) at (20,30) {\Huge Example};
  \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,
        rotate=180,
        label/name suffix=attribute concept,
        label/at=center,
        solid,
```

```
      }
    },
      shift={(0pt,0pt)}]
  \Node[/tikz/draw=orange](1)(20,10)
  \Node(2)(35,20)
  \Node[/tikz/opacity=0.3](3)(5,30)%
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge[/tikz/opacity=0.3](1)(3)
  \Edge[/tikz/arrows=<->](2)(4)
  \Edge[/tikz/opacity=0.3](3)(5)
  \Edge(4)(5)
  \leftAttbox[/tikz/opacity=0.3](3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox[/tikz/opacity=0.3](3){Hamilton}
  \rightObjbox(2){Massa}
  \rightObjbox(4){Alonso}
  \path[draw,red] (\pgfkeysvalueof{/fca/name
  prefix}2\pgfkeysvalueof{/fca/name suffix}.west)
  -- +(4\unitlength,0pt);
  \path[draw,red!50!yellow] (2.west)
  -- +(4\unitlength,0pt);
  \draw[green] (fca node 5.west)--(fca node 1.north east);
  \end{diagram}
\end{tikzpicture}
```

The same diagram can be typeset using PGF:
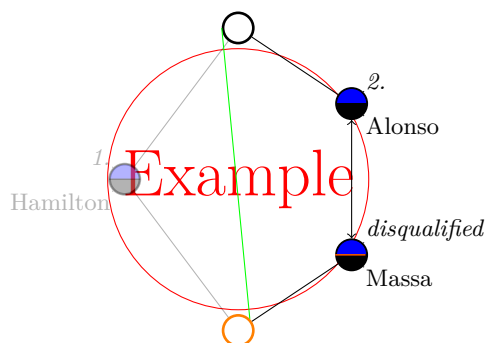
Example 30: A pgf enhanced diagram

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)
```
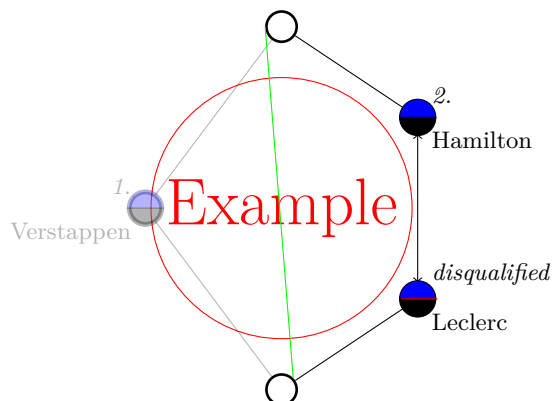
```
   \begin{pgfscope}
     \pgfsetstrokeopacity{0.3}
     \pgfsetfillopacity{0.3}
     \leftAttbox(3){1.}
   \end{pgfscope}
   \rightAttbox(2){disqualified}
   \rightAttbox(4){2.}
   \begin{pgfscope}
     \pgfsetstrokeopacity{0.3}
     \pgfsetfillopacity{0.3}
     \leftObjbox(3){Verstappen}
 \end{pgfscope}
   \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 31: A diagram
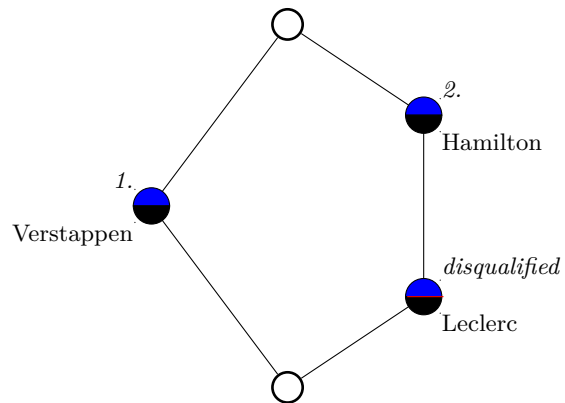
Code:

```
\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.

```
1 \newif\iffca@compat@macros
2 \fca@compat@macrosfalse
3 \DeclareOption{compat}{%
4   \fca@compat@macrostrue
5 }
6 \DeclareOption{nocompat}{%
7   \fca@compat@macrosfalse
8 }
```

Opt `draft`
`\iffca@draft`

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

```
 9 \newif\iffca@draft
10 \fca@draftfalse
11 \DeclareOption{draft}{%
12   \fca@drafttrue
13 }
14 \DeclareOption{final}{%
15   \fca@draftfalse
16 }
```

Evaluate the packages options.

```
17 \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.

```
18 \RequirePackage{pgf}
19 \usepgflibrary{shapes.geometric}
20 \RequirePackage{amssymb,graphics,color,ifthen,xspace}%%
21 \@ifundefined{AfterPackage}{%
22   \RequirePackage{afterpackage}%
23 }{}%
24 \AfterPackage{tikz}{%
25   \usetikzlibrary{fca}%
26 }%
```

\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⟩.

```
27 \newcommand*\fca@parselength[3]{%
28   \pgfmathparse{#2}%
29   \ifpgfmathunitsdeclared
30     #1=\pgfmathresult pt\relax
31   \else
32     #1=\pgfmathresult #3\relax%
33   \fi
34 }
```

9.3 The context environment cxt

9.3.1 Some configurations

\fca@cxt@Kreuz   First we define some symbols that are used in formal contexts.
\fca@cxt@Punkt
\fcaCxtArrowStyle
```
35 \newcommand{\fca@cxt@Kreuz}{$\times$}%
```
\cxtArrowStyle
```
36 \newcommand{\fca@cxt@Punkt}{}%
```
\fca@cxt@down
```
37 \iffca@compat@macros
```
\fca@cxt@up
```
38   \newcommand{\cxtArrowStyle}{\footnotesize}
```
\fca@cxt@both
```
39   \def\fcaCxtArrowStyle{\cxtArrowStyle}
40 \else
41   \newcommand{\fcaCxtArrowStyle}{\footnotesize}
42 \fi
43 \newcommand{\fca@cxt@down}{\fcaCxtArrowStyle$\Runterpfeil$}%
44 \newcommand{\fca@cxt@up}{\fcaCxtArrowStyle$\Hochpfeil$}%
45 \newcommand{\fca@cxt@both}{\fcaCxtArrowStyle$\Doppelpfeil$}%
```

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

```
46 \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.

```
47 \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.

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

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

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

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

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

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

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

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

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

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

48

```
60        \addtolength{\fca@cxt@nameraise}{-1\fca@cxt@ttnameheight}%
61        \raisebox{.5\fca@cxt@nameraise}{#1}%
62    }%
63  }%
64  \ignorespaces
65 }%
66 \iffca@compat@macros
67  \newcommand{\cxtName}{\fcaCxtName}
68 \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.

```
69 \newcommand{\fca@cxt@att}[1]{%
70  \ifthenelse{\boolean{fca@cxt@ttributes}}{%
71    \settoheight{\fca@cxt@ttnameheight}{#1}\adjcxt@name%
72    \expandafter\def\expandafter\fca@cxt@tabtop\expandafter{%
73      \fca@cxt@tabtop&#1}%
74    \stepcounter{fca@cxt@mAnz}%
75  }{%
76    \PackageWarning{fca}{Attribute following object in
77      cxt-environment%
78      has been ignored}{}}%
79  \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.

```
80 \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.

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

```
82    \fca@cxt@tabdef
83    #2\strut
84    \fca@cxt@Line{#1}%
85    \\
86 }%
```

\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 &.

```
87 \newcommand{\fca@cxt@freeobj}[2]{%acrocod}
88    \fca@cxt@tabdef%
89    #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.

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

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

```
91 \def\fca@cxt@tabdef@@{%
92    \expandafter\fca@cxt@tabdef@\fca@cxt@align
93    \fca@cxtn@me%&%
94    \fca@cxt@tabtop\strut\\\hline\hline
95 }
```

\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.

```
 96 \def\fca@cxt@tabdef@#1{%
 97   \def\fca@cxt@tabdef{\hline}%
 98   \tabcolsep0.5ex\relax%
 99   \begin{tabular}[#1]{|l||*{\value{fca@cxt@mAnz}}{c|}}%
100     \hline%
101     }%
```

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 32: 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 33: 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 34: An empty named context.

Code:

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

Result:

| Formula 0 |

Example 35: An empty cxt environment.

Code:

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

Result:

| |

```
102    \newenvironment{cxt}[1][t]{%
103    \begingroup
104    \fca@cxt@resetDefaults
105    \def\fca@cxt@align{#1}%
106    \ignorespaces
107    }{%
108    \fca@cxt@tabdef% open the tabular in case there are no objects
109    \end{tabular}%
110    \endgroup }%
```

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

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

```
114  \setcounter{fca@cxt@mAnz}{0}%
115  \setboolean{fca@cxt@ttributes}{true}%
116  \let\cxtName\fcaCxtName
117  \let\alignBottom\fcaCxtAlignBottom
118  \let\alignCenter\fcaCxtAlignCenter
119  \let\alignTop   \fcaCxtAlignTop
120  \let\att\fca@cxt@att
121  \let\atr\fca@cxt@atr
122  \let\obj\fca@cxt@obj
123  \let\freeobj\fca@cxt@freeobj
124  \let\cxtphantom\fca@cxt@phantom
125  \let\cxtrlap\fca@cxt@rlap
126  \let\fca@cxtn@me\@empty%
127  \let\fca@cxt@tabdef\fca@cxt@tabdef@@
128 }%
```

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

```
129 \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.

```
130 \def\fca@cxt@rlap#1{%
131   \settowidth\@tempdima{\cxtphantom}%
132   \makebox[\@tempdima][c]{\hss #1\hss}%
133 }
```

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

```
134 \def\fca@cxt@M@kechar@newcommand{%
135   \newcommand*%
136 }
```

\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)

```
137 \def\fca@cxt@Makechar@newcommand#1{%
138   \expandafter \fca@cxt@M@kechar@newcommand \csname cxt@char@\string#1 \endcsname
139 }
```

\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*`.

```
140 \def\fcaNewContextCh@r@#1[#2][#3]#4{%
141   \fca@cxt@Makechar@newcommand{#1}[{#2}][{#3}]{#4\fca@cxt@read@line}%
142 }%
143 \def\fcaNewContextCh@r#1[#2]#3{%
144   \fca@cxt@Makechar@newcommand{#1}[{#2}]{#3\fca@cxt@read@line}%
145 }%
146 \def\fcaNewContextChar@@#1[#2]{%
147   \@ifnextchar[{\fcaNewContextCh@r@{#1}[{#2}]}{\fcaNewContextCh@r{#1}[{#2}]}%
148 }%
149 \def\fcaNewContextChar@#1#2{%
150   \fca@cxt@Makechar@newcommand{#1}{#2\fca@cxt@read@line}%
151 }%
152 \def\fcaNewContextChar#1{%
153   \@ifundefined{cxt@char@\string#1 }{}{%
154     \PackageError{fca}{The character '\string#1' is already defined.}%
155     \expandafter\let
156     \csname cxt@char@\string#1 \endcsname\@undefined
157   }%
```

```
158    \@ifnextchar[{\fcaNewContextChar@@{#1}}{\fcaNewContextChar@{#1}}%
159 }
```

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

```
160 \def\fca@cxt@M@kechar@providecommand{%
161    \providecommand*%
162 }
```

fca@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)

```
163 \def\fca@cxt@Makechar@providecommand#1{%
164    \expandafter \fca@cxt@M@kechar@providecommand \csname cxt@char@\string#1 \endcsname
165 }
```

\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*.

```
166 \def\fcaProvideContextCh@r@#1[#2][#3]#4{%
167    \fca@cxt@Makechar@providecommand{#1}[{#2}][{#3}]{#4\fca@cxt@read@line}%
168 }%
169 \def\fcaProvideContextCh@r#1[#2]#3{%
170    \fca@cxt@Makechar@providecommand{#1}[{#2}]{#3\fca@cxt@read@line}%
171 }%
172 \def\fcaProvideContextChar@@#1[#2]{%
173    \@ifnextchar[{\fcaProvideContextCh@r@{#1}[{#2}]}{\fcaProvideContextCh@r{#1}[{#2}]}%
174 }%
175 \def\fcaProvideContextChar@#1#2{%
176    \fca@cxt@Makechar@providecommand{#1}{#2\fca@cxt@read@line}%
177 }%
```

```
178 \def\fcaProvideContextChar#1{%
179   \@ifnextchar[{\fcaProvideContextChar@@{#1}}{\fcaProvideContextChar@{#1}}%
180 }
```

fca@cxt@M@kechar@renewcommand

Put a starred \renewcommand* into one single token. We will use it when we define a single character.

```
181 \def\fca@cxt@M@kechar@renewcommand{%
182   \renewcommand*%
183 }
```

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)

```
184 \def\fca@cxt@Makechar@renewcommand#1{%
185   \expandafter \fca@cxt@M@kechar@renewcommand \csname cxt@char@\string#1 \endcsname
186 }
```

\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*.

```
187 \def\fcaRenewContextCh@r@#1[#2][#3]#4{%
188   \fca@cxt@Makechar@renewcommand{#1}[{#2}][{#3}]{#4\fca@cxt@read@line}%
189 }%
190 \def\fcaRenewContextCh@r#1[#2]#3{%
191   \fca@cxt@Makechar@renewcommand{#1}[{#2}]{#3\fca@cxt@read@line}%
192 }%
193 \def\fcaRenewContextChar@@#1[#2]{%
194   \@ifnextchar[{\fcaRenewContextCh@r@{#1}[{#2}]}{\fcaRenewContextCh@r{#1}[{#2}]}%
195 }%
196 \def\fcaRenewContextChar@#1#2{%
197   \fca@cxt@Makechar@renewcommand{#1}{#2\fca@cxt@read@line}%
```

56

```
198 }%
199 \def\fcaRenewContextChar#1{%
200   \@ifundefined{cxt@char@\string#1 }{%
201     \PackageError{fca}{The character '\string#1' is undefined.^^J
202       It must have been defined in order to be redefined.}%
203     \fcaNewContextChar{#1}{}%
204   }{}%
205   \@ifnextchar[{\fcaRenewContextChar@@{#1}}{\fcaRenewContextChar@{#1}}%
206 }
```

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.

```
207 \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.

