

ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ МАТЕМАТИКИ, МЕХАНИКИ И КОМПЬЮТЕРНЫХ НАУК
КАФЕДРА МАТЕМАТИЧЕСКОГО АНАЛИЗА

Ю. А. Кирютенко
TikZ & PGF
Некоторые библиотеки и утилиты

Ростов-на-Дону
2015

Ю. А. Кирютенко.

TikZ & PGF. Некоторые библиотеки и утилиты. — Ростов-на-Дону, 2015, 205 с.

Пособие к курсу «Информационные технологии: среда $\text{\LaTeX} 2_{\epsilon}$ » является продолжением пособия «TikZ & PGF. Создание графики в $\text{\LaTeX} 2_{\epsilon}$ -документах». Как и предыдущее, оно основано на руководстве Тилла Тантау (Till Tantau) [1], собравшего разработанные до него части системы pgf и их документацию, и написавшего большой объем своего кода. Из руководства выбраны и описываются только те библиотеки и утилиты, которые необходимы при создании несложной графики в среде MiKTeX . В пособии не рассматривается почти все, что касается изменений и дополнений к библиотекам TikZ & PGF.

Последняя глава пособия посвящена механизму объектно-ориентированного программирования в TikZ & PGF, позволяя определять классы (без наследования), методы, атрибуты (переменные) и объекты (экземпляры). Как и механизм математики, механизм ООП может использоваться независимо от пакета pgf. В этой главе предполагается, что читатель знаком с основами объектно-ориентированного программирования.

Все примеры рисунков (иногда слегка измененные) взяты из руководства Тилла Тантау (Till Tantau) [1]. Многие формы, механизм художественного оформления (декорации) и матрицы были написаны и зарегистрированы Марком Виброу (Mark Wibrow).

Предполагается, что читатель знает среду $\text{\LaTeX} 2_{\epsilon}$ (см. [3], [4], [5]) и умеет работать в ней. Чтобы работать с TikZ & PGF в среде MiKTeX , в нее обычным образом нужно установить последние версии пакетов pgf и xcolor (их можно взять на сайте www.ctan.org).

Часть I

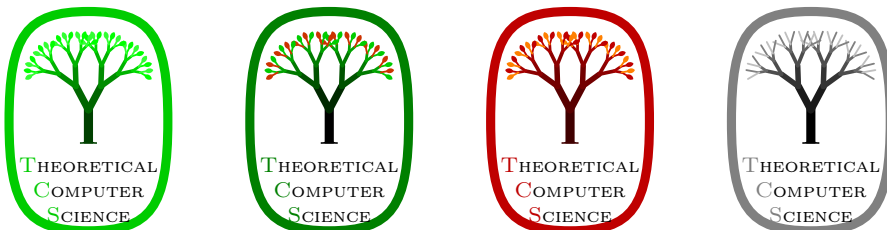
Основные библиотеки TikZ&PGF

В этой части рассматривается большинство основных библиотек, поставляемых с TikZ & PGF. Библиотеки обеспечивают дополнительные графические объекты, например, новые наконечники стрелок, новые графические метки и формы, а также расширяют возможности пакета pgf или интерфейса TikZ. Библиотеки не загружаются по умолчанию, так как не все и не всем они бывают нужны. Чтобы воспользоваться библиотекой, в преамбуле документа нужно ее загрузить:

$$\backslashusetikzlibrary{\langle\text{name library}\rangle}$$

Некоторые библиотеки из руководства [1, часть III] не рассматриваются. Одни в силу их очевидности и подробных описаний и главах из первого пособия: например, возможности библиотеки `calc` подробно описаны в главе 10, а библиотеки `fadings` — в главе 17, библиотеки `fixedpointarithmetic` и `fpu`, обеспечивающие арифметические операции с числами, кратко описаны в разделе, посвященном математике. Другие, например, библиотека `arrows` в описании не нуждается, достаточно заглянуть в исходный документ в главу 23 и из представленной там таблицы выбрать нужный наконечник. Некоторые неописанные в пособии библиотеки не имеют прямого отношения к задаче создания рисунка, хотя и важны для решения других задач. Библиотека `external` обеспечивает инструменты для высокоуровневого автоматического или полу-автоматического экспорта TikZ-изображений. Библиотеки `pgfkeys`, `plotters` загружаются TikZ автоматически и используются многими другими библиотеками и командами, так что прямое обращение к ним едва ли потребуется при первом знакомстве с TikZ&Pgf. Так же как и библиотека `profiler`, которая позволяет упростить оптимизацию TeX-программ по скорости и времени их выполнения.

Библиотека `svg.path` вводит команду, позволяющую рисовать с помощью svg-синтаксиса (который в пособиях не используется).



Глава 1

Библиотека automata

1.1 Рисунок конечного автомата

Библиотека позволяет достаточно просто создавать рисунки хорошего качества для представления конечных автоматов и машины Тьюринга. Чтобы нарисовать конечный автомат, используется следующий алгоритм:

1. Для каждого состояния конечного автомата, определить одну вершину с опцией `state` (состояние).

2. Чтобы разместить вершины состояния, использовать или абсолютные позиции или относительные позиции, определяя опции типа `above` или `right`.

3. Определить уникальное имя для каждой вершины состояния.

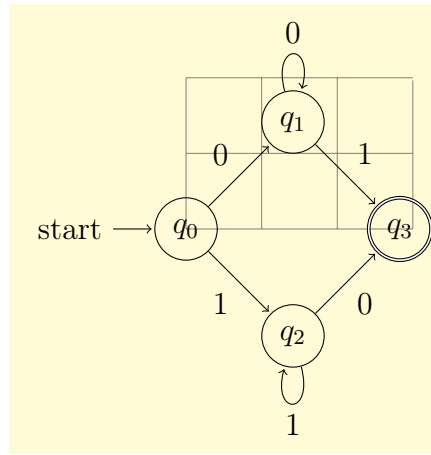
4. Указать поглощающие и начальные (исходные) состояния автомата, добавляя опции `accepting` и `initial`, соответственно, к узлам состояния.

5. Как только состояния зафиксированы, добавить дуги (переходы между состояниями). Для этого потребуется операция `edge`. Однако, дуги можно добавлять сразу же после размещения каждой вершины состояния.

6. Для циклов следует использовать операцию `edge [loop]`.

Как работает алгоритм, рассмотрим на реальном примере. Рассмотрим автомат с четырьмя недетерминированными состояниями, который проверяет содержит ли он последовательность 0^*1 или последовательность 1^*0 .

```
\begin{tikzpicture}
[shorten >=1pt,node distance=2cm,on grid,auto,
state/.style={circle,draw}]
\draw[help lines] (0,0) grid (3,2);
\node[state,initial] (q_0) {$q_0$};
\node[state] (q_1) [above right=of q_0] {$q_1$};
\node[state] (q_2) [below right=of q_0] {$q_2$};
\node[state,accepting] (q_3) [below right=of q_1] {$q_3$};
\path[->]
(q_0) edge node {0} (q_1)
edge node [swap] {1} (q_2)
(q_1) edge node {1} (q_3)
edge [loop above] node {0} ()
(q_2) edge node [swap] {0} (q_3)
edge [loop below] node {1} ();
\end{tikzpicture}
```



1.2 Состояния с выводом и без вывода данных

Стиль `state` фактически только устанавливает значение по умолчанию, лежащее в основе стиля. Но можно определить стиль для нового сложного состояния и затем просто установить стиль `state` в дополнение к новому стилю, чтобы получить желаемое. По умолчанию, определены следующие стили состояний:

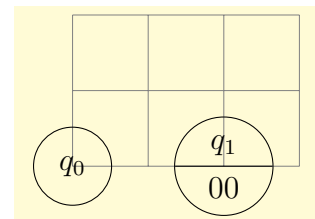
`/tikz/state without output` (style, no value)

Стиль рисует узел (вершину) в виде круга. Вызывает стиль `every state`.

`/tikz/state with output` (style, no value)

Стиль рисует узел (вершину) в виде круга, разделенного на части, используя форму `circle split`. В верхней части формы располагается название узла, в нижней части — выходные данные. Чтобы определить выходные данные в нижней части круга, внутри узла используется команда `\nodepart{lower}`. Вызывает стиль `every state`.

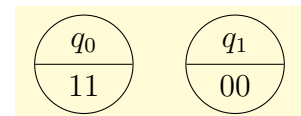
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\node[state without output] {$q_0$};
\node[state with output] at (2,0)
  {$q_1$ \nodepart{lower} $00$};
\end{tikzpicture}
```



`/tikz/state` (style, initially state without output)

Стиль надо переопределить, если нужно использовать состояния другой природы.

```
\begin{tikzpicture}[state/.style=state with output]
\node[state] {$q_0$ \nodepart{lower} $11$};
\node[state] at (2,0) {$q_1$ \nodepart{lower} $00$};
\end{tikzpicture}
```



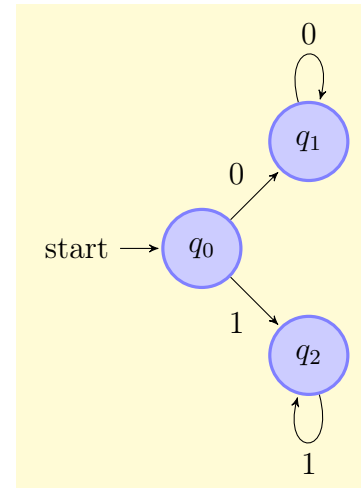
`/tikz/every state` (style, initially empty)

Этот стиль используется стилями `state without output` и `state with output`. По умолчанию, он ничего не делает, но его можно использовать для того, чтобы заставить узел состояния выглядеть должным образом.

```

\begin{tikzpicture}
[shorten >=1pt,node distance=2cm,on grid,
 >=stealth',every state/.style={draw=blue!50,
 very thick,fill=blue!20,circle}]
\node[state,initial] (q_0) {$q_0$};
\node[state] (q_1)[above right=of q_0]{$q_1$};
\node[state] (q_2)[below right=of q_0]{$q_2$};
\path[->]
(q_0) edge node[above left]{0}(q_1)
      edge node [below left]{1}(q_2)
(q_1) edge [loop above] node {0} ()
(q_2) edge [loop below] node {1} ();
\end{tikzpicture}

```



1.3 Начальные и поглощающие состояния

Стили `initial` и `accepting` подобны стилю `state`, и являются только основой для стиля, который устанавливает фактические параметры для начальных и поглощающих состояний.

`/tikz/initial` (style, initially initial by arrow)

Используется для рисования исходных состояний.

`/tikz/initial by arrow` (style, no value)

Добавляет к вершине стрелку и, возможно, некоторый текст. Стрелка указывает от текста на узел. Текст узла и направление, а также расстояние могут устанавливаться, используя следующие специальные ключи.

`/tikz/initial text=<text>` (no default, initially start)

Устанавливает текст. Чтобы подавить текст, надо использовать пустую строку.

`/tikz/initial where=<direction>` (no default, initially left)

Устанавливает место для текста. Допустимые значения: `above`, `below`, `left`, `right`.

`/tikz/initial distance=<distance>` (no default, initially 3ex)

Устанавливает длину стрелки, ведущей от текста к узлу состояния.

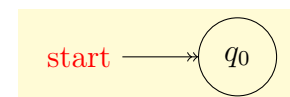
`/tikz/every initial by arrow` (style, initially empty)

Данный стиль выполняется в начале каждого пути, который содержит стрелку и текст. Его можно использовать, для того, чтобы сделать текст, например, красным.

```

\begin{tikzpicture}
[every initial by arrow/.style={text=red,->>}]
\node[circle,state,
 initial,initial distance=1cm]{$q_0$};
\end{tikzpicture}

```



`/tikz/initial above` (style, no value)

Сокращение для `initial by arrow`, `initial where=above`.

Следующие три стиля подобны предыдущему.

`/tikz/initial below` (style, no value)

`/tikz/initial left` (style, no value)

`/tikz/initial right` (style, no value)

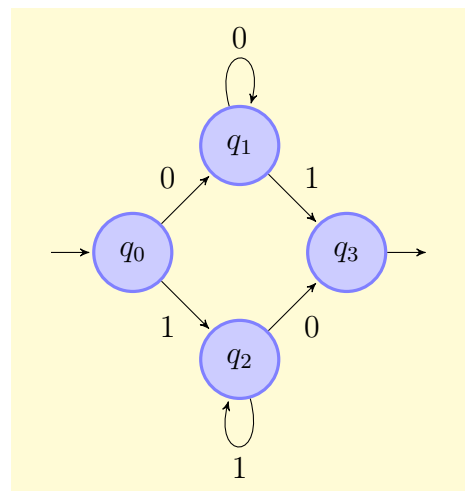
`/tikz/initial by diamond` (style, no value)

Стиль использует форму `diamond` (алмаз), чтобы указать начальный узел.

Для поглощающих состояний автомата, ситуация аналогичная: существует стиль `accepting`, определяющий способ представления узла для поглощающего состояния. Есть две опции: `accepting by arrow`, аналогичная `initial by arrow`, но с противоположным направлением стрелки, и `accepting by double`, которая рисует двойную линию вокруг узла. Значение по умолчанию стиля `accepting` — `accepting by double`.

Можно использовать те же стили и опции, что для узлов начального состояния, только заменяя слово `initial` на слово `accepting`.

```
\begin{tikzpicture}[shorten >=1pt,circle
node distance=2cm,on grid,>=stealth',
initial text=,every state/.style=
{draw=blue!50,very thick,fill=blue!20},
accepting/.style=accepting by arrow]
\node[state,initial] (q_0) {$q_0$};
\node[state] (q_1) [above right=of q_0]
{$q_1$};
\node[state] (q_2) [below right=of q_0]
{$q_2$};
\node[state,accepting] (q_3)
[below right=of q_1]{$q_3$};
\path[->]
(q_0) edge node [above left]{0}(q_1)
edge node [below left] {1}(q_2)
(q_1) edge node [above right] {1}(q_3)
edge [loop above] node {0}()
(q_2) edge node [below right] {0}(q_3)
edge [loop below] node {1}();
\end{tikzpicture}
```



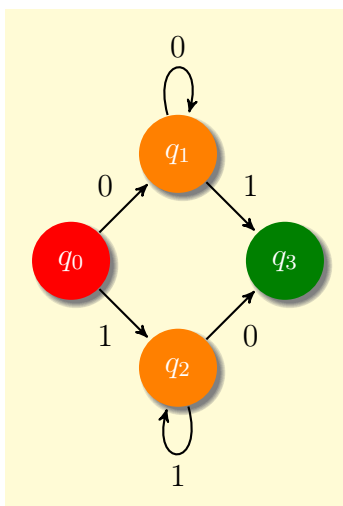
1.4 Примеры

В первом примере еще раз создается конечный автомат, представленный в предыдущих разделах. Но используется следующее правило для представления вершин поглощающих состояний и исходных состояний: вершины исходного состояния — красные, вершины поглощающего состояния — зеленые, а вершины нормального состояния — оранжевые. Затем, нужно найти путь от красного состояния к зеленому.


```

\begin{tikzpicture}
[shorten >=1pt,node distance=2cm,on grid,>=stealth',thick,
every state/.style={fill,draw=none,orange,text=white,
circular drop shadow,circle},
accepting/.style ={green!50!black,text=white},
initial/.style ={red,text=white}]
\node[state,initial] (q_0) {$q_0$};
\node[state] (q_1) [above right=of q_0] {$q_1$};
\node[state] (q_2) [below right=of q_0] {$q_2$};
\node[state,accepting](q_3) [below right=of q_1] {$q_3$};
\path[->] (q_0) edge node [above left] {0} (q_1)
edge node [below left] {1} (q_2)
(q_1) edge node [above right] {1} (q_3)
edge [loop above] node {0} ()
(q_2) edge node [below right] {0} (q_3)
edge [loop below] node {1} ();
\end{tikzpicture}

```

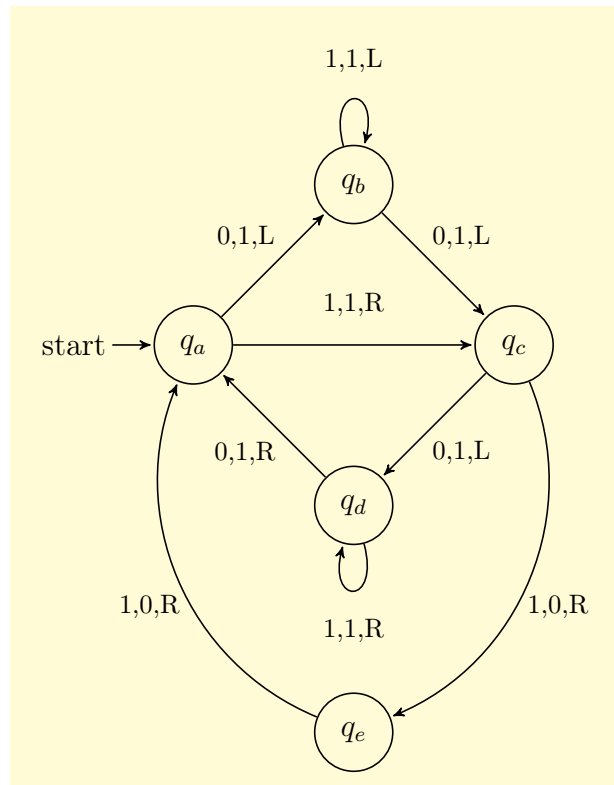


Второй и последний пример — пример автомата с пятью состояниями.

```

\begin{tikzpicture}[->,>=stealth',shorten >=1pt,auto,circle,
node distance=3cm,on grid,semithick,inner sep=2pt,bend angle=45]
\node[initial,state] (A) {$q_a$};
\node[state] (B) [above right=of A] {$q_b$};
\node[state] (D) [below right=of A] {$q_d$};
\node[state] (C) [below right=of B] {$q_c$};
\node[state] (E) [below=of D] {$q_e$};
\path [every node/.style={font=\footnotesize}]
(A) edge node {0,1,L} (B)
edge node {1,1,R} (C)
(B) edge [loop above] node {1,1,L} (B)
edge node {0,1,L} (C)
(C) edge node {0,1,L} (D)
edge [bend left] node {1,0,R} (E)
(D) edge [loop below] node {1,1,R} (D)
edge node {0,1,R} (A)
(E) edge [bend left] node {1,0,R} (A);
\end{tikzpicture}

```



Глава 2

Библиотека `backgrounds`

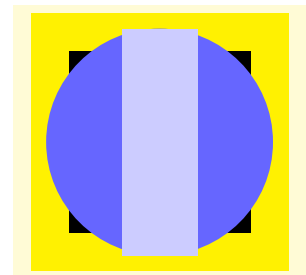
Библиотека `backgrounds` позволяет определить «фон» рисунка, делая доступными новые стили и опции. Библиотека не обращается непосредственно к фону, а обращается к рамкам, нарисованным вокруг и позади рисунка. Например, она позволяет добавить к рисунку опцию `framed`, чтобы получить прямоугольное поле вокруг него, или опцию `gridded`, чтобы поместить сетку позади него.

При первом использовании этой библиотеки нужно сделать доступным ключ

`/tikz/on background layer` (no value)

Этот ключ может использоваться с окружением `{scope}`, заставляя все объекты, расположенные внутри окружения, набираться на фоновом уровне. Заметим, что окружения `{scope}` не должны быть «глубоко вложенными» в рисунок, так как изменения графического состояния (например, цвета) не затрагивают уровни (относительно уровней см. в [1, глава 82]).

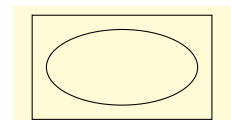
```
\begin{tikzpicture}
% На основном уровне:
\fill[blue!60] (0,0) circle (1.5cm);
% На фоновом уровне:
\begin{scope}[on background layer]
\fill[yellow] (-1.7,-1.7) rectangle (1.7,1.7);
\end{scope}
\begin{scope}[on background layer]
\fill[black] (-1.2,-1.2) rectangle (1.2,1.2);
\end{scope}
% Вновь на основном уровне:
\fill[blue!20] (-.5,-1.5) rectangle (.5,1.5);
\end{tikzpicture}
```



`/tikz/show background rectangle` (style, no value)

Рисует прямоугольник позади рисунка (опция работает только в команде `\tikz` и в среде `{tikzpicture}`).

```
\begin{tikzpicture}[show background rectangle]
\draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



Размер фонового прямоугольника определяется размерам ограничивающего прямоугольника рисунка, к которым прибавляются некоторые расстояния. Эти расстояния могут быть различными по оси OX и оси OY и устанавливаются опциями

`/tikz/inner frame xsep=<dimension>` (no default, initially 1ex)

Устанавливает для фонового прямоугольника дополнительное горизонтальное расстояние.

`/tikz/inner frame ysep=<dimension>` (no default, initially 1ex)

Устанавливает для фонового прямоугольника дополнительное вертикальное расстояние.

`/tikz/inner frame sep=<dimension>` (no default)

Устанавливает для фонового прямоугольника дополнительное горизонтальное и вертикальное расстояния равными `<dimension>`.

Следующие два стиля делают установку внутреннего дополнительного расстояния немного проще:

`/tikz/tight background` (style, no value)

Устанавливает внутреннее дополнительное расстояние равным 0pt. Таким образом, фоновый прямоугольник будет иметь размер ограничивающего прямоугольника рисунка.

`/tikz/loose background` (style, no value)

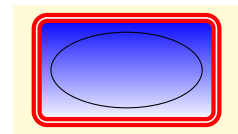
Устанавливает внутреннее дополнительное расстояние для фоновой рамки равным 2ex.

Можно повлиять на то, как выглядит фоновый прямоугольник, устанавливая следующий стиль:

`/tikz/background rectangle` (style, initially draw)

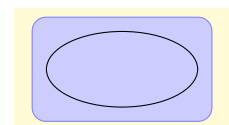
Стиль определяет, как рисуется или заполняется фоновый прямоугольник. По умолчанию фоновый прямоугольник рисуется обычным образом. Установка этого стиля, скажем, в виде `fill=blue!20` создаст светло-голубой фон изображения. Но можно использовать и более причудливые параметры, что и показано в следующем примере:

```
\begin{tikzpicture}
[background rectangle/.style={double,ultra thick,
draw=red,top color=blue,rounded corners},
show background rectangle]
\draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



Естественно, никто в здравом уме не будет использовать такой фон. Вот пример хорошего фона:

```
\begin{tikzpicture}
[background rectangle/.style={draw=blue!50,
fill=blue!20,rounded corners=1ex},
show background rectangle]
\draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



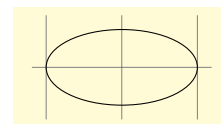
`/tikz/framed` (style, no value)

Сокращение для стиля `show background rectangle`.

`/tikz/show background grid` (style, no value)

Стиль ведет себя подобно стилю `show background rectangle`, но использует не прямоугольник, а сетку. Нижний левый и верхний правый углы сетки вычисляются так же, как и размеры фонового прямоугольника.

```
\begin{tikzpicture}[show background grid]
  \draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



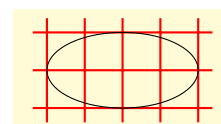
`/tikz/gridded` (style, no value)

Сокращение для стиля `show background grid`.

Можно повлиять на вид фоновой сетки, устанавливая следующий стиль:

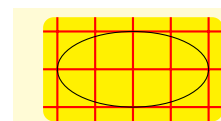
`/tikz/background grid` (style, initially draw,help lines)

```
\begin{tikzpicture}
[background grid/.style={thick,draw=red,step=.5cm},
 show background grid]
  \draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



Этот стиль можно комбинировать с опцией `framed`, которая должна быть первой:

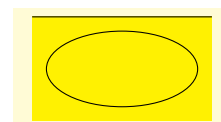
```
\tikzset{
background grid/.style={thick,draw=red,step=.5cm},
background rectangle/.style={rounded corners,
                             fill=yellow}}
\begin{tikzpicture}[framed,gridded]
  \draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



`/tikz/show background top` (style, no value)

Рисует наверху фонового прямоугольника одиночную линию, которая обычно точно совпадает с верхней границей фонового прямоугольника.

```
\begin{tikzpicture}[
  background rectangle/.style={fill=yellow},
  framed,show background top]
  \draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```

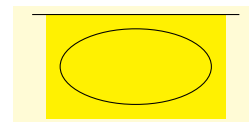


Следующая опция позволяет удлинить (или укоротить) эту линию:

`/tikz/outer frame xsep=<dimension>` (no default, initially 0pt)

Величина `<dimension>` прибавляется слева и справа к линии.

```
\begin{tikzpicture}
[background rectangle/.style={fill=yellow},framed,
show background top,outer frame xsep=1ex]
\draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



Следующие три опции ведут себя аналогично опции `show background top`.

`/tikz/show background bottom` (style, no value)
`/tikz/show background left` (style, no value)
`/tikz/show background right` (style, no value)

Нарисованные слева и справа линии можно сделать длиннее или короче.

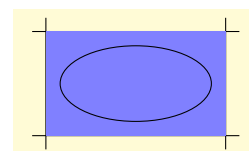
`/tikz/outer frame ysep=<dimension>` (no default, initially 0pt)

Величина `<dimension>` прибавляется снизу и сверху к линии, нарисованной на левой или правой границе фонового прямоугольника.

`/tikz/outer frame sep=<dimension>` (no default)

Добавляет `<dimension>` и по x - и по y - направлениям.

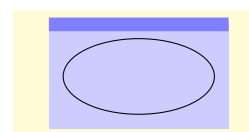
```
\begin{tikzpicture}
[background rectangle/.style={fill=blue!50},
outer frame sep=1ex,show background top,%
show background bottom,show background left,%
show background right,framed]
\draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



Можно повлиять на то, как проводится линия, устанавливая следующий стиль:

`/tikz/background top` (style, initially draw)

```
\tikzset{background rectangle/.style={fill=blue!20},
background top/.style={draw=blue!50,line width=1ex}}
\begin{tikzpicture}[framed,show background top]
\draw (0,0) ellipse (10mm and 5mm);
\end{tikzpicture}
```



Глава 3

Библиотека `calendar`

3.1 Команда `\calendar`

Библиотека `calendar` определяет команду `\calendar` окружения `{tikzpicture}`, которую используют для верстки календарей. Команда `\calendar` использует команду `\pgfcalendar` из пакета `pgfcalendar`, который загружается автоматически.

```
/tikz/\calendar<calendar specification>;
```

Синтаксис команды подобен синтаксису команд `\node` или `\matrix`, но она имеет свой синтаксический анализатор, и только команды, описанные ниже, будут им распознаваться.

Спецификации календаря. Параметр `<calendar specification>` должен быть набором элементов, каждый из которых — одна из следующих структур:

- `[<options>]` — может содержать любые опции TikZ, отделяемые запятыми, которые относятся к календарю в целом (например, `[red,draw=none]`). Можно представлять опции несколько раз, эффект накапливается.
- `(<name>)` — то же, что и `[name=<name>]`. Эффект от введения имени календаря объясняется позже. Заметим, что календарь — не узел, а `<name>` — не имя узла.
- `at (<coordinate>)` — то же, что и `[at=(<coordinate>)]`.
- `if (<date condition>)<options or commands>else<else options or commands>` — эффект от применения опции `if` объясняется позже.

В начале каждого календаря, используется стиль

```
/tikz/every calendar (style, initially empty)
```

Диапазон дат. Основной результат команды `\calendar` состоит в том, чтобы выполнить специальный код для каждого дня из указанного диапазона дат, который устанавливается, используя опцию

```
/tikz/dates=<start date> to <end date> (no default)
```

Начальная и конечная даты определяются в соответствии с форматом ISO (Международной организации по стандартизации — `<год>-<месяц>-<день>`), например, `2006-01-02`. Можно заменить последний день месяца на `last`, например, `2006-02-last` — это то же, что и `2006-02-28`. Можно добавить знак `+`, сопровождаемый числом, чтобы определить сдвиг, например, `2006-01-01+-1` — это то же, что и `2005-12-31`.

Команда `\calendar` выполняет итерации по датам из указанного диапазона. Термин текущая дата относится к дате, обрабатываемой командой. Для каждой текущей даты выполняется код, который называется `current date code`.

Центральная часть `current date code` — выполнение кода `\tikzdaycode`. По умолчанию, это код просто создает узел, текст которого отражает день месяца. Чтобы эти узлы не налегали друг на друга, следует определить порядок размещения дат. Есть предопределенные порядки их размещения: `day list downward` и `week list`, но можно определить и собственный порядок размещения.

Рассмотрим пример с предопределенным порядком размещения дат.

```
\tikz \calendar[dates=2010-02-01 to
                2010-02-14,week list];
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14

Изменение пробелов. В календаре, пробелы между днями определяются множеством опций. Большинство операций расположения дат не использует все такие опции, а только те, применение которых естественно и понятно.

`/tikz/day xshift=<dimension>` (no default, initially 3.5ex)

Определяет горизонтальный сдвиг между якорями узлов, содержащих даты.

```
\tikz \calendar[dates=2010-02-01 to
                2010-02-14,week list,day xshift=3ex];
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14

`/tikz/day yshift=<dimension>` (no default, initially 3.5ex)

Определяет вертикальный сдвиг между якорями узлов, содержащих даты.

```
\tikz \calendar[dates=2010-02-01 to
                2010-02-14,week list,day yshift=2ex];
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14

`/tikz/month xshift=<dimension>` (no default)

Определяет дополнительный горизонтальный сдвиг между разными месяцами.

`/tikz/month yshift=<dimension>` (no default)

Определяет дополнительный вертикальный сдвиг между разными месяцами.

```
\tikz \calendar[dates=2010-02-15 to
                2010-03-14,week list,month yshift=0pt];
```

15	16	17	18	19	20	21
22	23	24	25	26	27	28
1	2	3	4	5	6	7
8	9	10	11	12	13	14

```
\tikz \calendar[dates=2010-02-15 to
                2010-03-14,week list,month yshift=1cm];
```

15	16	17	18	19	20	21
22	23	24	25	26	27	28
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Изменение позиции календаря. Календарь размещается так, что якорь метки первого дня является началом. Эту ситуацию можно изменить, используя опцию `at`. Если сказать `at={(1,1)}`, то этот якорь первого дня будет лежать в точке (1,1).

Вообще говоря, не обязательно якорь первого дня будет помещаться в начало. Иногда, этому мешают правила установки дополнительных пробелов. Есть разные способы решения этой задачи. Во-первых, можно все это проигнорировать. Так как календари

часто размещаются в собственном окружении `{tikzpicture}`, и так как их размеры вычисляются автоматически, точная позиция начала часто вообще не имеет значения. Во-вторых, можно поместить календарь в узел (`\tikz \calendar ...`), что позволит позиционировать узел обычным образом, используя якоря узла. В-третьих, можно использовать одноклеточную матрицу. Преимущество последнего подхода состоит в том, что матрица позволяет использовать любой якорь любого узла внутри матрицы, как якорь для всей матрицы. Например, следующий календарь размещен так, что центр даты 2010-02-17 лежит в точке (2,2):

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\matrix[anchor=cal-2010-02-17.center] at (2,2)
{\calendar(cal) [dates=2010-02-01
to 2010-02-28,week list]; \ \ };
\end{tikzpicture}
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

К сожалению, матричный подход, который является самым естественным, не столь мобилен, как другие подходы.

Изменение внешнего вида дат. Как сказано выше, для каждого дня в календаре выполняется команда `\tikzdaycode`. Каждый раз, когда такой код выполняется, система координат может устанавливаться заново, чтобы правильно поместить день месяца. Можно изменить как сам код, так и внешний вид, используя следующие опции.

`/tikz/day code={<code>}` (no default, initially see below)

Позволяет изменить код, который выполняется для каждого дня. По умолчанию создается узел с соответствующим именем, но можно изменить и это.

```
\tikz \calendar[dates=2010-02-01 to 2010-02-14,
week list, day code={\fill[blue]
(0,0) circle (2pt)}];
```



Значение по умолчанию команды `\tikzdaycode` таково:

```
\node[name=\pgfcalendarsuggestedname, every day]{\tikzdaytext};
```

Первая часть делает узлы дней доступными через формируемые имена: если `<name>` имя, данное календарю через опцию `name=` или через определяющий элемент (`<name>`), то макрос `\pgfcalendarsuggestedname` расширяется до `<name>-<date>`, где `<date>` — дата того дня в формате ISO, которая в настоящий момент обрабатывается. Например, если обрабатывается дата 1 января 2006 — January 1, 2006 — и календарь назвали именем `mycal`, то узел, содержащий 1 для этой даты, будет называться `mycal-2006-01-01` и по этому имени на данный узел можно позже сослаться.

```
\begin{tikzpicture}
\calendar (mycal)
[dates=2010-02-01 to 2010-02-28,week list];
\draw[red](mycal-2010-02-17) circle (8pt);
\end{tikzpicture}
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

`/tikz/day text=<text>` (no default)

Изменяет определение макроса `\tikzdaytext`, который по умолчанию просто возвращает текущий день месяца. Вот достаточно глупый, но показательный, пример:

```
\tikz \calendar
  [dates=2010-02-01 to 2010-02-14,week list,
   day text=x];
```

x	x	x	x	x	x	x
x	x	x	x	x	x	x

Более содержательные примеры основаны на использовании команды `%`, которая переопределяется внутри `\pgfcalendar`, чтобы означать то же самое, что и команда `\pgfcalendarshorthand`. (Оригинальное значение `%` в календаре теряется, и его нужно заранее сохранить, если предполагается использовать.)