```
208 \def\fca@cxt@executechar#1{%
209   &\@ifundefined{cxt@char@\string#1 }{%
210     \PackageWarning{fca}{Undefinded character \string#1 \space in the context}%
211     \let\fca@cxt@tmp=\fca@cxt@aPunkt%
212   }{%
213     \expandafter\let\expandafter\fca@cxt@tmp \csname
214     cxt@char@\string#1 \endcsname }%
215   \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.

```
216 \def\fca@cxt@read@line#1{%
217   \ifx#1\fca@cxt@stop \let\fca@cxt@zeile@excecutechar\@gobble \else
218   \let\fca@cxt@zeile@excecutechar\fca@cxt@executechar \fi
219   \fca@cxt@zeile@excecutechar{#1}%
220 }%
221 % \end{macro}
222 %
223 % \begin{macro}{\fca@cxt@Line}\marg{\line}
224 %   Process a whole crosstable line. We add \cs{fca@cxt@stop} at the end and start the loop
225 % \begin{macrocode}
226 \def\fca@cxt@Line#1{%
227   \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.

```
228 \fcaNewContextChar .{\cxtphantom}
229 \fcaNewContextChar x{\fca@cxt@Kreuz}
230 \fcaNewContextChar X{\fca@cxt@Kreuz}
231 \fcaNewContextChar u{\cxtrlap{\fca@cxt@up}}
232 \fcaNewContextChar d{\cxtrlap{\fca@cxt@down}}
233 \fcaNewContextChar b{\cxtrlap{\fca@cxt@both}}
234 \@for\tmp:= 0,1,2,3,4,5,6,7,8,9\do{
235   \edef\@tmp{
236     \noexpand\fcaNewContextChar\tmp{%
237       \noexpand\cxtrlap\tmp
238     }%
239   }%
240   \@tmp
241 }
```

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.

```
242 \newcount\fca@cxt@input@obj
243 \newcount\fca@cxt@input@attr
244 \newcount\fca@cxt@input@line
245 \newtoks\fca@cxt@input@contextlines
246 \fca@cxt@input@line0
247 \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.

```
248 %
249 \newcommand\cxtinput[1]{%
250 %  \begingroup
251   \fca@cxt@input@contextlines{}%
252   \fca@cxt@input@save@nl@active%
253   \fca@cxt@input@make@nl@active%
254   \fca@cxt@input@input{#1}%
255   \fca@cxt@input@restore@nl%
256   \xdef\fca@cxt@input@tempa{\the\fca@cxt@input@contextlines}%
257 %\aftergroup
258   \fca@cxt@input@tempa%
259 %  \endgroup%
260 }
```

\fca@cxt@input@newline   Macro holding the command for the next line

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

262 \def\fca@cxt@input@head{%
263   \fca@cxt@input@getline\fca@cxt@input@check@B
264 }
```

Check the "B" at the beginning of the file

```
265 \def\fca@cxt@input@check@B#1{%
266   \def\tempa{B}\def\tempb{#1}%
267   \ifx\tempa\tempb
268   \typeout{Burmeister format detected}%
269   \else
270     \fca@cxt@input@error{No Burmeister format detected}{The \string\cxtinput macro can input
271   \fi
272   \def\fca@cxt@input@newline{%
273     \fca@cxt@input@getline{\fca@cxt@input@read@cxtname}%
274   }%
275 }
```

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

```
276 \def\fca@cxt@input@match@mptyline#1#2{%
277   \edef\tempa{#2}%
278   \ifx\tempa\@empty
279   \else
280     \fca@cxt@input@error{Error in Burmeister format.}{At the current position an empty line
281   \fi
282   \def\fca@cxt@input@newline{%
283     \fca@cxt@input@getline{#1}%
284   }%
285 }
```

Read the context name

```
286 \def\fca@cxt@input@read@cxtname#1{%
287   \ifx\fca@cxtn@me\@empty
288     \fcaCxtName{#1}%
289   \fi
290   \def\fca@cxt@input@newline{%
291     \fca@cxt@input@getline{\fca@cxt@input@readobjcount}%
292   }%
293 }
```

Read number of objects from the file

```
294 \def\fca@cxt@input@readobjcount#1{%
295   \fca@cxt@input@obj=#1\relax
296   \def\fca@cxt@input@newline{%
297     \fca@cxt@input@getline\fca@cxt@input@readattrcount%
298   }%
299 }
```

Read number of attributes

```
300 \def\fca@cxt@input@readattrcount#1{%
301   \def\fca@cxt@input@newline{%
302     \fca@cxt@input@getline{\fca@cxt@input@match@mptyline\fca@cxt@input@readobjects}%
303   }%
304   \fca@cxt@input@attr=#1\relax
305 }
```

initializes the reading of the object names

```
306 \def\fca@cxt@input@readobjects{%
307   \ifnum\fca@cxt@input@obj>0\relax
308     \def\fca@cxt@input@newline{%
309       \fca@cxt@input@getline\fca@cxt@input@readobjname
310     }%
311     \@tempcnta=1\relax
312     \let\tempa\fca@cxt@input@readobjname%
313   \else
314     \let\tempa\fca@cxt@input@readattributes%
315   \fi
316   \tempa
317 }
```

Read the object names

```
318 \def\fca@cxt@input@readobjname#1{%
319   \expandafter\def\csname cxt@input@objname@\the\@tempcnta\endcsname{#1}%
320   \ifnum\@tempcnta < \fca@cxt@input@obj
321     \advance\@tempcnta by 1\relax
322   \else
323     \def\fca@cxt@input@newline{%
324       \fca@cxt@input@getline\fca@cxt@input@readattributes%
325     }%
326   \fi
327 }
```

Initialize reading of attibute names

```
328 \def\fca@cxt@input@readattributes{%
329   \ifnum\fca@cxt@input@attr>0\relax
330     \def\fca@cxt@input@newline{%
331       \fca@cxt@input@getline\fca@cxt@input@readattrname
332     }%
333     \@tempcnta=1\relax
334     \def\tempa{\fca@cxt@input@readattrname}%
335   \else
336     \def\tempa{\fca@cxt@input@readcontext}%
337   \fi
338   \tempa
339 }
```

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

```
340 \def\fca@cxt@input@readattrname#1{%
341   \fca@cxt@input@appendtotok{#1}\atr%
342   \ifnum\@tempcnta < \fca@cxt@input@attr
343     \advance\@tempcnta by 1\relax
344   \else
345     \def\fca@cxt@input@newline{%
346       \fca@cxt@input@getline\fca@cxt@input@readcontext
347     }%
348   \fi
349 }
```

initalize reading of the cross table

```
350 \def\fca@cxt@input@readcontext{%
351   \ifnum\fca@cxt@input@obj>0\relax
352     \def\fca@cxt@input@newline{%
353       \fca@cxt@input@getline
354       \fca@cxt@input@readcontextline
355     }%
356     \@tempcnta=1\relax
357     \def\tempa{\fca@cxt@input@readcontextline}%
358   \else
359     \def\tempa{}%
360   \fi
361   \tempa
362 }
```

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

```
363 \def\fca@cxt@input@readcontextline#1{%
364   \expandafter\expandafter\expandafter\fca@cxt@input@appendtotok
365   \expandafter\expandafter\expandafter{%
366     \csname cxt@input@objname@\the\@tempcnta\endcsname}%
367   {\obj{#1}}%
368   \ifnum\@tempcnta < \fca@cxt@input@obj
369     \advance\@tempcnta by 1\relax
370   \else
371     \let\fca@cxt@input@newline\relax
372     \let\fca@cxt@input@endoffile\relax
373   \fi
374 }
```

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

```
375 \def\fca@cxt@input@appendtotok#1#2{%
376   \expandafter\fca@cxt@input@contextlines\expandafter{%
377     \the\fca@cxt@input@contextlines
378     #2{#1}%
379   }%
380 }
```

```
381
382 \def\fca@cxt@input@error#1#2{%
383   \PackageError{fca}{At line \the\fca@cxt@input@line : #1}{#2}%
384   \def\fca@cxt@input@newline{}%
385 }%
```

Some end of file mark

```
386 \def\fca@cxt@input@endoffile{%
387   \fca@cxt@input@error{unexpected end of file}{The context file is somehow
388     inconsistent.\MessageBreak The last lines of it seem to be lost.}%
389 }
```

macro for usage with \ifx

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

swich catcode of newline to runtime mode

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

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

```
393 \gdef\fca@cxt@input@make@nl@active{%
394   \catcode`\^^M\active%
395 % \let\fca@cxt@input@oldcr^^M%
396 % \def^^M{\fca@cxt@input@newline}%
397 }%
398
399 \gdef\fca@cxt@input@save@nl@active{%
400   \chardef\fca@cxt@input@catcode@nl=\catcode`\^^M%
401 % \let\fca@cxt@input@oldcr^^M%
402 % \def^^M{\fca@cxt@input@newline}%
403 }%
```

Define a macro to set the catcode.

```
404 \gdef\fca@cxt@input@restore@nl{%
405   \catcode`\^^M\fca@cxt@input@catcode@nl\relax%
406 % \let\fca@cxt@input@oldcr^^M%
407 % \def^^M{\fca@cxt@input@newline}%
408 }%
409
410 \fca@cxt@input@make@nl@active%
```

reads a line from the context file.

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

```
414    \ifx\tempa\fca@cxt@input@@endoffile%
415      \tempa%
416    \fi%
417    #1{#2}%
418    \fca@cxt@input@newline%
419 }%
```

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

```
420 \gdef\fca@cxt@input@input#1{%
421    \expandafter\fca@cxt@input@head\@@input #1 %
422    \fca@cxt@input@endoffile%
423 }
```

restore newline catcode

```
424 \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 36: 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}}
```
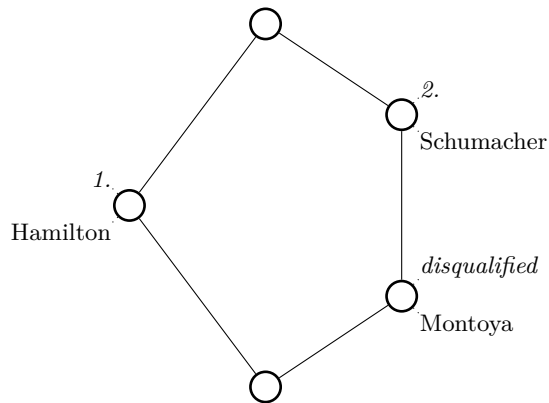
Result:

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**.

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

65

```
426  \newcommand{\fcaObjectLabelStyle}{%
427    \small\baselineskip1em\rmfamily\upshape%
428  }% %
429  \newcommand{\fcaAttributeLabelStyle}{%
430    \small\baselineskip1em\rmfamily\itshape
431  }% %
432  \newcommand{\fcaLabelBoxWidth}{40mm}%
433  \let\Node\fca@node
434  \let\Edge\fca@edge
435  \iffca@compat@macros
436  \else
437    \def\ObjectLabelStyle{\fcaObjectLabelStyle}%
438    \def\AttributeLabelStyle{\fcaAttributeLabelStyle}%
439    \let\LabelBoxWidth\fcaLabelBoxWidth
440    \let\EdgeThickness\fcaEdgeThickness
441    \let\NodeThickness\fcaNodeThickness
442    \let\Numbers\fcaNumbers
443    \let\NoNumbers\fcaNoNumbers
444    \let\CircleSize\fcaCircleSize
445    \let\NodeColor\fcaNodeColor
446    \let\ColorNode\fcaColorNode
447    \let\NoDots\fcaNoDots
448    \let\Dots\fcaDots
449  \fi
450  \let\leftAttbox\fca@leftAttbox
451  \let\centerAttbox\fca@centerAttbox
452  \let\rightAttbox\fca@rightAttbox
453  \let\leftObjbox\fca@leftObjbox
454  \let\centerObjbox\fca@centerObjbox
455  \let\rightObjbox\fca@rightObjbox
456 }% %
457 \let\fca@Defaults\fca@notikz@Defaults
458 \newboolean{fca@connectors}\setboolean{fca@connectors}{true}% %
459 \iffca@compat@macros
460   \newcommand{\diagramXoffset}{0}% %
461   \newcommand{\diagramYoffset}{0}% %
462   \newcommand{\fcaDiagramXoffset}{\diagramXoffset}% %
463   \newcommand{\fcaDiagramYoffset}{\diagramYoffset}% %
464 \else
465   \newcommand{\fcaDiagramXoffset}{0}% %
466   \newcommand{\fcaDiagramYoffset}{0}% %
467 \fi
468 \newcommand*{\fca@xunitlength}{\unitlength}%
469 \newcommand*{\fca@yunitlength}{\unitlength}%
470 \newcommand*{\fca@edge@thickness}{.8pt}% %
471 \newcommand*{\fca@node@thickness}{1pt}% %
472 \newcommand*{\fca@transform}{}%
473 \newcommand*{\fca@options}{}%
474 \newcommand*{\fca@defaultoptions}{}%
475 \newcommand*{\fca@usepath}{}%
476 \newcommand*{\fca@node@number@prefix}{\pgfkeysvalueof{/fca/namespace}\space node\space}%
477 \newcommand*{\fca@node@number@suffix}{\space number}%
478 \newcommand*{\fca@label@edge@width}{\@wholewidth}%
479 \newcommand*{\fca@label@at}{center}% anchor of node
```

```
480 \newcommand*{\fca@label@shift@x}{0pt}%
481 \newcommand*{\fca@label@shift@y}{0pt}%
482 \newcommand*{\fca@label@shift@x@sign}{}
483 \newcommand*{\fca@label@shift@y@sign}{}
484 \newcommand*{\fca@label@type}{attributes}%
485 \newcommand*{\fca@label@position}{right}%
486
487 \newcommand*\fca@none{none}
488 \newcommand*\fca@firstofthree[3]{#1}
489 \newcommand*\fca@secondofthree[3]{#2}
490 \newcommand*\fca@thirdofthree[3]{#3}
491 \newcommand*\fca@testoption[1]{%
492   \def\fca@tempa{#1}%
493   \ifx\empty\fca@tempa
494     \let\fca@tempb\fca@thirdofthree
495   \else
496     \def\fca@tempb{\pgfkeysnovalue}%
497     \ifx\fca@tempa\fca@tempb
498       \let\fca@tempb\fca@thirdofthree
499     \else
500       \ifx\fca@tempa\fca@none
501         \let\fca@tempb\fca@secondofthree
502       \else
503         \let\fca@tempb\fca@firstofthree
504       \fi
505     \fi
506   \fi
507   \fca@tempb
508 }
```

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 TikZ 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 (TikZ, 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 TikZ 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.

```
509 \newcommand*\fcaLayer[2]{%
510   \def\@tempa{main}%
511   \edef\@tempb{#2}%
512   \ifx\@tempb\@empty\let\@tempb\@tempa\fi
513   \ifx\@tempa\@tempb
514     \def\@tempa{}%
515     \def\@tempb{}%
516   \else
517     \def\@tempa{%
518       \begin{pgfonlayer}{#2}%
519       }
520     \def\@tempb{%
521       \end{pgfonlayer}%
522     }
523   \fi
524   \expandafter\let\csname fca@#1layer\endcsname\@tempa
525   \expandafter\let\csname fca@end#1layer\endcsname\@tempb
526 }
```

\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.

```
527 \newcommand*{\fca@labelslayer}{\csname fca@\fca@label@type layer\endcsname}
528 \newcommand*{\fca@endlabelslayer}{\csname fca@end\fca@label@type layer\endcsname}
529 \newcommand*{\fca@labelconceptslayer}{\csname fca@\fca@label@type conceptlayer\endcsname}
530 \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.

```
531 \newcommand*{\fcaNewLayer}[2]{%
532   \pgfdeclarelayer{#2}%
533   \fcaLayer{#1}{#2}%
534 }
```

\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

```
535 \newcommand*{\fcaDeclareLayers}{%
536   \fcaLayer{nodes}{main}
537   \fcaNewLayer{nodenames}{\pgfkeysvalueof{/fca/namespace} node numbers}
538   \fcaNewLayer{attributes}{\pgfkeysvalueof{/fca/namespace} attributes}
539   \fcaNewLayer{objects}{\pgfkeysvalueof{/fca/namespace} objects}
540   \fcaNewLayer{attributesconcept}{%
541     \pgfkeysvalueof{/fca/namespace} attribute concepts}
542   \fcaNewLayer{objectsconcept}{%
543     \pgfkeysvalueof{/fca/namespace} object concepts}
544   \fcaNewLayer{edges}{\pgfkeysvalueof{/fca/namespace} edges}
545   \fcaNewLayer{connectors}{\pgfkeysvalueof{/fca/namespace} connectors}
546   \pgfdeclarelayer{\pgfkeysvalueof{/fca/namespace} above nodes}
547   \pgfdeclarelayer{\pgfkeysvalueof{/fca/namespace} below nodes}
548 }
```

\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.

```
549 \newcommand*{\fcaSetLayers}{%
550   \pgfsetlayers{%
551     \pgfkeysvalueof{/fca/namespace} edges,%
552     \pgfkeysvalueof{/fca/namespace} connectors,%
553     \pgfkeysvalueof{/fca/namespace} below nodes,%
554     main,%
555     \pgfkeysvalueof{/fca/namespace} above nodes,%
556     \pgfkeysvalueof{/fca/namespace} node numbers,%
557     \pgfkeysvalueof{/fca/namespace} attribute concepts,%
558     \pgfkeysvalueof{/fca/namespace} object concepts,%
559     \pgfkeysvalueof{/fca/namespace} attributes,%
560     \pgfkeysvalueof{/fca/namespace} objects%
561   }%
562 }
```

\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.

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

```
564   \fcaLayer{nodes}{main}%
565   \fcaLayer{nodenames}{main}%
566   \fcaLayer{attributes}{main}%
567   \fcaLayer{objects}{main}%
568   \fcaLayer{attributesconcept}{main}%
569   \fcaLayer{objectsconcept}{main}%
570   \fcaLayer{edges}{main}%
571   \fcaLayer{connectors}{main}%
572 }
573 \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.

```
574 \newcommand*{\fcaLayers}{%
575   \fcaDeclareLayers
576   \fcaSetLayers
577 }
```

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

Example 37: 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)
```
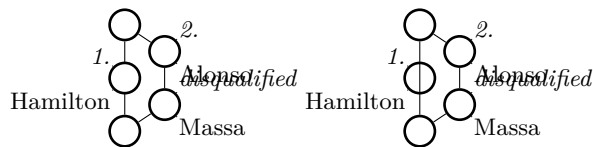
```
\Node(5)(20,50)
\Edge(1)(2)
\Edge(1)(5)
\Edge(2)(4)
\Edge(4)(5)
\leftAttbox(3){1.}
\rightAttbox(2){disqualified}
\rightAttbox(4){2.}
\leftObjbox(3){Hamilton}
\rightObjbox(2){Massa}
\rightObjbox(4){Alonso}
\end{diagram}
```

Result:

Example 38: A Ti*k*Z diagram (left: with layers, right: without layers)

Code:

```
\begin{tikzdiagram}[/tikz/x=.1cm,/tikz/y=.1cm]%
  \fcaLayers
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node[/tikz/opacity=0.8](3)(20,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  \Edge(1)(2)
  \Edge[draw=black,/tikz/opacity=0.3](1)(5)
  \Edge[/tikz/arrows=<->](2)(4)
  \Edge[/tikz/opacity=0.3](3)(5)
  \Edge(4)(5)
  \leftAttbox[/tikz/opacity=0.3](3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox[/tikz/opacity=0.3](3){Hamilton}
  \rightObjbox(2){Massa}
  \rightObjbox(4){Alonso}
  \draw (1.center) -- (4.center);
  \begin{pgfonlayer}{fca below nodes}
    \draw (5.center)--(2.center);
  \end{pgfonlayer}
\end{tikzdiagram}
\begin{tikzdiagram}[/tikz/x=.1cm,/tikz/y=.1cm]%
  \Node(1)(20,10)
  \Node(2)(35,20)
```
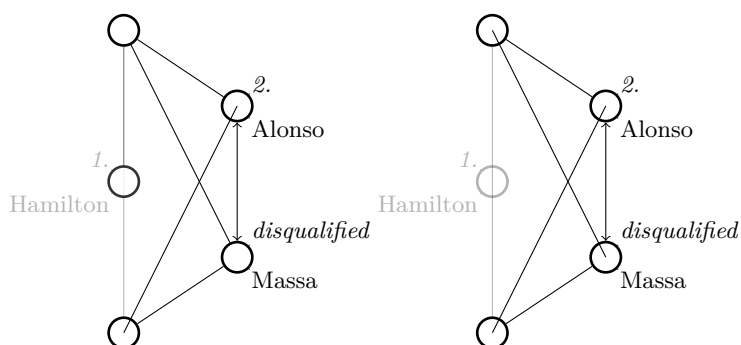
```
    \Node[/tikz/opacity=0.3](3)(20,30)
    \Node(4)(35,40)
    \Node(5)(20,50)
    \Edge(1)(2)
    \Edge[/tikz/draw=black,/tikz/opacity=0.3](1)(5)
    \Edge[/tikz/arrows=<->](2)(4)
    \Edge(4)(5)
    \leftAttbox[/tikz/opacity=0.3](3){1.}
    \rightAttbox(2){disqualified}
    \rightAttbox(4){2.}
    \leftObjbox[/tikz/opacity=0.3](3){Hamilton}
    \rightObjbox(2){Massa}
    \rightObjbox(4){Alonso}
    \draw (1.center) -- (4.center);
    \draw (5.center)--(2.center);
  \end{tikzdiagram}
```

Result:


```
578 % We allow to select the label type and positioning independently.
579 % This function executes the right positioning settings.
580 \newcommand\fca@dolabelposition{%
581   \pgfkeysgetvalue{/fca/label/type}{\@tempa}%
582   \pgfkeysgetvalue{/fca/label/position}{\@tempb}%
583   \ifx\pgfnovalue\@tempa
584   \else
585     \ifx\pgfnovalue\@tempb
586     \else
587       \pgfkeysalso{/fca/label/\@tempa{} position=\@tempb}%
588     \fi
589   \fi
590 }
591
592 \newcommand*\fcaNewLabelType{%
593   \@ifstar{\fca@NewLabelType}{\fca@NewL@belType}%\fistar
594 }
595 \newcommand*\fca@NewLabelType[1]{%
596   \pgfqkeys{/fca/label}{%
597     type/#1/.style={type=#1},
598     #1 position/.is choice,
```

```
599    }%
600 }
601 \newcommand*\fca@NewL@belType[1]{%
602    \fca@NewLabelType{#1}%
603    \pgfqkeys{/fca/label}{%
604      #1/.style={type=#1}%
605    }%
606 }
607 \newcommand*\fcaNewLabelPosition{%
608    \@ifstar{\fca@NewLabelPosition}{\fca@NewL@belPosition}%\fistar
609 }
610 \newcommand*\fca@NewLabelPosition[1]{%
611    \pgfqkeys{/fca/label}{%
612      position/#1/.style={position=#1},
613    }%
614 }
615 \newcommand*\fca@NewL@belPosition[1]{%
616    \fca@NewLabelPosition{#1}%
617    \pgfqkeys{/fca/label}{%
618      #1/.style={position=#1}%
619    }%
620 }
621
```

Changing the default values

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

```
622 \newcommand*\fca@generate@tikz@parameter[3]{%
623    \pgfqkeys{/fca/#1}{.initial={#2},.value required}
624 }
625 \newcommand*\fca@generate@tikz@parameters{%
626    \@for \fca@tmp:=%
627    {font}{}{\tikz@textfont},%
628    {node font}{}{\tikz@node@textfont},%
629    {text opacity}{}{\tikz@textopacity},%
630    {text width}{\pgfkeysnovalue}{\tikz@text@width},%
631    {text height}{}{\tikz@text@height},
632    {text depth}{}{\tikz@text@depth},
633    {text action}{}{\tikz@text@action},
634    {options}{}{\tikz@options},%\newcommand*{\fca@node@}{white}%
635    {anchor}{}{\tikz@anchor},
636    {shape}{}{\tikz@shape}
637    \do {
638      \expandafter\fca@generate@tikz@parameter\fca@tmp
639    }
640 }
641 \fca@generate@tikz@parameters
```

10.2 set up what we need from tikz

```
642 \RequirePackage{fca}
643 \let\fca@tikz@without@library\relax
644 \def\fca@generate@tikz@parameter#1#2#3{%
645   \pgfkeysdef{/fca/#1}{\pgfkeys{/tikz/#1=##1}}
646   \pgfkeyssetvalue{/fca/#1}{#3}%
647 }
```