Команда `%` вставляет в текст текущий день/месяц/год/день недели (d, m, y, w) в определенном формате. Первый символ после `%` указывает тип (допустимые значения: d, m, y, w), второй символ определяет, как значение должно отображаться (- численно, = численно с предшествующим пробелом, 0 численно с начальными нулями, t (или .) в виде текста). Например, `%d0` задает день с начальным нулем (детали см. в описании команды `\pgfcalendarshorthand` в [1, раздел 57.2]). В примере ниже, используется `day text` так, чтобы даты отображались с начальным нулем:

```
\tikz \calendar
  [dates=2010-02-01 to 2010-02-14,week list,
   day text=%d0];
```

01	02	03	04	05	06	07
08	09	10	11	12	13	14

`/tikz/every day (initially anchor=base east)` (style, no value)

Стиль выполняется кодом узла для каждого дня и потому полезен для изменения способа представления дат. Например, задать отображение красных дат можно так:

```
\tikz[every day/.style={red}]
  \calendar[dates=2010-02-01 to 2010-02-14,
   week list];
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14

Изменение вида меток месяца и года. В дополнение к дням календаря можно добавить метки для месяца и даже года (по умолчанию это не происходит). Нужные метки добавляются явно только один раз в месяц или год. Для этого используются специальные стили, которые начинаются со слов `month label`, делают метку видимой и указывают место ее расположения:

```
\tikz
  \calendar[dates=2010-02-01 to 2010-02-14,
   week list,month label above centered];
```

February						
1	2	3	4	5	6	7
8	9	10	11	12	13	14

Изменить вид метки месяца и года позволяют следующие опции:

`/tikz/month code=<code>` (no default)

Определяет как будет расширяться макрос `\tikzmonthcode`. По умолчанию он устанавливается в код `\node[every month]{\tikzmonthtext}`; . Этот узел по умолчанию не именуется.

`/tikz/month text=<text>` (no default)

Позволяет изменить макрос `\tikzmonthtext`. По умолчанию, текст месяца — длинное текстовое представление текущего месяца.

`/tikz/every month` (style, initially empty)

Позволяет изменить вид меток для всех месяцев.

```
\tikz \calendar
  [dates=2010-02-01 to 2010-02-14,week list,
   month label above centered,
   month text=\textcolor{red}{\%mt} \%y-];
```

February 2010

1	2	3	4	5	6	7
8	9	10	11	12	13	14

Следующие опции для лет работают как аналогичные опции для месяцев.

`/tikz/year code=<code>` (no default)

`/tikz/year text=<text>` (no default)

`/tikz/every year` (no value)

Команда if для работы с датами. Мощность команды `\calendar` определяется также возможностью использовать условные выражения. Существуют два эквивалентных способа определить условное выражение. Первый путь: можно добавить код `if (<conditions>)<code or options>` в `<calendar specification>`, в котором, возможно, содержится и код `else<else code or options>`. Можно указать несколько условных выражений (но их нельзя просто вкладывать). Второй путь: использовать следующую опцию:

`/tikz/if=(<conditions>)<code or options>else<else code or options>` (no default)

Опция дает тот же результат, что и `if` в `<calendar specification>` и часто полезна в стиле `every calendar`, если условные выражения нельзя организовать иначе.

Теперь, независимо от того, как определяется условное выражение, получится следующий результат (индивидуально и независимо для каждой даты в календаре):

1. Проводится проверка, является ли текущая дата одной из возможностей, перечисленных в `<conditions>`. Например, запись `if (Saturday,Sunday) {foo}`, означает, что код `{foo}` будет выполняться для каждого дня в календаре, который является субботой (Saturday) или воскресеньем (Sunday).

Чтобы выполнить проверку условий `<conditions>`, используются команда `\ifdate` и, таким образом, макрос `\pgfcalendarifdate` (см. [1, раздел 57.1]), где приведен полный список возможных критериев). Самые полезные критерии — критерии, совпадающие с именем дня недели: `workday` для рабочих дней недели, `weekend` для выходных дней (субботы и воскресенья), `equals` для проверки, равна ли текущая дата заданной дате, и `at least` для сравнения текущей даты с заданной датой.

2. Если дата проходит проверку, выполняется `<code or options>`; если дата не проходит проверку, выполняется `<else code or options>`, если такой присутствует. Часть `<code or options>` может быть кодом, на что указывают, окружая его фигурными скобками, или списком опций, на что указывают, окружая опции квадратными скобками. Например, запись `if (Sunday) {\draw...} else {\fill...}` приводит к выполнению одного из двух кусков кода, а запись `if (Sunday) [red] else [green]` определяет две возможные опции.

Если `<code or options>` — код, он просто выполняется (для текущего дня). Если это список опций, они передаются окружению текущей даты.

Приведем некоторые примеры. Сначала, используем условное выражение, чтобы выделить все воскресенья красным цветом. Затем, выполним нечто на конкретной дате.

```
\tikz\calendar
  [dates=2010-02-01 to 2010-02-28,week list]
  if (Sunday) [red];
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

```
\tikz\calendar
  [dates=2010-02-01 to 2010-02-28,week list]
  if (Sunday) [red]
  if (equals=2010-02-17)
    {\draw (0,0) circle (10pt)};};
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

Возникает вопрос: почему круг сдвинут относительно указанной даты? Дело в том, что метка даты использует якорь `base east`, который сдвигает метку вверх и вправо. Чтобы исправить ситуацию, нужно изменить якорь:

```
\tikz [every day/.style={anchor=mid}]
\calendar
  [dates=2010-02-01 to 2010-02-28,week list]
  if (Sunday) [red]
  if (equals=2010-02-17)
    {\draw (0,0) circle (10pt)};};
```

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

Точнее управлять кодом, работающим с датами, позволяют такие опции:

`/tikz/execute before day scope=<code>` (no default)

Код `<code>` выполняется для каждой даты прежде любого другого кода. Множество вызовов этой опции накапливаются. Таким образом, если использовать эту опцию дважды, код от первой опции используется первым для каждого дня и сопровождается выполнением кода, заданного второй опцией.

`/tikz/execute at begin day scope=<code>` (no default)

Код выполняется раньше всего остального внутри окружения текущей даты. Снова, результат нескольких опций накапливается.

`/tikz/execute at end day scope=<code>` (no default)

Код выполняется непосредственно перед тем, как будет закрыто окружение текущего дня. Снова, результат нескольких опций накапливается, однако, в обратном порядке.

`/tikz/execute after day scope=<code>` (no default)

Код выполняется в самом конце обработки текущей даты, вне ее окружения. Накопление происходит, но также в обратном порядке.

В следующих подразделах рассматривается, как можно, используя разный код окружений, создать разные размещения дней в календаре.

3.1.1 Создание простого списка дней

Создадим список дней календаря, в котором даты расположены сверху вниз (следующая ниже предыдущей). Для этого, система координат сдвигается вниз в конец кода для каждого дня. Этот сдвиг должен быть вне окружения дня, поскольку сдвиг должен накапливаться.

```
\tikz\calendar
  [dates=2010-02-01 to 2010-02-05,
  execute after day scope=
  {\pgftransformyshift{-1em}}];
```

Используя такой подход, можно создать список дней, растущий вниз, вправо, влево или по диагонали.

3.1.2 Добавление метки месяца

Добавим метку месяца слева от начала каждого месяца. Идея состоит в том, чтобы сделать две вещи: (1) добавить код, который выполняется первым для каждого месяца; (2) выполнить код перед реальной датой, гарантируя, что опции, относящиеся к дням, не окажут влияния на отображение месяца.

Есть две опции, где можно определить код для месяца: или добавить его в начале окружения дня или до этого окружения. Любой вариант будет работать, но, возможно, более безопасно, поместить такой код в окружение, чтобы гарантировать, что эти установки по неосторожности не сработают снаружи окружения.

```
\tikz\calendar[dates=2010-02-01 to 2010-02-07,
  execute after day scope={\pgftransformyshift{-1em}},
  execute at begin day scope=
  {\ifdate{day of month=1}{\tikzmonthcode}{}},
  every month/.append style=
  {anchor=base east,xshift=-2em}];
```

В коде выше использовалась описанная ранее команда

```
\ifdate{<condition>}{<then code>}{<else code>}
```

Эта команда дает результат, аналогичный результату от кода

```
if (<condition>){<then code>} else {<else code>}
```

3.1.3 Создание недельного списка

Рассмотрим построение календаря с более сложным расположением дней: с недельным списком дат. При таком расположении, для каждой недели выделяется одна строка. По горизонтали дни размещаются так, что все понедельники лежат один ниже другого, аналогично — все вторники, и так далее.

Чтобы создать такое расположение, используется следующий подход: начало системы координаты располагается на якоре понедельника для каждой недели. Это означает,

что в конце каждой недели начало перемещается вниз на одну строку. Чтобы правильно позиционировать каждую дату, используется код внутри и в начале окружения дня, горизонтально сдвигая день согласно его положению в неделе.

```
\tikz\calendar
[dates=2011-02-01 to 2011-02-17,day text=\%d0,
% каждый день сдвинуть вправо,
% согласно его положению в неделе
execute at begin day scope=
  {\pgftransformxshift{\pgfcalendarcurrentweekday em}},
% после конца недели, начало сдвинуть вниз
execute after day scope=
  {\ifdate{Sunday}{\pgftransformyshift{-1em}}{}}];
```

```
010203040506
07080910111213
14151617
```

3.1.4 Создание помесячного списка дней

Создадим календарь, в котором дни расположены помесячно, то есть дни каждого месяца содержатся на одной строке. Это достаточно просто сделать, как и для понедельного календаря, но добавим одно требование: дни месяца должны выравниваться по столбцам так, чтобы в одном столбце располагались одноименные дни недели. Так как месяцы начинаются в разные дни недели, это означает, что каждая строка имеет индивидуальный сдвиг.

Один из возможных способов решения задачи состоит в том, чтобы после каждого месяца (или в начале каждого месяца) производить сдвиг вниз по вертикали на одну строку; для правильного горизонтального размещения в окружении дня локально производить сдвиг дня в соответствии с его местом в месяце; кроме того, нужно дополнительно сдвигать день, чтобы гарантировать, что первый день месяца лежит в правильном столбце недели, для этого нужно запомнить этот день недели.

```
\newcount\mycount
\tikz\calendar[dates=2010-01-01 to 2010-03-last,
execute before day scope={\ifdate{day of month=1} {
  \mycount=\pgfcalendarcurrentweekday % Запомнить день первой даты месяца
  \pgftransformyshift{-1em}}{} % Сдвинуть вниз },
execute at begin day scope=
  {% каждый день сдвинуть вправо согласно дню месяца
  \pgftransformxshift{\pgfcalendarcurrentday em}
  % и дополнительно согласно дню месяца первого дня
  \pgftransformxshift{\the\mycount em}  }];
```

```
1 2 3 4 5 6 7 8 910111213141516171819202122232425262728293031
1 2 3 4 5 6 7 8 910111213141516171819202122232425262728
1 2 3 4 5 6 7 8 910111213141516171819202122232425262728293031
```

3.2 Размещение дней календаря

Алгоритм расположения дат определяет, как даты календаря упорядочиваются по странице. Библиотека `calendar` определяет несколько алгоритмов размещения.

`/tikz/day list downward` (style, no value)

Дни месяца располагаются в столбик сверху вниз. Сдвиг между датами задается опцией `day yshift`. Между месяцами добавляется дополнительный сдвиг, определяемый опцией `month yshift`.

```
\tikz\calendar [dates=2010-01-30 to 2010-02-02,
                day list downward,month yshift=1em];
```

30
31
1
2

`/tikz/day list upward` (style, no value)

Дни месяца располагаются в столбик, только снизу вверх.

```
\tikz\calendar [dates=2010-01-30 to 2010-02-02,
                day list upward,month yshift=1em];
```

2
1
31
30

`/tikz/day list right` (style, no value)

Дни месяца располагаются в строку слева направо, но вместо опций `day yshift`, `month yshift`, используются опции `day xshift`, `month xshift`.

```
\tikz\calendar
  [dates=2010-01-30 to 2010-02-02,
   day list right,month xshift=1em];
```

30 31 1 2

`/tikz/day list left` (style, no value)

Дни месяца располагаются в строку, только справа налево.

```
\tikz\calendar
  [dates=2010-01-30 to 2010-02-02,
   day list left,month xshift=1em];
```

2 1 31 30

Следующее стили располагают даты по неделям.

`/tikz/week list` (style, no value)

Стиль создает одну строку для каждой недели в указанном диапазоне дат. Значение опции `day xshift` используется как расстояние между датами в каждой строке недели, значение `day yshift` используется как расстояние между строками. В обоих случаях, “расстояние” относится к расстоянию между якорями узлов дат (или, более общо, расстоянию между началами небольших рисунков, созданных для каждого дня).

Дни в каждой неделе сдвигаются так, что понедельники всегда в первой позиции строки (чтобы изменить это, нужно соответственно изменить код). Если диапазон дат начинается не с понедельника, первая строка не будет начинаться в первом столбце, а в столбце, соответствующем первой дате диапазона.

В начале каждого месяца (за исключением первого месяца в диапазоне) добавляется дополнительное вертикальное пространство `month yshift`. Если эта опция имеет значение `Opt`, получится непрерывный список дат.

```
\tikz\calendar [dates=2010-03-22
                to 2010-04-10,week list,
                month yshift=5mm];
```

```
22 23 24 25 26 27 28
29 30 31
      1 2 3 4
5 6 7 8 9 10
```

Следующий стиль позволяет очень компактно расположить даты целого года.

`/tikz/month list` (style, no value)

Для каждого месяца выделяется одна строка. Как и в стиле `week list`, значение опции `day xshift` используется для горизонтального сдвига, а опции `month yshift` для вертикального сдвига. В каждой строке, дни месяца перечисляются друг за другом, в том столбце, который соответствует дню недели. Таким образом, первый столбец содержит только понедельники. Если месяц начинается не с понедельника, его дни сдвигаются вправо в соответствующий столбец.

```
\sffamily\scriptsize
\tikz\calendar
  [dates=2014-09-01 to 2014-12-31,day xshift=1.3em,
  month list,month label left,month yshift=1.25em]
  if (Sunday) [red];
```

```
September 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
October    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
November   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
December  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

3.3 Метки месяцев

Для разных календарей желательно иметь возможность располагать по-разному метки к каждому месяцу. По умолчанию метки месяцев не печатаются вообще. Чтобы их напечатать, следует использовать одну из следующих опций.

`/tikz/month label left` (style, no value)

Размещает метку месяца слева от первого дня месяца (месяц не всегда начинается с понедельника, но позиция выбирается, исходя из предположения, что месяц начинается с понедельника).

```
\tikz\calendar
  [dates=2010-01-31 to 2010-02-02,
  day list downward,month yshift=1em,
  month label left];
```

```
31
February 1
          2
```

`/tikz/month label left vertical` (style, no value)

Стиль, аналогичный предыдущему, но метка поворачивается против часовой стрелки на 90° .


```
\tikz\calendar
  [dates=2010-01-31 to 2010-02-03,
   day list downward,month yshift=1em,
   month label left vertical];
```

	31
February	1
	2
	3

`/tikz/month label right`

(style, no value)

Размещает метку месяца справа на уровне первого дня месяца. Это означает, что при перечислении дней, метка лежит справа от первого дня, при перечислении недель — справа от первой недели, при перечислении месяцев — справа от целого месяца.

```
\tikz\calendar
  [dates=2010-01-31 to 2010-02-02,
   day list downward,month yshift=1em,
   month label right];
```

31	
	1 February
	2

`/tikz/month label right vertical`

(style, no value)

Стиль, аналогичный предыдущему, но размещает метку месяца, поворачивая ее по часовой стрелке на 90°.

```
\tikz\calendar
  [dates=2010-01-31 to 2010-02-03,
   day list downward,month yshift=1em,
   month label right vertical];
```

31	
	February
	1
	2
	3

`/tikz/month label above left`

(style, no value)

Размещает метку месяца выше строки его первого дня, сдвигая влево в крайний левый столбец. Метка поднимается на фиксированное расстояние, равное 1.25em. Чтобы изменить это значение, следует использовать для узла месяца опцию `yshift`.

```
\tikz\calendar
  [dates=2010-01-30 to 2010-02-03,day list right,
   month xshift=1em, month label above left];
```

			February		
	30	31	1	2	3

```
\tikz\calendar[dates=2010-01-30 to 2010-02-06,
  week list,month yshift=1em,
  month label above left];
```

					30	31
February						
1	2	3	4	5	6	

`/tikz/month label above centered`

(style, no value)

Размещает метку месяца по центру и выше строки, содержащей первый день месяца.

```
\tikz\calendar [dates=2010-02-01 to 2010-02-last,day xshift=1.3em,
  day list right,month label above centered];
```

February																											
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

`/tikz/month label above right` (style, no value)

Размещает метку месяца справа и выше строки, содержащей первый день месяца.

```
\tikz\calendar
  [dates=2010-01-25 to 2010-02-07,
   week list,month yshift=1em,
   month label above right];
```

```
25 26 27 28 29 30 31
                                     February
1   2   3   4   5   6   7
```

`/tikz/month label below left` (style, no value)

Стиль, аналогичный стилю `month label above left`, только метка месяца размещается слева ниже строки месяца.

```
\tikz\calendar [dates=2010-02-01 to 2010-02-last,day xshift=1.3em,
                 day list right,month label below left];
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
February
```

`/tikz/month label below centered` (style, no value)

Стиль, аналогичный стилю `month label above centered`, только метка месяца размещается по центру ниже строки месяца.

```
\tikz\calendar [dates=2010-02-01 to 2010-02-last,day xshift=1.3em,
                 day list right,month label below centered];
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
                February
```

3.4 Примеры

Рассмотрим примеры создания календарей, имеющих некоторые особенности. Начнем с календаря, в котором вычеркиваются рабочие дни, приближаясь к большому празднику — Новому году. Для этих дат, устанавливаем форму `strike out`.

```
\begin{tikzpicture}
\calendar
  [dates=2014-12-01 to 2015-01-last,
   week list,inner sep=2pt,
   month label above centered,
   month text=\%mt \%y0]
if (at most=2014-12-29)
  [nodes={strike out,draw}]
if (weekend)
  [black!50,nodes={draw=none}];
\end{tikzpicture}
```

```
December 2014
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

January 2015
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Следующий календарь показывает некий предельный срок, равный 10 дням (без воскресений) после текущей даты. Последние три даты перед крайним сроком отображаются красными, как предупреждение о приближении времени «Ч».

```
\begin{tikzpicture}
\calendar[dates=\year-\month-\day+-25
           to \year-\month-\day+25,
           week list,inner sep=2pt,
           month label above centered,
           month text=\textit{\%mt \%y0}]
if (at least=\year-\month-\day) {}
  else [nodes={strike out,draw}]
if (at most=\year-\month-\day+7)
  [green!50!black]
if (between=\year-\month-\day+8
    and \year-\month-\day+10) [red]
if (Sunday)[gray,nodes={draw=none}];
\end{tikzpicture}
```

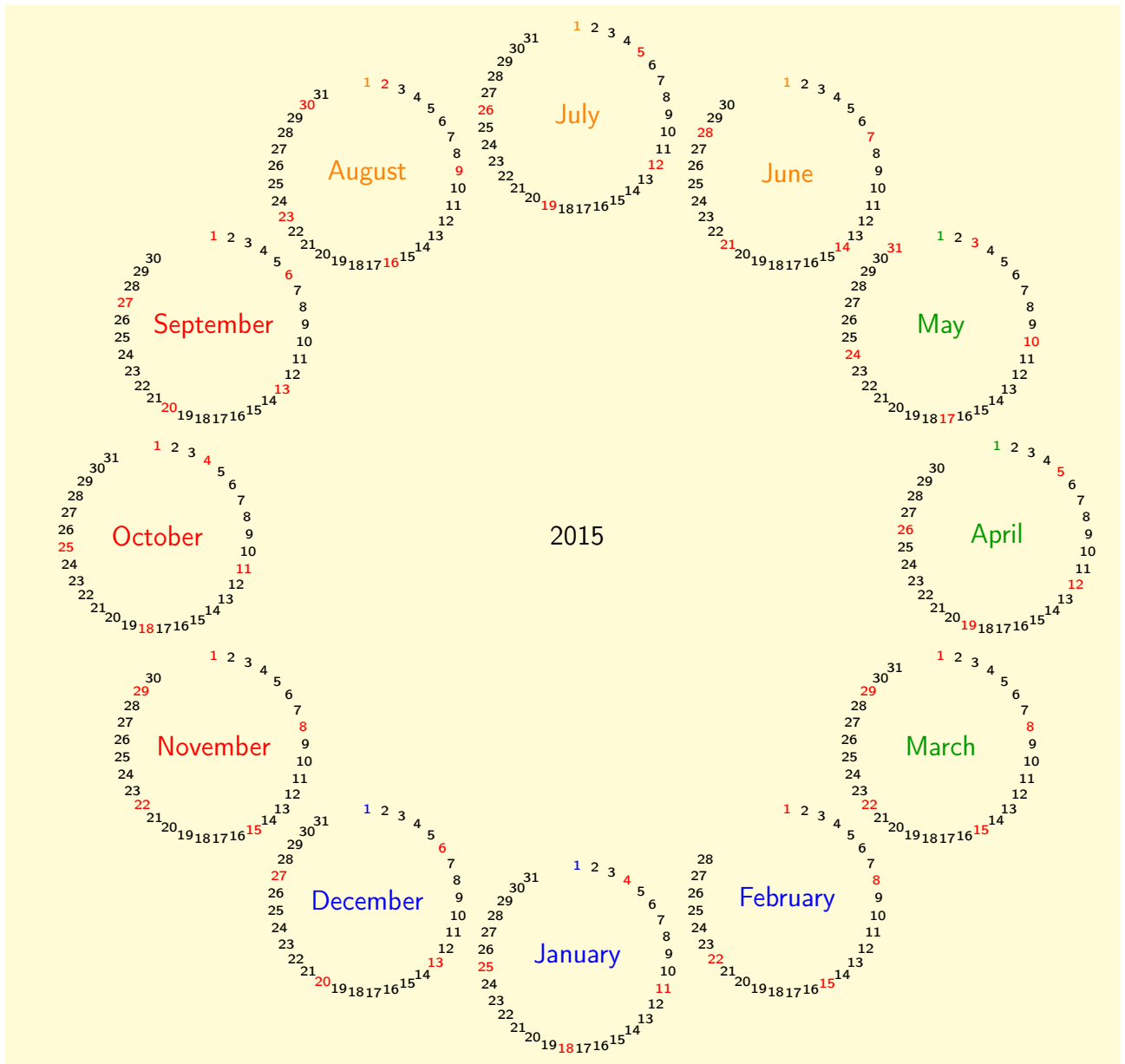
							11
12	13	14	15	16	17		18
19	20	21	22	23	24		25
26	27	28	29	30	31		
<i>February 2015</i>							
							1
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28		
<i>March 2015</i>							
							1
2							

И, наконец, пример календаря, располагающего даты по окружностям.

```
\sffamily
\colorlet{winter}{blue}      \colorlet{spring}{green!60!black}
\colorlet{summer}{orange}   \colorlet{fall}{red}
% Счетчик, так как TikZ не умеет работать с произвольными углами.
\newcount\mycount

\begin{tikzpicture}[transform shape,scale=0.95,
  every day/.style={anchor=mid,font=\fontfamily{serif}\fontseries{m}\fontsize{6}\selectfont}
  \node{\normalsize\the\year};
  \foreach \month/\monthcolor in
    {1/winter,2/winter,3/spring,4/spring,5/spring,6/summer,
     7/summer,8/summer,9/fall,10/fall,11/fall,12/winter}
{% Вычисление углов
  \mycount=\month          \advance\mycount by -1
  \multiply\mycount by 30  \advance\mycount by -90

% Построение календаря
\calendar at (\the\mycount:6.4cm)
  [dates=\the\year-\month-01 to \the\year-\month-last,]
  if (day of month=1) {\color{\monthcolor}\tikzmonthcode}
  if (Sunday)[red]
  if (all) {
  % Снова вычисление угла
  \mycount=1              \advance\mycount by -\pgfcalendarcurrentday
  \multiply\mycount by 11 \advance\mycount by 90
  \pgftransformshift{\pgfpointpolar{\mycount}{1.4cm}} };}
\end{tikzpicture}
```



Глава 4

Библиотека chains

Библиотека `chains` определяет опции для создания цепочек. Цепочка — последовательность узлов, упорядоченных по строке или столбцу, и связанных дугами. Цепочки используются для упрощения размещения узлов в разветвленной сети. Для позиционирования узлов по строкам и столбцам часто используются матрицы (см. [1, глава 17]), но цепочки можно использовать и для описания связей между узлами, которые уже были связаны, используя, например, матрицы. Поэтому, имеет смысл использовать матрицы для размещения элементов, и цепочки для построения связей между ними.

4.1 Начало и продолжение цепочки

Как правило, на рисунке одномоментно создается одна цепочка, но можно конструировать и несколько цепочек. В этом случае, цепочки нужно называть разными именами и для каждого узла указывать, какой цепочке он принадлежит.

Первый шаг к созданию цепочки — опция `start chain`.

`/tikz/start chain=<chain name><direction>` (no default)

Опция может (но не обязательно) задаваться, как опция окружения (`{scope}` или `{tikzpicture}`), включающего все узлы цепочки или пути, в котором должны находиться все узлы цепочки. Если не задано имя цепочки (нет `<chain name>`), цепочка по умолчанию получит имя `chain`.

Опция запускает цепочку с именем `<chain name>`, делая ее активной — конструируемой в данное время. Ключ `start chain` может использоваться в окружении для активизации цепочки только один раз (для цепочки с тем же именем). Цепочка перестает быть активной в конце того окружения, в котором задан ключ `start chain`.

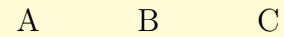
Хотя цепочки активны локально (активны в окружении, в котором запущены опцией `start chain`), информация о цепочках хранится глобально, и цепочку можно продолжить и после выхода из ее окружения. Для этого используется опция `continue chain`, позволяющая «оживить» существующую цепочку в другом окружении.

Параметр `<direction>` используется в опции `chain` для определения правила размещения узлов в цепочке. Если он отсутствует, используется текущее значение ключа

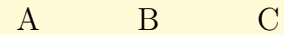
`/tikz/chain default direction=<direction>` (no default, initially going right)

Параметр `<direction>` может иметь две разные формы: `going <options>` или `placed <options>` (см. ниже описание опции `on chain`). Здесь же отметим, что направление `<direction>` применяется ко всей цепочке. Ничего другого данный ключ не делает. И, чтобы поместить узел в цепочку, нужно использовать опцию `on chain`.

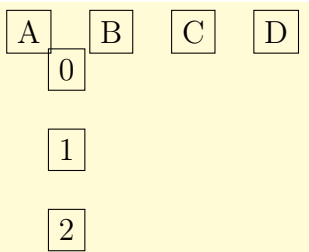
```
\begin{tikzpicture}[start chain]
  % цепочка с именем "chain"
  \node [on chain] {A};
  \node [on chain] {B};
  \node [on chain] {C};
\end{tikzpicture}
```



```
\begin{tikzpicture}
  % Как и выше, но используя окружение в {}
  { [start chain]
    \node [on chain] {A};
    \node [on chain] {B};
    \node [on chain] {C};}
\end{tikzpicture}
```



```
\begin{tikzpicture}[start chain=1 going right,
  start chain=2 going below,
  node distance=5mm,
  every node/.style=draw]
  \node[on chain=1] {A};
  \node[on chain=1] {B};
  \node[on chain=1] {C};
  \node[on chain=2] at (0.5,-.5) {0};
  \node[on chain=2] {1};
  \node[on chain=2] {2};
  \node[on chain=1] {D};
\end{tikzpicture}
```



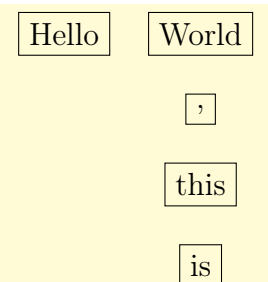
`/tikz/continue chain=<chain name><direction>`

(no default)

Опция позволяет (повторно) активизировать существующую цепочку и, возможно, изменить направление по умолчанию. Если параметр `<chain name>` отсутствует, используется имя самой внутренней из активизированных цепочек. Если таковых цепочек нет, используется цепочка с именем `chain`.

Рассмотрим два различных применения этой опции. В первом, изменяется направление конструируемой цепочки. Для этого опция используется где-то в окружении цепочки. Во втором, «оживляется» цепочка после того, как она уже была создана.

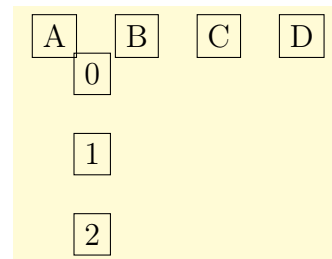
```
\begin{tikzpicture}
  [start chain=going right,node distance=5mm]
  \node[draw,on chain] {Hello};
  \node[draw,on chain] {World};
  \node[draw,continue chain=going below,on chain]{,};
  \node[draw,on chain] {this};
  \node[draw,on chain] {is};
\end{tikzpicture}
```



```

\begin{tikzpicture}[start chain=1 going right,
  node distance=5mm, every node/.style=draw]
\node [on chain] {A}; \node [on chain] {B};
\node [on chain] {C};
\begin{scope}[start chain=2 going below]
\node [on chain=2] at (0.5,-.5) {0};
\node [on chain=2] {1}; \node [on chain=2] {2};
\end{scope}
{[continue chain=1]\node [on chain] {D};}
\end{tikzpicture}

```



4.2 Узлы в цепочке

`/tikz/on chain=<chain name><direction>` (no default)

Ключ нужно задавать как опцию узла. Когда она так используется, имя цепочки `<chain name>` должно быть именем той цепочки, которая была запущена опцией `start chain`. Если `<chain name>` — пустая строка, используется текущее значение самой внутренней активизированной цепочки. Если эта опция используется для узла несколько раз, только последняя опция будет «победителем». (Чтобы поместить узел в несколько цепочек, используется команда `\chainin`.) Часть `<direction>` не обязательна. А если существует, устанавливает направление, используемое для этого узла, иначе используется направление `<direction>`, заданное первоначальной опцией `start chain` (или последней опцией `continue chain`, которая позволяет изменить значение по умолчанию). Результаты от применения этой опции следующие:

1. Внутренний счетчик (один локальный счетчик встраивается в каждую цепочку) увеличивается на 1. Счет начинается с 1: первый узел цепочки получает номер 1, второй узел — 2, и так далее. Значение этого внутреннего счетчика глобально хранится в макросе `\tikzchaincount`.

2. Если узел не имеет имени (заданной опцией узла `name` или через специальный синтаксис), имя узла устанавливается равными `<chain name>-<value counter>` (если имя цепочки `myChain`, то узлы получают имена `myChain-1`, `myChain-2`,...).

3. Независимо от способа определения, имя узла глобально сохраняется в макросе `\tikzchaincurrent`.

4. За исключением первого узла, имя предыдущего узла в цепочке хранит макрос `\tikzchainprevious` (глобально). Для первого узла цепочки этот макрос устанавливается равным пустой строке.

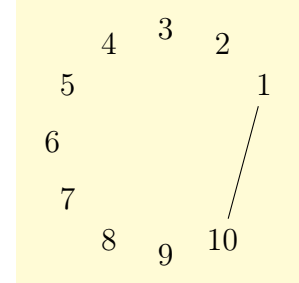
5. Кроме, возможно, первого узла цепочки, применяется правило размещения узла. Это правило — опция `TikZ`, которая применяется автоматически к каждому узлу в цепочке. В зависимости от формы параметра `<direction>` (или локально заданного или заданного опцией `start chain`), применяются разные правила.

Сначала, проверяется, что использовалось: `going` или `placed`. Различие состоит в том, что в первом случае правило размещения не применяется к первому узлу цепочки, в то время как во втором случае, правило размещения применяется и к первому узлу.

Независимо от используемой формы, `<text>` внутри `<direction>`, следующий за `going` или `placed` (отделенный пробелом), может давать два различных результата:

- (а) если содержит `=`, то `<text>` – правило размещения, которое и выполняется; тогда внутри `<text>` есть доступ к макросам `\tikzchainprevious` и `\tikzchaincount`, что позволяет выполнить вычисления по позиционированию узла.
- (б) если не содержит `=`, то правило размещения — `<text>=of \tikzchainprevious`.

```
\begin{tikzpicture}[start chain=circle placed
                    {at=(\tikzchaincount*30:1.5)}]
\foreach \i in {1,...,10} \node [on chain] {\i};
\draw (circle-1) -- (circle-10);
\end{tikzpicture}
```



6. Для **всех** узлов в цепочке выполняется следующий стиль:

`/tikz/every on chain`

(style, no value)

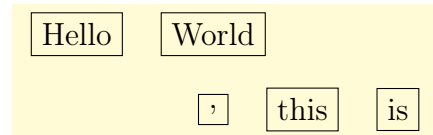
Правило `right=of (\tikzchainprevious)` — стандартное правило размещения узла: новый узел помещается вправо от предыдущего на расстоянии `node distance`.

```
\begin{tikzpicture}[start chain,node distance=5mm]
\draw [draw,on chain] {};
\draw [draw,on chain] {Hello};
\draw [draw,on chain] {Welt};
\end{tikzpicture}
```



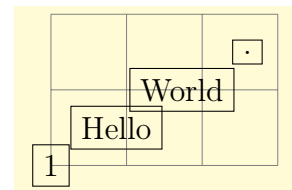
Необязательная опция `<direction>` ключа `on chain` позволяет временно изменить направление в середине цепочки:

```
\begin{tikzpicture}
[start chain,node distance=5mm]
\draw [draw,on chain] {Hello};
\draw [draw,on chain] {World};
\draw [draw,on chain=going below] {,};
\draw [draw,on chain] {this};
\draw [draw,on chain] {is};
\end{tikzpicture}
```



В `direction>` можно использовать и более сложные вычисления:

```
\begin{tikzpicture}
[start chain=going {at=(\tikzchainprevious),
                    shift=(30:1)}]
\draw [help lines] (0,0) grid (3,2);
\draw [draw,on chain] {1};
\draw [draw,on chain] {Hello};
\draw [draw,on chain] {World};
\draw [draw,on chain] {.};
\end{tikzpicture}
```



Для каждой цепочки, создаются два специальных псевдоузла:

(1) `<chain name>-begin`, то же, что первый узел в цепочке, но он определяется только после определения первого узла;

(2) `<chain name>-end`, то же, что последний текущий узел в цепочке. По мере расширения цепочки, этот узел меняется.

Опция `on chain` может использоваться вместе с опцией `late options`, чтобы добавить уже существующий узел в цепочку. Следующая команда, которая должна появляться только в том окружении, в котором была использована опция `start chain`, упрощает этот процесс.

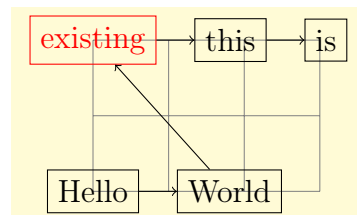
```
/tikz/\chainin(<existing name>)[<options>]
```

Команда облегчает добавление уже созданного узла в цепочку. Этот узел может быть даже частью другой цепочки. Когда говорится `\chainin(some node)`; узел с именем `<some node>` должен существовать, и тогда он становится частью текущей цепочки. Это не означает, что узел может измениться (он уже создан), но может использоваться опция `join` (см. ниже раздел 4.3), присоединяющая узел `some node` к последнему из предыдущих узлов в цепочке, а последующие узлы будут размещаться уже относительно узла `some node`. Допустимо задавать опцию `on chain` в `<options>`, чтобы определить, в какую цепочку должен быть помещен узел. Эта команда — только сокращение для кода

```
\path(<existing name>) [late options={on chain, every chain in, <options>}]
```

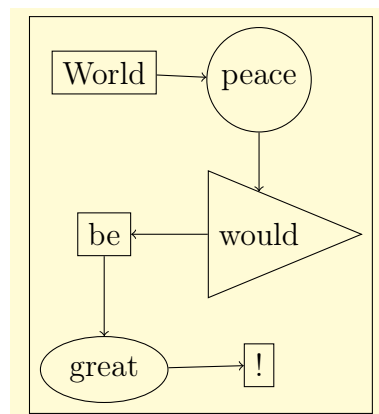
В частности, можно продолжить путь и после команды `\chainin`.

```
\begin{tikzpicture}[node distance=5mm,
every node/.style=draw, every join/.style=->]
\draw [help lines] (0,0) grid (3,2);
\node[red] (existing) at (0,2) {existing};
  {\start chain
  \node [draw,on chain,join] {Hello};
  \node [draw,on chain,join] {World};
  \chainin (existing) [join];
  \node [draw,on chain,join] {this}; \node [draw,on chain,join] {is};}
\end{tikzpicture}
```



Рассмотрим пример, в котором узлы позиционированы, используя матрицу, а затем соединены, используя цепочки.

```
\begin{tikzpicture}[every node/.style=draw]
\matrix [matrix of nodes, column sep=5mm,
row sep=5mm, ampersand replacement=\&]
{|(a)| World \& |(b)| [circle]| peace \\
|(c)| be \& |(d)| [isosceles triangle]| would \\
|(e)| [ellipse]| great \& |(f)| ! \\ };
{\start chain,
every on chain/.style={join=by ->}}
\chainin (a); \chainin (b); \chainin (d);
\chainin (c); \chainin (e); \chainin (f);}
\end{tikzpicture}
```



4.3 Соединение узлов в цепочке

`/tikz/join=with<with> by<options>` (no default)

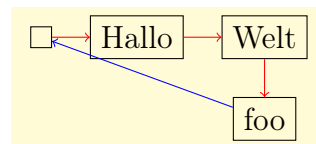
Когда задается эта опция, к любому узлу цепочки (кроме, возможно, первого) после узла добавляется команда `join`. Часть `with` определяет, какой узел должен использоваться в качестве начальной точки дуги. Если часть `with` отсутствует, используется макрос `\tikzchainprevious`. Команда `edge` получает `<options>` в качестве параметра и текущий узел в качестве ее адресата. Если нет предыдущего узла и не задана часть `with`, команда `edge` не выполняется.

`/tikz/every join` (style, no value)

Стиль выполняется каждый раз, когда используется команда `join`.

Заметим, что для узла нужно вызывать эту опцию несколько раз, чтобы связать его с несколькими узлами. Это особенно полезно при соединении ветвей (см. следующий раздел).

```
\begin{tikzpicture}
[start chain,node distance=5mm,
every join/.style={->,red}]
\node [draw,on chain,join] {};
\node [draw,on chain,join] {Hallo};
\node [draw,on chain,join] {Welt};
\node [draw,on chain=going below,join,
join=with chain-1 by {blue,<-}] {foo};
\end{tikzpicture}
```



4.4 Ветви

Ветвь — цепочка, которая (обычно, только временно) расширяет существующую цепочку. Идея состоит в следующем: предположим, что создается цепочка и в некотором узле `x` происходит ветвление. В этом случае, в точке ветвления начинается одна (или несколько) ветвей. Для каждой ветви создается цепочка, но первым узлом в этой цепочке должен быть узел `x`. Для этого, в узле `x` следует использовать команду `\chainin`, чтобы заставить разделиться цепочку на разные ветви и именовать каждую ветвь так, чтобы указать имя основной цепочки. Именно это позволяет сделать опция

`/tikz/start branch=<branch name><direction>` (no default)

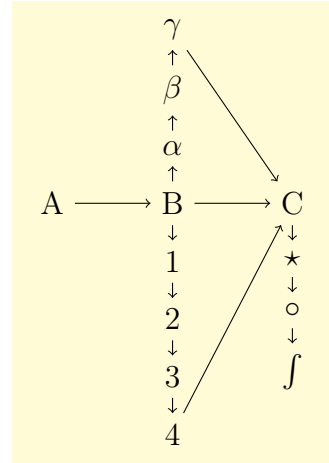
Опция используется так же, как и `start chain`, но результат немного другой:

- Опция может использоваться, если уже есть некоторая активная цепочка и в ней есть (последний) узел, который станет узлом ветвления — `<fork node>`.
- Цепочка получает имя `<current chain>/<branch name>`. Например, если цепочка имеет имя `trunk` и в ее узле `<fork node>` создается ветвь, для которой опция `<branch name>` определяется как `left`, то полное имя ветви будет `trunk/left`. Значение параметра `<branch name>` нужно задать явно, для него нет значения по умолчанию.
- Узел `<fork node>` автоматически встраивается в ветвь как начальный узел. Затем вызывается опция `join`, чтобы связать первый узел создаваемой ветви с узлом ветвления `<fork node>`.

```

\begin{tikzpicture}[every on chain/.style=join,
                    every join/.style={arrows=->},
                    node distance=2mm and 1cm]
{[start chain=trunk]
 \node [on chain] {A};
 \node [on chain] {B};
   {[start branch=numbers going below]
    \node [on chain] {1}; \node [on chain] {2};
    \node [on chain] {3}; \node [on chain] {4};}
   {[start branch=greek going above]
    \node [on chain] {$\alpha$};
    \node [on chain] {$\beta$};
    \node [on chain] {$\gamma$}; }
 \node [on chain,join=with trunk/numbers-end,
        join=with trunk/greek-end] {C};
   {[start branch=symbols going below] \node [on chain] {$\star$};
    \node [on chain] {$\circ$};
    \node [on chain] {$\int$}; } }
\end{tikzpicture}

```



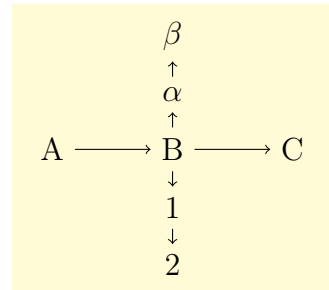
`/tikz/continue branch=<branch name><direction>` (no default)

Опция работает подобно опции `continue chain`, только в качестве имени цепочки используется имя `<current chain>/<branch name>`.

```

\begin{tikzpicture}[every on chain/.style=join,
                    every join/.style={arrows=->},
                    node distance=2mm and 1cm]
{[start chain=trunk]
 \node [on chain] {A};
 \node [on chain] {B};
   { [start branch=numbers going below] }
   { [start branch=greek going above]   }
   % только объявление, к ним вернемся позже
 \node [on chain] {C};
   % Переходим к созданию объявленных ветвей
   { [continue branch=numbers] \node [on chain] {1};
    \node [on chain] {2}; }
   { [continue branch=greek] \node [on chain] {$\alpha$};
    \node [on chain] {$\beta$}; }
}
\end{tikzpicture}

```



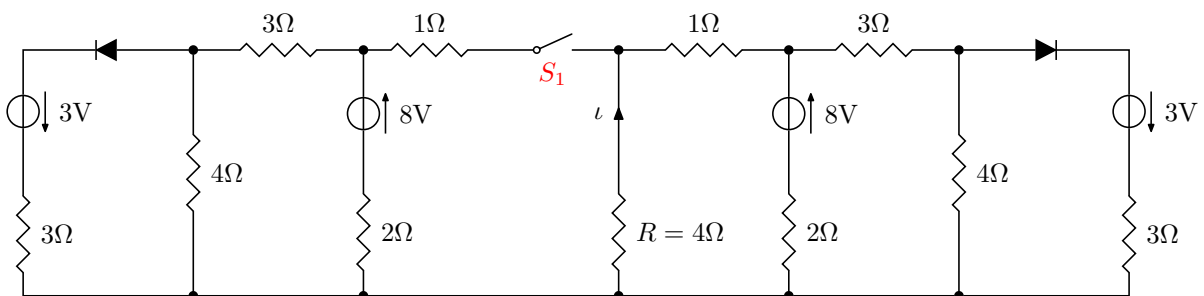
Глава 5

Библиотеки схем и цепей

5.1 Введение

Библиотеки для схем и цепей (контуров) позволяют рисовать различные типы электрических или логических схем. Для этого нужна не одна, а целая иерархия библиотек, которые работают вместе. Основная цель их создания состояла в том, чтобы установить равновесие между простотой использования и легкостью расширения при создании высококачественных графических представлений разных схем.

5.1.1 Первый пример



```
\begin{tikzpicture}[rotate=-90,circuit ee IEC,x=3.25cm,y=2.25cm,
semithick,every info/.style={font=\footnotesize},
small circuit symbols,
set resistor graphic=var resistor IEC graphic,
set diode graphic=var diode IEC graphic,
set make contact graphic=var make contact IEC graphic]

\foreach \contact/\y in {1/1,2/2,3/3.5,4/4.5,5/5.5}
{\node [contact] (left contact \contact) at (0,\y) {};}
\node [contact] (right contact \contact) at (1,\y) {};}
\draw (right contact 1) -- (right contact 2) -- (right contact 3)
-- (right contact 4) -- (right contact 5);
\draw (left contact 1) to [diode] ++(down:1)
to [voltage source={near start,
direction info={volt=3}},
resistor={near end,ohm=3}] ++(right:1)
to (right contact 1);
\draw (left contact 1) to [resistor={ohm=4}] (right contact 1);
```

```

\draw (left contact 1) to [resistor={ohm=3}] (left contact 2);
\draw (left contact 2) to [voltage source={near start,
                                direction info={<-,volt=8}},
                            resistor={ohm=2,near end}] (right contact 2);
\draw (left contact 2) to [resistor={near start,ohm=1},
                            make contact={near end,info'={ [red]$S_1$}}]
    (left contact 3);
\draw (left contact 3) to [current direction'={near start,info=${\iota$},
                            resistor={near end,info={\$R=4\Omega$}}]
    (right contact 3);
\draw (left contact 4) to [voltage source={near start,
                                direction info={<-,volt=8}},
                            resistor={ohm=2,near end}] (right contact 4);
\draw (left contact 3) to [resistor={ohm=1}] (left contact 4);
\draw (left contact 4) to [resistor={ohm=3}] (left contact 5);
\draw (left contact 5) to [resistor={ohm=4}] (right contact 5);
\draw (left contact 5) to [diode] ++(up:1)
    to [voltage source={near start,
                                direction info={volt=3}},
        resistor={near end,ohm=3}] ++(right:1)
    to (right contact 5);

\end{tikzpicture}

```

Важная особенность библиотек схем та, что видом схемы можно управлять самым общим образом, а метки размещаются автоматически по умолчанию. Пример выше можно сгенерировать еще раз почти из того же исходного текста, например, удаляя из окружения `tikzpicture` опцию `rotate=-90`, чтобы расположить схему вертикально.

5.1.2 Символы

Схема обычно состоит из многочисленных элементов, таких как логические блоки, резисторы или диоды, которые связаны проводами. В `pgf/TikZ` используются узлы для представления элементов электрических цепей и обычные линии для проводов.

`TikZ` предлагает много различных способов позиционирования и связывания узлов, и все они могут использоваться при создании схем. Кроме того, библиотека `circuits` определяет дополнительный специальный путь, который особенно полезен для таких элементов, как резисторы.

Есть много различных названий, которые используются для обращения к элементам схем, но для дальнейшего потребуются только несколько стандартных терминов: будем называть все такие элементы символами. Символьная форма — форма `pgf`, объявленная, используя команду `\pgfdeclareshape`. Символьный узел — узел, форма которого — символьная форма.

5.1.3 Графика символов

Символы могут создаваться кодом `\node [shape=some symbol shape]`. Однако, чтобы представить некоторые символы правильно, только использование стандартных `pgf`-форм не достаточно. Например, большинство символов имеет визуально обязательный «размер по умолчанию», но размер символьной формы зависит только от текущих значений параметров, таких как `minimum height` или `inner xsep`.

По этим причинам, библиотеки схем вводят понятие *рисунка символа* (символьного рисунка). Это — стиль, который заставляет узел `\node` иметь не только правильную форму, но также и правильный размер, и правильно используемый путь. Вообще говоря, этот стиль может устанавливать элементы так, чтобы символ выглядел правильно. И, когда написано, например, `\node[diode]`, то используется стиль, называемый `diode graphic`, который устанавливает все элементы для сборки корректного символьного рисунка.

Дадим краткий обзор различных видов библиотек для схем:

(1) TikZ-библиотека `circuits` определяет общие ключи для создания схем, которые используются при определении специализированных библиотек. Но библиотека `circuits` прямо не используется, так как не определяет символьных рисунков.

(2) TikZ-библиотека `circuits.logic` определяет ключи для создания логических блоков, подобных `and-gate` (И-вентилю) или `or-gate` (ИЛИ-вентилю). Эта библиотека также не определяет символьных рисунков; это делают две ее подбиблиотеки:

(2a) Библиотека `circuits.logic.US` определяет символьные рисунки, которые представляют логические блоки в так называемом US-стиле (американском стиле). Она включает все вышеупомянутые библиотеки, и эту библиотеку уже можно использовать для построения схем.

(2b) Библиотека `circuits.logic.IEC` также определяет символьные рисунки, которые представляют логические блоки, но в стиле Международной Электротехнической Комиссии (IEC), в которой используются прямоугольные формы, а не круглые, как в американском стиле. Эта библиотека может мирно сосуществовать с US-библиотекой, и потому можно «на лету» изменить используемые символьные рисунки.

(3) TikZ-библиотека `circuits.ee` определяет ключи для символов из электротехники, таких как резисторы или конденсаторы. Снова, ее (под)библиотеки определяют фактические символьные рисунки.

(3a) Библиотека `circuits.ee.IEC` определяет формы символов, соответствующие стандартам Международной Электротехнической Комиссии.

(4) Pgf-библиотеки `shapes.gates.*` определяют собственные символьные формы для схем. Обычно эти формы не используют напрямую, а используют стиль, который использует соответствующий символьный рисунок, который, в свою очередь, использует одну из возможных форм.

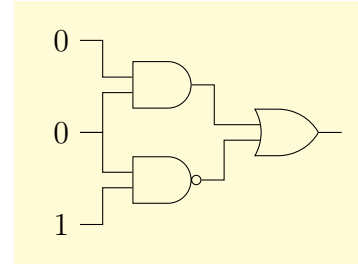
Рассмотрим простой пример. Предположим, что нужно создать логическую схему. Тогда, прежде всего надо решить, какие символьные рисунки будут использоваться. Если будем использовать американский стиль, включим библиотеку `circuits.logic.US`. Если желательно использовать символы в стиле Международной Электротехнической Комиссии, следует включить библиотеку `circuits.logic.IEC`. Но можно использовать в одном документе и обе: `\usetikzlibrary{circuits.logic.US,circuits.logic.IEC}`.

Чтобы создать рисунок, который содержит схему в американском стиле, в окружении `{tikzpicture}` нужно использовать опцию `circuit logic US`, устанавливающую ключи, позволяющие создать соответствующий символ. Чтобы создать схему в IEC-стиле, нужно использовать опцию `circuit logic IEC`.

```

\begin{tikzpicture}[circuit logic US]
\matrix[column sep=7mm,ampersand replacement=\&]
{\node (i0) {0}; \& \& \\
\& \node [and gate] (a1) {}; \& \\
\node (i1) {0}; \& \& \node [or gate] (o) {};\& \\
\& \node [nand gate] (a2) {};\& \\
\node (i2) {1}; \& \& };\& \\
\draw (i0.east) -- ++(right:3mm) |- (a1.input 1);
\draw (i1.east) -- ++(right:3mm) |- (a1.input 2);
\draw (i1.east) -- ++(right:3mm) |- (a2.input 1);
\draw (i2.east) -- ++(right:3mm) |- (a2.input 2);
\draw (a1.output) -- ++(right:3mm) |- (o.input 1);
\draw (a2.output) -- ++(right:3mm) |- (o.input 2);
\draw (o.output) -- ++(right:3mm);
\end{tikzpicture}

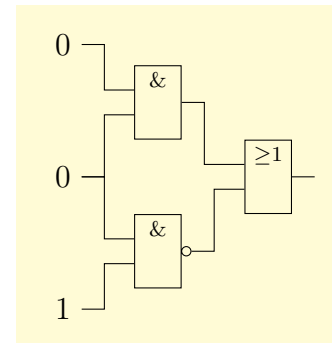
```



```

\begin{tikzpicture}[circuit logic IEC]
\matrix[column sep=7mm,ampersand replacement=\&]
{ \node (i0) {0}; \& \& \\
\& \node [and gate] (a1) {};\& \\
\node (i1) {0}; \& \& \node [or gate] (o) {};\& \\
\& \node [nand gate] (a2) {};\& \\
\node (i2) {1}; \& \& };\& \\
\draw (i0.east) -- ++(right:3mm) |- (a1.input 1);
\draw (i1.east) -- ++(right:3mm) |- (a1.input 2);
\draw (i1.east) -- ++(right:3mm) |- (a2.input 1);
\draw (i2.east) -- ++(right:3mm) |- (a2.input 2);
\draw (a1.output) -- ++(right:3mm) |- (o.input 1);
\draw (a2.output) -- ++(right:3mm) |- (o.input 2);
\draw (o.output) -- ++(right:3mm);
\end{tikzpicture}

```



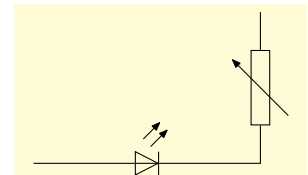
5.1.4 Аннотации

Аннотация (примечание) — небольшое рисуемое пространство, добавляемое к символу. Например, когда добавляются две небольшие параллельные стрелки, выходящие из элемента электрической схемы, это означает, что этот элемент — источник излучения. Вместо того, чтобы иметь один символ для диода, а другой для «излучающего диода», используют один символ для всех диодов, и, когда нужно, добавляют к нему аннотацию, указывающую на излучающий диод. Аннотация передается символу как параметр:

```

\tikz [circuit ee IEC]
\draw (0,0) to [diode={light emitting}] (3,0)
to [resistor={adjustable}] (3,2);

```



5.2 Основная библиотека для схем

Из основной библиотеки `circuits`, которая используется другими библиотеками для схем, обычно нужны только некоторые общие опции, описанные ниже.

`/tikz/circuits` (no value)

Ключ должен передаваться в виде опции рисунка или окружения, которые содержат схему. Он выполняет некоторые внутренние установки, обычно вызывая специализированные ключи, такие как `circuit ee IEC`.

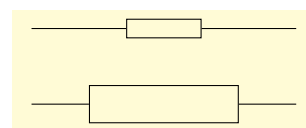
5.2.1 Размер символа

`/tikz/circuits symbol unit=<dimension>` (no default, initially 7pt)

Ключ определяет «единицу» для вычисления размера символов. Библиотеки определяют размеры символов по этой «единице». Например, длинная сторона катушки индуктивности в библиотеке IEC, по умолчанию, равна `<dimension>*5`. Когда изменяется `<dimension>`, размеры всех символов будет автоматически изменяться.

Заметим, что можно переопределить размер любого конкретного символа. Эти установки применимы только к размерам по умолчанию.

```
\begin{tikzpicture}[circuit ee IEC]
  \draw (0,1) to [resistor] (3.5,1);
  \draw[circuit symbol unit=14pt]
    (0,0) to [resistor] (3.5,0);
\end{tikzpicture}
```



`/tikz/huge circuit symbols` (style, no value)

Устанавливает единицу размера символа схемы по умолчанию равной 10pt.

`/tikz/large circuit symbols` (style, no value)

Устанавливает единицу размера символа схемы по умолчанию равной 8pt.

`/tikz/medium circuit symbols` (style, no value)

Устанавливает единицу размера символа схемы по умолчанию равной 7pt.

`/tikz/small circuit symbols` (style, no value)

Устанавливает единицу размера символа схемы по умолчанию равной 6pt.

`/tikz/tiny circuit symbols` (style, no value)

Устанавливает единицу размера символа схемы по умолчанию равной 5pt.

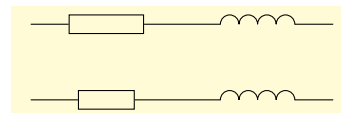
`/tikz/circuit symbol size=width <width> height <height>` (no default)

Ключ устанавливает опцию `minimum height` (минимальная высота) равной величине `<height>`, умноженной на текущее значение единицы для символов схемы, а минимальную ширину (опцию `minimum width`) равной величине `<width>`, умноженной на то же значение. Таким образом, эта опция может использоваться с командой узлов, чтобы установить размеры узлов кратными значению единицы для символов схемы.


```

\begin{tikzpicture}[circuit ee IEC]
  \draw (0,1) to [resistor] (2,1)
           to [inductor] (4,1);
  \begin{scope}[every resistor/.style=
    {circuit symbol size=width 3 height 1}]
    \draw (0,0) to [resistor] (2,0) to [inductor] (4,0);
  \end{scope}
\end{tikzpicture}

```



5.2.2 Объявление новых символов

`/tikz/circuit declare symbol=<name>` (no default)

Используется для объявления символа. Но не отображает символ, не устанавливает для него графический образ, а просто подготавливает несколько ключей, которые могут позже использоваться, чтобы нарисовать символ и настроить его.

Первый, так определяемый ключ, называется `<name>`. Этот ключ нужно передавать как опцию `node` или `to` пути так, как сказано ниже. Ключ будет принимать опции, которые могут использоваться для того, чтобы повлиять на способ отображения графического образа символа.

Рассмотрим пример. Предположим, что нужно определить символ с именем `foo`, который похож на простой прямоугольник: `\tikzset{circuit declare symbol=foo}`. Теперь определенный символ можно использовать следующим образом:

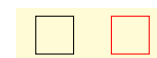
```
\node [foo] at (1,1) {}; \node [foo={red}] at (2,1) {};
```

Но при таком определении ничего не отобразится, так как не определен графический образ, который используется для `foo`. Для этого нужно использовать ключ `set foo graphic`, а точнее, `set <name> graphic`. Этот ключ получает в качестве параметра графические опции, которые устанавливаются, когда символ `foo` отображается:

```

\begin{tikzpicture}[circuit declare symbol=foo,
  set foo graphic={draw,rectangle,minimum size=5mm}]
  \node [foo] at (1,1) {}; \node [foo={red}] at (2,1) {};
\end{tikzpicture}

```



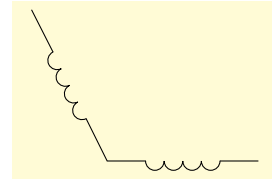
Детали объявления новых символов см. в [1, 29.2.2].

5.2.3 Размещение символов в правильном направлении

В отличие от нормальных узлов, которые вообще не должны вращаться, так как это делает их текст трудно читаемым, символы часто приходится вращать. Есть два способа добиться вращения символа.

(1) Когда символ размещается в пути командой `to`, его графический образ автоматически поворачивается, располагаясь вдоль пути. Вот пример, который показывает, как форма, представляющая катушку индуктивности, автоматически вращается:

```
\tikz [circuit ee IEC]
  \draw (3,0) to[inductor] (1,0)
         to[inductor] (0,2);
```



(2) Многие формы так размещаться в пути не могут, а именно, те, которые имеют больше двух возможных вводов. Кроме того, иногда нужно сначала разместить узлы, возможно используя матрицу, и только затем соединить их. В этом случае, можно использовать опцию вращения, подобную `rotate=90`, в той форме, которую нужно вращать. Следующие четыре ключа позволяют немного проще решить эту задачу:

`/tikz/point up`

То же, что и опция `rotate=90`.

`/tikz/point down`

То же, что и опция `rotate=-90`.

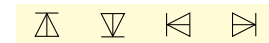
`/tikz/point left`

То же, что и опция `rotate=-180`.

`/tikz/point right`

Ключ ничего не делает.

```
\tikz[circuit ee IEC] \node[diode,point up] {};
\tikz[circuit ee IEC] \node[diode,point down] {};
\tikz[circuit ee IEC] \node[diode,point left] {};
\tikz[circuit ee IEC] \node[diode,point right] {};
```



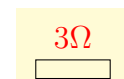
5.2.4 Информационные метки

Информационные метки позволяют добавить текст к символу в схеме. В отличие от нормальных узлов символы схемы обычно не имеют текста, а нужный текст размещается, как правило, рядом с ними (например, текст 3Ω рядом с резистором на схеме ниже). В TikZ определена опция `label` (детали опции `label` см. в [1, 16.10] или в разделе 13.9 пособия «TikZ & PGF. Создание графики в Л^AT_EX2_ε-документах»). Опция `info` строится на основе этой опции, но имеет варианты, полезные при создании схем.

`/tikz/info=[<options>] <angle>:<text>` (no default)

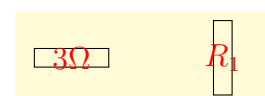
Ключ дает почти такой же результат, как и ключ `label`, только автоматически еще использует стиль `/tikz/every info`, который позволяет управлять информационными метками, обеспечивая простой способ изменения их вида в схемах, не затрагивая другие типы меток. Параметры `<options>` и `<angle>` передаются прямо команде `label`.

```
\tikz[circuit ee IEC,every info/.style=red]
  \node [resistor,info=$3\Omega$] {};
```



Чтобы разместить некоторый текст на основном узле, следует использовать `center` в качестве `<angle>`:

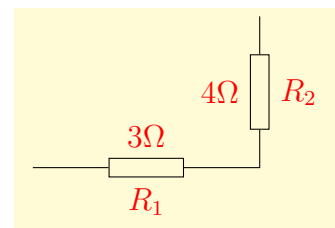
```
\begin{tikzpicture}
[circuit ee IEC,every info/.style=red]
  \node [resistor,info=center:$3\Omega$] {};
  \node [resistor,point up,info=center:$R_1$] at (2,0) {};
\end{tikzpicture}
```



`/tikz/info'=[<options>] <angle>:<text>` (no default)

Работает так же, как и ключ `info` только со значением по умолчанию — `below`. Это означает, что когда используется ключ `info`, метка устанавливается выше узла, а когда используется ключ `info'`, метка устанавливается ниже узла. В случае, когда узел вращается, позиции информационных узлов вращаются соответствующим образом.

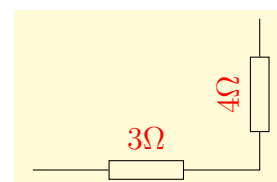
```
\begin{tikzpicture}
  [circuit ee IEC, every info/.style=red]
\draw (0,0)
  to[resistor={info={\$3\Omega\$}, info'={\$R_1\$}}] (3,0)
  to[resistor={info={\$4\Omega\$}, info'={\$R_2\$}}] (3,2);
\end{tikzpicture}
```



`/tikz/info sloped=[<options>] <angle>:<text>` (no default)

Работает подобно ключу `info`, только, когда рисуется метка, устанавливается опция `transform shape`, заставляя метку следовать наклону основного узла.

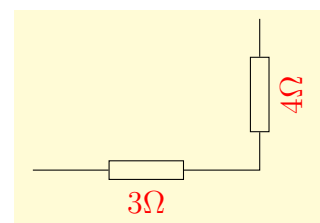
```
\begin{tikzpicture}
  [circuit ee IEC, every info/.style=red]
\draw (0,0)
  to[resistor={info sloped={\$3\Omega\$}}] (3,0)
  to[resistor={info sloped={\$4\Omega\$}}] (3,2);
\end{tikzpicture}
```



`/tikz/info' sloped` (no default)

Комбинация опций `info'` и `info sloped`.

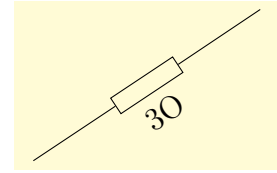
```
\begin{tikzpicture}
  [circuit ee IEC, every info/.style=red]
\draw (0,0)
  to[resistor={info' sloped={\$3\Omega\$}}] (3,0)
  to[resistor={info' sloped={\$4\Omega\$}}] (3,2);
\end{tikzpicture}
```



`/tikz/circuit declare unit={<name>}{<unit>}` (no default)

Позволяет объявлять другие ключи, что облегчает привязку физических единиц к узлу. Идея состоит в том, чтобы сказать `circuit declare unit={ohm}{\Omega}`, после чего запись `ohm=5k` означает то же, что и `info={every ohm}\mathrm{5k\Omega}`. Здесь, `every ohm` — стиль, который позволяет управлять видом этой единицы. Так как ключ `info` используется внутренне, изменяя стиль `every info`, можно изменить вид всех информационных единиц.

```
\begin{tikzpicture}
  [circuit ee IEC,circuit declare unit={my ohm}{0}]
  \draw (0,0) to [resistor={my ohm' sloped=3}] (3,2);
\end{tikzpicture}
```



5.2.5 Объявление и использование аннотаций

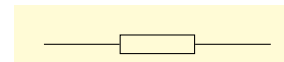
Аннотации очень похожи на информационные метки. Главное их отличие в том, что аннотации, вообще говоря, вызывают нечто, позволяющее по умолчанию нарисовать их, а не рассматривать их как некоторый дополнительный текст (хотя аннотация может добавить и некоторый текст). Аннотацию можно объявить, используя следующий ключ:

```
/tikz/circuit declare annotation={<name>}{<distance>}{<path>} (no default)
```

Объявляет аннотацию с именем <name>. Создает стиль `every <name>`. После того, как аннотация объявлена, она может использоваться как аргумент символа, который добавит графический образ в путь <path> к символу. Опция `label distance` устанавливается равной <distance>. Поясним на примере, как работает этот ключ. Предположим, что нужно создать аннотацию, которая похожа на небольшую круговую стрелку. Нужно сделать объявление такой аннотации, а затем ее использовать:

```
\tikzset{circuit declare annotation=
  {circular annotation} {9pt} {(0pt,8pt) arc (-270:80:3.5pt)} }
```

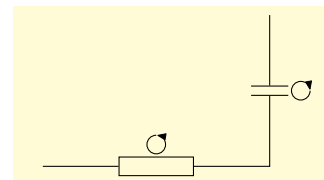
```
\tikz[circuit ee IEC] \draw (0,0) to
  [resistor={circular annotation}] (3,0);
```



Что произошло, ничего не видно? Это следствие того, что параметр <path> становится частью пути, который содержит только символ узла и больше ничего. Такой путь не рисуется и не заполняется. Поэтому следует использовать операцию пути `edge`:

```
\tikzset{circuit declare annotation=
  {circular annotation}{9pt}
  {(0pt,8pt)edge[to path={arc(-270:80:3.5pt)}] ()}}
```

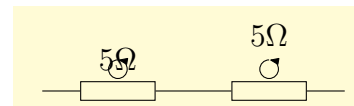
```
\tikz[circuit ee IEC] \draw (0,0)
  to[resistor={circular annotation}] (3,0)
  to[capacitor={circular annotation'}] (3,2);
```



Запись имени аннотации в виде <name'> позволяет разместить аннотацию с другой стороны символа (как и в случае ключа `info'`).

Параметр расстояния <distance> очень важен для правильного размещения дополнительных информационных меток. Когда присутствует аннотация, информационные метки иногда следует размещать подальше от символа. Поэтому аннотация и определяет параметр <distance>, который применяется ко всем информационным меткам, заданным как параметры к аннотации. Вот пример, показывающий эти различия:

```
\tikz[circuit ee IEC] \draw (0,0)
  to[resistor={circular annotation,ohm=5}] (2,0)
  to[resistor={circular annotation={ohm=5}}] (4,0);
```



5.2.6 Графический образ символа

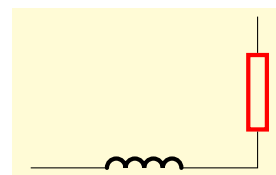
Для каждого символа выбирается определенный графический образ, реально представляющий символ. Графический образ можно изменить несколькими способами:

(1) Можно выбирать разные библиотеки и использовать различные `circuit`-ключи, что приведет к изменению всех графических образов, используемых для символов.

(2) Можно изменить размер графических образов, устанавливая различные значения для `circuit size unit` или используя ключи подобные `small circuit symbols`.

(3) Можно добавить опции к графическому образу символа, или глобально, используя стиль `every circuit symbol`, или локально, используя стиль `every <name>`, где `<name>` — имя символа. Например, на следующем рисунке, символы представляются толстыми, а резисторы — красными.