10.3   Backports and bugfixes for Ti*k*Z

```
648 \@ifpackagelater{tikz}{2020/09/28}{}{%
649   \PackageWarning{fca}{Your TikZ version does not fully support fca
650     name spaces.^^J
651     I'm trying to fix that..., trying to apply^^J
652     patch 88951be592b558b94b14a97aaffe9df6c1ce61ee from TikZ}
653   \def\tikz@calc@anchor#1.#2\tikz@stop{%
654     % Check if a shape with name prefix exists, otherwise try the global name
655     % without prefix.
656     \ifcsname pgf@sh@ns@\tikz@pp@name{#1}\endcsname%
657       \pgfpointanchor{\tikz@pp@name{#1}}{#2}%
658     \else
659       \pgfpointanchor{#1}{#2}%
660     \fi
661   }%
662 \def\tikz@subpicture@handle@#1{
663   \pgfkeys{/tikz/pics/.cd,#1}%
664   \tikz@node@transformations%
665   \let\tikz@transform=\relax%
666   \let\tikz@picmode\tikz@mode%
667   \tikzset{name prefix ../.style/.expanded={/tikz/name prefix=\pgfkeysvalueof{/tikz/name pre
668   \ifx\tikz@fig@name\pgfutil@empty\else%
669     \tikzset{name prefix/.expanded=\tikz@fig@name}%
670   \fi%
671   \pgfkeysvalueof{/tikz/pics/setup code}%
672   \pgfkeysgetvalue{/tikz/pics/code}{\tikz@pic@code}
673   \ifx\tikz@pic@code\pgfutil@empty\else%
674   \setbox\tikz@whichbox=\hbox\bgroup%
675     \unhbox\tikz@whichbox%
676       \hbox\bgroup
677       \bgroup%
678       \pgfinterruptpath%
679       \pgfscope%
680       \tikz@options%
681       \setbox\tikz@figbox=\box\pgfutil@voidb@x%
682       \setbox\tikz@figbox@bg=\box\pgfutil@voidb@x%
683       \tikz@atbegin@scope%
684       \scope[every pic/.try]%
685       \tikz@pic@code%
686       \endscope%
687       \tikz@atend@scope%
688       \endpgfscope%
689       \endpgfinterruptpath%
690       \egroup
691       \egroup%
692       \egroup%
```

74

```
693    \fi%
694    \pgfkeysgetvalue{/tikz/pics/foreground code}{\tikz@pic@code}
695    \ifx\tikz@pic@code\pgfutil@empty\else%
696       \setbox\tikz@figbox=\hbox\bgroup%
697       \unhbox\tikz@figbox%
698       \hbox\bgroup
699       \bgroup%
700       \pgfinterruptpath%
701       \pgfscope%
702       \tikz@options%
703       \setbox\tikz@figbox=\box\pgfutil@voidb@x%
704       \setbox\tikz@figbox@bg=\box\pgfutil@voidb@x%
705       \tikz@atbegin@scope%
706       \scope[every front pic/.try]%
707       \tikz@pic@code%
708       \endscope%
709       \tikz@atend@scope%
710       \endpgfscope%
711       \endpgfinterruptpath%
712       \egroup
713       \egroup%
714       \egroup%
715    \fi%
716    \pgfkeysgetvalue{/tikz/pics/background code}{\tikz@pic@code}
717    \ifx\tikz@pic@code\pgfutil@empty\else%
718       \setbox\tikz@figbox@bg=\hbox\bgroup%
719       \unhbox\tikz@figbox@bg%
720       \hbox\bgroup
721       \bgroup%
722       \pgfinterruptpath%
723       \pgfscope%
724       \tikz@options%
725       \setbox\tikz@figbox=\box\pgfutil@voidb@x%
726       \setbox\tikz@figbox@bg=\box\pgfutil@voidb@x%
727       \tikz@atbegin@scope%
728       \scope[every behind pic/.try]%
729       \tikz@pic@code%
730       \endscope%
731       \tikz@atend@scope%
732       \endpgfscope%
733       \endpgfinterruptpath%
734       \egroup
735       \egroup%
736       \egroup%
737    \fi%
738    \tikz@node@finish%
739  }%
740 \def\tikz@parse@node#1(#2){%
741    \pgfutil@in@.{#2}% Ok, flag this
742    \ifpgfutil@in@
743       \tikz@calc@anchor#2\tikz@stop%
744    \else%
745       \tikz@calc@anchor#2.center\tikz@stop% to be on the save side, in
746                                 % case iftikz@shapeborder is ignored...
```

```
747    \ifcsname pgf@sh@ns@\tikz@pp@name{#2}\endcsname
748      \expandafter\ifx\csname pgf@sh@ns@\tikz@pp@name{#2}\endcsname\tikz@coordinate@text%
749      \else
750        \tikz@shapebordertrue%
751        \def\tikz@shapeborder@name{\tikz@pp@name{#2}}%
752      \fi%
753    \else\ifcsname pgf@sh@ns@#2\endcsname
754      \expandafter\ifx\csname pgf@sh@ns@#2\endcsname\tikz@coordinate@text%
755      \else
756        \tikz@shapebordertrue%
757        \def\tikz@shapeborder@name{#2}%
758      \fi%
759    \fi\fi
760  \fi%
761  \edef\tikz@marshal{\noexpand#1{\noexpand\pgfqpoint{\the\pgf@x}{\the\pgf@y}}}%
762  \tikz@marshal%
763 }%
764 }%
```

Options that are directly forwarded to TikZ   If they are not documented
somewhere else, their implementation needs `fca.sty` fröm the TikZlibrary `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 TikZ for filling paths. The algorithm is described in the [TikZ 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

76

Opt	`blend mode`	**blend mode:** This option defines, how thransparency changes the colors of semi-transparent objects. See the [TikZ 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

```
765 \fca@generate@tikz@parameters
766 \@for \@tempa:=%
767 shape,
768 text ragged,%
769 text badly ragged,%
770 text ragged left,%
771 text badly ragged left,%
772 text justified,%
773 text centered,%
774 text badly centered,%
775 align,%
776 even odd rule,%
777 nonzero rule,%
778 fill opacity,%
779 opacity,%
780 blend mode,%
781 color,%
782 rotate,%
783 solid,%
784 dotted,%
785 loosely dotted,%
786 densely dotted,%
787 line width,%
788 x,%
789 y%
790 \do {
791    \edef\@tempb{%
792      \noexpand\pgfkeysdef{/fca/\@tempa}{\noexpand\pgfqkeys{/tikz}{\@tempa={##1}}}%
793    }%
794    \@tempb
795 }
796 \pgfkeysdef{/fca/options}{\tikz@addoption{#1}}
797 \pgfkeysdef{/fca/stroke}{%
798    \pgfqkeys{/tikz}{draw={#1}}%
799    \fca@testoption{#1}{%
800      \fca@append\fca@usepath{stroke,}%
801    }{}{%
802      \fca@append\fca@usepath{stroke,}%
803    }%
```

77

```
804 }
805 \pgfkeysdef{/fca/fill}{%
806   \pgfqkeys{/tikz}{fill={#1}}%
807   \fca@testoption{#1}{%
808     \fca@append\fca@usepath{fill,}%
809   }{}{%
810     \fca@append\fca@usepath{fill,}%
811   }%
812 }
813 \def\fca@options{\tikz@options}%
814 \def\fca@defaultoptions{\tikz@options}%
815 \def\fca@transform{\tikz@transform}%
816 \@ifundefined{tikz@transform}{\let\tikz@transform\empty}{}
817 \let\fca@ifintikz\@secondoftwo
818 \tikzaddtikzonlycommandshortcutlet\fca@ifintikz\@firstoftwo
819 \pgfqkeys{/fca/label}{%
820   attributes left/.append style={
821     /tikz/.cd,
822     align=flush right,
823     /fca/.cd
824   },
825   attributes center/.append style={%
826     /tikz/.cd,
827     align=flush center,
828     /fca/.cd%
829   },
830   attributes right/.append style={
831     /tikz/.cd,
832     align=flush left,
833     /fca/.cd
834   },
835   objects left/.append style={
836     /tikz/.cd,
837     align=flush right,
838     /fca/.cd
839   },
840   objects center/.append style={
841     /tikz/.cd,
842     align=flush center,
843     /fca/.cd
844   },
845   objects right/.append style={
846     /tikz/.cd,
847     align=flush left,
848     /fca/.cd
849   }
850 }

851 \let\fca@notikz@parse@paren@vector\fca@parse@paren@vector
852 \newcommand\fca@tikz@parse@paren@vector[3]{%
853   \def\tikz@coordinate@caller{%
854     \fca@notikz@parse@paren@vector{#1}{#2}{#3}%
855   }%
856   \tikz@scan@one@point \tikz@@coordinate@at@math
```

```
857 }
858
859 \newcommand\fca@tikz@oldnode[4]{%
860   \begin{pgfscope}%
861     \fca@nodeslayer
862     \let\tikz@text@width\pgfutil@empty
863     \path (#3,#4) node[
864       anchor=center,
865       shape=circle,
866       line width=\fca@node@thickness,
867       /fca/.cd,
868       /fca/every node/.try,
869      /fca/every concept/.try,
870      #1]
871      (#2) {} ;%
872     \fca@endnodeslayer
873    \iffca@draft
874     \ifthenelse{\boolean{fca@CircledNumbers}}{%
875      \fca@nodenameslayer
876      \path[overlay] (#2)
877      node[
878      shape=circle,
879      anchor=center,
880      /fca/.cd,
881      /fca/every node/.try,
882      /fca/every concept/.try,
883      #1
884      /fca/.cd,
885      /fca/node number,
886      draw=none]%
887      (#2 number)
888      {#2};%
889      \fca@endnodenameslayer
890     }{}%
891    \fi
892   \end{pgfscope}%
893   \ignorespaces
894 }
```

Example 39: Testing lines with TikZ

Code:

```
  \definecolor{darkgreen}{rgb}{0.05,0.5,0.}
\begin{tikzdiagram}[x=1.2mm,y=1.2mm]
\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)
```
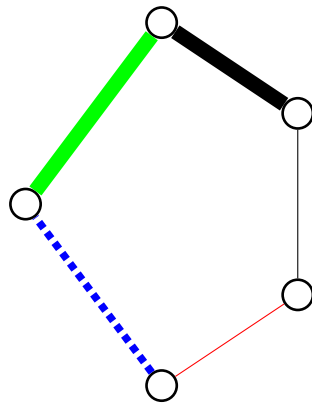
```
 \EdgeThickness{2mm}
 \Edge[draw=green](3)(5)
 \Edge(4)(5)
\end{tikzdiagram}
```

Result:

```
895 \newcommand{\fca@tikz@oldedge}[3]{%
896   \fca@edgeslayer
897   \path[draw,/fca/.cd,every edge/.try,#1] (#2) edge (#3);%
898   \fca@endedgeslayer
899   \ignorespaces
900 }
901
902 \newcommand\fca@tikz@labelBox@label[1]{%
903   \typeout{fca@tikz@labelBox@label: |attributes\space \fca@label@position|}%
904   \fca@labelslayer
905   \path[/fca/.cd,every node/.try,every label/.try,#1]
906   (\fca@temp@node@name.\fca@label@at)
907   ++(\fca@label@shift@x,\fca@label@shift@y)
908   node[/fca/.cd,every node/.try,/tikz/.cd,shape=rectangle,/fca/.cd,every label/.try,#1] (%
909     \pgfkeysvalueof{/fca/label/name prefix}%
910     \pgfkeysvalueof{/fca/node}%
911     \pgfkeysvalueof{/fca/label/name suffix}%
912   ){%
913     \pgfkeysvalueof{/fca/node contents}%
914   };%
915   \fca@endlabelslayer
916 }
917
918 \newcommand\fca@tikz@labelBox@connector[1]{%
919   \typeout{fca@tikz@labelBox@connector}%
920   \begin{scope}
921     \fca@connectorslayer
922     \path[draw,/fca/.cd,#1,every label edge/.try,/tikz/.cd]
923     (\pgfkeysvalueof{/fca/node}) --
924     (%
925     \pgfkeysvalueof{/fca/label/name prefix}%
926     \pgfkeysvalueof{/fca/node}%
```

```
927    \pgfkeysvalueof{/fca/label/name suffix}%
928    .\pgfkeysvalueof{/fca/anchor}) ;
929    \fca@endconnectorslayer
930  \end{scope}%
931 }
932
933 \newcommand\fca@tikz@labelBox@concept[1]{%
934  \typeout{fca@tikz@labelBox@concept}%
935  \fca@labelconceptslayer
936  % \edef\fca@tempa{
937  \path
938  (\fca@temp@node@name.\fca@label@at)%
939  node[draw,
940  /fca/.cd,
941  shape=coordinate,
942  #1,
943  every label concept/.try,
944  label concept/.try]
945  (\pgfkeysvalueof{/fca/label/name prefix}%
946  \pgfkeysvalueof{/fca/node}%
947  \pgfkeysvalueof{/fca/label/name suffix}%
948  ){};
949  % \pgfsetstrokecolor{\fca@node@color}%
950  % \pgfnode{}{\pgfkeysvalueof{/fca/anchor}}{}{}%
951  \fca@endlabelconceptslayer
952 }
953
954 \newcommand\fca@tikz@labelBox[1]{%
955  \begin{scope}%
956    \fcaset{%
957      draw/.forward to=/tikz/draw,
958      every node/.try,every label/.try,#1}%
959    \edef\@tempa{\pgfkeysvalueof{/fca/pgfnode}}%
960    \xdef\fca@temp@node@name{%
961      \expandafter\fca@remove@anchor\expandafter{\@tempa}%
962    }%
963    \fca@tikz@labelBox@label{#1}%
964    \iffca@connectors
965      \fca@tikz@labelBox@connector{#1}%
966    \fi
967    \begin{scope}
968      \fcaset{shape=coordinate,#1}%
969      \fcaset{%
970        every label concept/.try}%
971      \fcaset{%
972        label concept/.try
973      }%
974      \def\@tempa{coordinate}%
975      \ifx\@tempa\tikz@shape
976      \else
977        \fca@tikz@labelBox@concept{#1}%
978      \fi
979    \end{scope}
980  \end{scope}%
```

```
981   \typeout{end fca@tikz@labelBox}%
982   \ignorespaces
983 }
984 \newcommand\fca@tikz@startdiagram[1][]{%
985   \fca@notikz@startdiagram%
986   \fca@tikz@diagram@
987 }
988 \newcommand\fca@tikz@Defaults{
989   \fca@notikz@Defaults
990   \pgfkeysgetvalue{/tikz/name prefix}{\fca@tikz@origprefix}%
991   \pgfkeysgetvalue{/fca/name prefix}{\@tempb}%
992   \let\@tempa\fca@tikz@origprefix
993   \fca@concat\@tempa\@tempb%
994   \pgfqkeys{/tikz}{%
995     name prefix/.expand once=\@tempa
996   }%
997   \pgfkeysgetvalue{/fca/name suffix}{\@tempa}%
998   \pgfkeysgetvalue{/tikz/name suffix}{\@fca@tikz@origsuffix}%
999   \fca@concat\@tempa\@fca@tikz@origsuffix%
1000   \pgfqkeys{/tikz}{%
1001     name suffix/.expand once=\@tempa
1002   }%
1003   \pgfkeyssetvalue{/fca/name prefix}{%
1004     \pgfkeysvalueof{/tikz/name prefix}%
1005   }%
1006   \pgfkeyssetvalue{/fca/name suffix}{%
1007     \pgfkeysvalueof{/tikz/name suffix}%
1008   }%
1009 }
1010 \tikzaddtikzonlycommandshortcutlet\fca@oldnode\fca@tikz@oldnode
1011 \tikzaddtikzonlycommandshortcutlet\fca@oldedge\fca@tikz@oldedge
1012 \tikzaddtikzonlycommandshortcutlet\fca@@labelBox\fca@tikz@labelBox
1013 \tikzaddtikzonlycommandshortcutlet\fca@parse@paren@vector\fca@tikz@parse@paren@vector
1014 \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 docu-
mentation can be found in the PGF/TikZ Manual in `pgfmanual.pdf` of your PGF
documentation.

```
1015 \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 TikZ** Stores the alignment setup macros TikZ nodes.
Opt	/fca/options	**/fca/options: Undocumented extension to TikZ** 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 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*⟩)} ⚠ changes in 3.0	**/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. 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: ```\n{\n text=pgfstrokecolor,\n /pgf/outer sep=0pt,\n /pgf/inner sep=0pt,\n}\n```
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. |

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

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

/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.

/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.

/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.

/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.

/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.

/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.

```
1016
1017 \newcommand\fca@append[2]{%
1018   \expandafter\def\expandafter#1\expandafter{#1#2}%
1019 }
1020 \newcommand\fca@concat[2]{%
1021   \expandafter\fca@append\expandafter#1\expandafter{#2}%
1022 }
1023
1024 \newcommand\fca@node@fill{white}
1025
1026 \fcaset{%
1027   connector/.is if=fca@connectors,
1028   namespace/.initial=fca,
1029   namespace/.value required,
1030   name prefix/.initial=\pgfkeysvalueof{/fca/namespace}\space node\space,
1031   name prefiv/.value required,
1032   name suffix/.initial={},
1033   name suffix/.value required,
1034   every node/.style = {},
1035   every concept/.style = {
1036     radius = 2mm,
1037     fill=\fca@node@fill,
1038     draw=black
```

```
1039     },
1040     every attributes/.style = {
1041       font=\AttributeLabelStyle
1042     },%\newcommand*{\fca@node@fill}{white}%%\newcommand*{\fca@node@fill}{white}%
1043     every objects/.style = {
1044       font=\ObjectLabelStyle
1045     },
1046     every edge/.style = {},
1047     every label edge/.style = { dotted, draw=black },
1048     font=\small\baselineskip1em\rmfamily,
1049     shape/.initial=circle,
1050     minimum width/.forward to=/pgf/minimum width,
1051     minimum height/.forward to=/pgf/minimum height,
1052     minimum size/.forward to=/pgf/minimum size,
1053     inner xsep/.forward to=/pgf/inner xsep,
1054     inner ysep/.forward to=/pgf/inner ysep,
1055     inner sep/.forward to=/pgf/inner sep,
1056     outer xsep/.forward to=/pgf/outer xsep,
1057     outer ysep/.forward to=/pgf/outer ysep,
1058     outer sep/.forward to=/pgf/outer sep,
1059     radius/.code={
1060       \@tempdimb\dimexpr\pgf@xx+\pgf@yx\relax
1061       \fca@parselength\@tempdima{#1}{\@tempdimb}%
1062       \@tempdima=.70710678118654752440084436210484\@tempdima
1063       \edef\@tempa{\noexpand\pgfkeysalso{/pgf/inner sep=\the\@tempdima}}%
1064       \@tempa
1065     },
1066     % anchor/.store in=\fca@label@anchor,
1067     % anchor/.value required,
1068     node number/.style = { node font=\tiny, font=\tiny },
1069     color/.code={\fca@append\fca@options{\color{#1}}},
1070     line width/.code={%
1071       \pgfmathsetlength\pgflinewidth{#1}%
1072       \fca@append\fca@options{\pgfsetlinewidth{#1}}%
1073     },
1074     stroke/.code={%
1075       \fca@testoption{#1}{%
1076         \fca@append\fca@options{%
1077           \pgfsetstrokecolor{#1}%
1078         }%
1079         \fca@append\fca@usepath{stroke,}%
1080       }{%
1081         \fca@append\fca@options{%
1082           \let\pgf@up@stroke\pgfutil@empty
1083         }%
1084       }{%
1085         \fca@append\fca@usepath{stroke,}%
1086       }%
1087     },
1088     draw/.forward to=/fca/stroke,
1089     fill/.code={%
1090       \fca@testoption{#1}{%
1091         \fca@append\fca@options{%
1092           \pgfsetfillcolor{#1}%
```

```
1093        }%
1094        \fca@append\fca@usepath{fill,}%
1095      }{%
1096        \fca@append\fca@options{%
1097          \let\pgf@up@fill\pgfutil@empty
1098        }%
1099      }{%
1100        \fca@append\fca@usepath{fill,}%
1101      }%
1102    },
1103    text/.initial=pgfstrokecolor,
1104    text/.value required,
1105    label type/.forward to=/fca/label/type,
1106    label position/.forward to=/fca/label/position,
1107    pgfnode/.initial={%
1108      \pgfkeysvalueof{/fca/name prefix}%
1109      \pgfkeysvalueof{/fca/node}%
1110      \pgfkeysvalueof{/fca/name suffix}%
1111    },
1112    pgfnode/.value required,
1113    node/.initial={},
1114    node/.value required,
1115    node contents/.initial={},
1116    node contents/.value required,
1117    shift/x/.store in=\fca@label@shift@x,
1118    shift/x/.value required,
1119    shift/y/.store in=\fca@label@shift@y,
1120    shift/y/.value required,
1121    shift/.value required,
1122    shift/.style args={(#1,#2)}{%
1123      /fca/shift/x=#1,
1124      /fca/shift/y=#2
1125    },%
1126    shift/x/signv2.1/.store in=\fca@label@shift@x@sign,
1127    shift/x/signv2.1/.value required,
1128    shift/y/signv2.1/.store in=\fca@label@shift@y@sign,
1129    shift/y/signv2.1/.value required,
1130    rotate/.code={%
1131      \fca@append\fca@transform{%
1132        \pgftransformrotate{#1}%
1133      }%
1134    },
1135    solid/.code={%
1136      \fca@append\fca@options{%
1137        \pgfsetdash{}{0pt}%
1138      }%
1139    },%
1140    dotted/.code={%
1141      \fca@append\fca@options{%
1142        \pgfsetdash{{\pgflinewidth}{2pt}}{0pt}%
1143      }%
1144    },%
1145    loosely dotted/.code={%
1146      \fca@append\fca@options{%
```

```
1147        \pgfsetdash{{\pgflinewidth}{4pt}}{0pt}%
1148      }%
1149    },%
1150    densely dotted/.code={%
1151      \fca@append\fca@options{%
1152        \pgfsetdash{{\pgflinewidth}{4pt}}{0pt}%
1153      }%
1154    }%
1155  }
1156  \fcaset{%
1157    every label/.style = {
1158      text=pgfstrokecolor,
1159      /pgf/outer sep=0pt,
1160      /pgf/inner sep=0pt,
1161    }
1162  }
1163
1164  \pgfqkeys{/fca/node}{%
1165    line width/.forward to=/fca/line width
1166    line width/.value required,
1167    radius/.style={
1168      /fca/radius=#1
1169    },
1170    color/.forward to=/fca/fill,
1171    layer/.code=\fcalayer{node},%
1172    numbers/.is if=fca@CircledNumbers,
1173    numbers/.default=true,
1174    number prefix/.store in=\fca@node@number@prefix,
1175    number prefix/.value required,
1176    number suffix/.store in=\fca@node@number@suffix,
1177    number suffix/.value required,
1178  }
1179
1180  % Line thickness in standard \LaTeX{}:
1181  % thin lines are .4pt
1182  % thick lines are 0.8pt
1183
1184  % Dotted: \pgfsetdash{{\pgflinewidth}{2pt}}
1185
1186  \pgfqkeys{/fca/label}{%
1187    name prefix/.initial=\pgfkeysvalueof{/fca/namespace}\space node\space,
1188    name prefix/.value required,
1189    name suffix/.initial=\space\fca@label@type\space\fca@label@position,
1190    name suffix/.value required
1191    fill/.store in=\fca@node@fill,
1192    fill/.value required,
1193    text width/.store in=\fca@label@text@width,
1194    text width/.value required,
1195    node font/.initial={},
1196    node font/.value required,
1197    edge width/.store in=\fca@label@edge@width,
1198    edge width/.value required,
1199    at/.store in=\fca@label@at,
1200    at/.value required,
```

```
1201  type/.store in=\fca@label@type,
1202  type/.is choice,
1203  type/.append code={%
1204    \def\fca@label@type{#1}% the choice does not store the value
1205    \pgfqkeysalso{/fca}{every #1/.try}%
1206    \pgfqkeysalso{/fca/label}{\fca@label@type\space\fca@label@position}%
1207  },
1208  type/attributes/.style={},
1209  type/objects/.style={},
1210  position/.store in=\fca@label@position,
1211  position/.is choice,
1212  position/.append code={%
1213    \def\fca@label@position{#1}% the choice does not store the value
1214    \pgfqkeysalso{/fca/label}{\fca@label@type\space\fca@label@position}%
1215  },
1216  position/left/.style={},
1217  position/right/.style={},
1218  position/center/.style={},
1219  %
1220  attributes left/.style={
1221    at=north west,
1222    /fca/anchor=south east,
1223    /fca/shift={(-1pt,1pt)},
1224    /fca/shift/x/signv2.1=-,
1225    /fca/shift/y/signv2.1={},
1226    /fca/font/.append=\raggedleft
1227  },
1228  attributes center/.style={
1229    at=north,
1230    /fca/anchor=south,
1231    /fca/shift={(0,1pt)},
1232    /fca/shift/x/signv2.1={},
1233    /fca/shift/y/signv2.1={},
1234    /fca/font/.append=\centering
1235  },
1236  attributes right/.style={
1237    at=north east,
1238    /fca/anchor=south west,
1239    /fca/shift={(1pt,1pt)},
1240    /fca/shift/x/signv2.1={},
1241    /fca/shift/y/signv2.1={},
1242    /fca/font/.append=\raggedright
1243  },
1244  objects left/.style={
1245    at=south west,
1246    /fca/anchor=north east,
1247    /fca/shift={(-1pt,-1pt)},
1248    /fca/shift/x/signv2.1=-,
1249    /fca/shift/y/signv2.1=-,
1250    /fca/font/.append=\raggedleft
1251  },
1252  objects center/.style={
1253    at=south,
1254    /fca/anchor=north,
```

```
1255    /fca/shift={(0,-1pt)},
1256    /fca/shift/x/signv2.1={},
1257    /fca/shift/y/signv2.1=-,
1258    /fca/font/.append=\centering
1259  },
1260  objects right/.style={
1261    at=south east,
1262    /fca/anchor=north west,
1263    /fca/shift={(1pt,-1pt)},
1264    /fca/shift/x/signv2.1={},
1265    /fca/shift/y/signv2.1=-,
1266    /fca/font/.append=\raggedright
1267  }
1268 }
1269
1270 \newcommand*\fca@tikz@without@library{%
1271  \PackageWarning{fca}{FCA TikZ integration is not activated.^^J
1272    You should consider \tikzlibrary{fca} instead of
1273    \string\usepackage{fca}
1274  }%
1275 }
1276
1277 \AtBeginDocument{%
1278  \@ifundefined{tikz@color}{}{\fca@tikz@without@library}
1279 }
1280
1281
1282 \newcommand*{\fcaNodeThickness}[1]{%
1283  \def\fca@node@thickness{#1}%
1284  \ignorespaces
1285 }% %
1286 \newcommand*{\fcaEdgeThickness}[1]{%
1287  \fcaset{every edge/.append style={%
1288      line width={#1}%
1289    }%
1290  }%
1291  \ignorespaces
1292 }
1293 \newcommand*{\fcaNodeColor}[1]{%
1294  \def\fca@node@fill{#1}%
1295  \ignorespaces
1296 }%
1297 \iffca@draft
1298  \newcommand*{\fcaNumbers}{%
1299    \setboolean{fca@CircledNumbers}{true}%
1300  }%
1301 \else
1302  \newcommand*{\fcaNumbers}{%
1303    \setboolean{fca@CircledNumbers}{false}%
1304  }
1305 \fi
1306 \newcommand*{\fcaNoNumbers}{%
1307  \setboolean{fca@CircledNumbers}{false}%
1308 }%
```

```
1309 \newcommand*{\fcaCircleSize}[1]{%
1310   \fcaset{every concept/.append style={radius=(#1)*0.5\unitlength}}%
1311   \ignorespaces
1312 }%
1313 \iffca@compat@macros
1314   \def\fcaObjectLabelStyle{\fcaObjectLabelStyle}%
1315   \def\fcaAttributeLabelStyle{\AttributeLabelStyle}%
1316   \def\fcaLabelBoxWidth{\LabelBoxWidth}%
1317   \def\EdgeThickness{\fcaEdgeThickness}%
1318   \def\NodeThickness{\fcaNodeThickness}%
1319   \def\NodeColor{\fcaNodeColor}%
1320   \def\Numbers{\fcaNumbers}%
1321   \def\NoNumbers{\fcaNoNumbers}
1322   \def\CircleSize{\fcaCircleSize}%
1323   \def\NoDots{\fcaNoDots}
1324   \def\Dots{\fcaDots}
1325 \fi
```