```
\begin{tikzpicture}[circuit ee IEC,
  every circuit symbol/.style={ultra thick},
  every resistor/.style={red}]
  \draw (0,0) to [inductor] ++(right:3)
    to [resistor] ++(up:2);
\end{tikzpicture}
```



(4) Можно выборочно изменить графический образ, используемый для символа, говоря `set resistor graphic=`.

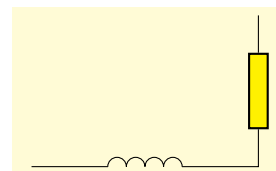
(5) Можно изменить один или несколько из описанных ниже стилей.

`/tikz/circuit symbol open`

(style, initially draw)

Стиль используется с символами, состоящими из линий, окружающими область. Например, IEC-версия резистора является открытым символом.

```
\tikz [circuit ee IEC,circuit symbol open/.style=
  {thick,draw,fill=yellow}]
  \draw (0,0) to [inductor] ++(right:3)
    to [resistor] ++(up:2);
```



`/tikz/circuit symbol filled`

(style, initially draw, fill=black)

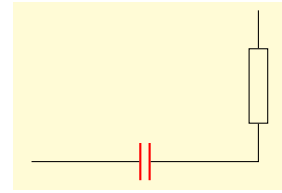
Стиль используется с символами, которые полностью заполняются. Например, катушка индуктивности в IEC является заполняемой (черный прямоугольник).

`/tikz/circuit symbol lines`

(style, initially draw)

Стиль используется с символами, состоящими только из линий, но ничего не окружающими (например, конденсатор).

```
\tikz[circuit ee IEC,circuit symbol lines/.style=
      {thick,draw=red}]
  \draw (0,0) to [capacitor] ++(right:3)
      to [resistor] ++(up:2);
```

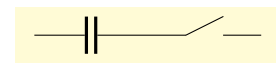


`/tikz/circuit symbol wires`

(style, initially draw)

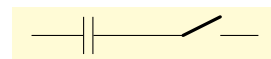
Стиль используется для символов, состоящих только из проводов. Отличие от предыдущего стиля то, что символ, состоящий из проводов, будет выглядеть странно, когда его линии будут более толстыми (например, символ электрического контакта), чем линии нормальных проводов, хотя символы, состоящие из линий (но не проводов) могут хорошо выглядеть, если сделать их более толстыми. Сравните два представления:

```
\tikz [circuit ee IEC,circuit symbol lines/.style=
      {draw,very thick}]
  \draw (0,0) to [capacitor={near start},
      make contact={near end}] (3,0);
```



% хорошо

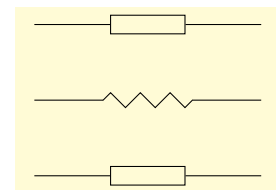
```
\tikz [circuit ee IEC,circuit symbol wires/.style=
      {draw,very thick}]
  \draw (0,0) to [capacitor={near start},
      make contact={near end}] (3,0);
```



% странно

Иногда библиотека для одного символа определяет вариант графического образа; например, в библиотеке IEC есть два варианта для резистора. Потому можно установить вариант графического образа для резистора и вернуться назад к оригиналу, используя ключ `set resistor graphic`:

```
\begin{tikzpicture}[circuit ee IEC]
  \draw (0,2) to [resistor] (3,2); % Стандарт
  \begin{scope}
    [set resistor graphic=var resistor IEC graphic]
    \draw (0,1) to [resistor] (3,1); % Вариант
    \draw[set resistor graphic=resistor IEC graphic]
      (0,0) to [resistor] (3,0); % Вновь стандарт
  \end{scope}
\end{tikzpicture}
```



5.3 Логические схемы

Логическая схема обычно содержит то, что называют логическими вентилями: `and-gate` (И-вентиль), `or-gate` (ИЛИ-вентиль), `xor-gate` (вентиль исключающего ИЛИ) и так далее. Рассмотрим различные библиотеки, которые в принципе могут применяться, и то, как выглядят и используются логические символы.

Есть разные способы изображения логических вентилях, поэтому существуют разные (под)библиотеки для их графического представления. Они определяют графический образ для любого символа, объявленного в библиотеке `circuits.logic`. Эта библиоте-

ка также определяет следующий стиль, который неявно используется многими другими библиотеками.

`/tikz/circuit logic` (no value)

Стиль вызывает ключ `circuit` (который внутренне вызывает стиль `every circuit`), затем определяет ключ `inputs`, а он вызывает стиль `every circuit logic`.

`/tikz/inputs=<inputs>` (no default)

Этот ключ определяется только в окружении `circuit logic`, в котором он дает тот же результат, что и ключ `logic gate inputs` (см. далее).

`/tikz/every circuit logic` (style, no value)

Позволяет управлять внешним видом логических схем.

Так как библиотека `circuit.logic` не определяет графических образов для символов, вместо нее нужно в среде \LaTeX использовать одну из ее (под)библиотек.

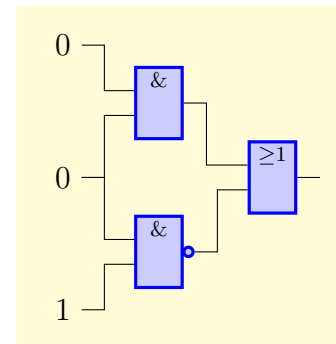
Библиотека `circuits.logic.IEC` определяет графические образы вентиляей, основанные на рекомендации Международной Электротехнической Комиссии. Когда она загружена, можно использовать следующий ключ для определения окружения, содержащего логические схемы, в которых вентили отображаются в данном стиле.

`/tikz/circuit logic IEC` (no value)

Вызывает стиль `circuit logic` и устанавливает IEC-графику для логических символов.

Для каждого графического образа из библиотеки существует стиль, который хранит особенности его представления. Эти ключи называются `and gate IEC graphic` или `gate IEC graphic`, и так далее.

```
\begin{tikzpicture}[circuit logic IEC,
  every circuit symbol/.style={
    logic gate IEC symbol color=black,
    fill=blue!20,draw=blue,very thick}]
\matrix[column sep=7mm,ampersand replacement=\&]
{\node(i0){0};\&\& \& \node[and gate] (a1) {};\&\& \\
\node(i1){0};\& \& \node[or gate] (o) {};\& \\
\node(i2){1};\& \& };\& \& \\
\draw (i0.east) -- ++(right:3mm) |- (a1.input 1);
\draw (i1.east) -- ++(right:3mm) |- (a1.input 2);
\draw (i1.east) -- ++(right:3mm) |- (a2.input 1);
\draw (i2.east) -- ++(right:3mm) |- (a2.input 2);
\draw (a1.output) -- ++(right:3mm) |- (o.input 1);
\draw (a2.output) -- ++(right:3mm) |- (o.input 2);
\draw (o.output) -- ++(right:3mm);
\end{tikzpicture}
```



Библиотека `circuits.logic.US` определяет графические образы, основанные на «Американском» варианте логических вентиляей. Она определяет ключ

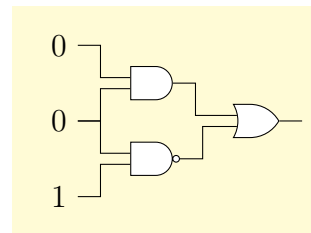
`/tikz/circuit logic US` (no value)

Вызывает стиль `circuit logic`, устанавливает US-графику для логических символов.

```

\begin{tikzpicture}[circuit logic US,
  tiny circuit symbols,
  every circuit symbol/.style={fill=white,draw}]
\matrix[column sep=7mm,ampersand replacement=\&]
{\node(i0){0};\& \& \\
  \& \node[and gate](a1){}; \& \\
  \node(i1){0};\& \& \node[or gate](o){}; \\
  \& \node[nand gate](a2){}; \& \\
  \node(i2){1};\& \& }; \\
\draw (i0.east) -- ++(right:3mm) |- (a1.input 1);
\draw (i1.east) -- ++(right:3mm) |- (a1.input 2);
\draw (i1.east) -- ++(right:3mm) |- (a2.input 1);
\draw (i2.east) -- ++(right:3mm) |- (a2.input 2);
\draw (a1.output) -- ++(right:3mm) |- (o.input 1);
\draw (a2.output) -- ++(right:3mm) |- (o.input 2);
\draw (o.output) -- ++(right:3mm);
\end{tikzpicture}

```



Библиотека `circuits.logic.CDH` определяет графические образы, основанные на варианте логических вентилях из работы А. Croft, R. Davidson, M. Hargreaves (1992), *Engineering Mathematics*, Addison-Wesley, 82–95. Они идентичны US-стилю, за исключением вентилях `and` и `nand`.

`/tikz/circuit logic CDH` (no value)

Вызывает опцию `circuit logic US` и устанавливает специальные вентиля для логических символов `and gate` и `nand gate`.

Различия в графических образах каждой из библиотек приведены в таблице 5.1.

Рассмотрим детали одной из библиотек, другие работают точно так же.

`/tikz/and gate` (no value)

Ключ должен передаваться команде `node`. Это заставляет узел «взглянуть на вентиль» `and`, и точный вид образа вентиля определить по используемому окружению.

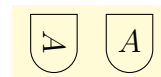
```
\tikz [circuit logic IEC] \node [and gate] {$A$};
```



```

\tikz [circuit logic US]
{\node[and gate,point down] {$A$};
 \node[and gate,point down,info=center:$A$] at (1,0) {}}

```



Для логического вентиля можно определить множество вводов. Однако есть верхний предел для их числа: 1024. Чтобы управлять вводами, используются специальные ключи, которые доступны только в окружении `circuit logic`.

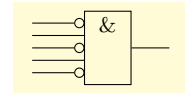
`/tikz/inputs=<input list>` (no default, initially `{normal,normal}`)

Если вентиль имеет k вводов, `<input list>` должен состоять из k символов: буквы i для инвертированного и буквы n для нормального вентиля. Инвертированные вентиля обозначаются небольшим кружком. Якоря для вводов будут устанавливаться соответственно символам, пронумерованным сверху вниз `input 1`, `input 2`, и так далее. Если вентиль имеет только один ввод, якорь называют `input` (без индекса).


```

\begin{tikzpicture}[circuit logic IEC]
  \node[and gate,inputs={inini}] (A) {};
  \foreach \a in {1,...,5}
    \draw (A.input \a -| -1,0) -- (A.input \a);
  \draw (A.output) -- ++(right:5mm);
\end{tikzpicture}

```



Фактически этот ключ — сокращение для `logic gate inputs` (см. далее). Там же описывается, как управлять размерами кругов на инвертированных вводах и как увеличить размер символа, когда слишком много вводов.

У каждого логического вентиля есть только один якорь вывода (`output`).

Ключ	Внешний вид circuit logic IEC	Внешний вид circuit logic US	Внешний вид circuit logic CDH
<code>/tikz/and gate</code>			
<code>/tikz/nand gate</code>			
<code>/tikz/or gate</code>			
<code>/tikz/nor gate</code>			
<code>/tikz/xor gate</code>			
<code>/tikz/xnor gate</code>			
<code>/tikz/not gate</code>			
<code>/tikz/buffer gate</code>			

Таблица 5.1: Графические образы для логических вентиляей.

5.3.1 Библиотека форм логических вентиляей

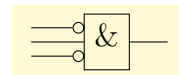
В предыдущих разделах описан интерфейс TikZ для создания логических схем. В этом разделе рассматриваются основные pgf-библиотеки форм. Начнем с основной библиотеки — `shapes.gates.logic`, которая определяет правила обработки вводов, и общие ключи, используемые всеми формами логического блока.

`/pgf//logic gate inputs=<input list>` (no default, initially `{normal,normal}`)

Определяет вводы для логического вентиля. Ключевое слово `inverted` указывает на инвертированный ввод, который будет рисоваться с кружком, прикрепленным к основной форме логического вентиля. Любое ключевое слово, отличное от `inverted` будет

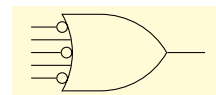
трактоваться как `normal` или не инвертированный ввод (однако, для точности, желательно использовать ключевое слово `normal` или `non-inverted`), и `pgf` не будет рисовать кружок. В обоих случаях якоря для вводов будут установлены и пронумерованы сверху вниз. Если вентиль поддерживает только один ввод, якорь просто называется `input` без числового индекса.

```
\begin{tikzpicture}[minimum height=0.75cm]
  \node[and gate IEC, draw, logic gate inputs=
    {inverted, normal, inverted}] (A) {};
  \foreach \a in {1,...,3}
    \draw (A.input \a -| -1,0) -- (A.input \a);
  \draw (A.output) -- ([xshift=0.5cm]A.output);
\end{tikzpicture}
```



При множестве вводов может возникать длинный список вида `{inverted, normal, inverted}`, в этом случае разрешается использовать сокращение¹: оставить только первые буквы *i* и *n*, и не использовать запятых (запись `ini` эквивалентна списку `{inverted, normal, inverted}`).

```
\begin{tikzpicture}[minimum height=0.75cm]
  \node[or gate US, draw, logic gate inputs=inini] (A) {};
  \foreach \a in {1,...,5}
    \draw (A.input \a -| -1,0) -- (A.input \a);
  \draw (A.output) -- ([xshift=0.5cm]A.output);
\end{tikzpicture}
```

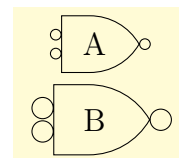


Высота вентиля может быть увеличена, если много вводов. Фактически, она зависит от трех переменных: k — числа вводов, r — радиуса круга, указывающего на инвертированный ввод, и s — расстояния между центрами вводов. Высота вентиля по умолчанию вычисляется по формуле $(k + 1) \times \max\{2r; s\}$, но может быть увеличена.

Радиус кружка для инвертированного ввода (такой же радиус используется для обозначения инвертированного вывода в вентилях `nand`, `nor`, `xnor`, `not`) и расстояние между центрами кружков можно настроить, используя следующие ключи:

`/pgf/logic gate inverted radius=<length>` (no default, initially 2pt)

```
\begin{tikzpicture}[minimum height=0.75cm]
  \tikzset{every node/.style={shape=nand gate CDH,
    draw, logic gate inputs=ii}}
  \node[logic gate inverted radius=2pt] {A};
  \node[logic gate inverted radius=4pt] at (0,-1) {B};
\end{tikzpicture}
```



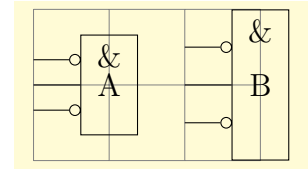
`/pgf/logic gate input sep=<length>` (no default, initially .125cm)

¹Это обобщение идей Джурджена Вербера (Juergen Werber) и Христофа Бартошека (Christoph Bartoschek).

```

\begin{tikzpicture}[minimum size=0.75cm]
\draw [help lines] grid (3,2);
\tikzset{every node/.style={shape=and gate IEC,
draw, logic gate inputs=ini}}
\node[logic gate input sep=0.33333cm]
      at (1,1)(A){A};
\node[logic gate input sep=0.5cm] at (3,1)(B){B};
\foreach \a in {1,...,3}
  \draw (A.input \a -| 0,0) -- (A.input \a)
        (B.input \a -| 2,0) -- (B.input \a);
\end{tikzpicture}

```



Когда pgf изменит высоту логического вентиля, подгоняя ее под число вводов и размерные характеристики радиуса кружка и расстояния между вводами, он изменит другие размеры вентиля так, чтобы их отношение не изменилось (это означает, что изменение высоты изменит ширину и наоборот).

5.3.2 Библиотека форм американских логических вентиляей

После загрузки библиотеки `shapes.gates.logic.US` можно использовать американские формы логических вентиляей и две формы для вентиляей `and` и `nand` с суффиксом CDH. Для каждой формы определяется допустимое число вводов: для `and`, `nand`, `or`, `nor` — два и более; для `xor`, `xnor` — два; для `not`, `buffer` — один. Якоря формы обращаются только к основной части формы и не включают инвертированные вводы или выводы. Еще библиотека определяет дополнительный ключ

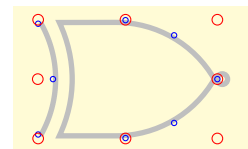
`/pgf/logic gate anchors use bounding box=<boolean>` (no default, initially false)

Когда для ключа установлено значение `true`, он гарантирует, что базовые якоря используют ограничивающий прямоугольник основной формы, который, игнорируя любые инвертированные вводы или выводы, включает все `outer sep`.

```

\begin{tikzpicture}[minimum height=1.5cm]
\node[xnor gate US, draw,
      gray!50,line width=2pt] (A) {};
\foreach \x/\y/\z in {false/blue/1pt,true/red/2pt}
\foreach \a in {north,south,east,west,north east,
               south east,north west,south west}
  \draw[logic gate anchors use bounding box=\x,
        color=\y] (A.\a) circle(\z);
\end{tikzpicture}

```



5.3.3 Библиотека форм логических IEC-вентиелей

После загрузки библиотеки `shapes.gates.logic.IEC` можно использовать прямоугольные формы логических вентиляей, основанные на рекомендации IEC. По умолчанию каждый вентиль рисуется с символом `&` для вентиляей `and`, `nand`, символом `≥ 1` для вентиляей `or`, `nor`, символом `1` для вентиляей `not`, `buffer`, символом `= 1` для вентиляей `xor`,

`xnor`. Эти символы рисуются автоматически, но, строго говоря, не являются частью содержимого узлов. Однако, при увеличении вентиля, гарантируется, что эти символы останутся в пределах границы узла. Для каждой формы определяется допустимое число вводов: оно такое же, как и для одноименных US-вентилей.

Можно изменить указанные символы и их позицию в пределах узла, используя указанные ниже ключи.

`/pgf/and gate IEC symbol=<text>` (no default, initially `\char'\&`)

Устанавливает символ для вентиля `and`. Если узел заполняется цветом, этот цвет будет использоваться и для символа, делая его невидимым, таким образом, необходимо набирать `<text>` подобно `\color {black}\char'\&`. Как вариант, чтобы установить цвет всех символов, можно использовать ключ `logic gate IEC symbol color`. В TikZ, когда используется опция `use IEC style logic gates` можно поменять ключ `and gate IEC symbol` на ключ `and gate symbol`.

Совершенно аналогично используются следующие ключи:

`/pgf/nand gate IEC symbol=<text>` (no default, initially `\char'\&`)

`/pgf/or gate IEC symbol=<text>` (no default, initially `≥ 1`)

`/pgf/nor gate IEC symbol=<text>` (no default, initially `≥ 1`)

`/pgf/xor gate IEC symbol=<text>` (no default, initially `{$=1$}`)

Здесь необходимы фигурные скобки, поскольку символ содержит `=!`

`/pgf/xnor gate IEC symbol=<text>` (no default, initially `{$=1$}`)

Здесь также необходимы фигурные скобки, поскольку символ содержит `=!`

`/pgf/not gate IEC symbol=<text>` (no default, initially `1`)

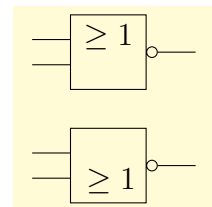
`/pgf/buffer gate IEC symbol=<text>` (no default, initially `1`)

В TikZ, когда используется опция `use IEC style logic gates` можно поменять ключ `* gate IEC symbol` на ключ `* gate symbol` для всех перечисленных ключей.

`/pgf/logic gate IEC symbol align=<align>` (no default, initially `top`)

Устанавливает тип выравнивания символа для логического вентиля (в TikZ, когда используется опция `use IEC style logic gates`, IEC можно опустить). Параметр `<align>` — разделяемый запятыми список из `slottop`, `bottom`, `left`, `right`. Расстояние между границей узла и внешней дугой символа определяется значениями опций `inner xsep`, `inner ysep`.

```
\begin{tikzpicture}[minimum size=1cm,
                    use IEC style logic gates]
  \tikzset{every node/.style={nor gate, draw}}
  \node (A) at (0,1.5) {};
  \node[logic gate symbol align={bottom, right}]
        (B) at (0,0) {};
  \foreach \g in {A, B}{ \foreach \i in {1,2}
                        \draw ([xshift=-0.5cm]\g.input \i) -- (\g.input \i);
                        \draw (\g.output) -- ([xshift=0.5cm]\g.output); }
\end{tikzpicture}
```



`/pgf/logic gate IEC symbol color=<color>` (no default)

Устанавливает цвет для всех символов одновременно. Этот цвет можно переопределить, определяя цвет при установке текста символа каждого символа.

5.4 Электротехнические схемы

5.4.1 Основные библиотеки

Электротехнические схемы содержат символы, подобные резисторам, конденсаторам, источникам напряжения и аннотации к ним. Именно такие символы и аннотации определяют электротехнические библиотеки (сокращенно, *ee*-библиотеки). Так же, как для логических вентилях, есть разные способы прорисовки *ee*-символов. В настоящее время, есть одна основная библиотека `circuits.ee`, и *ee* (под)библиотека `circuits.ee.IEC`, которая использует графику, предложенную IEC.

Библиотека `circuits.ee` объявляет *ee*-символы, стандартные аннотации и единицы, но не определяет графические образы символов, что делает специализированная библиотека `circuits.ee.IEC`. Как и в случае библиотек логических вентилях, библиотека `circuits.ee` определяет ключи, обычно используемые другими библиотеками.

`/tikz/circuit ee` (style, no value)

Стиль вызывает ключ `circuit`, который вызывают стиль `every circuit` и стиль `every circuit ee`, позволяющие управлять внешним видом схем.

Чтобы реально нарисовать схему, следует загрузить библиотеку `circuits.ee.IEC` и воспользоваться стилем

`/tikz/circuit ee IEC` (style, no value)

Вызывает стиль `circuit ee` и устанавливает IEC-графику для элементов *ee*-схем.

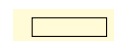
В окружении `circuit ee IEC` можно использовать ключи для символов, единиц и аннотаций, перечисленные далее. Рассмотрим детально только один из них — `resistor` (резистор), все другие работают точно также.

`/tikz/resistor=<options>` (no default)

Ключ должен использоваться только с командами пути `node` или `to`.

Когда ключ `resistor` используется с `node`, он превращает узел в резистор (по умолчанию в прямоугольник).

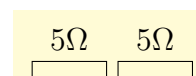
```
\tikz [circuit ee IEC] \node [resistor] {};
```



В отличие от нормальных узлов, узел с резистором не может использовать текст (нельзя написать `node [resistor] {foo}`). Вместо этого, маркировка резистора должна быть проведена, используя одну из опций `label`, `info` или `ohm`.

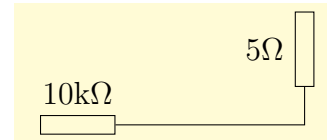
Параметр `<options>` не имеет смысла для опции `resistor`, когда последняя используется вне узла. Параметр `<options>` относится непосредственно к узлу и потому, следующие два выражения дают один и тот же результат:

```
\tikz [circuit ee IEC] \node [resistor,ohm=5] {};  
\tikz [circuit ee IEC] \node [resistor={ohm=5}] {};
```



В схемах часто нужно вращать элементы. Для этого полезны опции `point up`, `point down`, `point left`, `point right` (это сокращения для вращений, аналогичных `rotate=90`).

```
\tikz [circuit ee IEC]{
  \node(R1)[resistor,point up,ohm=5] at (3,1){};
  \node(R2)[resistor,ohm=10k] at (0,0){};
  \draw (R2) -| (R1); }
```

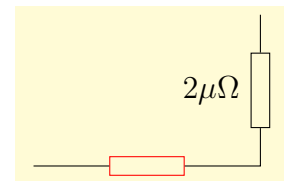


Когда ключ `resistor` в окружении `circuit ee IEC` используется с `to`, внутренне вызывается ключ `circuit handle symbol`, что приводит к следующей последовательности результатов:

- (1) Путь во время построения разрезается, чтобы сформировать место для узла.
- (2) Этот узел будет представлять резистор, который вращается так, чтобы располагаться "вдоль" пути (если нет опции, изменяющей такое поведение).
- (3) Параметры, передаваемые ключу `resistor`, передаются в узел.
- (4) Параметры в `<options>` предварительно анализируются, чтобы выделить ключ `pos` или ключ, подобный `at start` или `midway`, чтобы использовать их для определения, где в пути будет лежать узел.

Так как `<options>` передается узлу резистора в пути, это можно использовать для того, чтобы добавить метку к узлу.

```
\tikz [circuit ee IEC]
  \draw (0,0) to [resistor=red] (3,0)
    to [resistor={ohm=2\mu}] (3,2);
```



Можно добавить несколько меток к резистору, можно иметь несколько резисторов (или других элементов) в одном пути.

Подобно логическим вентилям, все `ee`-символы имеют якоря ввода и вывода. Специальные узлы могут иметь много якорей такого типа. Кроме того, узлы `ee`-символа имеют стандартные якоря для основных направлений.

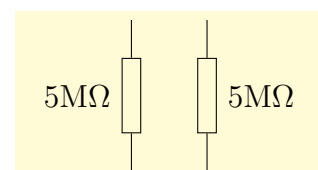
Управлять внешним видом `ee`-символов можно точно так же, как и внешним видом логических вентилях.

Обратимся к определению единиц, например единицы ом (другие предопределенные единицы перечислены в следующем разделе).

`/tikz/ohm=<value>` (no default)

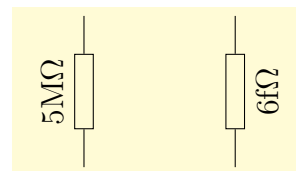
Добавляет к узлу информационную метку $\mathrm{\langle value \rangle \Omega}$. Так как `<value>` набирается внутри `\mathrm`, то набор `ohm=5k` даст в результате метку $5k\Omega$, `ohm=5p` — $5p\Omega$, `ohm=5.6\cdot 10^{\{2\}\mu}` — $5.6 \cdot 10^2 \mu\Omega$. Вместо опции `ohm` можно использовать опцию `ohm'`, которая размещает метку с другой стороны узла.

```
\tikz [circuit ee IEC]
  \draw (0,0) to [resistor={ohm=5M}] (0,2)
    (2,0) to [resistor={ohm'=5M}] (2,2);
```



Наконец, есть опции `ohm sloped` и `ohm' sloped`, которые заставляют информационную метку вращаться вместе с основным узлом.

```
\tikz [circuit ee IEC]
  \draw (0,0) to [resistor={ohm sloped=5M}] (0,2)
    (2,0) to [resistor={ohm' sloped=6f}] (2,2);
```

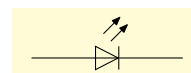


Рассмотрим опции аннотации, и, как пример, опцию аннотацию `light emitting` (другие предопределенные опции аннотации перечислены в следующем разделе). Здесь же отметим, что есть еще и опция аннотация `light emitting'`, которая размещает аннотацию с другой стороны узла.

`/tikz/light emitting=<options>` (no default)

Как и единицы, аннотации должны задаваться через дополнительную опцию узла, что создаст некоторый рисунок (в этом случае, две параллельные линии), размещенный рядом с узлом.

```
\tikz [circuit ee IEC]
  \draw (0,0) to [diode=light emitting] (2,0);
```



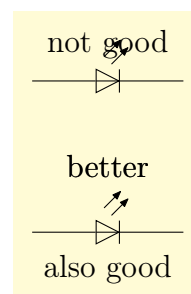
Параметр `<options>` может использоваться для трех разных вещей:

(1) Можно использовать ключи, подобные `red`, чтобы локально изменить внешний вид аннотации.

(2) Можно использовать ключи, подобные `<-` или `-latex`, чтобы изменить направление и вид стрелок, используемых в аннотации.

(3) Можно в `<options>` использовать информационные метки, подобные `ohm=5` или `info=foo`, которые будут добавляться к основному узлу (не к аннотации), но расстояние до метки должно быть изменено так, чтобы поднять ее над аннотацией. Например:

```
\tikz [circuit ee IEC]
  {\draw (0,2)
    to [diode={light emitting,info=not good}] (2,2);
  \draw (0,0)
    to [diode={light emitting={info=better},
              info'=also good}] (2,0);}
```



Внешним видом аннотации можно управлять тремя способами:

(1) Можно установить стиль `every circuit annotation`.

(2) Можно установить стиль `every light emitting`.

(3) Можно использовать стиль `annotation arrow`, по умолчанию устанавливающий стрелку вида `>`:

5.4.2 Предопределенные ee-символы библиотеки IES

Основные ee-символы

Ключ	Внешний вид	Ключ	Внешний вид
/tikz/resistor		/tikz/inductor	
/tikz/capacitor		/tikz/battery	
/tikz/bulb		/tikz/current source	
/tikz/voltage source		/tikz/ground	

Альтернативный вариант основных ee-символов

Ключ	Внешний вид	Ключ	Внешний вид
/tikz/resistor		/tikz/inductor	

Символы, указывающие текущее направление

Ключ	Внешний вид	Ключ	Внешний вид
/tikz/current direction		/tikz/current direction'	

Диоды

Ключ	Внешний вид	Альтернативный вид
/tikz/diode		
/tikz/Zener diode		
/tikz/Schottky diode		
/tikz/tunnel diode		
/tikz/backward diode		
/tikz/breakdown diode		

Контакты

Ключ	Внешний вид	Альтернативный вид
/tikz/contact		
/tikz/make contact		
/tikz/break contact		

Электротехнические единицы

Ключ	Внешний вид	Ключ	Внешний вид
/tikz/ampere	1A	/tikz/volt	1V
/tikz/ohm	1Ω	/tikz/siemens	1S
/tikz/henry	1H	/tikz/farad	1F
/tikz/coulomb	1C	/tikz/voltampere	1VA
/tikz/watt	1W	/tikz/hertz	1Hz

Аннотации

Ключ	Внешний вид	Ключ	Внешний вид
/tikz/light emitting		/tikz/light dependent	
/tikz/direction info		/tikz/adjustable	

Глава 6

Библиотеки декорирования

6.1 Обзор библиотек и общие опции

Библиотеки для декораций определяют много разных декораций, которые могут применяться к пути. На декорации влияет множество параметров, которые можно установить, используя опцию `decoration`. Далее рассматриваются *почти все* опции, определенные для декораций: часть материала о декорациях содержится в главе 18 первого пособия, многие детали надо искать в [1, главы 21, 72], поскольку в пособии не рассматриваются мета-декорации и вопросы построения новых декораций.

`/pgf/decoration/amplitude=<dimension>` (no default, initially 2.5pt)

Определяет желаемую высоту (или амплитуду) тех декораций, для которых это имеет смысл. Для примера, начальное значение в 2.5pt означает, что декорации деформации должны деформировать путь на расстояние в 2.5pt от первоначального пути.

`/pgf/decoration/segment length=<dimension>` (no default, initially 10pt)

Многие декорации состояются из маленьких сегментов. Ключ определяет желаемую длину каждого такого сегмента.

`/pgf/decoration/angle=<degree>` (no default, initially 45)

Причина, по которой некоторые декорации похожи, зависит от настраиваемого угла. Например, декорация `wave` (волна) составлена из дуг и угол раствора для этих дуг задается через `angle`.

`/pgf/decoration/aspect=<factor>` (no default, initially 0.5)

Для некоторых декораций существует естественный коэффициент сжатия. Например, для декорации `brace` опция `aspect` определяет, будет точка фигурной скобки.

`/pgf/decoration/start radius=<dimension>` (no default, initially 2.5pt)

Для некоторых декораций существует естественный начальный радиус (некоторого круга, по-видимому).

`/pgf/decoration/end radius=<dimension>` (no default, initially 2.5pt)

Для некоторых декораций существует естественный конечный радиус (некоторого круга, по-видимому).

`/pgf/decoration/radius=<dimension>` (style, no default)

Устанавливает начальный и конечный радиусы одновременно.

`/pgf/decoration/path has corners=<boolean>` (no default, initially false)

Подсказка к коду декорации относительно того, имеет ли путь углы или нет. Если путь имеет форму угла, устанавливая эту опцию равной `true`, можно получить лучший

вид для декорации, поскольку стыки между исходными сегментами аппроксимируются тщательнее, чем тогда, когда ключ равен `false`. Однако, если путь, скажем, круг, устанавливая ключ равным `true`, получим плохой результат. Большинство декораций этот ключ игнорируют.

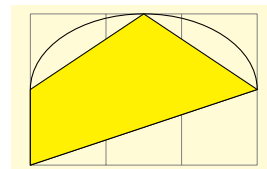
6.2 Декорации трансформации пути

Библиотека `decorations.pathmorphing` определяет декорации трансформации пути, которые изменяют внешний вид декорируемого пути, но не изменяют структуру исходного пути и число его подпутей. Например, если путь состоял из двух кругов и открытой дуги, после процесса декорирования путь будет состоять из двух замкнутых подпутей и одного открытого подпути.

6.2.1 Декорации, использующие только прямые

Декорация **lineto** заменяет путь прямыми линиями. Для каждой кривой, путь просто идет по прямой от начальной точки до конечной точки. В следующем примере, дуга декорируется двумя прямолинейными отрезками.

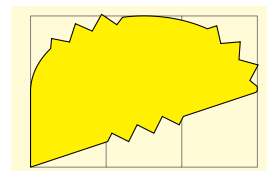
```
\begin{tikzpicture}[decoration=lineto]
\draw [help lines] grid (3,2);
\draw (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```



Декорация **straight zigzag** изменяет путь, чередуя декорации `curveto` и `zigzag`, заканчивая декорацией `curveto`. На вид декорации влияют параметры:

- `amplitude` определяет отклонение от прямой линии;
- `segment length` определяет длину полного цикла «вверх - вниз»;
- `meta-segment length` определяет длину декораций `curveto` и `zigzag`.

```
\begin{tikzpicture}[decoration={straight zigzag,
meta-segment length=1.1cm}]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```



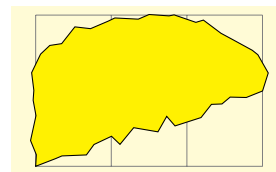
Декорация **random steps** составляется из прямолинейных сегментов, которые располагаются по направлению к концу пути, но на каждом шаге немного сдвигаются случайным образом. На вид декорации влияют параметры:

- `segment length` определяет основную длину каждого шага;
- `amplitude` определяет сегмент $[-d, d]$, в котором случайно выбираются те два значения, на которые сдвигается конец каждого шага в x - и в y -направлении.