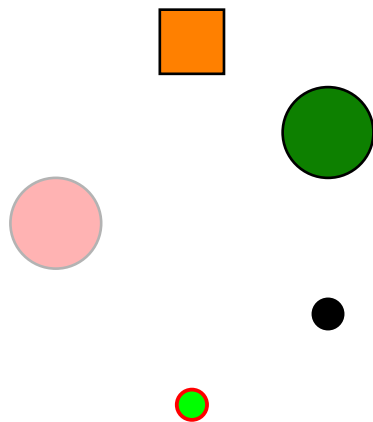## Example 40: Testing node parameters

Code:

```
{\unitlength 1.2mm
\definecolor{darkgreen}{rgb}{0.05,0.5,0.}
\begin{diagram}
 \Node[draw=red, fill=green,/tikz/line width=.5mm](1)(20,10)
 \Node[fill=black, radius=2mm](2)(35,20)
 \fcaCircleSize{10}%
 \Node[opacity=0.3,fill=red](3)(5,30)
 \fcaset{node/line width=2mm}
 \NodeColor{darkgreen}
 \Node(4)(35,40)
 \fcaNodeColor{orange}
 \Node[/tikz/rectangle](5)(20,50)
\end{diagram}}
```

Result:

```latex
1326 \newcounter{fca@minNode}%
1327 \newcounter{fca@maxNode}%
1328 \newcounter{fca@runNode}%
1329 \newboolean{fca@CircledNumbers}%
1330 \newcounter{fca@CircleDiameter}%
1331 \newcounter{fca@AuxCounter}%
1332 \newcounter{fca@BuxCounter}%
1333 \def\fca@adjNode#1{%
1334    \ifthenelse{#1<\value{fca@minNode}}{\setcounter{fca@minNode}{#1}}{}%
1335    \ifthenelse{\value{fca@maxNode}<#1}{\setcounter{fca@maxNode}{#1}}{}}%
1336 \newcommand\fca@typeset@node@content[1]{%
1337    \pgfkeysgetvalue{/fca/text width}\@tempa
1338    \edef\@tempb{\@tempa}%
1339    \expandafter\fca@testoption\expandafter{\@tempb}{%
1340       \def\fca@node@end@content{\end{minipage}}%
1341       \fca@parselength\@tempdima{\fca@tempa}\fca@xunitlength
1342       \begin{minipage}{\@tempdima}%
1343    }{%
1344       \let\fca@node@end@content\egroup
1345       \makebox\bgroup%
1346    }{%
1347       \let\fca@node@end@content\egroup
1348       \makebox\bgroup
1349    }%
1350    \pgfkeysvalueof{/fca/font}%
1351    \color{\pgfkeysvalueof{/fca/text}}%
1352    #1%
1353    \fca@node@end@content%
1354 }
1355 \newcommand\fca@node[1][]{%
1356    \@ifnextchar\bgroup{\fca@oldnode{#1}}{%
1357       \@ifnextchar({\fca@pictnode{#1}}{\fca@tikznode{#1}}%)
1358    }%
1359 % \egroup
1360 }
1361 \newcommand\fca@pictnode[1]{%
1362    \fca@parse@parenlabel{\fca@pictn@de{#1}}%
1363 }
1364 \newcommand\fca@pictn@de[2]{%
1365    \fca@parse@paren@vector{%
1366       \edef\fca@tempa{{\the\@tempdima}{\the\@tempdimb}}%
1367       \def\fca@tempb{\fca@oldnode{#1}{#2}}%
1368       \expandafter\fca@tempb\fca@tempa
1369    }\@tempdima\@tempdimb
1370 }
1371 \newcommand\fca@oldnode[4]{%
1372    \begin{pgfscope}%
1373       \fcaset{shape=circle,
1374          line width=\fca@node@thickness,
1375          color/.forward to=/fca/fill}%
1376       \fcaset{every node/.try}%
1377       \fcaset{every concept/.try}%
1378       \fcaset{#1}%
1379       \fca@nodeslayer
```

95

```
1380     \pgfkeysvalueof{/fca/node font}%
1381     \fca@parselength\@tempdima{#3}\unitlength
1382     \fca@parselength\@tempdimb{#4}\unitlength
1383     \pgftransformshift{\pgfpoint{\@tempdima}{\@tempdimb}}%
1384     \fca@options
1385     % \pgfsetstrokecolor{\fca@node@color}%
1386     \pgfnode{\pgfkeysvalueof{/fca/shape}}{center}{}{%
1387       \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name suffix}%
1388     }{\expandafter\pgfusepath\expandafter{\fca@usepath}}%
1389     \fca@endnodeslayer
1390     \ifthenelse{\boolean{fca@CircledNumbers}}{%
1391       \begin{pgfscope}
1392         \fca@nodenameslayer
1393         \pgf@relevantforpicturesizefalse
1394         \def\fca@usepath{}%
1395         \fcaset{node number}%
1396         \fca@options \pgfnode{\pgfkeysvalueof{/fca/shape}}{center}{%
1397           \fca@typeset@node@content {%
1398             \centering \pgfkeysvalueof{/fca/font}%
1399             #2%
1400           }%
1401         }{%
1402           \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name
1403             suffix} number%
1404         }{\expandafter\pgfusepath\expandafter{\fca@usepath}}%
1405         \fca@endnodenameslayer
1406       \end{pgfscope}
1407     }{}%
1408   \end{pgfscope}%
1409   \ignorespaces
1410 }
1411 \newcommand{\fca@edge}[1][]{%
1412   \@ifnextchar\bgroup{\fca@oldedge{#1}}{\fca@newedge{#1}}%
1413 % \egroup
1414 }
1415 \newcommand\fca@newedge[1]{%
1416   \fca@parse@parenlabel{\fca@new@dge{#1}}%
1417 }
1418 \newcommand\fca@new@dge[2]{%
1419   \fca@parse@parenlabel{\fca@oldedge{#1}{#2}}%
1420 }
```

Example 41: 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)
```
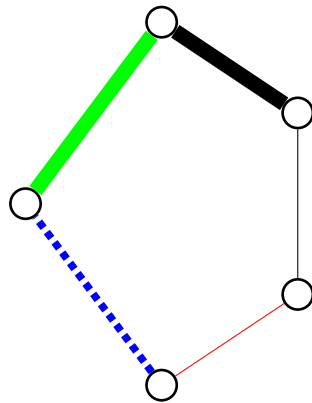
```
\Node(5)(20,50)
{\color{red}\Edge(1)(2)}
\Edge[draw=blue,dotted,line width=1mm](1)(3)
\Edge(2)(4)
\EdgeThickness{2mm}
\Edge[draw=green](3)(5)
\Edge(4)(5)
\end{diagram}}
```

Result:

```
1421  \newcommand{\fca@oldedge}[3]{%
1422    \begin{pgfscope}%
1423      \fca@edgeslayer
1424      \fcaset{every edge/.try,#1}%
1425      \fca@options
1426      \fca@transform
1427      %\pgfsetlinewidth{\pgfkeysgetvalue{/fca/line width}}%
1428      \pgfpathmoveto{%
1429        \pgfpointshapeborder{%
1430          \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name suffix}%%
1431        }{%
1432          \pgfpointanchor{%
1433            \pgfkeysvalueof{/fca/name prefix}#3\pgfkeysvalueof{/fca/name suffix}%%
1434          }{center}%
1435        }%
1436      }%
1437      \pgfpathlineto{%
1438        \pgfpointshapeborder{%
1439          \pgfkeysvalueof{/fca/name prefix}#3\pgfkeysvalueof{/fca/name suffix}%%
1440        }{%
1441          \pgfpointanchor{%
1442          \pgfkeysvalueof{/fca/name prefix}#2\pgfkeysvalueof{/fca/name suffix}%%
1443          }{center}%
1444        }%
1445      }%
1446      \pgfusepath{stroke}%
1447      \fca@endedgeslayer
1448    \end{pgfscope}%
```

```
1449   \ignorespaces
1450 }
1451 %
1452 \def\fca@changeaValue#1#2#3{\setcounter{fca@AuxCounter}{#1}%
1453   \addtocounter{fca@AuxCounter}{#2}\def#3{\value{fca@AuxCounter}}}%
1454 \def\fca@changebValue#1#2#3{\setcounter{fca@BuxCounter}{#1}%
1455   \addtocounter{fca@BuxCounter}{#2}\def#3{\value{fca@BuxCounter}}}%
1456 \newcommand{\fcaNoDots}{\setboolean{fca@connectors}{false}}
1457 \newcommand{\fcaDots}{\setboolean{fca@connectors}{true}}
```
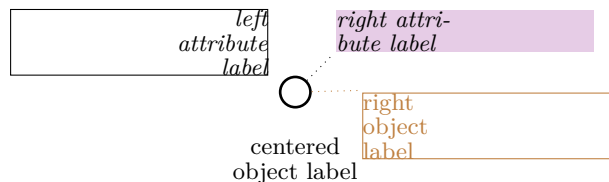

Example 42: Testing nodots

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:

```
1458 \def\fca@parse@parenlabel#1(#2){#1{#2}}
1459 \def\fca@parse@paren@vector#1#2#3(#4,#5){%
1460   \fca@parse@brace@vector{#1}{#2}{#3}{#4}{#5}%
1461 }
```

\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.

```
1462 \newcommand*\fca@parse@label[2]{%
1463   \@ifnextchar\bgroup{%\egroup
1464     #1%
```

```
1465   }{%
1466     \@ifnextchar(%)
1467     {%
1468       \fca@parse@parenlabel{#1}%
1469     }{%
1470       #2%
1471     }%
1472   }%
1473 }
1474 \newcommand*\fca@parse@vector[4]{%
1475   \@ifnextchar(%)
1476   {%
1477     \fca@parse@paren@vector{#1}{#3}{#4}%
1478   }{%
1479     \@ifnextchar\bgroup{%\egroup
1480       \fca@parse@brace@vector{#1}{#3}{#4}%
1481     }{#2}%
1482   }%
1483 }
1484 \newcommand*\fca@parse@brace@vector[5]{%
1485   \fca@parselength#2{#4}\fca@xunitlength
1486   \fca@parselength#3{#5}\fca@yunitlength
1487   #1%
1488 }
```

\fca@labelBox $[\langle options\rangle]$

\fca@labelBox $[\langle options\rangle]\{\langle label\rangle\}$

\fca@labelBox $[\langle options\rangle]\{\langle label\rangle\}\{\langle content\rangle\}$

\fca@labelBox $[\langle options\rangle]\{\langle label\rangle\}\{\langle xoffset\rangle\}\{\langle yoffset\rangle\}\{\langle content\rangle\}$

\fca@labelBox $[\langle options\rangle]\{\langle label\rangle\}(\langle xoffset,yoffset\rangle)\{\langle content\rangle\}$

\fca@labelBox $[\langle options\rangle](\langle label\rangle)\{\langle content\rangle\}$

\fca@labelBox $[\langle options\rangle](\langle label\rangle)(\langle xoffset,yoffset\rangle)\{\langle content\rangle\}$

Example 43: Syntax test for labels

Code:

```
{\unitlength 1.2mm\relax
\fcadocDraft
\begin{diagram}{40}{55}
  \Numbers
  \Node{1}{20}{10}
  \Node{2}{35}{20}
```

99

```
        \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}
}
```
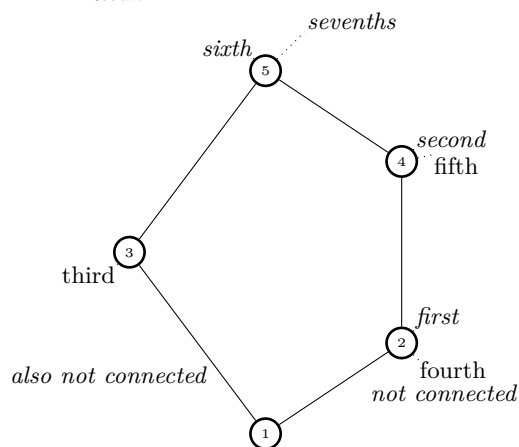

Result:

1489 \newcommand*\fca@labelBox[1][]{%
1490   \fca@parse@label{\fca@l@belBox{#1}}{\fca@@labelBox{#1}}%
1491 }
1492 \newcommand*\fca@l@belBox[2]{%
1493   \@ifnextchar\bgroup{%
1494   \fca@lab@lBox{#1}{#2}%
1495   }{%
1496   \@ifnextchar({%
1497   \fca@parse@vector{%
1498   \def\fca@tempa{outer sep=1pt,#1,node=#2,shift=}%

```
1499        \edef\fca@tempb{{(\the\@tempdima,\the\@tempdimb)}}%
1500        \fca@concat\fca@tempa\fca@tempb%
1501        \@ifnextchar\bgroup{%
1502          \expandafter\fca@l@b@lBox\expandafter{\fca@tempa}%
1503        }{%
1504          \expandafter\fca@@labelBox\expandafter{\fca@tempa}%
1505        }%
1506      }{}{\@tempdima}{\@tempdimb}%
1507    }{%
1508      \fca@@labelBox{#1,node={#2}}%
1509    }%
1510  }%
1511 }
1512 \newcommand*\fca@lab@lBox[3]{%
1513   \@ifnextchar\bgroup{%
1514     \fca@parse@vector{%
1515       \@ifnextchar\bgroup{%
1516         \fca@oldlabelBox{#1,node=#2}{\the\@tempdima}{\the\@tempdimb}%
1517       }{%
1518         \fca@oldlabelBox{#1,node=#2}{\the\@tempdima}{\the\@tempdimb}{}%
1519       }%
1520     }{}{\@tempdima}{\@tempdimb}{#3}%
1521   }{%
1522     \fca@@labelBox{#1,node={#2},node contents={#3}}%
1523   }%
1524 }
1525 \newcommand*\fca@l@b@lBox[2]{%
1526   \fca@@labelBox{#1,node contents=#2}
1527 }
1528 \newcommand*\fca@label@shift@sign@{}%
1529 \expandafter\newcommand\expandafter*\csname fca@label@shift@sign@+\endcsname[1]{+(#1)}
1530 \expandafter\newcommand\expandafter*\csname fca@label@shift@sign@-\endcsname[1]{-(#1)}
1531 \newcommand*\fca@addshiftsign[1]{%
1532   \expandafter\fca@@ddshiftsign\expandafter{\csname
1533   fca@label@shift@#1@sign\endcsname}
1534 }
1535 \newcommand*\fca@@ddshiftsign[1]{%
1536   \csname fca@label@shift@sign@#1\endcsname
1537 }
1538
1539 \newcommand\fca@oldlabelBox[4]{%
1540   \def\fca@tempa{%
1541     #1,
1542     label/at=center,
1543     shift=
1544   }%
1545   \edef\fca@tempb{%
1546     {(({\noexpand\fca@addshiftsign x{#2}},{\noexpand\fca@addshiftsign y{#3}}))}%
1547   }%
1548   \expandafter\expandafter\expandafter\fca@@labelBox
1549   \expandafter\expandafter\expandafter{%
1550     \expandafter\fca@tempa
1551     \fca@tempb,
1552     node contents={#4}%
```

```
1553    }%
1554 }
```

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

```
1555 \newcommand\fca@remove@anchor[1]{%
1556   \fca@remove@anch@r#1.\fca@remove@anchor%
1557 }
```

\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*⟩}}.

```
1558 \newcommand\fca@prepare@node[5]{%
1559   \def\@tempa{\fca@split@node@{#1}{#2}{#3}{#4}}%
1560   \edef\@tempb{#5.center.}%
1561   \fca@concat\@tempa{\@tempb\fca@split@node@}%
1562   \@tempa
1563 }
```

\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*⟩}}.

```
1564 \def\fca@split@node@#1#2#3#4#5.#6.#7\fca@split@node@{%
1565   \def\@tempa{#7}%
1566   \ifx\@tempa\empty\relax
1567     \def#1##1{\pgfpointshapeborder{#3#5#4}{##1}}%
1568   \else
```

102

```
1569    \def#1##1{\pgfpointanchor{#3#5#4}{#6}}%
1570    \fi
1571    \def#2{%
1572      \pgfpointanchor{#3#5#4}{#6}%
1573    }%
1574 }

1575 \def\fca@def@expanded@node#1#2#3#4{%
1576    \edef\@tempa{\noexpand\def\noexpand#1\noexpand\fca@node@full@node{#2}{#3}#4..\fca@node@ful
1577    \@tempa
1578 }
```

\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.

```
1579 \def\fca@node@anchor#1.#2\fca@node@anchor{%
1580    \pgfpointanchor{#1}{#2}%
1581 }
```

\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*⟩}.
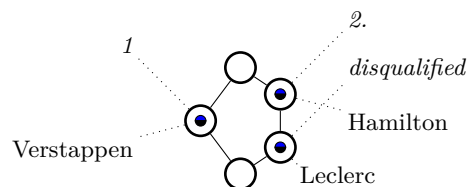
```
1582 \def\fca@node@border#1,#2.#3.#4\fca@node@border{%
1583    \pgfpointshapeborder{#1}{\pgfpointanchor{#2}{#3}}%
1584 }

1585 \newcommand\fca@calculate@edgepoint[3]{%
1586    \edef\@tempa{\noexpand\pgfutil@in@{.}{#2}}%
1587    \@tempa
1588    \ifpgfutil@in@
1589      \def#1{%
1590        \fca@node@anchor#2\fca@node@anchor
1591      }%
1592    \else
1593      \def#1{%
1594        \fca@node@border#2,#3.center.\fca@node@border
1595      }%
1596    \fi
1597 }
```
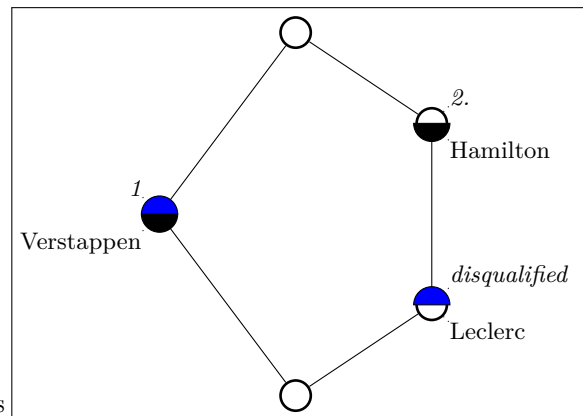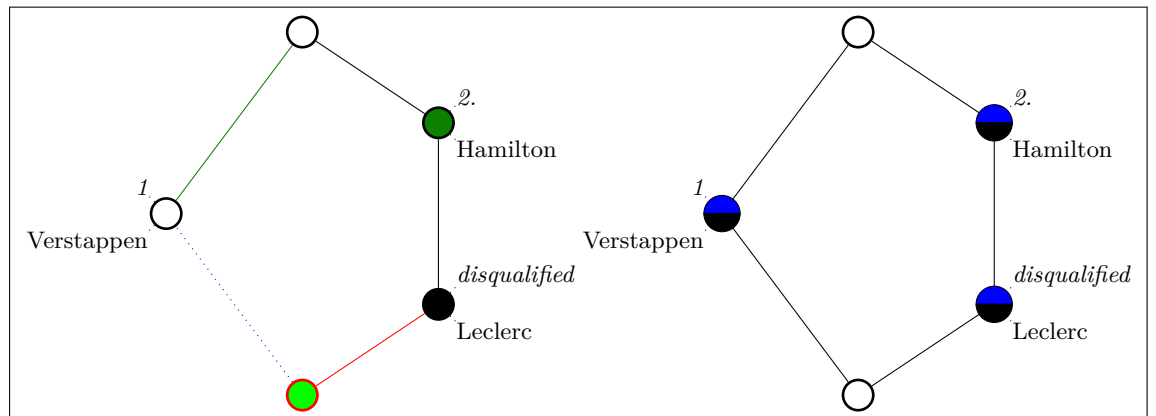
Using the style only for special nodes

Example 44: 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)
```
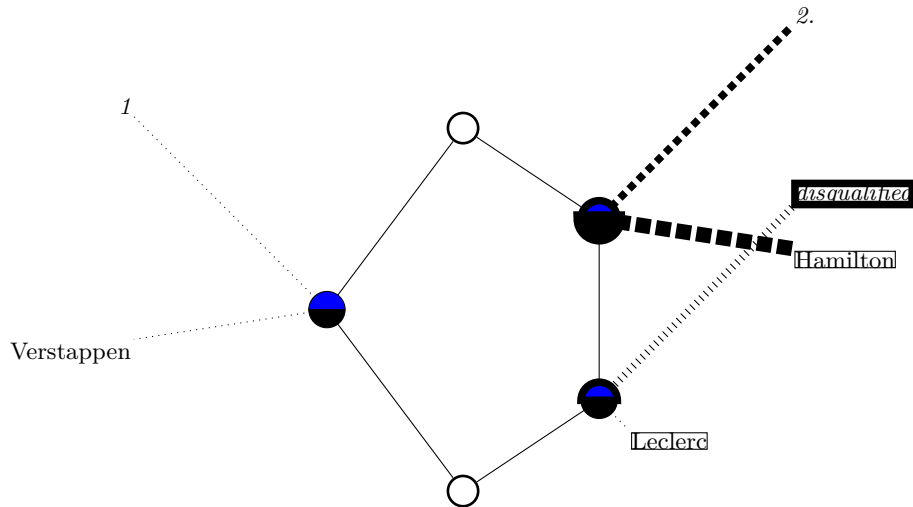
```
  \leftAttbox(3)(-20,20){1}
  \rightAttbox[draw,line width=1mm](2)(20,20){disqualified}
  \rightAttbox[label/edge width=1mm](4)(20,20){2.}
  \leftObjbox(3)(-20,-2){Verstappen}
  \rightObjbox[draw](2)(2,-2){Leclerc}
  \fcaset{label/edge width=2mm}
  \rightObjbox[draw](4)(20,-2){Hamilton}
\end{diagram}
```

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

```
1598 \newcommand\fca@@labelBox[1]{%
1599   \begin{pgfscope}%
1600     \fcaset{every node/.try,
1601       node font=\pgfkeysvalueof{/fca/label/node font},
1602       every label/.try,#1}%
1603     \fca@labelslayer
1604     \pgfkeysvalueof{/fca/node font}%
1605     \fca@options
1606     \edef\@tempa{\pgfkeysvalueof{/fca/pgfnode}}%
1607     \xdef\fca@temp@node@name{%
1608       \expandafter\fca@remove@anchor\expandafter{\@tempa}%
1609     }%
1610     \pgftransformshift{%
1611       \pgfpointanchor{%
1612         \fca@temp@node@name
1613       }{\fca@label@at}%
1614     }%
1615     \fca@parselength\@tempdima{\fca@label@shift@x}\unitlength
1616     \fca@parselength\@tempdimb{\fca@label@shift@y}\unitlength
1617     \pgftransformshift{\pgfpoint{\@tempdima}{\@tempdimb}}%
1618     \fca@transform
1619     \pgfnode{rectangle}{\pgfkeysvalueof{/fca/anchor}}{%
```

```
1620        \fca@typeset@node@content
1621        {%
1622          \pgfkeysvalueof{/fca/node contents}%
1623        }%
1624      }{%
1625        \pgfkeysvalueof{/fca/label/name prefix}%
1626        \pgfkeysvalueof{/fca/node}%
1627        \pgfkeysvalueof{/fca/label/name suffix}%
1628      }{%
1629        \expandafter\pgfusepath\expandafter{\fca@usepath}%
1630      }%
1631      \fca@endlabelslayer
1632      % pin edge
1633      \iffca@connectors
1634        \fca@connectorslayer
1635        \pgftransformreset
1636        \let\fca@options\fca@defaultoptions
1637        \fcaset{%
1638          line width=\fca@label@edge@width
1639        }
1640        \fcaset{every label edge/.try}%
1641        \fcaset{#1}%
1642        \fca@options
1643        \fca@transform
1644        %\pgfsetlinewidth{\pgfkeysgetvalue{/fca/line width}}%
1645        \fca@prepare@node\fca@temp@start@point\fca@temp@start@point@{}{}{\pgfkeysvalueof{/fca/
1646        \fca@prepare@node\fca@temp@end@point\fca@temp@end@point@
1647        {\pgfkeysvalueof{/fca/label/name prefix}}%
1648        {\pgfkeysvalueof{/fca/label/name suffix}}%
1649        {\pgfkeysvalueof{/fca/node}.\pgfkeysvalueof{/fca/anchor}}%
1650        \pgfpathmoveto{\fca@temp@start@point\fca@temp@end@point@}%
1651        \pgfpathlineto{\fca@temp@end@point\fca@temp@start@point@}%
1652        \pgfusepath{stroke}%
1653        \fca@endconnectorslayer
1654      \fi
1655      % overlay concept
1656      \pgftransformreset
1657      \let\fca@options\fca@defaultoptions
1658      \pgfsetdash{}{0pt}%
1659      \fcaset{shape=coordinate,#1}%
1660      \fcaset{every label concept/.try}%
1661      \fcaset{label concept/.try}%
1662      \ifx\pgfkeysvalueof{/fca/shape}\pgfkeysnovalue
1663      \else
1664        \fca@labelconceptslayer
1665        \pgftransformshift{%
1666          \pgfpointanchor{%
1667            \fca@temp@node@name
1668          }{\fca@label@at}%
1669        }%
1670        \fca@transform
1671        \fca@options
1672        % \pgfsetstrokecolor{\fca@node@color}%
1673        \pgfnode{\pgfkeysvalueof{/fca/shape}}{\pgfkeysvalueof{/fca/anchor}}{}{%
```