```

\begin{tikzpicture}
[decoration={random steps,segment length=2mm}]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}

```



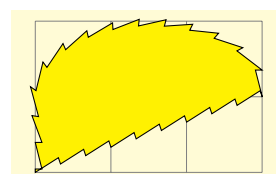
Декорация **saw** похожа на лезвие пилы. На вид декорации влияют параметры:

- `amplitude` определяет, на сколько каждый зубец поднимается над прямой;
- `segment length` определяет длину каждого зубца.

```

\begin{tikzpicture}[decoration=saw]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}

```



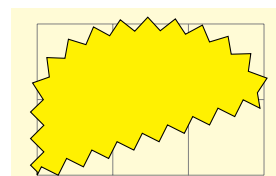
Декорация **zigzag** похожа на зигзагообразную линию. На вид декорации влияют параметры:

- `amplitude` определяет, на насколько каждый зигзаг поднимается над и опускается под прямую;
- `segment length` определяет длину полного цикла «вверх - вниз».

```

\begin{tikzpicture}[decoration=zigzag]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}

```



6.2.2 Декорации, использующие кривые

Декорация **bent** рисует слегка изогнутую линию от начальной до конечной точки:

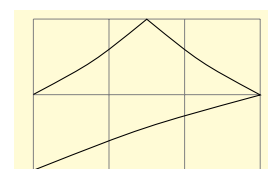
- `amplitude` определяет величину изгиба; амплитуда, равная 0pt, дает прямую.
- `аспект` определяет, насколько плотно прилегает изгиб. Хорошее значение — приблизительно 0.3.

Эту декорацию следует применять только к прямым линиям.

```

\begin{tikzpicture}[decoration=bent]
\draw [help lines] grid (3,2);
\draw [decorate] (0,0) -- (3,1) -- (1.5,2) -- (0,1);
\end{tikzpicture}

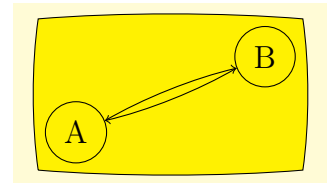
```



```

\begin{tikzpicture}[decoration={bent,aspect=.3}]
\draw [decorate,fill=yellow]
      (0,0) rectangle (3.5,2);
\node[circle,draw] (A) at (.5,.5) {A};
\node[circle,draw] (B) at (3,1.5) {B};
\draw[->,decorate] (A) -- (B);
\draw[->,decorate] (B) -- (A);
\end{tikzpicture}

```



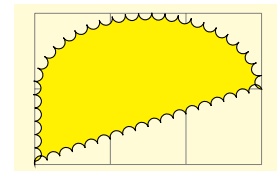
Декорация **bumps** заменяет путь маленькими половинками эллипсов. На вид декорации влияют параметры:

- `amplitude` определяет высоту половины эллипса;
- `segment length` определяет ширину половины эллипса.

```

\begin{tikzpicture}[decoration=bumps]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
      (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}

```



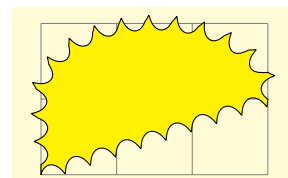
Декорация **coil**. Декорация заменяет путь спиральной линией (она получается при прокатывании по пути трехмерной пружины, которую можно расположить под разными углами к пути). На вид декорации влияют параметры:

- `amplitude` определяет, насколько верхушка спиральной линии возвышается над путем и опускается ниже него (это радиус катушки, прокатываемой по пути).
- `segment length` определяет расстояние между двумя последовательными кривыми (это длина волны спиральной кривой).
- `aspect` определяет «направление взгляда» на перемещаемую пружину: 0 означает «взгляд со стороны», а 0.5 (значение по умолчанию) — «взгляд с торца».

```

\begin{tikzpicture}[decoration=coil]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
      (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}

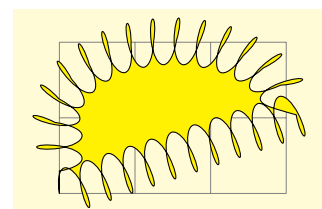
```



```

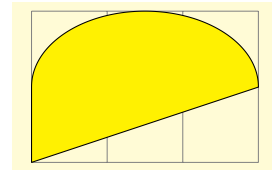
\begin{tikzpicture}[decoration={coil,aspect=0.3,
      segment length=3mm,amplitude=3mm}]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
      (0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}

```



Декорация `curveto` просто приводит линии, следуя первоначальному пути, визуально вроде бы ничего не изменяя. Но, если обратиться к внутренней реализации декорации, она фактически заменяет путь многочисленными маленькими прямыми линиями. Главным образом эта декорация полезна только вместе с мета-декорациями.

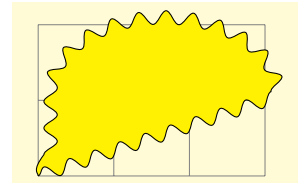
```
\begin{tikzpicture}[decoration=curveto]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```



Декорация `snake` заменяет путь линией, похожей на змею, когда на змею смотрят сверху. Более точно, это — синусоида с сглаженными началом и концом. На вид декорации влияют параметры:

- `amplitude` определяет амплитуду синусоиды;
- `segment length` определяет длину волны синусоиды.

```
\begin{tikzpicture}[decoration=snake]
\draw [help lines] grid (3,2);
\draw [decorate,fill=yellow]
(0,0) -- (3,1) arc (0:180:1.5 and 1) -- cycle;
\end{tikzpicture}
```



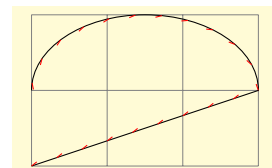
6.3 Декорации, заменяющие путь

Библиотека `decorations.pathreplacing` определяет декорации, которые заменяют декорируемый путь другим путем. В отличие от декорации трансформации, новый путь может сильно отличаться от исходного, например прямая линия может заменяться рядом кругов. Заметим, что заполнение пути, который декорирован, используя одну из декораций этой библиотеки, обычно не заполнит первоначальную область, а скорее несколько меньшую область, ограниченную вновь созданными сегментами пути.

Декорация `border` добавляет к пути прямые линии, которые разворачиваются по отношению в декорируемому пути на определяемый угол. Идея состоит в том, чтобы добавить эти небольшие линии, выделяя границу или область. На вид декорации влияют параметры:

- `segment length` определяет расстояние между добавляемыми линиями;
- `amplitude` определяет длину добавляемой линии;
- `angle` определяет угол между добавляемыми линиями и линией пути.

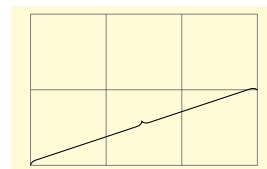
```
\begin{tikzpicture}[decoration=border]
\draw [help lines] grid (3,2);
\draw [postaction={decorate,draw,red}]
(0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



Декорация `brace` заменяет прямую линию длинной фигурной скобкой, левый и правый конец которой располагаются в начальной и конечной точке декорации. Декорация имеет смысл только для прямолинейных путей.

- `amplitude` определяет, на сколько фигурная скобка возвышается над декорируемым путем;
- `aspect` определяет часть от полной длины, в которой должна находиться середина фигурной скобки.

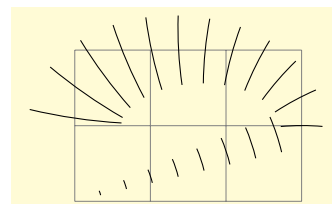
```
\begin{tikzpicture}[decoration=brace]
\draw [help lines] grid (3,2);
\draw [decorate] (0,0) -- (3,1);
\end{tikzpicture}
```



Декорация `expanding waves` добавляет дуги, которые располагаются вдоль декорируемого пути. На вид декорации влияют параметры:

- `segment length` определяет расстояние между последовательными дугами;
- `angle` определяет раствор угла ниже и выше пути. Таким образом, общий раствор угла — этот удвоенный угол.

```
\begin{tikzpicture}
[decoration={expanding waves,angle=5}]
\draw [help lines] grid (3,2);
\draw [decorate]
(0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

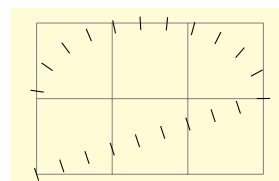


Декорация `moveto` просто переходит в конец пути, используя операцию пути `move-to`. Декорация всегда определена, когда загружена библиотека, описывается здесь ради полноты картины и полезна как `pre=moveto` или `post=moveto` декорация.

Декорация `ticks` заменяет путь прямолинейными отрезками, ортогональными декорируемому пути. На вид декорации влияют параметры:

- `segment length` определяет расстояние между прямолинейными отрезками;
- `amplitude` определяет половину длины прямолинейного отрезка.

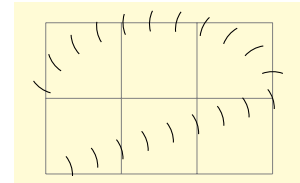
```
\begin{tikzpicture}[decoration=ticks]
\draw [help lines] grid (3,2);
\draw [decorate]
(0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



Декорация `waves` заменяет путь дугами фиксированного размера. На вид декорации влияют параметры:

- `segment length` определяет расстояние между дугами.
- `angle` определяет раствор угла ниже и выше пути. Таким образом, общий раствор угла — этот удвоенный угол;
- `radius` определяет радиус каждой дуги.

```
\begin{tikzpicture}
    [decoration={waves,radius=4mm}]
\draw [help lines] grid (3,2);
\draw [decorate]
    (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



Декорация **show path construction** позволяет сделать что-то свое для каждого типа исходного сегмента в пути (а именно, для сегментов типа `moveto`, `lineto`, `curveto` или `closepath`). Как правило, каждый такой сегмент заменяется другим путем, но не обязательно. Построение такой декорации требует программирования с помощью макросов базового уровня, что выходит за рамки данного пособия (см. примеры [1, р.326–327] и [1, глава 72]).

6.4 Маркировка декораций

При маркировке пути в качестве метки может использоваться любой рисунок, который помещается в определенной позиции пути. Метки нужны в разных ситуациях: их можно использовать, например, для отображения небольшой вспомогательной или подсказывающей информации. По историческим причинам существуют три разных библиотеки меток, которые отличаются видом устанавливаемых маркеров.

6.4.1 Метки общего характера

Метки общего характера из библиотеки `decorations.markings` могут представляться узлами, стрелками и даже целыми рисунками.

Декорация **markings** — основная декорация, определяемая этой библиотекой. О метке можно думать как о небольшой картинке или, более точно, о некотором окружении с содержимым, размещаемом на пути в определенной точке. Предположим, что метка должна быть простым значком пересечения, который строится следующим кодом:

```
\draw (-2pt,-2pt) - (2pt, 2pt); \draw (2pt,-2pt) - (-2pt, 2pt);
```

Если использовать этот код в качестве метки в точке пути `2cm`, то произойдет следующее: `pgf` определит точку, которая отстоит на `2cm` от начала пути; затем переместит сюда систему координат и повернет ее так, что положительная часть x -оси будет касательной к пути; затем создаст окружение, внутри которого будет выполняться приведенный выше код приводящий к появлению на пути небольшого знака пересечения).

Декорация `markings` позволяет разместить на пути одну или несколько меток. Декорация разрушает исходный путь (кроме случаев, рассматриваемых позже) в том смысле, что использует путь для определения точки, но после создания декорации, этот

путь исчезает. Обычно, чтобы добавить метку, надо использовать опцию `postaction` (см. примеры ниже).

Для определения метки используется опция декорации `mark`. Существуют две версии ее использования.

Вот первая версия.

`/pgf/decoration/mark=at position <pos> with <code>` (no default)

Определяет, что, когда применяется декорация `markings`, метка должна располагаться в позиции `<pos>` пути, а код метки задается параметром `<code>`. Параметр `<pos>` может иметь четыре различные формы:

1. Это может быть неотрицательный размер, например, `0pt`, `2cm` или `5cm/2`. В этом случае, он относится к точке пути, которая удалена от начала на указанное расстояние.

2. Это может быть отрицательный размер, например, `-1cm-2pt` или `-1sp`. В этом случае, точка вычисляется от конца пути.

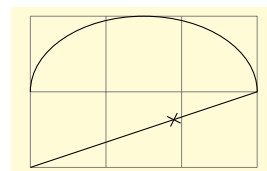
3. Это может быть безразмерное неотрицательное число, например, `1/2`, `0.333+2*0.1`. В этом случае, `<pos>` интерпретируется как частное от полной длины пути. Таким образом, значение `<pos>` равное `0.5` относится к середине пути, а равное `0.1` — к точке близкой к началу пути, и так далее.

4. Это может быть безразмерное отрицательное число, например, `-0.1`. Тогда, снова, оно интерпретируется как частное от полной длины пути, но с отсчетом от конца пути.

Параметр `<pos>` определяет точку пути и, когда метка применяется, система координат преобразуется так, что ее начало помещается в эту точку, а положительная часть x -оси размещается вдоль пути. В этой системе координат выполняется `<code>`. Он может содержать все виды графических команд рисования, включая (даже именованные) узлы. Если точка оказывается вне пути (например, `<pos>` равен `1.2`), то метка не ставится.

Используем приведенный выше код пересечения в простых примерах.

```
\begin{tikzpicture}
[decoration={markings, mark= at position 2cm with
  {\draw (-2pt,-2pt) -- (2pt,2pt);
  \draw (2pt,-2pt) -- (-2pt,2pt);}}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



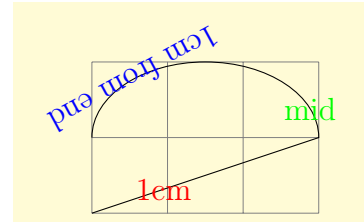
Одну и ту же метку можно разместить на пути несколько раз, но это удобно сделать, используя вторую версию опции `mark` (см. ниже).

Опцию `mark` можно задавать несколько раз, устанавливая разные метки. Но в этом случае нужно, чтобы точки устанавливались на пути строго в порядке возрастания.


```

\begin{tikzpicture}
[decoration={markings,
mark=at position 1cm with \node[red]{1cm};,
mark=at position .5 with \node[green]{mid};,
mark=at position -1cm with
{\node[blue,transform shape]{1cm from end};}}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}

```

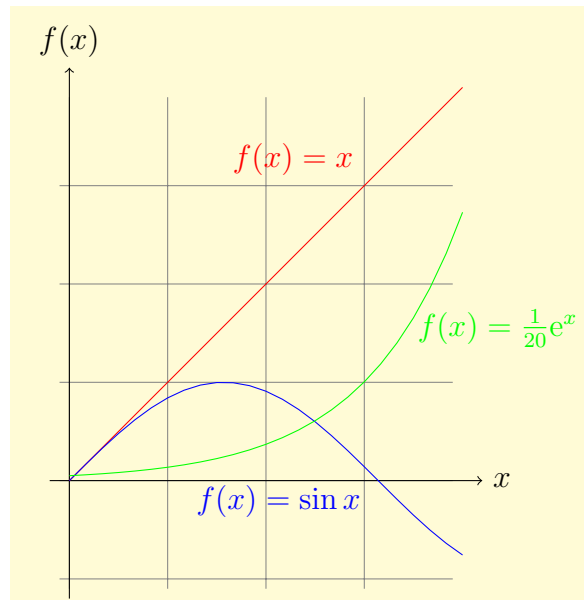


Следующий пример показывает, как, используя метки, поместить текст на график:

```

\begin{tikzpicture} [scale=1.3, domain=0:4,
label/.style={postaction={decorate,
decoration={markings,mark=at position .75 with \node #1;}}}]
\draw[very thin,gray] (-0.1,-1.1) grid (3.9,3.9);
\draw[->] (-0.2,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2) node[above] {$f(x)$};
\draw[red,label={[above left] {$f(x)=x$}}] plot (\x,\x);
\draw[blue,label={[below left] {$f(x)=\sin x$}}] plot (\x,{sin(\x r)});
\draw[green,label={[right] {$f(x)=\frac{1}{20}\mathrm{e}^x$}}]
plot (\x,{0.05*exp(\x)});
\end{tikzpicture}

```



Когда код `<code>` начинает выполняться, устанавливаются два специальных ключа, значения которых интересны и могут оказаться очень полезными:

`/pgf/decoration/mark info/sequence number` (no value)

Данный ключ можно только читать. Его значение (которое можно получить, используя команду `\pgfkeysvalueof`) — последовательный номер метки. Первая метка, добавленная в путь, получает номер 1, вторая — номер 2, и так далее. Ключ полезен для последующей работы с меткой (см. ниже).

`/pgf/decoration/mark info/distance from start` (no value)

Данный ключ тоже можно только читать. Его значение — расстояние метки от начала пути в pt. Например, если длина пути 100pt, и метка находится в его середине, значение этого ключа — 50.0pt.

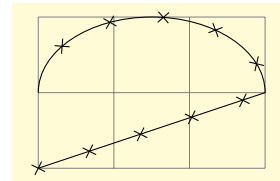
Вот вторая версия использования опции mark.

`/pgf/decoration/mark=`between positions `<start pos>` and `<end pos>`
step `<stepping>` with `<code>` (no default)

Работает подобно первой версии опции, но размещает множество меток, начиная с позиции `<start pos>` с шагом `<stepping>` до позиции `<end pos>`. Все три параметра могут определяться так же, как и параметр `<pos>` в первой версии.

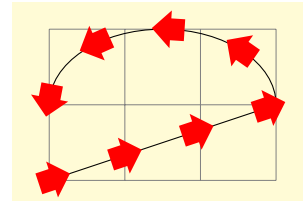
Начнем с обещанного выше примера, в котором на пути размещается несколько (10) однотипных меток пересечений, начиная с начальной точки пути (`<start pos>=0`) и заканчивая конечной точкой (`<end pos>=1`).

```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0 and 1 step 0.1
  with {\draw (-2pt,-2pt) -- (2pt,2pt);
  \draw (2pt,-2pt) -- (-2pt,2pt);}}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



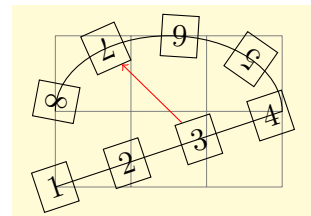
В следующем примере размещаются стрелки вместо пересечений. Обратите внимание на использование опции `transform shape`, которая гарантирует вращение узлов.

```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0 and 1 step 1cm
  with {\node [single arrow,fill=red,
  single arrow head extend=3pt,
  transform shape] {};}]}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



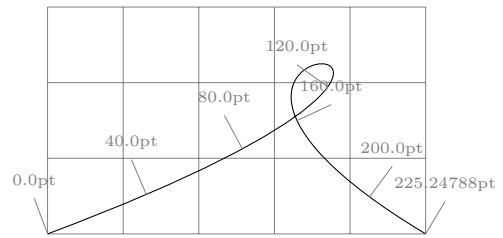
Используя ключ `sequence number` можно нумеровать узлы и обращаться к ним:

```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0 and 1 step 1cm with {
  \node [draw,name=mark-\pgfkeysvalueof{
  /pgf/decoration/mark info/sequence number},
  transform shape]
  {\pgfkeysvalueof{
  /pgf/decoration/mark info/sequence number}};}]}]
\draw [help lines] grid (3,2);
\draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\draw [red,->] (mark-3) -- (mark-7);
\end{tikzpicture}
```



В следующем примере используется информация о расстоянии, которая и размещается на пути:

```
\begin{tikzpicture} [decoration={markings,
  mark= %Главные метки
  between positions 0 and 1 step 40pt with
  {\draw [help lines] (0,0) -- (0,0.5)
  node[above,font=\tiny]{
  \pgfkeysvalueof{/pgf/decoration/mark info/
  distance from start}}; },
  mark=at position -0.1pt with
  {\draw [help lines] (0,0) -- (0,0.5)
  node[above,font=\tiny]{
  \pgfkeysvalueof{/pgf/decoration/mark info/distance from start}};}]
\draw [help lines] grid (5,3);
\draw [postaction={decorate}] (0,0) .. controls (8,3) and (0,3) .. (5,0);
\end{tikzpicture}
```



`/pgf/decoration/reset marks` (no value)

Так как опции `mark` накапливаются, должен быть способ переустановить их так, чтобы все опции `mark`, установленные в замкнутом окружении, не пересекались. Эта опция делает именно это.

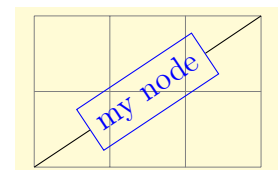
Ранее было сказано, что декорация обычно разрушает путь. Однако, этого не произойдет, если будет установлен следующий ключ.

`/pgf/decoration/mark connection node=<node name>` (no default, initially empty)

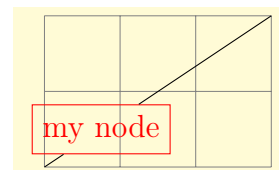
Когда ключ устанавливает непустое имя `<node name>`, в то время как декорация обрабатывается, происходит следующее: код метки должен, среди прочего, определить узел с именем `<node name>`; затем, путь вывода декорации будет содержать `line-to` в одном конце этого узла и `moveto` в другом конце узла. Более точно, первый конец задается позицией на границе узла `<node name>`, который лежит в направлении, «из которого путь подходит к узлу», в то время как другой конец лежит на границе, «в которой путь покидает узел». Кроме того, эта опция заставляет декорацию заканчиваться операцией `line-to` вместо операции до `move-to`. В результате, когда прямая декорируется одной или несколькими метками, которые содержат только узел, прямая будет эффективно связывать эти узлы.

Приведем два примера, показывающие, как это работает.

```
\begin{tikzpicture}[decoration={markings,
  mark connection node=my node,
  mark=at position .5 with
  {\node [draw,blue,transform shape]
  (my node) {my node}}}]
\draw [help lines] grid (3,2);
\draw decorate { (0,0) -- (3,2) };
\end{tikzpicture}
```



```
\begin{tikzpicture}[decoration={markings,
  mark connection node=my node,
  mark=at position .25 with
    {\node [draw,red](my node) {my node};}}]
  \draw [help lines] grid (3,2);
  \draw decorate { (0,0) -- (3,2) };
\end{tikzpicture}
```



6.4.2 Метки в виде стрелок

Создать правильно метки в виде стрелок позволяют две специальные команды, доступные во время выполнения кода `<code>` опции `mark` (они возможны только в таком коде):

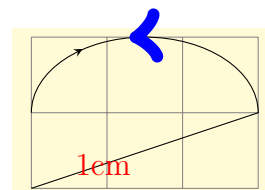
```
\arrow[<options>]{<arrow end tip>}
```

Рисует наконечник стрелки (`<tip>`). Это именно то, что обычно нужно, когда рисуется метка в виде стрелки. Когда используется TikZ, можно задать опции `<options>`, которые выполняются в окружении, содержащем наконечник стрелки.

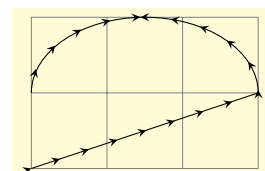
```
\arrowreversed[<options>]{<arrow end tip>}
```

Команда аналогична предыдущей, только наконечник стрелки переворачивается и указывает в другом направлении.

```
\begin{tikzpicture}[decoration={markings,
  mark=at position 1cm with {\node[red]{1cm};},
  mark=at position .75 with
    {\arrow[blue,line width=2mm]{>}},
  mark=at position -1cm with
    {\arrowreversed[black]{stealth};}}]
  \draw [help lines] grid (3,2);
  \draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



```
\begin{tikzpicture}[decoration={markings,
  mark=between positions 0 and .75 step 4mm
    with {\arrow{stealth}},
  mark=between positions .75 and 1 step 4mm
    with {\arrowreversed{stealth}}}]
  \draw [help lines] grid (3,2);
  \draw [postaction={decorate}] (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```

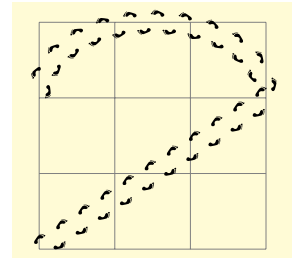


6.4.3 Метки в виде следа

Чтобы ставить такие метки, следует загрузить библиотеку `decorations.footprints`.

Декорация **footprints** рисует небольшие следы вдоль декорируемого пути (первый шаг — с левой ноги).

```
\begin{tikzpicture}[decoration={footprints,
    foot length=5pt, stride length=10pt}]
\draw [help lines] grid (3,3);
\fill [decorate]
    (0,0) -- (3,2) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



Можно повлиять на вид этой декорации, используя несколько опций.

`/pgf/decoration/foot length` (initially 10pt)

Определяет длину или размер следа (но не влияет на длину шага).

```
\begin{tikzpicture}[decoration={
    footprints, foot length=20pt}]
\fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```



`/pgf/decoration/stride length` (initially 30pt)

Определяет длину шага — расстояние между двумя левыми следами.

```
\begin{tikzpicture}[decoration={
    footprints, stride length=50pt}]
\fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```



`/pgf/decoration/foot sep` (initially 4pt)

Отодвигает следы от пути (вверх и вниз) наполовину указанной величины.

```
\begin{tikzpicture}[decoration={
    footprints, foot sep=10pt}]
\fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```



`/pgf/decoration/foot angle` (initially 10)

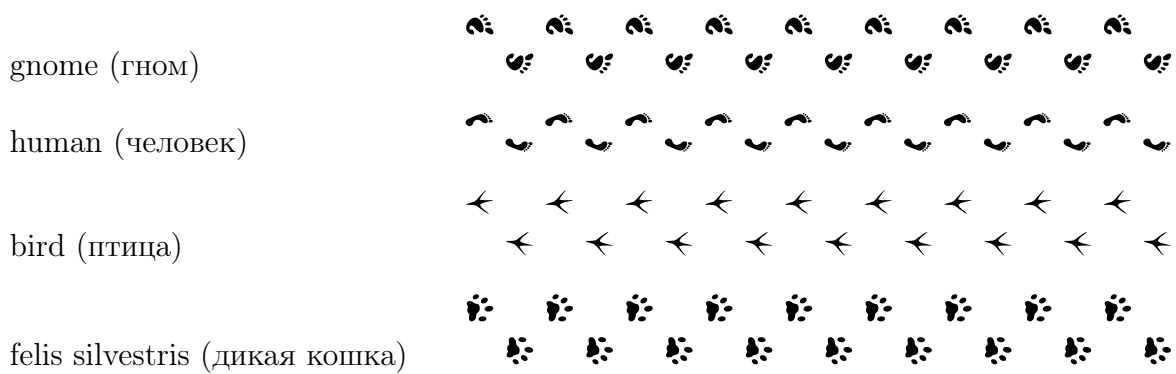
Определяет угол разворота ступней.

```
\begin{tikzpicture}[decoration={
    footprints, foot angle=60}]
\fill [decorate] (0,0) -- (3,0);
\end{tikzpicture}
```



`/pgf/decoration/foot of` (initially human)

Определяет, кто «оставляет» следы. Возможные значения:



6.4.4 Формы фоновых меток

Третья библиотека `decorations.shapes` для добавления меток использует фоновые пути определенных форм¹. Она определяет декорации, которые используют формы (или нечто, подобное формам) для декорирования пути, а следующие опции являются общими для декораций этой библиотеки.

`/pgf/decoration/shape width=<dimension>` (no default, initially 2.5pt)

Определяет желаемую ширину форм. Для декораций, которые поддерживают формы переменных размеров, опция устанавливает как начальную, так и конечную ширину (которые могут переопределяться, используя такие опции как `shape start width`).

`/pgf/decoration/shape height=<dimension>` (no default, initially 2.5pt)

Опция аналогична предыдущей, но определяет высоту форм.

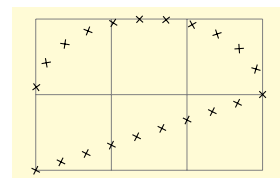
`/pgf/decoration/shape size=<dimension>` (no default)

Опция одновременно устанавливает ширину и высоту форм.

Декорация `crosses` заменяет исходный путь метками пересечения. На вид декорации влияют следующие параметры:

- `segment length` определяет расстояние между соседними метками;
- `shape height` определяет высоту каждой метки;
- `shape width` определяет ширину каждой метки.

```
\begin{tikzpicture}[decoration=crosses]
\draw [help lines] grid (3,2);
\draw [decorate]
(0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}
```



Декорация `triangles` заменяет исходный путь треугольниками, которые указывают направление пути своей вершиной. На вид декорации влияют параметры:

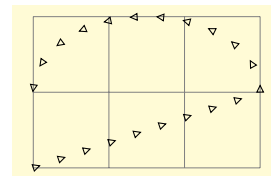
- `segment length` определяет расстояние между соседними треугольниками;
- `shape height` определяет высоту стороны треугольника, ортогональной пути;
- `shape width` определяет ширину каждого треугольника.

¹Эта библиотека рассматривается главным образом по историческим причинам, и обычно предпочтительнее использовать библиотеку `decorations.markings`.

```

\begin{tikzpicture}[decoration=triangles]
\draw [help lines] grid (3,2);
\draw [decorate]
      (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}

```



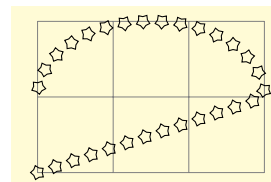
Декорация **shape backgrounds** является общей декорацией, которая заменяет исходный путь повторяемыми копиями фонового пути произвольной формы, которая предварительно определяется, используя команду `\pgfdeclareshape` (это означает, что можно использовать любую форму из библиотеки форм).

Заметим, что используются формы фонового пути, но узлы не создаются. Это означает, что нельзя расположить текст внутри форм этого пути, нельзя их именовать или обращаться к ним. Наконец, эта декорация не будет работать с формами, которые зависят от размера текстового поля (как, например, стреловидные формы). Если какое-то из этих ограничений приводит к проблемам, следует использовать библиотеку `markings`.

```

\begin{tikzpicture}[decoration={shape backgrounds,
                               shape=star,shape size=5pt}]
\draw [help lines] grid (3,2);
\draw [decorate]
      (0,0) -- (3,1) arc (0:180:1.5 and 1);
\end{tikzpicture}

```



```

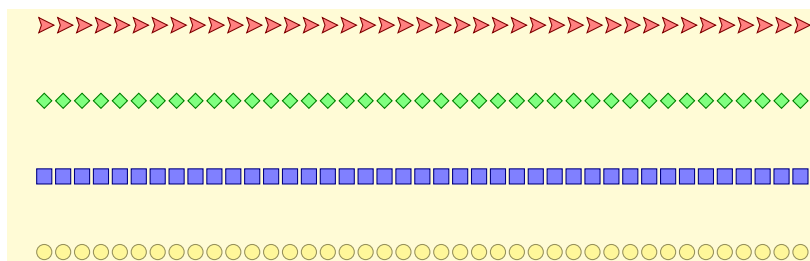
\tikzset{
  paint/.style={draw=#1!50!black, fill=#1!50},
  decorate with/.style={
    decorate,decoration={shape backgrounds,shape=#1,shape size=2mm}}}

```

```

\begin{tikzpicture}[scale=2]
\draw [decorate with=dart,      paint=red]      (0,1.5) -- (5,1.5);
\draw [decorate with=diamond,  paint=green] (0,1) -- (5,1);
\draw [decorate with=rectangle, paint=blue]    (0,0.5) -- (5,0.5);
\draw [decorate with=circle,   paint=yellow]  (0,0) -- (5,0);
\end{tikzpicture}

```



Все формы позиционируются посредством якоря, который определяется через опцию декорации `anchor`:

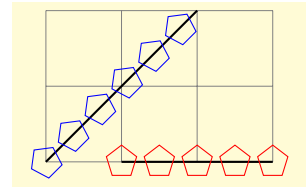
`/pgf/decoration/anchor=<anchor>` (no default, initially center)

Форма фонового пути добавляется в начальную точку пути и, если расстояние между формами соответствующее, в конечную точку пути.