106

```
1674        \pgfkeysvalueof{/fca/label/name prefix}%
1675        \pgfkeysvalueof{/fca/node}%
1676        \pgfkeysvalueof{/fca/label/name suffix}%
1677      }{\expandafter\pgfusepath\expandafter{\fca@usepath}}%
1678      \fca@endlabelconceptslayer
1679    \fi
1680  \end{pgfscope}%
1681 }


1682 \newcommand\fca@leftAttbox[1][]{%
1683   \fca@labelBox[label type=attributes,label position=left,#1]
1684 }
1685 \newcommand\fca@centerAttbox[1][]{%
1686   \fca@labelBox[label type=attributes,label position=center,#1]
1687 }
1688 \newcommand\fca@rightAttbox[1][]{%
1689   \fca@labelBox[label type=attributes,label position=right,#1]
1690 }
1691 \newcommand\fca@leftObjbox[1][]{%
1692   \fca@labelBox[label type=objects,label position=left,#1]
1693 }
1694 \newcommand\fca@centerObjbox[1][]{%
1695   \fca@labelBox[label type=objects,label position=center,#1]
1696 }
1697 \newcommand\fca@rightObjbox[1][]{%
1698   \fca@labelBox[label type=objects,label position=right,#1]
1699 }
1700 %
1701 \def\fca@ResetDefaults{\setcounter{fca@minNode}{60}%
1702   \setcounter{fca@maxNode}{0}%
1703   \setcounter{fca@CircleDiameter}{4}%
1704   \setboolean{fca@CircledNumbers}{false}%
1705   \fca@Defaults}%
1706 %
1707 \def\fcaColorNode#1{{%
1708     \PackageWarning{fca}{The \string\ColorNode\space macro is
1709       deprecated.^^J
1710       Use '\string\Node[fill=#1]' instead}
1711     \linethickness{\fca@node@thickness}%
1712     \color{#1}{\circle*{\value{fca@CircleDiameter}}}%
1713     \color{black}{\circle{\value{fca@CircleDiameter}}}}}%
1714 %
1715 \def\fca@circle{{%
1716     \linethickness{\fca@node@thickness}%
1717     \color{\fca@node@fill}{\circle*{\value{fca@CircleDiameter}}}%
1718     \color{black}{\circle{\value{fca@CircleDiameter}}}}}%
1719 %
1720 \def\fca@DrawCircles{\setcounter{fca@runNode}{\value{fca@minNode}}%
1721   \stepcounter{fca@maxNode}%
1722   \whiledo{\value{fca@runNode}<\value{fca@maxNode}}%
1723   {\fca@getNode{\value{fca@runNode}}%
1724     \put(\fca@x,\fca@y){\fca@circle}%
1725     \ifthenelse{\boolean{fca@CircledNumbers}}%
```

```
1726        {\put(\fca@x,\fca@y){\makebox(0,0){\tiny\arabic{fca@runNode}}}}{}%
1727        \stepcounter{fca@runNode}}}%
1728 %

1729
1730 \newcommand*\fca@setb@undingbox{%
1731 }%
1732 \newcommand*\fca@ignoreboundingbox{}%

1733
1734 % set bounding box using brace notation
1735 \newcommand*\fca@olddiagram[2]{%
1736        \fca@parselength\@tempdima{#1}\unitlength
1737        \edef\fca@right@border{\the\@tempdima}%
1738        \fca@parselength\@tempdimb{#2}\unitlength
1739        \edef\fca@top@border{\the\@tempdimb}%
1740        \let\fca@boundingbox\fca@setboundingbox
1741        \fca@parselength\@tempdima{-(\fcaDiagramXoffset)}\unitlength%
1742        \edef\fca@left@border{\the\@tempdima}%
1743        \fca@parselength\@tempdima{-(\fcaDiagramYoffset)}\unitlength%
1744        \edef\fca@bottom@border{\the\@tempdima}%
1745        \fca@startdiagram
1746        \fca@setboundingbox{\fca@left@border}{\fca@bottom@border}
1747        {\fca@right@border}{\fca@top@border}%
1748 }
1749 \newcommand*\fca@diagram{%
1750        \@ifnextchar(\fca@diagr@m\fca@di@gram
1751 % )
1752 }
1753 % set bounding box using parentheses
1754 \newcommand*\fca@diagr@m{%(#1,#2)
1755        \PackageError{fca}{not implemented}{not implemented}%
1756 }
1757 % set bounding box using defaults
1758 \newcommand*\fca@di@gram{%
1759        \fca@startdiagram
1760 }
1761 \newcommand*\fca@startdiagram{%
1762        \fca@begindiagram
1763        \fca@Defaults
1764 }
1765 \newcommand*\fca@setboundingbox[4]{%
1766        \pgfpathrectangle{\pgfpoint{#1}{#2}}{\pgfpoint{#3}{#4}}%
1767        \pgfusepath{use as bounding box}%
1768 }
1769 % \todo{Lattice diagrams inside pgf pictures should be drawn inside
1770 % their own scope}
1771 \newenvironment{diagram}[1][]
1772 {%
1773        \ifpgfpicture
1774          \def\fca@begindiagram{%
1775            \begin{pgfscope}%
1776              \fcaset{#1}%
1777              \unitlength=\dimexpr\pgf@xx+\pgf@yx\relax
1778              \def\fca@enddiagram{%
1779                \end{pgfscope}%
```

```
1780            }%
1781        }%
1782    \else
1783        \def\fca@begindiagram{%
1784            \begin{pgfpicture}%
1785                \fcaset{#1}%
1786                \def\fca@enddiagram{\end{pgfpicture}}%
1787        }%
1788    \fi
1789    \noindent
1790    \@ifnextchar\bgroup\fca@olddiagram\fca@diagram% This line must be
1791                                    % executed before \begin{pgfpicture}
1792 }%
1793 {%
1794    %\fca@DrawCircles
1795    \fca@enddiagram%
1796 }%
```

Env   **tikzdiagram**   This environment places a diagram in a TikZ picture.

```
1797 \newenvironment{tikzdiagram}[1][]{%
1798    \def\fca@begindiagram{%
1799        \begin{tikzpicture}%
1800            \fcaset{#1}%
1801            \unitlength=\dimexpr\pgf@xx+\pgf@yx\relax
1802        }%
1803    \noindent
1804    \@ifnextchar\bgroup\fca@olddiagram\fca@diagram% This line must be
1805    % executed before \begin{pgfpicture}
1806 }{%
1807    \end{tikzpicture}%
1808 }
```

Opt   /fca/copy unitlength   This option copys the value of \unitlength to the x and y coordinates of TikZ.

```
1809 \fcaset{copy unitlength/.style={%
1810        x=\unitlength,
1811        y=\unitlength,
1812    }
1813 }
```

Example 45: Using \unitlength in TikZ pictures

Code:

```
\begin{tikzpicture}
  \begin{diagram}[copy unitlength]
    \Node(1)(20,10)
    \Node(2)(35,20)
    \Node(3)(5,30)
    \Node(4)(35,40)
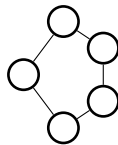```

```
        \Node(5)(20,50)
        \Edge(1)(2)
        \Edge(1)(3)
        \Edge(2)(4)
        \Edge(3)(5)
        \Edge(4)(5)
     \end{diagram}
  \end{tikzpicture}
```

Result:

Example 46: Using \unitlength in Ti*k*Z diagrams
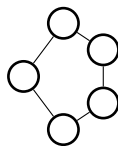
Code:

```
\begin{tikzdiagram}[copy unitlength]
    \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)
\end{tikzdiagram}
```

Result:

End of diagram environment definition.

10.4   Some simple macros for FCA texts

```
1814 \providecommand{\GMI}{(G,M,\relI)}
1815 \newcommand{\context}[1][K]{\ensuremath{\mathbb{#1}}}
1816 \providecommand{\'}{\ensuremath{\sp\prime}}% derivation operator
1817 \providecommand{\extent}[1]{\textrm{ext}(#1)}
1818 \providecommand{\intent}[1]{\textrm{int}(#1)}
1819 \providecommand{\extents}[1]{\textrm{Ext}(#1)}
```

```
1820 \providecommand{\intents}[1]{\textrm{Int}(#1)}
1821 \providecommand{\BV}{\underline{{\mathfrak B}}}
1822 \providecommand{\CL}{\BV}
1823 \providecommand{\BGMI}{{\mathfrak B} (G,M,\relI)}
1824 \providecommand{\CGMI}{\BGMI}
1825 \providecommand{\BVGMI}{\BV (G,M,\relI)}
1826 \providecommand{\CLGMI}{\BVGMI}
1827 \providecommand{\HNI}{(H,N,\relI \cap\; H{\times}N)}
1828 \providecommand{\relI}{\mathrel{I}}
1829 \providecommand{\notI}{\mathrel{\mbox{\rlap{\char'40}%
1830       {\it I}\hspace*{-0.09em}\raisebox{.27ex}{\char'40}}}}%
1831 %
1832 \providecommand{\bigtimes}{\mathop{%
1833     \mathchoice{\raisebox{-2pt}{\huge$\times$}}{\mbox{\LARGE$\times$}}%
1834     {\raisebox{0pt}{\Large$\times$}}{\times}}\displaylimits}%
1835 %
1836 \providecommand{\Runterpfeil}{\mathrel{\swarrow}}
1837 \providecommand{\DownArrow}{\Runterpfeil}
1838 \providecommand{\Hochpfeil}{\mathrel{\nearrow}}
1839 \providecommand{\UpArrow}{\Hochpfeil}
1840 \providecommand{\IRunterpfeil}{\mathrel{\searrow}}
1841 \providecommand{\IDownArrow}{\IRunterpfeil}
1842 \providecommand{\IHochpfeil}{\mathrel{\nwarrow}}
1843 \providecommand{\IUpArrow}{\IHochpfeil}
1844 \providecommand{\Doppelpfeil}{\mathrel{\!\!\rlap{$\;\nearrow$}\swarrow}}
1845 \providecommand{\DoubleArrow}{\Doppelpfeil}
1846 %
1847 \newcommand{\DDPfeil}{\mathrel{\mathchoice%
1848     {\mbox{$\displaystyle\swarrow\hspace{-.7em}\swarrow$}}
1849     {\mbox{$\textstyle\swarrow\hspace{-.7em}\swarrow$}}
1850     {\mbox{$\scriptstyle\swarrow\hspace{-.5em}\swarrow$}}
1851     {\mbox{$\scriptscriptstyle\swarrow\hspace{-.35em}\swarrow$}}}}
1852 %
1853 \newcommand{\NDDPfeil}{\mathrel{\mathchoice%
1854     {\mbox{$\displaystyle\swarrow\hspace{-.7em}\swarrow%
1855         \hspace{-1.2em}\backslash\hspace{.4em}$}}
1856     {\mbox{$\textstyle\swarrow\hspace{-.7em}\swarrow
1857         \hspace{-1.1em}\backslash\hspace{.4em}$}}
1858     {\mbox{$\scriptstyle\swarrow\hspace{-.5em}\swarrow
1859         \hspace{-.6em}\backslash\hspace{.2em}$}}
1860     {\mbox{$\scriptscriptstyle\swarrow\hspace{-.35em}\swarrow
1861         \hspace{-.5em}\backslash\hspace{.1em}$}}}}
1862 %
1863 \providecommand{\DPfeil}{\DDPfeil}
1864 \providecommand{\NDPfeil}{\NDDPfeil}
1865 \newcommand{\DDArrow}{\DDPfeil}
1866 \newcommand{\NDDArrow}{\NDDPfeil}
1867 \providecommand{\Semi}{\mathrel{\mbox{\tiny\rlap{\raisebox{2.0ex}%
1868         {$\bigtriangledown$}}\raisebox{-0.0ex}{$\bigtriangleup$}}}}
1869 \def\ovee{\mbox{\small$\mathrel{\hspace{.35em}\raisebox{-1pt}%
1870     {$\smash{\vee}$}\hspace*{-.835em}\bigcirc\hspace{.2em}}$}}
1871 \def\owedge{\mbox{\small$\mathrel{\hspace{.35em}\smash{\wedge}%
1872     \hspace*{-.835em}\bigcirc\hspace{.2em}}$}}
1873 %
```
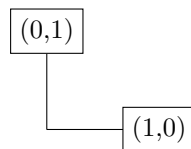
```
1874 \providecommand{\ptimes}[1]{\mathrel{\stackrel{#1}{\times}}}
1875 %
1876 \providecommand{\FCA}{Formal Concept Analysis\xspace}
1877 \providecommand{\FBA}{Formale Begriffsanalyse\xspace}
1878 \providecommand{\FnBA}{Formalen Begriffsanalyse\xspace}
```

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.

```
1879 B
1880 Formula 1
1881 3
1882 3
1883
1884 Verstappen
1885 Hamilton
1886 Leclerc
1887 1.
1888 2.
1889 disqualified
1890 X..
1891 .X.
1892 .XX

1893 B
1894
1895 3
1896 3
1897
1898 Verstappen
1899 Hamilton
1900 Leclerc
1901 1.
1902 2.
1903 disqualified
1904 X..
1905 .X.
1906 .XX
```