```

\begin{tikzpicture}
  [decoration={shape backgrounds,
    shape=regular polygon,shape size=4mm}]
  \draw[help lines] grid (3,2);
  \draw[thick] (0,0) -- (2,2) (1,0) -- (3,0);
  \draw[red,decorate,decoration={shape sep=.5cm}] (1,0) -- (3,0);
  \draw[blue,decorate,decoration={shape sep=.5cm}] (0,0) -- (2,2);
\end{tikzpicture}

```



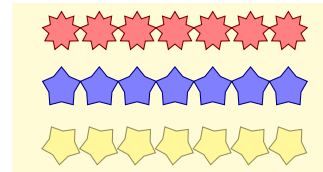
Для настройки конкретных форм могут определяться ключи (например, такие как `star points`, `cloud puffs`, `kite angles` и так далее). Размер формы можно устанавливать принудительно, используя трансформацию. Это означает, что сначала форма набирается с пустым текстовым полем и некоторыми размерами по умолчанию, определяемыми начальной формой. Затем эта форма масштабируется, используя координатные преобразования так, чтобы иметь желаемый размер (который может изменяться по мере продвижения по исходному пути). Это означает, что настройки для используемых углов поворота и расстояний слегка меняются. Наиболее общие опции, типа `inner sep` и `minimum size` игнорируются, но преобразования могут применяться к каждому сегменту, как описано ниже.

```

\tikzset{
  paint/.style={draw=#1!50!black,fill=#1!50},
  my star/.style={decorate,decoration={
    shape backgrounds,shape=star},star points=#1}}

\begin{tikzpicture}[decoration={shape sep=.5cm,
  shape size=.5cm}]
\draw[my star=9, paint=red] (0,1.5) -- (3,1.5);
\draw[my star=5, paint=blue] (0,.75) -- (3,.75);
\draw[my star=5,paint=yellow,
  shape border rotate=30] (0,0) -- (3,0);
\end{tikzpicture}

```



Есть различные ключи, позволяющие управлять прорисовкой форм декорации.

`/pgf/decoration/shape=<shape name>` (no default, initially circle)

Определяет форму, фоновый путь которой используется.

`/pgf/decoration/shape sep=<spacing>` (no default, initially .25cm, between centers)

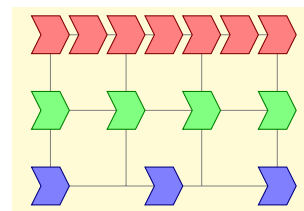
Устанавливает интервал между формами на декорируемом пути. Можно указывать расстояние и, дополнительно, ключевые слова `between centers` и `between borders` (которым должна предшествовать запятая), определяющие, что расстояние устанавливается между якорями центров форм или границами форм.


```

\begin{tikzpicture}[decoration={shape backgrounds,
                             shape size=.5cm,shape=signal},
                  signal from=west, signal to=east,
                  paint/.style={decorate, draw=#1!50!black,
                                fill=#1!50}]
\draw[help lines] grid (3,2);
\draw[paint=red,decoration={shape sep=.5cm}]
      (0,2) -- (3,2);
\draw[paint=green,decoration={shape sep={1cm, between center}}]
      (0,1) -- (3,1);
\draw[paint=blue,decoration={shape sep={1cm, between borders}}]
      (0,0) -- (3,0);

\end{tikzpicture}

```



`/pgf/decoration//shape evenly spread=<number>` (no default)

Этот ключ переопределяет ключ `shape sep` и вынуждает декорацию разместить `<number>` форм равномерно вдоль пути. Если значение `<number>` меньше 1, формы не используются. Если значение `<number>` равно 1, одна форма помещается в середине пути. Можно использовать дополнительные ключевые слова `by centers` (значение по умолчанию, если нет другого ключевого слова) `by borders` (с предшествующей запятой), которые определяют, как устанавливается расстояние между формами. Эти ключевые слова будут иметь значение, если только размеры форм меняются со временем.

`/pgf/decoration/shape scaled` (no default, initially false)

Если этот ключ устанавливается равным `false` (значение по умолчанию), то используются только начальная ширина и высота. Заметим, что ключи `shape width` и `shape height` устанавливают начальные и конечные размеры одновременно.

Можно масштабировать ширину и высоту форм вдоль длины декорируемого пути. Формы масштабируются равномерно от стартового размера до конечного. Следующие ключи настраивают механизм масштабирования форм декорации:

`/pgf/decoration/shape start width=<length>` (no default, initially 2.5pt)

Определяет стартовую ширину формы.

`/pgf/decoration/shape start height=<length>` (no default, initially 2.5pt)

Определяет стартовую высоту формы.

`/pgf/decoration/shape start size=<length>` (style, no default)

Позволяет одновременно определить стартовую ширину и высоту формы.

`/pgf/decoration/shape end width=<length>` (no default, initially 2.5pt)

Определяет рекомендуемую конечную ширину формы. Это та ширина, которую форма примет, только если она будет нарисована точно в конечной точке пути.

`/pgf/decoration/shape end height=<length>` (no default)

Определяет рекомендуемую конечную высоту формы.

`/pgf/decoration/shape end size=<length>` (style, no default)

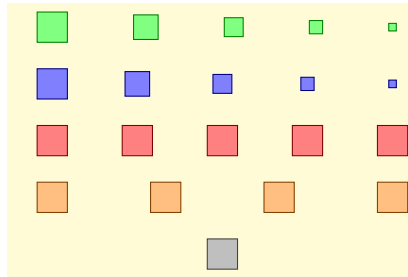
Позволяет одновременно определить конечную ширину и высоту формы.

Проиллюстрируем применение этих опций примерами.

```

\tikzset{paint/.style={draw=#1!50!black,fill=#1!50},
  spreading/.style={decorate,decoration={shape backgrounds,
    shape=rectangle, shape start size=4mm,shape end size=1mm,
    shape evenly spread={#1}}}}
\begin{tikzpicture}
\fill[paint=green,spreading={5,by borders},decoration={shape scaled}
(0,2) -- (3,2);
\fill[paint=blue,spreading={5, by centers},decoration={shape scaled}
(0,1.5) -- (3,1.5);
\fill[paint=red, spreading=5] (0,1) -- (3,1);
\fill[paint=orange, spreading=4] (0,.5) -- (3,.5);
\fill[paint=gray, spreading=1] (0,0) -- (3,0);
\end{tikzpicture}

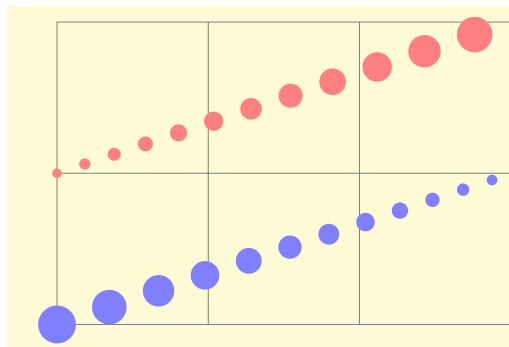
```



```

\tikzset{
  bigger/.style={decoration={shape start size=.125cm,shape end size=.5cm}},
  smaller/.style={decoration={shape start size=.5cm,shape end size=.125cm}},
  decoration={shape backgrounds,
    shape sep={.25cm, between borders},shape scaled}
}
\begin{tikzpicture}[scale=2]
\draw [help lines] grid (3,2);
\fill [decorate,bigger,red!50] (0,1) -- (3,2);
\fill [decorate,smaller,blue!50] (0,0) -- (3,1);
\end{tikzpicture}

```



```

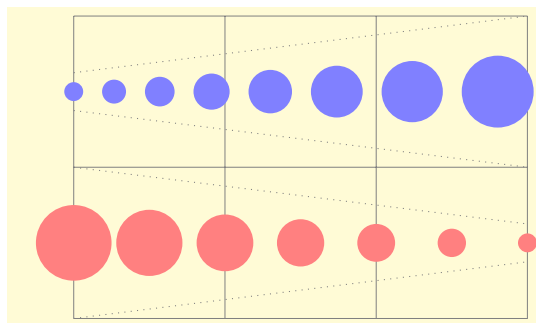
\tikzset{
  bigger/.style={decoration={shape start size=.25cm,shape end size=1cm}},
  smaller/.style={decoration={shape start size=1cm,shape end size=.25cm}},
  decoration={shape backgrounds,shape sep={.25cm,between borders},

```

```

\begin{tikzpicture}[scale=2]
  \draw [help lines] grid (3,2);
  \fill [decorate,bigger,decoration={shape sep={.25cm, between borders}},
        blue!50] (0,1.5) -- (3,1.5);
  \fill [decorate,smaller, decoration={shape sep={1cm, between centers}},
        red!50] (0,.5) -- (3,.5);
  \draw [color=gray, dotted] (0,1.625) -- (3,2) (0,1.375) -- (3,1)
        (0,1) -- (3,.625) (0,0) -- (3,.375);
\end{tikzpicture}

```



`/pgf/decoration/shape sloped=<boolean>` (no default, initially false)

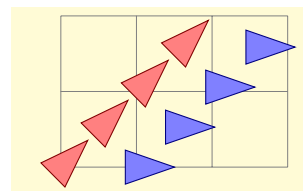
По умолчанию, формы вращаются по направлению пути декорации. Если значение `<boolean>` равно `false` (ложь), то вращения не происходит.

```

\tikzset{
  paint/.style={draw=#1!50!black,fill=#1!50}}

\begin{tikzpicture}[decoration={
  shape width=.65cm, shape height=.45cm,
  shape=isosceles triangle, shape sep=.75cm,
  shape backgrounds}]
\draw[help lines] grid (3,2);
\draw[paint=red,decorate] (0,0) -- (2,2);
\draw[paint=blue,decorate,decoration={shape sloped=false}]
      (1,0) -- (3,2);
\end{tikzpicture}

```



6.5 Текстовые декорации

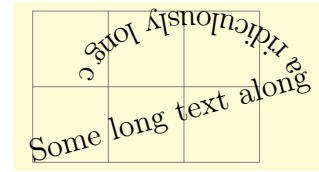
Библиотека `decorations.text` определяет декорации, которые позволяют украсить путь некоторым текстом. Ими можно, например, воспользоваться, чтобы нарисовать текст вдоль некоторой кривой.

Декорация `text along path` украшает путь текстом. Но рисование текста — побочный эффект декорации. Исходный путь используется только для того, чтобы определить, куда должны помещаться символы, и отбрасывается после того, как декорация создана. По этой причине в следующем примере не видна исходная линия.

```

\catcode'\|12
\begin{tikzpicture}
  [decoration={text along path,text={
    Some long text along a ridiculously
    long curve that}}]
\draw [help lines] grid (3,2);
\draw [decorate] (0,0) -- (3.6,1) arc (0:180:1.5 and 1);
\end{tikzpicture}

```



Pgf позволяет «красиво» набирать текст, однако отметим следующие моменты:

- Каждый символ в тексте набирается в отдельном боксе (`\hbox`). Это означает, что если нужны такие вещи, как кернинг или лигатура, нужно это делать в тексте декорации в пределах группы вручную, например, `W{\kern-1ptA}TER` (WATER).
- Каждый символ позиционируется, используя центр его базовой линии. Чтобы переместить текст вертикально (относительно пути), должен использоваться дополнительный ключ.
- Нет гарантии, что символы не будут накладываться, если угол между сегментами много меньше 180° . Но это не должно быть слишком большой проблемой.
- Можно использовать текст в математическом режиме только при значительных ограничениях, поскольку математический режим вводит и широко использует специальные символы. Но даже скромный математический текст вряд ли будет успешно размещен вдоль пути.
- Некоторые неточности позиционирования могут быть особенно заметны на границах. Эту проблему можно решить в ручную (к сожалению) только в каждом конкретном случае.

В текстовой декорации используются следующие ключи:

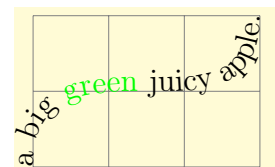
`/pgf/decoration/text=<text>` (no default, initially empty)

Устанавливает текст, набираемый по кривой. Пробелы игнорируются, таким образом, чтобы вставить пробел нужно использовать `\` (или `\space` в L^AT_EX'e). Можно форматировать текст, используя нормальные команды форматирования, типа `\it`, `\bf` и `\color`, в пределах настраиваемых разделителей. Первоначально такими разделителями являются символы `|`.

```

\catcode'\|12
\begin{tikzpicture} \draw [help lines] grid (3,2);
\path [decorate,decoration={text along path,
  text={a big |\color{green}|green||
    juicy apple.}}]
(0,0) .. controls (0,2) and (3,0) .. (3,2);
\end{tikzpicture}

```

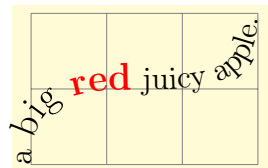


Благодаря первому разделителю со знаком `+`, форматизирующие команды добавляются к любому уже существующему форматированию.

```

\begin{tikzpicture}
\draw [help lines] grid (3,2);
\path [decorate,decoration={text along path,
text={a |\large|big |+\bf\color{red}|red||
juicy apple.}}]
(0,0) .. controls (0,2) and (3,0) .. (3,2);
\end{tikzpicture}

```



Внутренне, текст хранится в макросе `\pgfdecorationtext`. Любые символы, которые не были набраны, поскольку был достигнут конец пути, будут сохранены в макросе `\pgfdecorationrestoftext`.

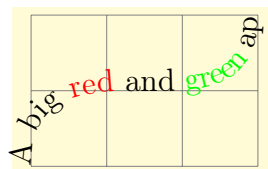
`/pgf/decoration/text format delimiters`={<before>}{<after>} (no default, initially `{|}{|}`)

Устанавливает символы, которые декорация будет использовать при анализе команд форматирования. Если `<after>` отсутствует, то символ `<before>` будет использоваться для обоих разделителей. Символ `+` указывает, что конкретные команды форматирования добавляются к уже существующим.

```

\begin{tikzpicture}
\draw[help lines] grid (3,2);
\path [decorate, decoration={text along path,
text format delimiters={|}{|},
text={A big [\color{red}]red[] and
[\color{green}]green[] apple.}}]
(0,0) .. controls (0,2) and (3,0) .. (3,2);
\end{tikzpicture}

```



`/pgf/decoration/text color`=<color> (no default, initially black)

Определяет цвет текста.

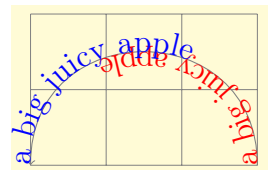
`/pgf/decoration/reverse path`=<boolean> (no default, initially false)

Обращает путь, что полезно для набора текста по разные стороны кривой.

```

\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [gray,->]
[postaction={decoration={text along path,
text={a big juicy apple}, text color=red},
decorate}]
(3,0) .. controls (3,2) and (0,2) .. (0,0);
\end{tikzpicture}

```



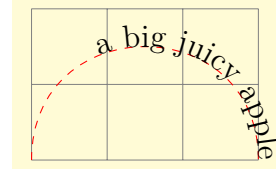
`/pgf/decoration/text align`=<alignment options> (no default)

Изменяет путь на основе опции `/pgf/decoration/text align`, выполняя ключ `<alignment options>`.

`/pgf/decoration/text align/align`=<alignment> (no default, initially left)

Выравнивает текст согласно параметру `<alignment>`, который должен содержать либо `left` (влево), либо `right` (вправо), либо `center` (по центру пути).

```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red, dashed]
  [postaction={decoration={text along path,
    text={a big juicy apple},
    text align={align=right}}, decorate}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```



`/pgf/decoration/text align/left indent=<length>` (no default, initially 0pt)

Определяет расстояние, которое нужно автоматически пройти, прежде чем начать набирать текст.

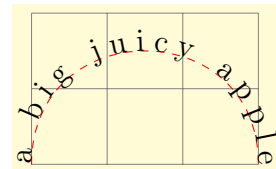
`/pgf/decoration/text align/right indent=<length>` (no default, initially 0pt)

Определяет расстояние до конца пути, на котором автоматически следует прекратить набор текста.

`/pgf/decoration/text align/fit to path=<boolean>` (no default, initially false)

Заставляет декорацию попыбовать заполнить текстом всю длину пути, автоматически смещая вперед на небольшую величину каждый символ заданного текста. Если текст длиннее пути, опция не даст никакого результата.

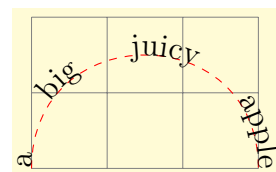
```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [color=red, dashed]
  [postaction={decoration={text along path,
    text={a big juicy apple},
    text align=fit to path}, decorate}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```



`/pgf/decoration/text align/fit to path stretching spaces=<boolean>` (initially false)

Опция аналогична предыдущей, но автоматически раздвигаются только пробелы (включая `\space`, но исключая `\`).

```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [red, dashed]
  [postaction={decoration={text along path,
    text={a big juicy apple}, text align={
    fit to path stretching spaces}}, decorate}]
  (0,0) .. controls (0,2) and (3,2) .. (3,0);
\end{tikzpicture}
```

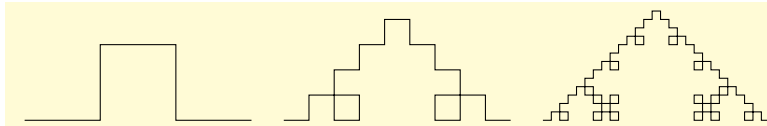


6.6 Рекурсивные декорации

После загрузки библиотеки `decorations.fractals` можно создавать декорации в виде рекурсивно строящихся линий. Для этого, обычно, нужная декорация многократно применяется к первоначально прямому пути.

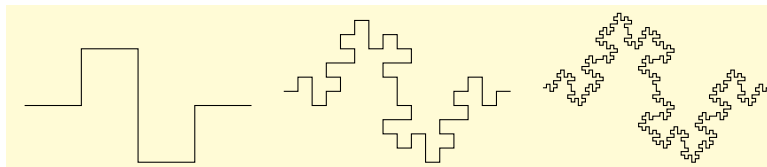
Декорация Koch curve type 1 заменяет среднюю треть отрезка прямоугольным выступом. Многократно применяя замену, можно создать фрактальную кривую Коха.

```
\tikz[decoration=Koch curve type 1]\draw decorate{(0,0) -- (3,0)};
\tikz[decoration=Koch curve type 1]\draw decorate{decorate{(0,0) -- (3,0)}};
\tikz[decoration=Koch curve type 1]
  \draw decorate{decorate{decorate{(0,0) -- (3,0)}}};
```



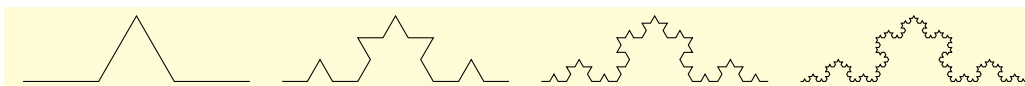
Декорация Koch curve type 2 заменяет отрезок прямоугольным синусом.

```
\tikz[decoration=Koch curve type 2]\draw decorate{(0,0) -- (3,0)};
\tikz[decoration=Koch curve type 2]\draw decorate{decorate{(0,0) -- (3,0)}};
\tikz[decoration=Koch curve type 2]
  \draw decorate{decorate{decorate{(0,0) -- (3,0)}}};
```



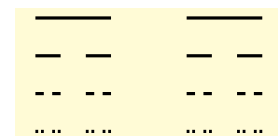
Декорация Koch snowflake заменяет среднюю треть прямолинейного отрезка треугольным выступом.

```
\tikz[decoration=Koch snowflake]\draw decorate{(0,0) -- (3,0)};
\tikz[decoration=Koch snowflake]\draw decorate{decorate{(0,0) -- (3,0)}};
\tikz[decoration=Koch snowflake]
  \draw decorate{decorate{decorate{(0,0) -- (3,0)}}};
\tikz[decoration=Koch snowflake]
  \draw decorate{decorate{decorate{decorate{(0,0) -- (3,0)}}}};
```



Декорация Cantor set выбрасывает из прямолинейного отрезка среднюю треть.

```
\begin{tikzpicture}[decoration=Cantor set,thick]
  \draw decorate{(0,0) -- (3,0)};
  \draw decorate{decorate{(0,-.5) -- (3,-.5)}};
  \draw decorate{decorate{
    decorate{(0,-1) -- (3,-1)}}};
  \draw decorate{decorate{decorate{
    decorate{(0,-1.5) -- (3,-1.5)}}}};
\end{tikzpicture}
```



Глава 7

Библиотека для ER-диаграмм

Библиотека `er` определяет несколько новых стилей, позволяющих рисовать диаграммы «сущность–связь» — ER-диаграммы (Entity-Relationship). Она определяет всего несколько новых стилей, но используя, например, стиль `entity` вместо того, чтобы сказать `rectangle,draw`, можно сделать код более понятным и выразительным.

7.1 Сущности

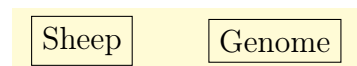
Библиотека определяет простой стиль для рисования сущностей.

`/tikz/entity` (style, no value)

Стиль должен использоваться с узлами, представляя сущности (различные типы объектов) через узлы прямоугольной формы, которые рисуются с минимально возможными разумными размерами.

Заметим, что стиль называется `entity` несмотря на то, что он должен использоваться для узлов, представляющих типы сущностей (различие между сущностью и типом сущности примерно то же, что и различие между экземпляром и классом в объектно-ориентированном программировании).

```
\begin{tikzpicture}
  \node [entity] (sheep) {Sheep};
  \node [entity] (genome)
    [right=of sheep] {Genome};
\end{tikzpicture}
```



`/tikz/every entity` (style, no value)

Этот стиль вызывается стилем `entity`. Чтобы изменить внешний вид сущностей, можно изменить этот стиль.

```
\begin{tikzpicture}
[every entity/.style={
  draw=blue!50,fill=blue!20,thick}]
  \node [entity] (sheep) {Sheep};
  \node [entity] (genome)
    [right=of sheep] {Genome};
\end{tikzpicture}
```



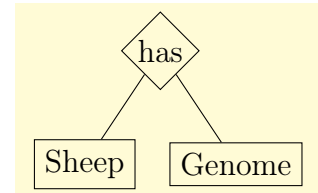
7.2 СВЯЗИ

Связи (или отношения) рисуются, используя стили, очень похожие на стили для сущностей.

`/tikz/relationship` (style, no value)

Стиль работает как `entity`, но только для связей. Снова, `relationships` (связи) — фактически это типы связей.

```
\begin{tikzpicture}
\node[entity] (sheep) at (0,0) {Sheep};
\node[entity] (genome) at (2,0) {Genome};
\node[relationship] at (1,1.5) {has}
edge (sheep) edge (genome);
\end{tikzpicture}
```

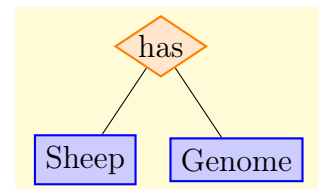


`/tikz/every relationship`

(style, no value)

Аналогичен стилю `every entity`.

```
\begin{tikzpicture}
[every entity/.style={
fill=blue!20,draw=blue,thick},
every relationship/.style={
fill=orange!20,draw=orange,thick,aspect=1.5}]
\node[entity] (sheep) at (0,0) {Sheep};
\node[entity] (genome) at (2,0) {Genome};
\node[relationship] at (1,1.5) {has} edge (sheep) edge (genome);
\end{tikzpicture}
```



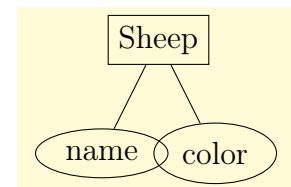
7.3 Атрибуты

`/tikz/attribute`

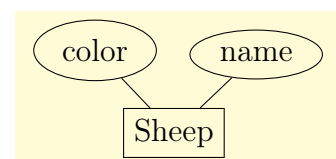
(style, no value)

Стиль используется для того, чтобы указать, что данный узел — атрибут. Можно связать атрибут с его сущностью, используя, например, команду `\child` или опцию `pin`.

```
\begin{tikzpicture}
\node[entity] (sheep) {Sheep}
child {node[attribute] {name}}
child {node[attribute] {color}};
\end{tikzpicture}
```



```
\begin{tikzpicture}
[every pin edge/.style=draw]
\node[entity,pin={[attribute]60:name},
pin={[attribute]120:color}] {Sheep};
\end{tikzpicture}
```



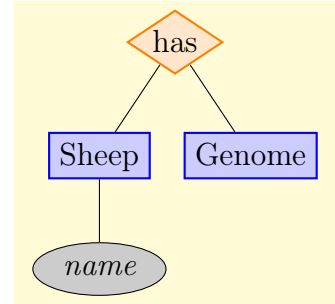
`/tikz/key attribute` (style, no value)

Стиль предназначен для ключевых атрибутов. По умолчанию, такой атрибут будет набираться курсивом. Вместо этого часто используется подчеркивание, но такой вариант выглядит плохо, и его труднее реализовать в \TeX 'е.

`/tikz/every attribute` (style, no value)

Стиль используется каждым (ключевым) атрибутом.

```
\begin{tikzpicture} [text depth=1pt,
  every attribute/.style={
    fill=black!20,draw=black},
  every entity/.style={
    fill=blue!20,draw=blue,thick},
  every relationship/.style={
    fill=orange!20,draw=orange,
    thick,aspect=1.5}]
\node [entity] (sheep) at (0,0) {Sheep}
  child {node [key attribute] {name}};
\node [entity] (genome) at (2,0) {Genome};
\node [relationship] at (1,1.5) {has} edge (sheep) edge (genome);
\end{tikzpicture}
```



Глава 8

Библиотека fit

Библиотека `fit` определяет опции для установки узла так, чтобы он содержал заданный набор координат. После загрузки этой библиотеки становятся доступными опции: `fit`, `every fit`, `rotate fit`.

`/tikz/fit=<coordinates or nodes>` (no default)

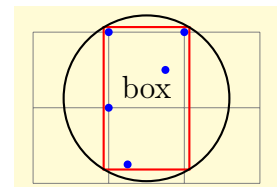
Опция должна задаваться в команде `node`. Параметр `<coordinates or nodes>` должен быть последовательностью из TikZ-координат (точек) или имен узлов, один после другого без запятых (как и в операции пути `plot coordinates`). Например,

`(1,0) (2,2)` или `(a) (1,0) (b)`, где `a` и `b` --- узлы.

Для заданной последовательности вычисляется минимальный ограничивающий прямоугольник, который охватывает все перечисленные точки и/или узлы. Для точек из списка гарантируется, что ограничивающий прямоугольник будет содержать их, для узлов гарантируется, что ограничивающий прямоугольник будет содержать якоря `east`, `west`, `north` и `south` перечисленных узлов. В принципе (детали чуть позже), все устанавливается так, что текстовое поле узла с опцией `fit` будет в точности этим ограничивающим прямоугольником.

Рассмотрим пример: заключим несколько точек в прямоугольный узел. Устанавливая `inner sep` равным нулю, получим точно текстовое поле узла. Затем заключим эти точки снова в круговой узел. Заметим, что круг охватывает первый ограничивающий прямоугольник.

```
\begin{tikzpicture}[inner sep=0pt,thick,
  dot/.style={fill=blue,circle,minimum size=3pt}]
\draw[help lines] (0,0) grid (3,2);
\node[dot] (a) at (1,1) {};
\node[dot] (b) at (2,2) {};
\node[dot] (c) at (1,2) {};
\node[dot] (d) at (1.25,0.25) {};
\node[dot] (e) at (1.75,1.5) {};
\node[draw=red, fit=(a) (b) (c) (d) (e)] {box};
\node[draw,circle,fit=(a) (b) (c) (d) (e)] {};
\end{tikzpicture}
```



Когда используется опция `fit`, к узлу также применяется стиль

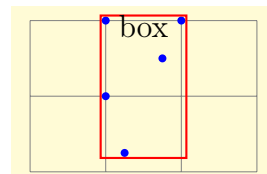
`/tikz/every fit` (style, initially empty)

Стиль позволяет изменить вид узлов, использующих опцию `fit`. Точный результат применения этой опции следующий:

1. Вычисляется минимальный ограничивающий прямоугольник, содержащий все координаты. Если в списке к опции задаются узлы, например, (a), то ограничивающий прямоугольник будет содержать точки (a.north), (a.south), (a.west) и (a.east). Если нужно учитывать только один якорь узла, например, (a.center), его и нужно задавать вместо (a).
2. Опция `text width` устанавливается равной ширине вычисленного ограничивающего прямоугольника.
3. Устанавливается опция `align=center`.
4. Опция `anchor` устанавливается равной `center`.
5. Позиция `at` узла устанавливается равной центру вычисленного ограничивающего прямоугольника.
6. После того, как узел набран, его высота и глубина корректируются так, чтобы они в целом составляли высоту вычисленного ограничивающего прямоугольника, а текст узлов вертикально центрируется внутри поля.

Разместить текст, не центрируя его, несколько сложнее. Для этого можно использовать опции `label` или `pin`. Но можно поступить и по-другому. Предположим, например, что в примере выше, слово `box` должно появиться в верхней части поля. Задачу можно решить следующим образом: создать узел с опцией `fit`, который не содержит текста, дать ему имя, а затем использовать обычный узел, чтобы поместить текст в нужное место.

```
\begin{tikzpicture}[inner sep=0pt,thick,
  dot/.style={fill=blue,circle,minimum size=3pt}]
\draw[help lines] (0,0) grid (3,2);
\node[dot] (a) at (1,1) {};
\node[dot] (b) at (2,2) {};
\node[dot] (c) at (1,2) {};
\node[dot] (d) at (1.25,0.25) {};
\node[dot] (e) at (1.75,1.5) {};
\node[draw=red,fit=(a) (b) (c) (d) (e)] (fit) {};
\node[below] at (fit.north) {box};
\end{tikzpicture}
```



Рассмотрим более сложный пример, в котором используется опция `fit`:

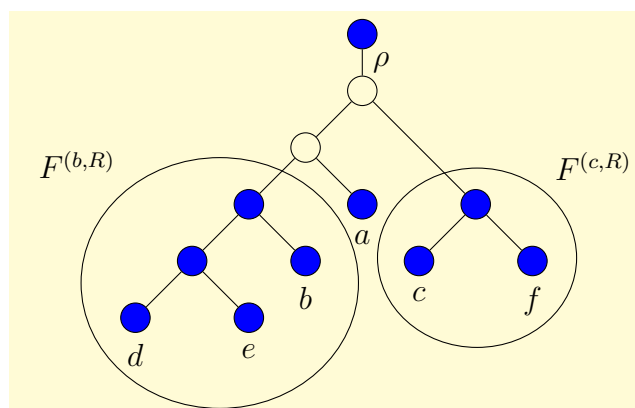
```
\begin{tikzpicture}
[vertex/.style={minimum size=2pt,fill=blue,draw,circle},
open/.style={fill=none},sibling distance=1.5cm,level distance=.75cm,
every fit/.style={ellipse,draw,inner sep=-2pt},
leaf/.style={label={[name=#1]below:$\#1$}},auto]
\node [vertex] (root) {}
child { node [vertex,open] {}
child { node [vertex,open] {}
child { node [vertex] (b's parent) {}
child { node [vertex] {}

```

```

    child { node [vertex,leaf=d] {} }
      child { node [vertex,leaf=e] {} } }
    child { node [vertex,leaf=b] {} } }
  child { node [vertex,leaf=a] {} } }
child { node [coordinate] {}
  child[missing]
  child { node [vertex] (f's parent) {}
    child { node [vertex,leaf=c] {} }
    child { node [vertex,leaf=f] {} } } }
edge from parent node {\rho} };
\node [fit=(d) (e) (b) (b's parent),label=above left:$F^{(b,R)}$] {};
\node [fit=(c) (f) (f's parent),label=above right:$F^{(c,R)}$] {};
\end{tikzpicture}

```



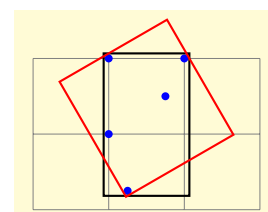
`/tikz/rotate fit=<angle>` (no default, initially 0)

Ключ позволяет поместить `<coordinates or nodes>` внутри узла, который повернут на угол `<angle>`. Дает то же эффект, что и опция `/tikz/rotate`.

```

\begin{tikzpicture}[inner sep=0pt,thick,
  dot/.style={fill=blue,circle,minimum size=3pt}]
\draw[help lines] (0,0) grid (3,2);
\node[dot] (a) at (1,1) {};
\node[dot] (b) at (2,2) {};
\node[dot] (c) at (1,2) {};
\node[dot] (d) at (1.25,0.25) {};
\node[dot] (e) at (1.75,1.5) {};
\node[draw, fit=(a) (b) (c) (d) (e)] {};
\node[draw=red, rotate fit=30,
  fit=(a) (b) (c) (d) (e)] {};
\end{tikzpicture}

```



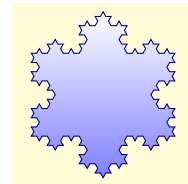
Глава 9

Библиотека `lindenmayersystems`

Системы Линдермайера или L-системы, были первоначально разработаны Аристидом Линдермайером (Aristid Lindenmayer) как теория роста морских водорослей и впоследствии использовались для моделирования типов ветвления растений и создания шаблонов фракталов. Как правило, L-системы состоят из ряда символов, каждый из которых ассоциируется с графическим действием (типа «повернуть налево», или «переместиться вперед»), и рядом правил (правило подстановки или правило перезаписи). Данная строка символов, правила подстановки применяются несколько раз и, когда следующая строка создана, выполняется действие, связанное с каждым символом.

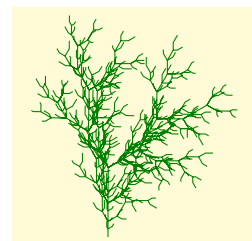
В `pgf` используются L-системы, позволяющие создать простые 2-мерные рекурсивные модели ...

```
\begin{tikzpicture}
\pgfdeclarelindenmayersystem{Koch curve}{
  \rule{F -> F-F++F-F}}
\shadedraw [top color=white, bottom color=blue!50,
            draw=blue!50!black]
[1-system={Koch curve, step=2pt, angle=60,
            axiom=F++F++F, order=3}]
lindenmayer system -- cycle;
\end{tikzpicture}
```



...и модели живых растений ...

```
\begin{tikzpicture}
\draw [green!50!black, rotate=90]
[1-system={rule set={F -> FF-[-F+F]+[+F-F]},
            axiom=F, order=4, step=2pt,
            randomize step percent=25, angle=30,
            randomize angle percent=5}] lindenmayer system;
\end{tikzpicture}
```



...но важно иметь ввиду, что даже умеренно сложные L-системы могут быстро исчерпать предоставленную системе `TeX` память, и потребовать для своего создания много времени. Превосходным введением в L-системы, содержащим некоторые красивые рисунки многие из которых невозможно создать в `pgf`), является работа «The Algorithmic Beauty of Plants», Przemyslaw Prusinkiewicz and Aristid Lindenmayer (Алгоритмическая

красота растений, Пржемыслав Присинкевич и Аристид Линдермайер), которая свободно доступна через Интернет.

Библиотека `lindenmayersystems` определяет команды для построения L-систем (это `pgf`-библиотека) и предоставляет внешний (пользовательский) интерфейс для построения L-систем в `TikZ` (`TikZ`-библиотека).

9.1 Объявление L-системы

До того, как использовать L-систему, ее нужно объявить командой

```
\pgfdeclarelindenmayersystem{<name>}{<specification>}
```

Объявляет L-систему с именем `<name>`. Параметр `<specification>` содержит описание символов L-системы и ее правил. Две команды `\symbol` и `\rule` определяются только тогда, когда выполняется параметр `<specification>`.

```
\symbol{<name>}{<code>}
```

Определяет символ с именем `<name>` для конкретной L-системы, и связанный с ним код `<code>`. Символ должен состоять из одного алфавитно-цифрового знака (то есть, A-Z, a-z или 0-9). Символы F, f, +, -, [,] доступны по умолчанию, так что их не надо определять для каждой L-системы. Однако, если требуется, их можно переопределить для конкретной L-системы. L-система обрабатывает символы со значениями по умолчанию следующим образом (команды, которые они выполняют, описаны ниже):

- F — продвинуться вперед на определенное расстояние, рисуя линию. Использует `\pgflsystemdrawforward`.
- f — продвинуться вперед на определенное расстояние, но не рисуя линию. Использует `\pgflsystemmoveforward`.
- + — повернуть налево под некоторым углом. Использует `\pgflsystemturnleft`.
- — — повернуть направо под некоторым углом. Использует `\pgflsystemturnright`.
- [— сохранить текущее состояние (то есть, позицию и направление). Использует `\pgflsystemsavestate`.
-] — восстановить последнее сохраненное ранее состояние L-системы. Использует `\pgflsystemrestorestate`.

Символы [и] воздействуют на стек: [помещает состояние L-системы в стеке, а] выталкивает состояние L-системы из стека.

Когда выполняется код `<code>`, матрица преобразования устанавливается так, чтобы начало было в текущей позиции и положительная часть x -оси «указывала вперед», так что код `\pgfpathlineto{\pgfpoint{1cm}{0cm}}` чертит линию в 1cm вперед.

Следующие ключи могут изменить результат построения L-системы. Однако, они в себе не сохраняют значения.

```
/pgf/lindenmayer system/step=<length> (no default, initially 5pt)
```

Определяет насколько далеко L-система продвигается вперед, если это требуется. Опция устанавливает \TeX -размер `\pgflsystemstep`.

```
/pgf/lindenmayer system/randomize step percent=<percentage> (no default, initially 0pt)
```

Если шаг является случайным, этот ключ определяет на сколько. Значение сохраняется в \TeX -макресе `\pgflsystemrandomizesteppercent`.

```
/pgf/lindenmayer system/left angle=<angle> (no default, initially 90)
```

Опция устанавливает угол, на который поворачивается L-система, когда поворачивается налево. Значение сохраняется в Т_ЕX-макросе `\pgflsystemrleftangle`.

`/pgf/lindenmayer system/right angle=<angle>` (no default, initially 90)

Опция устанавливает угол, на который поворачивается L-система, когда поворачивается направо. Значение сохраняется в Т_ЕX-макросе `\pgflsystemrrightangle`.

`/pgf/lindenmayer system/randomize angle percent=<percentage>` (no default, initially 0)

Если углы являются случайными, этот ключ определяет на сколько. Значение сохраняется в Т_ЕX-макросе `\pgflsystemrandomizeanglepercent`.

Ради скорости и удобства, когда код для символа выполняется, доступны следующие команды.

`\pgflsystemcurrentstep`

Текущий шаг L-системы (как далеко система продвинется, если будет нужно). Изначально устанавливает значение в Т_ЕX-размер `\pgflsystemstep`, но значение может быть изменено, если используется команда `\pgflsystemrandomizestep` (см. ниже).

`\pgflsystemcurrentleftangle`

Угол, на который L-система повернет, когда она поворачивает налево. Значение, которое хранится в этом макросе, может быть изменено, если используется команда `\pgflsystemrandomizeleftangle`.

`\pgflsystemcurrentrightangle`

Угол, на который L-система повернет, когда она поворачивает направо. Значение, которое хранится в этом макросе, может быть изменено, если используется команда `\pgflsystemrandomizerightangle`.

Следующие команды полезны, если нужно определять собственные символы.

`\pgflsystemrandomizestep`

Порождает случайное значение в `\pgflsystemcurrentstep` согласно значению ключа `randomize step percent`.

`\pgflsystemrandomizeleftangle`

Порождает случайное значение в `\pgflsystemcurrentleftangle` согласно значению ключа `randomize angle percent`.

`\pgflsystemrandomizerightangle`

Порождает случайное значение в `\pgflsystemcurrentrightangle` согласно значению ключа `randomize angle`.

`\pgflsystemdrawforward`

Порождает движение вперед в текущем направлении на `\pgflsystemcurrentstep` и чертит линию.

`\pgflsystemmoveforward`

Порождает движение вперед в текущем направлении на `\pgflsystemcurrentstep`, но не чертит линию.

`\pgflsystemturnleft`

Порождает поворот налево на `\pgflsystemcurrentleftangle`.

`\pgflsystemturnright`

Порождает поворот направо на `\pgflsystemcurrentrightangle`.

`\pgflsystemsavestate`

Сохраняет текущую позицию и ориентацию, просто запуская новую Т_ЕX-группу.

`\pgflsystemrestorestate`

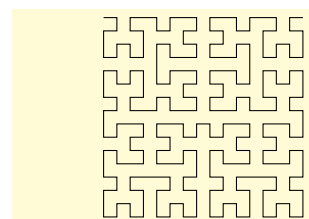
Восстанавливает последнюю сохраненную позицию и ориентацию.

`\rule{<head>-><body>}`

Объявляет правило; <head> — единственный символ, который еще не был объявлен, используя команду `\symbol` или уже должен существовать как символ по умолчанию (фактически, наиболее интересные L-системы зависят от использования символов без соответствующего кода, управляющего ростом системы); <body> — строка символов, которая снова не должна иметь кода, присоединенного к ней.

Как пример, следующая L-система использует некоторые из этих команд и иллюстрирует ту точку зрения, согласно которой некоторые символы, в этом случае A и B, не должны иметь кода, связанного с ними. Они просто управляют ростом системы.

```
\pgfdeclarelindenmayersystem{Hilbert curve}{
  \symbol{X}{\pgflsystemdrawforward}
  % Явно определяются символы + и -.
  \symbol{+}{\pgflsystemturnright}
  \symbol{-}{\pgflsystemturnleft}
  \rule{A -> +BX-AXA-XB+}
  \rule{B -> -AX+BXB+XA-}
}
```



```
\tikz\draw[lindenmayer system={Hilbert curve,axiom=A,order=4,angle=90}]
  lindenmayer system;
```

9.2 Использование L-систем

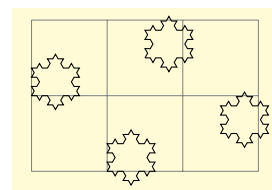
9.2.1 L-системы в PGF

Следующая команда используется, чтобы запустить L-систему в pgf:

`\pgflindenmayersystem{<name>}{<axiom>}{<order>}`

Строит L-систему с именем <name>, используя входную строку <axiom> для указанного числа <order> итераций. В дополнение должны быть установлены соответствующие ключи.

```
\begin{tikzpicture}
\draw [help lines] grid (3,2);
\pgfset{lindenmayer system/.cd,angle=60,step=2pt}
\foreach \x/\y in
  {0cm/1cm, 1.5cm/1.5cm, 2.5cm/0.5cm, 1cm/0cm}{
  \pgftransformshift{\pgfqpoint{\x}{\y}}
  \pgfpathmoveto{\pgfpointorigin}
  \pgflindenmayersystem{Koch curve}{F++F++F}{2}
  \pgfusepath{stroke}
}
\end{tikzpicture}
```



Заметим, что, для L-системы можно определять специальные символы, которые выполняют операции `move-to` и другие операции пути.

9.2.2 L-системы в TikZ

В TikZ, L-системы создаются, используя операцию пути. Однако, TikZ более гибок в вопросах позиционирования L-системы, а также обеспечивает ключи, чтобы создать L-системы «on-line».

```
\path ... lindenmayer system[<keys>]...;
```

Запускает L-систему согласно параметрам, определенным в опции <keys> (которая может содержать и обычные опции, типа `draw` или `thin`). Синтаксис достаточно гибок по отношению к параметрам L-системы, и следующие три выражения делают одно и то же (результат см. выше в конце раздела 9.1):

```
\draw lindenmayer system
  [lindenmayer system={Hilbert curve, axiom=A, order=4, angle=90}];

\draw [lindenmayer system={Hilbert curve, axiom=A, order=4, angle=90}]
  lindenmayer system;

\tikzset{lindenmayer system={Hilbert curve, axiom=A, order=4, angle=90}}
\draw lindenmayer system;

\path ... l-system[<keys>] ...;
```

Компактная версия команды `lindenmayer system`.

Библиотека `lindenmayersystems` определяет некоторые дополнительные ключи для L-систем, которые работают только в TikZ, должны располагаться в одном и том же пути, а именно, в пути `/pgf/lindenmayer system`, но сохранять повторно этот путь не нужно, так как уже обеспечены следующие ключи:

```
/pgf/lindenmayer system[<keys>] (style, no value)
```

Псевдоним: `/tikz/lindenmayer system`

Изменяет ключевой путь на `/pgf/lindenmayer system` и выполняет <keys>.

```
/pgf/l-system[<keys>] (style, no value)
```

Псевдоним: `/tikz/l-system`

Компактная версия предыдущего ключа.

```
/pgf/lindenmayer system/name={<name>} (no default)
```

Устанавливает имя для L-системы.

```
/pgf/lindenmayer system/axiom={<string>} (no default)
```

Устанавливает аксиому (или вводит строку) для L-системы.

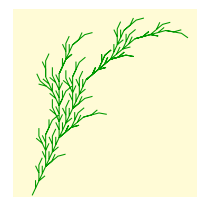
```
/pgf/lindenmayer system/order={<integer>} (no default)
```

Устанавливает число итераций при построении L-системы.

```
/pgf/lindenmayer system/rule set={<list>} (no default)
```

Ключ позволяет (анонимной) L-системе быть объявленной «on-line». Есть, однако, ограничение, позволяющее использовать для рисования только символы по умолчанию (пустые символы также могут использоваться, чтобы управлять ростом системы). Правила в <list> должны разделяться запятыми.

```
\tikz[rotate=65]\draw [green!60!black] l-system
  [l-system={rule set={F -> F[+F]F[-F]}, axiom=F,
    order=4, angle=25, step=3pt}};
```



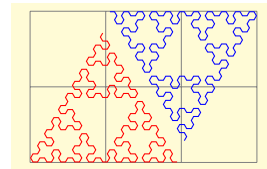
`/pgf/lindenmayer system/anchor=<anchor>` (no default)

По умолчанию, когда этот ключ не используется, L-система начнется с последней определенной точки. Используя этот ключ, L-система будет размещаться в специальном (прямоугольном) узле, который может быть позиционирован, используя указанный якорь `<anchor>`.

```

\begin{tikzpicture}
[l-system={step=1.75pt,order=5,angle=60}]
\pgfdeclarelindenmayersystem{Sierpinski triangle}{
  \symbol{X}{\pgflsystemdrawforward}
  \symbol{Y}{\pgflsystemdrawforward}
  \rule{X -> Y-X-Y} \rule{Y -> X+Y+X} }
\draw [help lines] grid (3,2);
\draw [red] (0,0) l-system [l-system={Sierpinski triangle,
axiom=+++X, anchor=south west}];
\draw [blue] (3,2) l-system [l-system={Sierpinski triangle,
axiom=X, anchor=north east}];
\end{tikzpicture}

```



Глава 10

Библиотека matrix

Библиотека `matrix` позволяет использовать дополнительные стили и опции для создания матриц.

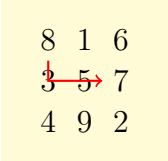
10.1 Матрица узлов

Матрица узлов — TikZ-матрица, в которой каждая клетка содержит узел. При создании такой матрицы быстро надоедает писать `\node{` в начале каждой клетки и `};` в конце. Следующая опция упрощает набор таких матриц.

`/tikz/matrix of nodes` (no value)

Опция добавляет `\node{` в начале и `};` в конце каждой клетки матрицы и устанавливает якорь узлов в `base`. Кроме того, она добавляет опцию `name` к каждому узлу, куда устанавливает имя `<matrix name>-<row number>-<column number>`.

```
\begin{tikzpicture} \matrix (magic)
[matrix of nodes,ampersand replacement=\&]
{8 \& 1 \& 6 \\
 3 \& 5 \& 7 \\
 4 \& 9 \& 2 \\
};
\draw[thick,red,->] (magic-1-1) |- (magic-2-3);
\end{tikzpicture}
```

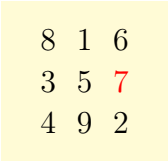


8	1	6
3	5	7
4	9	2

Можно тремя способами добавить опции к конкретным узлам в матрице:

1. Изменить стиль, скажем, строки 2 и столбца 5, чтобы передать этой клетке специальные опции.

```
\begin{tikzpicture} [row 2 column 3/.style=red]
\matrix [matrix of nodes,ampersand replacement=\&]
{8 \& 1 \& 6 \\
 3 \& 5 \& 7 \\
 4 \& 9 \& 2 \\
};
\end{tikzpicture}
```



8	1	6
3	5	7
4	9	2

2. Использовать специальный синтаксис в начале клетки: если клетка начинается с вертикальной черты, то всё между этой и следующей чертой передается команде `node`.

```

\begin{tikzpicture}
\matrix [matrix of nodes,ampersand replacement=\&]
{8 \& 1 \&          6 \\
 3 \& 5 \& |[red]| 7 \\
 4 \& 9 \&          2 \\
};
\end{tikzpicture}

```

8	1	6
3	5	7
4	9	2

Также можно использовать опцию подобную |[red] (seven)|, чтобы задавать имена узлам. Символ-разделитель между столбцами (в примерах это \&) принимает дополнительный параметр, определяющий дополнительный пробел между столбцами.

```

\begin{tikzpicture}
\matrix [matrix of nodes,ampersand replacement=\&]
{8 \&[1cm] 1 \&[3mm] |[red]| 6 \\
 3 \&      5 \& |[red]|      7 \\
 4 \&      9 \&          2 \\
};
\end{tikzpicture}

```

8	1	6
3	5	7
4	9	2

3. Если клетка начинается с команды \path или какой-то команды, которая расширяется до команды \path, такой как \draw, \node, \fill и другие, то первоначальный код для этой клетки подавляется, а код после \node{...}; выполняется. Это означает, что для данной конкретной клетки можно определить другое содержимое.

```

\begin{tikzpicture}
\matrix [matrix of nodes,ampersand replacement=\&]
{8 \& 1 \& 6 \\
 3 \& 5 \& \node[red]{7};\draw(0,0) circle(10pt); \\
 4 \& 9 \& 2 \\
};
\end{tikzpicture}

```

8	1	6
3	5	7
4	9	2

/tikz/matrix of math nodes

(no value)

Опция делает почти то же, что и предыдущая, только добавляет в начале и в конце каждого узла \$, включая математический режим во всех узлах матрицы.

```

\begin{tikzpicture}\matrix[matrix of math nodes,
ampersand replacement=\&]
{a_8 \& a_1 \&          a_6 \\
 a_3 \& a_5 \& |[red]| a_7 \\
 a_4 \& a_9 \&          a_2 \\
};
\end{tikzpicture}

```

a_8	a_1	a_6
a_3	a_5	a_7
a_4	a_9	a_2

/tikz/nodes in empty cells=<true or false>

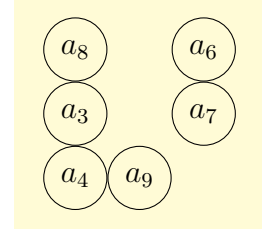
(default true)

Когда значение опции равно true, узел с пустым содержимым помещается в пустую клетку. Как правило, пустые клетки остаются просто пустыми. Различия объясняют следующие два примера. Такой стиль можно использовать вместе и опциями matrix of nodes и matrix of math nodes.

```

\begin{tikzpicture}
\matrix [matrix of math nodes,
nodes={circle,draw},ampersand replacement=\&]
{a_8 \&      \& a_6 \\
a_3 \&      \& a_7 \\
a_4 \& a_9 \&      \& \\
};
\end{tikzpicture}

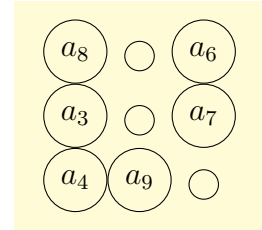
```



```

\begin{tikzpicture}
\matrix [matrix of math nodes,nodes={circle,draw},
nodes in empty cells,ampersand replacement=\&]
{a_8 \&      \& a_6 \\
a_3 \&      \& a_7 \\
a_4 \& a_9 \&      \& \\
};
\end{tikzpicture}

```



10.2 Особенности команды `\&`

Обратим особое внимание на команду `\&` в матрице узлов. Причина в том, что этот символ очень перегружен в \TeX : с одной стороны, он обозначает конец строки в обычном тексте; с другой стороны он обозначает конец строки в матрице. Теперь, если матрица содержит узлы, у которых, в свою очередь, присутствует несколько строк, становится неясно, какое значение команды `\&` должно использоваться. Такая проблема возникает только тогда, когда используется опция узлов `text width`. Предположим, что написан код вида

```

\matrix [text width=5cm,matrix of nodes,ampersand replacement=\&]
{first row \& upper line \& lower line \& second row \& hmm \& };

```

Он приведет \TeX в недоумение: сколько строк у матрицы — две строки, в которых правая верхняя клетка матрицы содержит текст из двух строк, или три строки, со второй строкой матрицы, имеющей только одну клетку? Так как \TeX не ясновидец, используются следующие правила:

1. В матрице, команда `\&`, по умолчанию, означает конец строки в матрице, а не конец строки в клетке матрицы.

2. Однако, есть исключение из первого правила: если клетка начинается с \TeX -группы (с символа `{}`), то внутри этой первой группы команда `\&` сохраняет значение конца строки символов. Заметим, что это специальное правило работает только для первой группы в клетке и эта группа должна быть в ее вначале.

Результат применения этих правил следующий: обычно, `\&` указывает на конец строки матрицы; если нужно использовать `\&` как указатель конца строки символов в клетке матрицы, следует поместить целую клетку в фигурные скобки. Следующий пример поясняет эти различия:

```
\begin{tikzpicture}
\matrix[matrix of nodes,
        ampersand replacement=\&,
        nodes={text width=19mm,draw}]
{row 1 \& upper line \& lower line \& \\
row 3 \& hmm \& \\ };
\end{tikzpicture}
```

row 1	upper line
lower line	
row 3	hmm

```
\begin{tikzpicture}
\matrix[matrix of nodes,
        ampersand replacement=\&,
        nodes={text width=19mm,draw}]
{row 1 \& {upper line \& lower line} \\
row 2 \& hmm \& \\ };
\end{tikzpicture}
```

row 1	upper line lower line
row 2	hmm

Заметим, что эта система не защищена от ошибок. Например, если написать нечто подобное `a & b {c \& d} \&` в матрице узлов, то возникнет ошибка, поскольку вторая клетка не начинается с фигурной скобки, таким образом, `\&` сохраняет свое обычное значение, и вторая клетка содержит текст `b {c`, что приводит к разбалансировке фигурных скобок.

10.3 Разделители

Разделители — круглые или фигурные скобки слева и справа от формулы или матрицы. Матричная библиотека предлагает опции для добавления таких разделителей к матрице. Однако, разделители можно добавить к любому узлу, который имеет стандартный набор якорей: `north`, `south`, `north west` и так далее. В частности, можно добавить разделители к любому прямоугольному полю.

`/tikz/left delimiter=<delimiter>` (no default)

Опцию можно использовать в любом узле, который имеет стандартные якоря. Параметр `<delimiter>` может быть любым разделителем, который приемлем в Т_ЕX'е для команды `\left`.

`/tikz/right delimiter=<delimiter>` (no default)

Опция аналогична предыдущей, только для правого разделителя.

```
\begin{tikzpicture}\matrix[matrix of math nodes,
        ampersand replacement=\&,
        left delimiter=(,right delimiter=\}]
{a_8 \& a_1 \& a_6 \& \\
a_3 \& a_5 \& a_7 \& \\
a_4 \& a_9 \& a_2 \& \\ };
\end{tikzpicture}
```

$\left(\begin{array}{ccc} a_8 & a_1 & a_6 \\ a_3 & a_5 & a_7 \\ a_4 & a_9 & a_2 \end{array} \right)$

```
\begin{tikzpicture}
\node [fill=red!20,ampersand replacement=\&,
      left delimiter=(,right delimiter=)]
{\displaystyle\int_0^1 x\,dx};
\end{tikzpicture}
```

/tikz/every delimiter (style, initially empty)

Стиль выполняется для каждого разделителя. Его можно использовать, чтобы сдвинуть или раскрасить разделители, или сделать что-то еще.

/tikz/every left delimiter (style, initially empty)

Стиль дополнительно выполняется для каждого левого разделителя.

/tikz/every right delimiter (style, initially empty)

Стиль дополнительно выполняется для каждого правого разделителя.

```
\begin{tikzpicture}
[every left delimiter/.style={red,xshift=1ex},
 every right delimiter/.style={xshift=-1ex}]
\matrix[matrix of math nodes,
        ampersand replacement=\&,
        left delimiter=(,right delimiter=)]
{a_8 \& a_1 \& a_6 \\
 a_3 \& a_5 \& a_7 \\
 a_4 \& a_9 \& a_2 };
\end{tikzpicture}
```

/tikz/above delimiter=<delimiter> (no default)

Опция позволяет добавить разделитель выше узлов матрицы. Разделитель реализуется вращением левого разделителя.

/tikz/below delimiter=<delimiter> (no default)

Опция позволяет добавить разделитель ниже узлов матрицы. Разделитель реализуется вращением правого разделителя.

```
\begin{tikzpicture}\matrix [matrix of math nodes,
                            ampersand replacement=\&,
                            left delimiter=|,right delimiter=\rmoustache,
                            above delimiter=(,below delimiter=)]
{a_8 \& a_1 \& a_6 \\
 a_3 \& a_5 \& a_7 \\
 a_4 \& a_9 \& a_2 };
\end{tikzpicture}
```

/tikz/every above delimiter (style, initially empty)

Стиль дополнительно выполняется для каждого верхнего разделителя.

/tikz/every below delimiter (style, initially empty)

Стиль дополнительно выполняется для каждого нижнего разделителя.

Глава 11

Библиотека mindmap

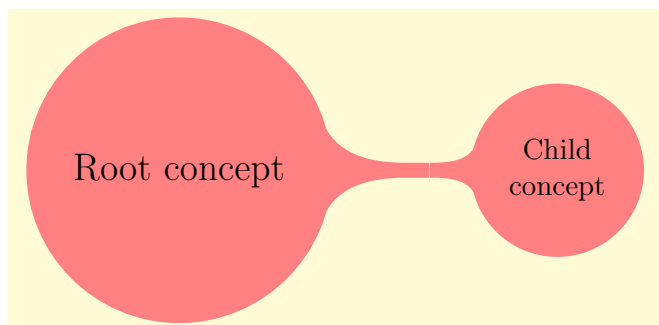
Библиотека `mindmap` определяет дополнительные стили и опции для создания специальных (понятийных) диаграмм — графического представления связанных между собой понятий, вместе с родственными понятиями и аннотациями. По сути, в `TikZ&PGF` понятийные диаграммы — деревья, возможно с несколькими дополнительными дугами, которые обычно рисуются в `TikZ` специальным образом: корневое понятие размещается в середине страницы и рисуется как большой круг, а связанные с ним понятия изображаются кругами меньшего размера и связываются с корневым понятием разумно представленными связями. Некоторые стили и макросы библиотеки работают лучше именно внутри дерева, однако, можно, а иногда и нужно, обрабатывать части понятийной диаграммы как граф.

11.1 Стил ь для понятийных диаграмм

`/tikz/mindmap` (style, no value)

Стил ь используется со всеми рисунками или, по крайней мере, со всеми окружениями, содержащими понятийные диаграммы, устанавливая многочисленные параметры настройки.

```
\tikz[mindmap,
      concept color=red!50]
\node[concept] {Root concept}
  child[grow=right]
    {node[concept]
      {Child concept}};
```



Размеры диаграмм предопределяются таким образом, чтобы размер среды для понятийной диаграммы более или менее учитывал размеры страницы формата A4.

`/tikz/every mindmap` (style, no value)

Используется стилем `mindmap`. Изменяя этот стил ь, можно добавить специальные параметры настройки для создаваемых понятийных диаграмм.

Замечание. Стил ь `mindmap` переопределяет параметры `font` и `sibling angle` в зависимости от текущего уровня расположения понятия (то есть, от понятия на 1-ом уровне, понятия на 2-ом уровне и т.д.). Всегда есть возможность переопределить эти параметры, используя *после* стилия `mindmap` выражения вида

```
\tikzset{level 1 concept/.append style={font=\small}}
```

или

```
\tikzset{level 2 concept/.append style={sibling distance=90}}
```

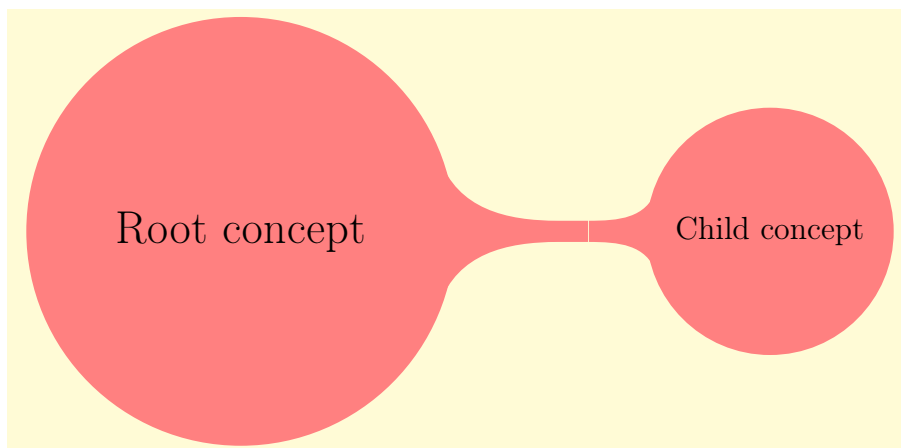
`/tikz/small mindmap` (style, no value)

Использует стиль `mindmap`, но дополнительно изменяет размеры по умолчанию для понятий, шрифты и расстояния так, чтобы понятийная диаграмма среднего размера поместилась на странице формата А5 (А5 — половина страницы размера А4). Понятийные диаграммы в стиле `small mindmap` заполняют стандартную рамку пакета `beamer`.

`/tikz/large mindmap` (style, no value)

Стиль использует стиль `mindmap`, но дополнительно изменяет размеры по умолчанию для понятий, шрифты и расстояния так, чтобы понятийная диаграмма среднего размера поместилась на странице формата А3 (А3 — удвоенная страницы размера А4).

```
\tikz[large mindmap,concept color=red!50]
  \node [concept] {Root concept}
    child[grow=right] {node[concept] {Child concept}};
```



`/tikz/huge mindmap` (style, no value)

Стиль еще больше увеличивает все размеры, и его лучше всего использовать на листе формата не меньше, чем А2.

11.2 Понятийные узлы

Основные объекты понятийных диаграмм (понятия) реализуются как узлы стиля `concept` (понятийные узлы) и должны быть кругами, чтобы некоторые макросы связи (между понятиями) корректно работали.

11.2.1 Изолированные понятия

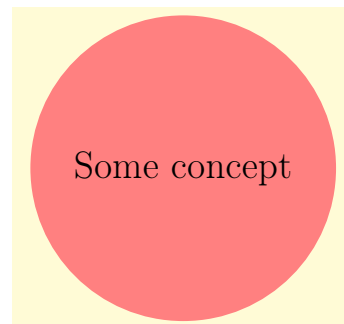
Следующие стили влияют на внешний вид изолированных понятий.

`/tikz/concept` (style, no value)

Стиль должен использоваться явно со всеми узлами, которые отражают понятия, хотя некоторые стили (подобные `extra concept`) устанавливают этот стиль автоматически. В основном, этот стиль создает понятийный узел в виде круга и устанавливает

однородный цвет, который имеет имя `concept color` (см. ниже). Дополнительно, вызывается стиль `every concept`.

```
\tikz[mindmap, concept color=red!50]
  node [concept] {Some concept};
```



`/tikz/every concept`

(style, no value)

Чтобы изменить внешний вид понятийных узлов, нужно изменить этот стиль. Но, используя этот стиль, нельзя изменять цвет или статус (`draw` или `fill`) понятийного узла. Данный стиль используется только для того, чтобы изменить цвет текста и шрифт.

`/tikz/concept color=<color>`

(no default)

Определяет цвет, который используется при рисовании и заполнении понятийного узла. Различие между этой опцией и установками цвета стилем `every concept` состоит в том, что эта опция позволяет TikZ следить за цветом, используемым для понятий, что очень важно, когда меняется цвет между двумя связываемыми дугой понятиями. Тогда TikZ может автоматически провести растушевывание, обеспечивая плавный переход между старым и новым цветом (см. следующий раздел).

`/tikz/extra concept`

(style, no value)

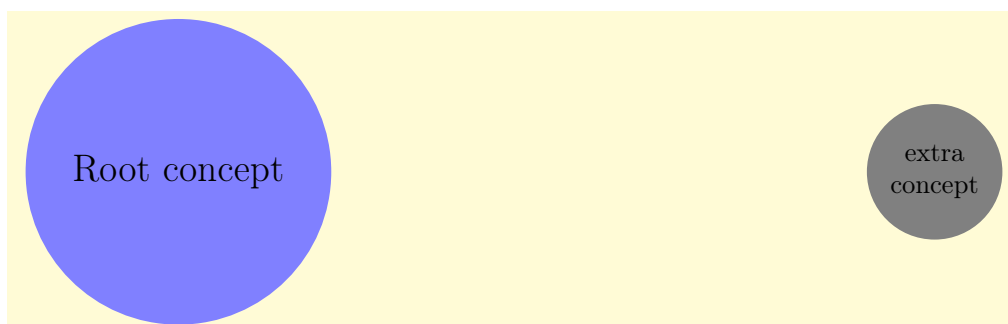
Предназначен для *дополнительных понятий*, которые не являются частью понятийного дерева, но располагаются рядом с ним. Как правило, они будут иметь приглушенный цвет и несколько меньше по размеру. Этот стиль нужен, чтобы единообразно использовать такие понятия и указать в коде, что они являются внешними.

`/tikz/every extra concept`

(style, no value)

Изменяя этот стиль можно изменить вид всех узлов внешних понятий.

```
\begin{tikzpicture} [mindmap,concept color=blue!50]
  \node [concept] {Root concept};
  \node [extra concept] at (10,0) {extra concept};
\end{tikzpicture}
```



11.2.2 Понятия в деревьях

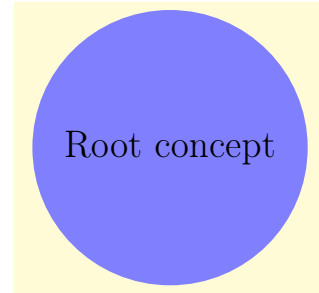
Как было сказано, TikZ предполагает, что понятийная диаграмма — конструкция, использующая опцию `child`, присущую деревьям. Существуют опции, которые влияют на внешний вид понятий, отображаемых на разных уровнях дерева.

`/tikz/root concept`

(style, no value)

Стиль используется для корня понятийного дерева и, изменяя его, можно изменить внешний вид корня.

```
\tikz[root concept/.append style={
  concept color=blue!50,
  minimum size=3.5cm},mindmap]
\node [concept] {Root concept};
```



Отметим, что такие стили, как `large mindmap`, переопределяют эти стили, так что при их использовании что-либо добавлять к этому стилю можно только внутри самого рисунка.

`/tikz/level 1 concept`

(style, no value)

Стиль `mindmap` добавляет этот стиль для понятий первого уровня. Это означает, что первый дочерний узел дерева будет использовать именно этот стиль. Аналогично определяются и используются следующие стили

`/tikz/level 2 concept`

(style, no value)

`/tikz/level 3 concept`

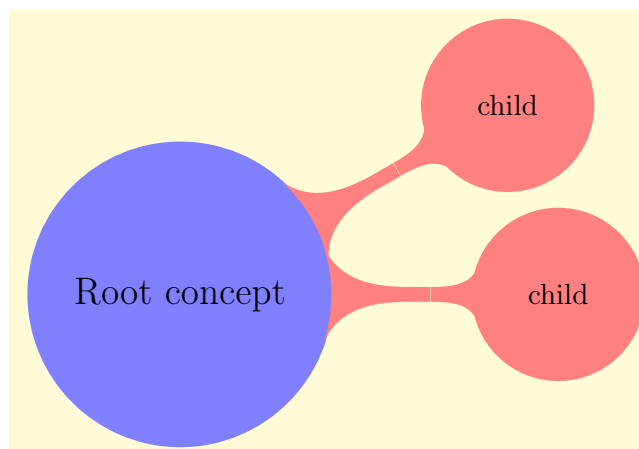
(style, no value)

`/tikz/level 4 concept`

(style, no value)

Нет стиля пятого и стиля более высокого уровня, их нужно создавать самостоятельно и использовать аналогичным образом.

```
\tikz
[ root concept/.append style={concept color=blue!50},
  level 1 concept/.append style={concept color=red!50},mindmap]
\node [concept] {Root concept}
  child[grow=30] {node[concept] {child}}
  child[grow=0 ] {node[concept] {child}};
```



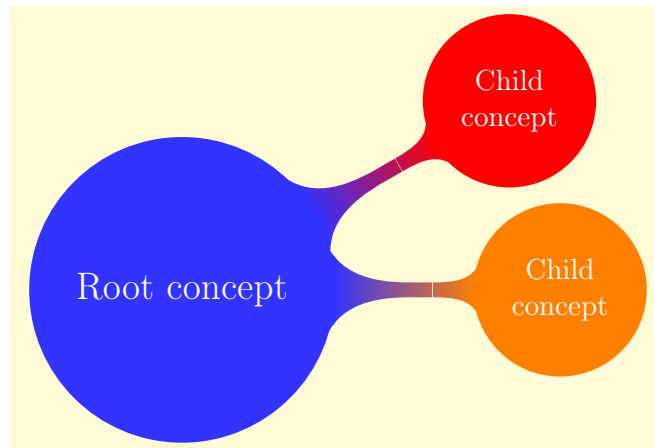
`/tikz/concept color=<color>`

(no default)

Опция используется, чтобы изменить цвет понятийного узла. Теперь можно посмотреть на её результат, когда опция используется в дочерних понятийных узлах. Обычно,

она позволяет изменить цвет дочерних узлов. Когда эта опция задается как опция операции `child` (а нет операции `node`, а также нет опции для всех дочерних узлов, определенной через стиль первого уровня), TikZ плавно изменит цвет от цвета родительского понятийного узла до цвета дочернего.

```
\tikz[mindmap,concept color=blue!50,text=white]
  \node [concept] {Root concept}
    child[concept color=red,grow=30] {node[concept] {Child concept}}
    child[concept color=orange,grow=0] {node[concept] {Child concept}};
```



Чтобы дочерние понятийные узлы одного уровня имели разные цвета, необходимо немного волшебства!

11.3 Связи между понятиями

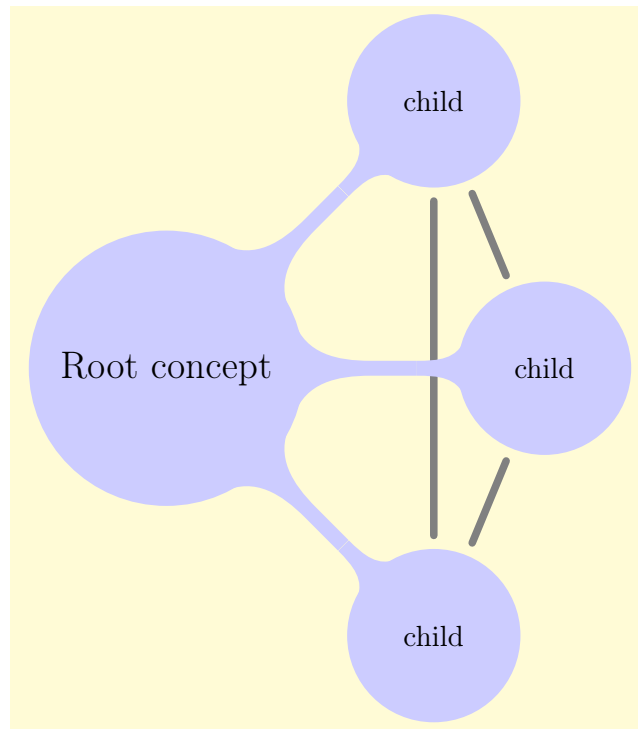
11.3.1 Простые связи

Самый простой способ связать два понятия — соединить их линией. Чтобы нарисовать такие линии в соответствующем виде, следует использовать стиль

`/tikz/concept connection` (style, no value)

Проблема возникает тогда, когда нужно связать понятия после того, как основная понятийная диаграмма была нарисована. В этом случае часто желательно, чтобы линии связи лежали вне (или за) основной понятийной диаграммы. Но такие линии можно провести только после того, как будут определены координаты понятий. В этом случае нужно поместить линии связи на фоновый уровень:

```
\begin{tikzpicture}
  [root concept/.append style={concept color=blue!20,minimum size=2cm},
  level 1 concept/.append style={sibling angle=45},mindmap]
  \node [concept] {Root concept}
    [clockwise from=45] child { node[concept] (c1) {child}}
    child { node[concept] (c2) {child}}
    child { node[concept] (c3) {child}};
\begin{pgfonlayer}{background}
  \draw [concept connection] (c1) edge (c2) edge (c3) (c2) edge (c3);
\end{pgfonlayer}
\end{tikzpicture}
```



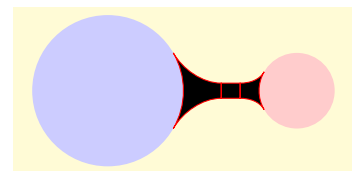
11.3.2 Декорирование линии связи

Вместо простой линии связи между двумя понятиями, можно добавить небольшую область между двумя узлами, которая имеет немного измененные концы. Такие области используются по умолчанию в понятийной диаграмме, как дуги из родительского узла. Для рисования таких областей используется специальная декорация.

Декорация `circle connection bar` связывает два круга. Начало декорации должно лежать на граница первого круг, конец — на границе второго. Два ключа декорации `start radius` и `end radius` должны определять радиусы этих кругов. Кроме того, на внешний вид декорации влияют опции декорации `amplitude` и `angle`.

Декорация превращает прямую линию в путь, который начинается на границе первого круга под определенным углом к линии, связывающей центры кругов. Путь затем меняется, превращаясь в прямоугольник, толщина которого определяется ключом `amplitude`. Наконец, путь заканчивается под тем же углом на границе второго круга. Приведем пример, который должен сделать все сказанное более понятным.

```
\begin{tikzpicture}
[decoration={start radius=1cm,
end radius=.5cm,amplitude=2mm,angle=30}]
\fill [blue!20] (0,0) circle (1cm);
\fill [red!20] (2.5,0) circle (.5cm);
\filldraw [draw=red,fill=black,decorate,
decoration=circle connection bar]
(1,0) -- (2,0);
\end{tikzpicture}
```

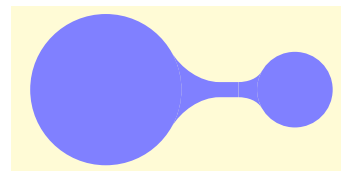


Как можно заметить, декорированный путь состоит из трех частей, что не очень хорошо смотрится. Однако можно заполнить декорированный путь тем же цветом, что и круги понятий.

```

\begin{tikzpicture}
  [blue!50,decoration={start radius=1cm,
    end radius=.5cm,amplitude=2mm,angle=30}]
  \fill (0,0) circle (1cm);
  \fill (2.5,0) circle (.5cm);
  \fill[decorate,decoration=circle connection bar]
    (1,0) -- (2,0);
\end{tikzpicture}

```

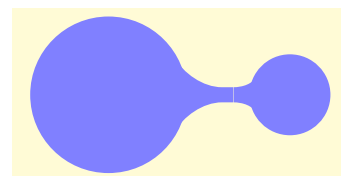


В примере, аналогичном приведенному выше, иногда можно заметить небольшую белую линию между кругами и декорированным путем, которая возникает из-за погрешностей округления. К сожалению, для больших расстояний, могут накапливаться весьма большие погрешности, тем более, что TikZ и TeX не очень хорошо умеют вычислять квадратные корни. Поэтому стоит сделать круги немного больше, чтобы компенсировать погрешности. Используя узлы в форме круга, можно добавить опцию `draw` с опцией `line width`, имеющей значение в 1pt или 2pt (для очень больших расстояния, возможно, потребуется ширина линии до 4pt).

```

\begin{tikzpicture}
  [blue!50,decoration={start radius=1cm,
    end radius=.5cm,amplitude=2mm,angle=30}]
  \fill (0,0) circle (1cm+1pt);
  \fill (2.4,0) circle (.5cm+1pt);
  \fill[decorate,decoration=circle connection bar]
    (1,0) -- (1.9,0);
\end{tikzpicture}

```



Декорация `circle connection bar` немного сложна в использовании. Особенно нелегко определение радиусов (ключи `amplitude` и `angle` можно установить раз и навсегда). Поэтому библиотека `mindmap` определяет специальный путь, позволяющий выполнить все необходимые вычисления автоматически.

`/tikz/circle connection bar` (style, no value)

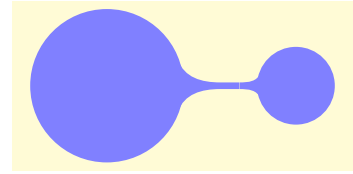
Стиль определяет достаточно сложный путь. В отличие от обычного пути, этот путь требует, чтобы начальный и конечный узлы имели форму круга (если дело обстоит не так, этот путь будет порождать ошибку). Сначала вычисляются радиусы этих кругов (измеряя расстояние от якоря `center` до некоторого якоря на границе), и, соответственно, устанавливается ключ `start circle`. Затем, опция `fill` устанавливает `concept color`, в то же время устанавливается `draw=none`. Устанавливается декорация `circle connection bar` и, наконец, включается следующий стиль, позволяющий изменить внешний вид пути связи между кругами:

`/tikz/every circle connection bar` (style, no value)

```

\begin{tikzpicture}
  [concept color=blue!50,blue!50,outer sep=0pt]
  \node (n1) at (0,0)
    [circle,minimum size=2cm,fill,draw,thick] {};
  \node (n2) at (2.5,0)
    [circle,minimum size=1cm,fill,draw,thick] {};
  \path (n1) to[circle connection bar] (n2);
\end{tikzpicture}

```



Обратите внимание на несколько странную опцию `outer sep=0pt`. Это нужно для того, чтобы декорированный путь лежал на границе заполненного круга, а не на границе нарисованной окружности (которая немного больше, и это немного тот немного больший размер, который следует использовать, чтобы скрыть ошибки округления).

Заметим также, что для одной операции `\path` не стоит указывать несколько операций `to` с опцией `circle connection bar`. Создать несколько параллельных связей позволяет операция `edge`, создающая новое окружение для каждой связи.

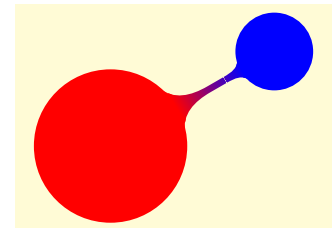
В понятийных диаграммах иногда желательно изменять цвет от одного понятия к другому. Тогда, области связи должны, в идеале, содержать гладкий переход между двумя заданными цветами. Задача непростая для решения «вручную», поэтому для ее решения существует специальный стиль, подобный стилю `circle connection bar`, который вместо заполнения пути одним цветом использует растушевывание.

`/tikz/circle connection bar switch color=from(<first color>)to(<second color>)`

```

\begin{tikzpicture}[outer sep=0pt]
  \node (n1) at (0,0)
    [circle,minimum size=2cm,fill,draw,thick,red] {};
  \node (n2) at (30:2.5)
    [circle,minimum size=1cm,fill,draw,thick,blue] {};
  \path (n1) to[circle connection bar switch color=
    from (red) to (blue)] (n2);
\end{tikzpicture}

```



11.3.3 Дуги дерева понятий

Понятия в понятийной диаграмме связываются автоматически, когда понятийная диаграмма конструируется в виде дерева. Причина этого в том, что опция `mindmap` в качестве дуги от коренного пути устанавливает путь `circle connection bar`. Кроме того, опция `mindmap` автоматически устанавливает «правильные» значения для многих параметров диаграммы. Не останавливаясь на этом подробно, рассмотрим поясняющий пример древовидной конструкции:

```

\begin{tikzpicture}
\path[mindmap,concept color=black,text=white]
  node[concept] {Computer Science} [clockwise from=0]
  % заметим, что 'sibling angle' может быть определен только на 1-ом
  % уровне, то есть в коде вида <<level 1 concept/.append style={}>>
  child[concept color=green!50!black]

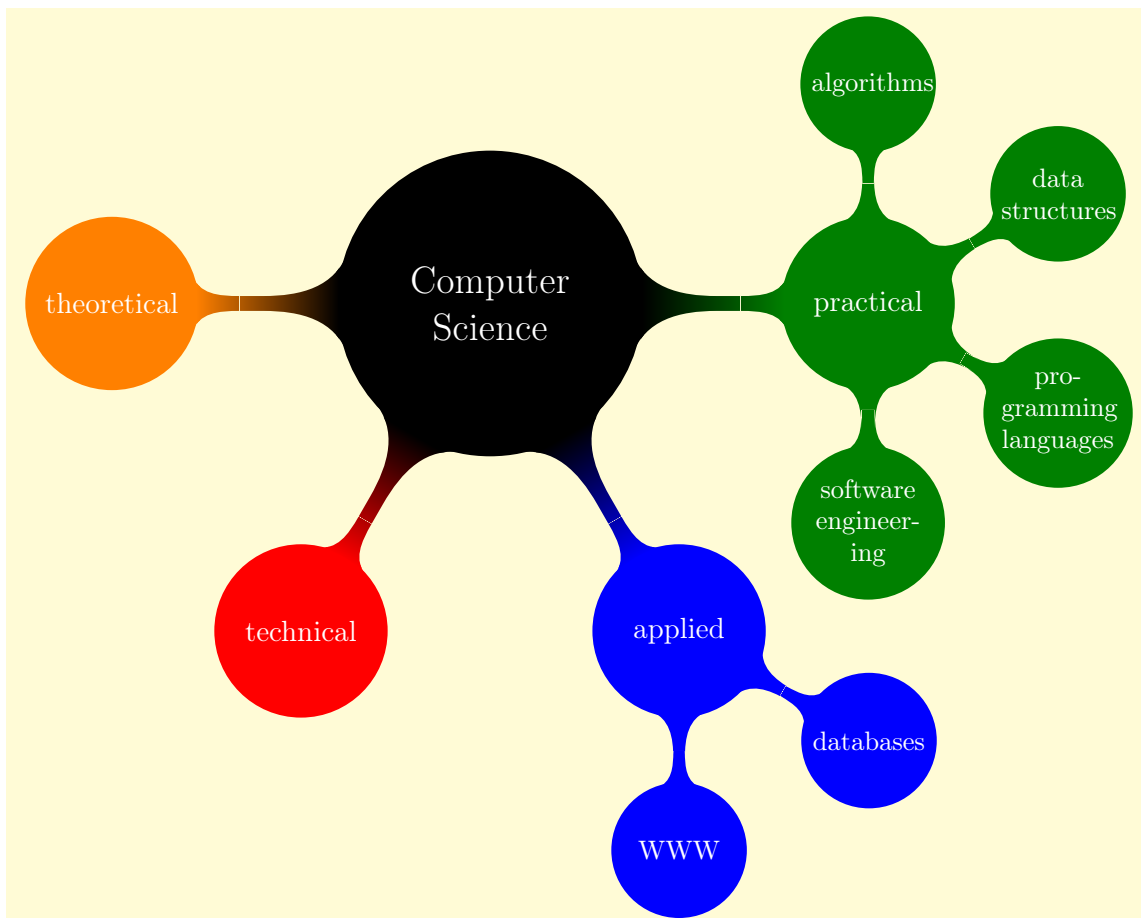
```



```

{ node[concept] {practical}{clockwise from=90}
  child { node[concept] {algorithms} }
  child { node[concept] {data structures} }
  child { node[concept] {pro\ -gramming languages} }
  child { node[concept] {software engineer\ -ing} } }
child[concept color=blue]
{ node[concept] {applied}{clockwise from=-30}
  child { node[concept] {databases} }
  child { node[concept] {WWW} } }
child [concept color=red] { node[concept] {technical} }
child [concept color=orange] { node[concept] {theoretical} };
\end{tikzpicture}

```



11.4 Добавление аннотаций

Аннотация — некоторый текст вне понятийной диаграммы, который, в отличие от дополнительного понятия, просто объясняет нечто, расположенное в понятийной диаграмме. Следующий стиль предназначен главным образом для того, чтобы помочь читателю кода увидеть, что этот узел является узлом аннотации.

`/tikz/annotation` (style, no value)

Стиль выделяет узел аннотации. Он использует стиль `every annotation`, который позволяет изменить вид узла аннотации.

`/tikz/every annotation` (style, no value)

```
\begin{tikzpicture} [mindmap,concept color=blue!40,  
                    every annotation/.style={fill=red!20}]  
  \node [concept] (root) {Root concept};  
  \node [annotation,right] at (root.east)  
    {Корневое понятие~---  
     это самое важное понятие.};  
\end{tikzpicture}
```



Глава 12

Библиотека `folding`

Библиотека `folding` определяет дополнительные команды, стили и опции для создания специальных диаграмм, как бы расположенных на согнутой специальным образом (фальцованной) бумаге. В настоящее время, есть только одна такая команда, которая позволяет создать диаграмму, содержащую, например, календари.

```
/tikz/\tikzfoldingdodecahedron[<options>];
```

Рисует на листе бумаги «складывающуюся» диаграмму, состоящую из 12-ти граней (в виде развертки додекаэдра). Синтаксис обязывает определить только `<options>` и ничего более не должно быть до заключительной точкой с запятой.

В `<options>` могут использоваться следующие ключи:

```
/tikz/folding line length=<dimension> (no default)
```

Устанавливает длину базисной линии для фальцовки. Для двенадцатигранника это длина всех сторон его пятиугольников.

```
/tikz/face 1=<code> (no default)
```

Параметр `<code>` выполняется для первой грани двенадцатигранника. Когда он выполняется, система координат сдвигается и вращается так, что располагается в середине первой грани.

Аналогичный смысл имеют ключи

```
/tikz/face 2=<code> (no default)
```

```
/tikz/face 3=<code> (no default)
```

.....

```
/tikz/face 11=<code> (no default)
```

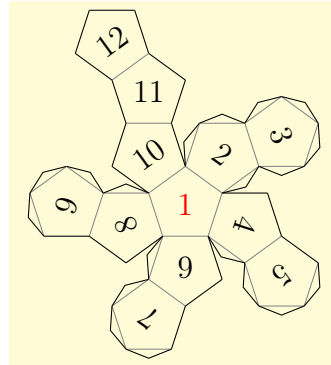
```
/tikz/face 12=<code> (no default)
```

Приведем поясняющий пример:

```

\begin{tikzpicture}[transform shape]
\tikzfoldingdodecahedron
[folding line length=6mm,
face 1={ \node[red] {1};},
face 2={ \node      {2};},
face 3={ \node      {3};},
face 4={ \node      {4};},
face 5={ \node      {5};},
face 6={ \node      {6};},
face 7={ \node      {7};},
face 8={ \node      {8};},
face 9={ \node      {9};},
face 10={\node      {10};},
face 11={\node      {11};},
face 12={\node      {12};}];
\end{tikzpicture}

```



На вид полей для склейки (линий разреза) и линий сгиба можно повлиять, используя следующие стили:

`/tikz/every cut` (style, initially empty)

Выполняется для каждой линии разреза.

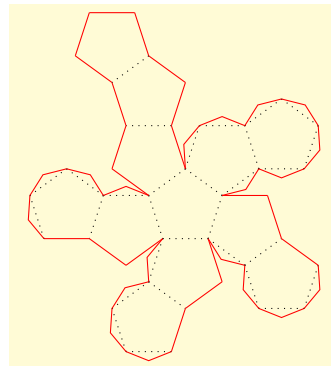
`/tikz/every fold` (style, initially help lines)

Выполняется для каждой линии сгиба.

```

\begin{tikzpicture}
[every cut/.style=red,
every fold/.style=dotted]
\tikzfoldingdodecahedron
[folding line length=6mm];
\end{tikzpicture}

```



Приведем большой пример, строящий диаграмму для календаря на текущий год.

```

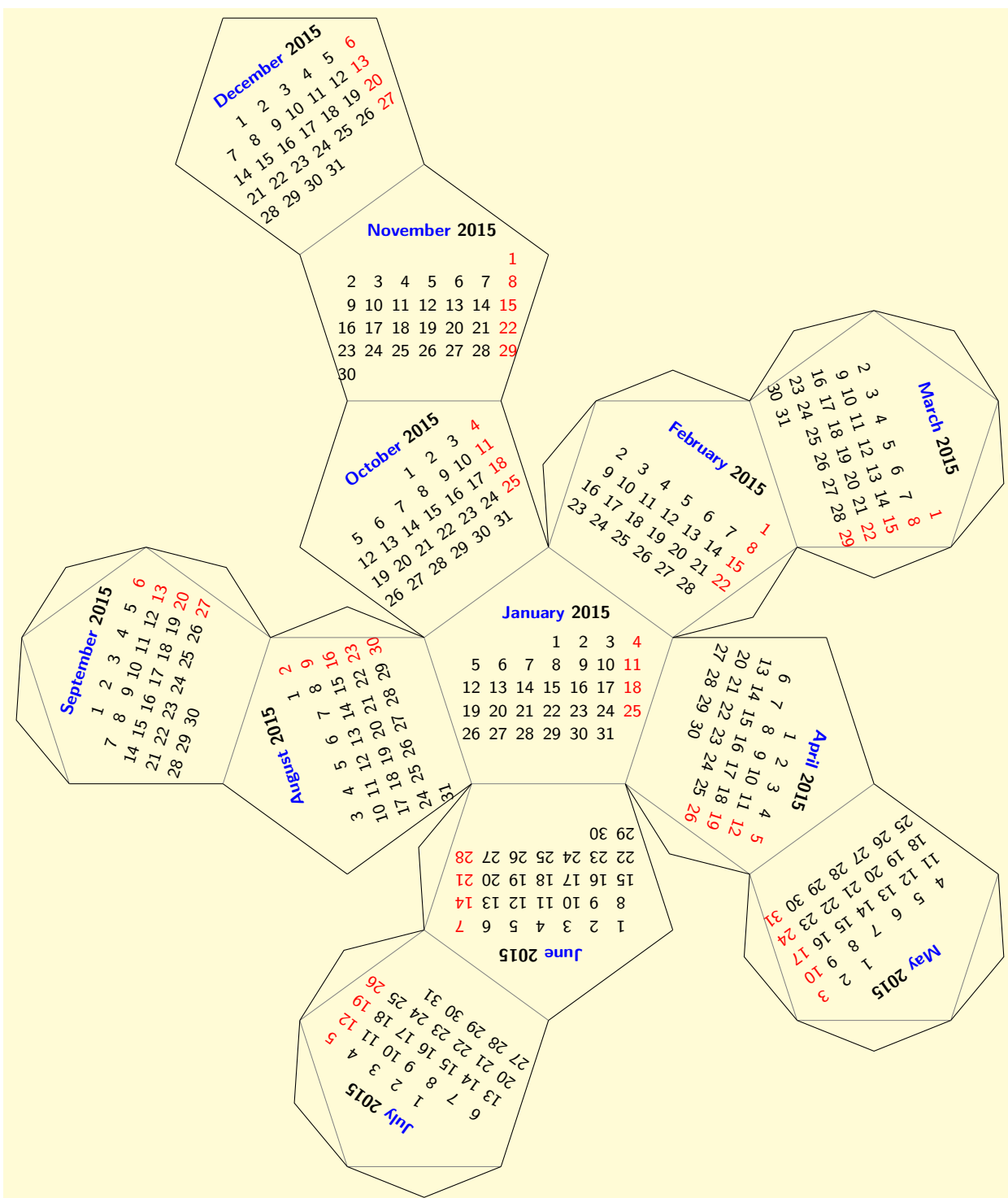
{\sffamily\scriptsize
\begin{tikzpicture}[transform shape,every calendar/.style={
at={(-8ex,4ex)}, week list, month label above centered,
month text=\bfseries\textcolor{blue}{\%mt} \%y0,
if={(Sunday) [red]}]}]
\tikzfoldingdodecahedron[folding line length=2.5cm,
face 1={ \calendar [dates=\the\year-01-01 to \the\year-01-last];},
face 2={ \calendar [dates=\the\year-02-01 to \the\year-02-last];},
face 3={ \calendar [dates=\the\year-03-01 to \the\year-03-last];},
face 4={ \calendar [dates=\the\year-04-01 to \the\year-04-last];},
face 5={ \calendar [dates=\the\year-05-01 to \the\year-05-last];},
face 6={ \calendar [dates=\the\year-06-01 to \the\year-06-last];},
face 7={ \calendar [dates=\the\year-07-01 to \the\year-07-last];},

```

```

face 8={ \calendar [dates=\the\year-08-01 to \the\year-08-last];},
face 9={ \calendar [dates=\the\year-09-01 to \the\year-09-last];},
face 10={\calendar [dates=\the\year-10-01 to \the\year-10-last];},
face 11={\calendar [dates=\the\year-11-01 to \the\year-11-last];},
face 12={\calendar [dates=\the\year-12-01 to \the\year-12-last];} ];
\end{tikzpicture}}

```



Глава 13

Библиотека patterns

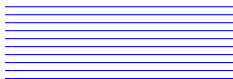
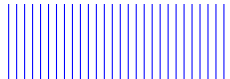
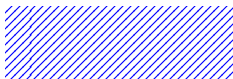
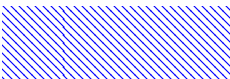
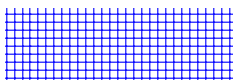
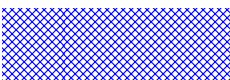

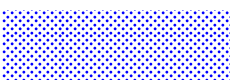




Библиотека `patterns` определяет несколько шаблонов, которые можно использовать для заполнения областей.

13.1 Шаблоны, содержащие только форму

Если для каждого указанного ниже в таблице шаблона выполнить код вида

```
\tikz\pattern [pattern=<pattern name>,pattern color=<color>]  
(0,0) rectangle (3,1);
```

где вместо `<pattern name>` стоит имя шаблона, указанное в первом столбце таблицы, а вместо `<color>`, например, `blue`, то получим результат, представленный во втором столбце таблицы.

Имя шаблона	Вид шаблона	Имя шаблона	Вид шаблона
horizontal lines		vertical lines	
north east lines		north west lines	
grid		crosshatch	
dots		crosshatch dots	
fivepointed stars		sixpointed stars	
bricks		checkerboard	

13.2 Окрашенные шаблоны

Если для каждого указанного ниже в таблице шаблона выполнить код вида

```
\tikz\pattern [pattern=<pattern name>] (0,0) rectangle (3,1);
```

где вместо `<pattern name>` стоит имя шаблона, указанное в первом столбце таблицы, то получим результат, представленный во втором столбце таблицы.

Имя шаблона	Вид шаблона
checkerboard light gray	
horizontal lines light gray	
horizontal lines gray	
horizontal lines dark gray	
horizontal lines light blue	
horizontal lines dark blue	
crosshatch dots gray	
crosshatch dots light steel blue	

Глава 14

Библиотека `petri`

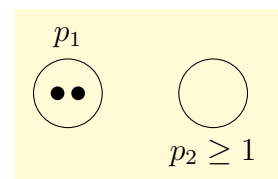
Библиотека `petri` определяет формы и стили, облегчающие создание сетей Петри.

14.1 Места

`/tikz/place` (style, no value)

Стиль указывает, что данный узел является местом в сети Петри. Обычно, текст узлов должен быть пустым, так как места не содержат текста. Нужно использовать опцию `label`, чтобы добавить текст вне узла, подобный его имени или его емкости, и опцию `tokens` (см. далее раздел 14.3), чтобы добавить лексемы в место.

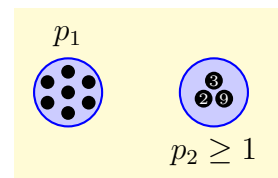
```
\begin{tikzpicture}
\node[place,label=above:$p_1$,tokens=2] (p1) {};
\node[place,label=below:$p_2\ge 1$,right=of p1]
      (p2) {};
\end{tikzpicture}
```



`/tikz/every place` (style, no value)

Стиль вызывается стилем `place`. Чтобы изменить внешний вид мест, нужно изменить ЭТОТ СТИЛЬ.

```
\begin{tikzpicture}[every place/.style={
  draw=blue,fill=blue!20,thick,minimum size=9mm}]
\node[place,tokens=7,label=above:$p_1$] (p1) {};
\node[place,structured tokens={3,2,9},
      label=below:$p_2\ge 1$,right=of p1] (p2) {};
\end{tikzpicture}
```



14.2 Переходы

Переходы — это также узлы и должны рисоваться, используя следующий стиль:

`/tikz/transition` (style, no value)

Указывает, что узел — переход. Как и для мест, текст перехода должен быть пустым, а чтобы добавить метки, следует использовать опцию `label`.

`/tikz/every transition` (style, no value)

Стиль вызывается стилем `transition`.

`/tikz/pre` (style, no value)

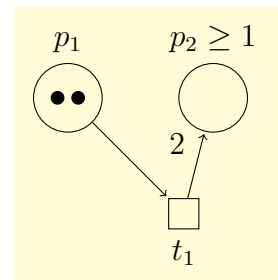
Стиль может использоваться с путями, ведущими от перехода к месту, чтобы указать на то, что место предшествует переходу. По умолчанию, этот стиль определяется, как `<-`, `shorten <=1pt`, но его всегда можно переопределить.

`/tikz/post` (style, no value)

Стиль также используется с путями, ведущими от перехода к месту, но на сей раз место находится после перехода. Стиль всегда можно переопределить.

Чтобы связать переход с местом, нужно использовать команду `edge`, как в следующем примере:

```
\begin{tikzpicture}
\node[place,tokens=2,label=above:$p_1$] (p1) {};
\node[place,label=above:$p_2\ge 1$,right=of p1]
      (p2) {};
\node[transition,below right=of p1,
      label=below:$t_1$] {}
      edge[pre] (p1)
      edge[post] node[auto] {2} (p2);
\end{tikzpicture}
```



`/tikz/pre and post` (style, no value)

Стиль должен использоваться, чтобы указать, что место как предшествует переходу, так и находится после перехода.

14.3 Лексемы

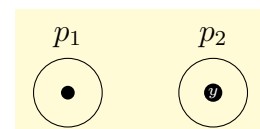
Самый сложный аспект рисования сетей Петри в TikZ — размещение лексем. Начнем с размещения единственной лексемы. Существуют узлы и простой стиль для их набора.

`/tikz/token` (style, no value)

Стиль указывает, что данный узел — лексема. Тогда, по умолчанию, в узле рисуется маленький черный круг. В отличие от мест и переходов, имеет смысл снабжать текстом узлы с лексемами. Такой текст будет набираться крошечным (`tiny`) шрифтом и белым на черном (естественно, можно изменить это, переопределяя стиль `every token`).

`/tikz/every token` (style, no value)

```
\begin{tikzpicture}
\node[place,label=above:$p_1$] (p1) {};
\node[token] at (p1) {};
\node[place,label=above:$p_2$,right=of p1] (p2) {};
\node[token] at (p2) {$y$};
\end{tikzpicture}
```



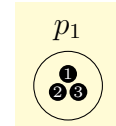
Сложнее, когда в узле две лексемы. Бывает трудно разместить без перекрытия два узла внутри одного узла. Решение этой проблемы: стиль `children are tokens`.

`/tikz/children are tokens`

(style, no value)

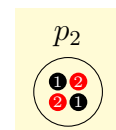
Идея стиля в том, чтобы использовать механизм деревьев для размещения лексем. Каждая лексема, лежащая на месте, трактуется как дочерний узел исходного узла. Обычно, это привело бы к тому, что лексемы были бы помещены ниже места и были бы связаны с местом дугой. Однако стиль `children are tokens` переопределяет функцию роста деревьев так, что лексемы размещаются рядом друг с другом внутри места, а дуга от корневого узла к дочернему не рисуется.

```
\begin{tikzpicture}
\node[place,label=above:$p_1$]{}
[children are tokens] child {node [token] {1}}
child {node [token] {2}} child {node [token] {3}};
\end{tikzpicture}
```



Специальная функция роста дерева для лексем имеет специальный механизм их отображения для каждого возможного числа лексем от одной до девяти. Это отображение размещает каждую лексему в должном месте. Например, единственная лексема размещается прямо на месте. Две лексем размещаются рядом друг с другом на расстоянии, определяемым ключом `token distance`. Три лексем размещаются в равнобедренном треугольнике, длина стороны которого равна `token distance`; и так далее до девяти лексем. Если нужно разместить больше лексем, надо написать собственный код для их размещения.

```
\begin{tikzpicture}
\node[place,label=above:$p_2$]{} [children are tokens]
child {node [token] {1}} child {node [token,fill=red] {2}}
child {node [token,fill=red] {2}} child {node [token] {1}};
\end{tikzpicture}
```

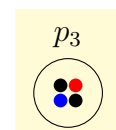


`/tikz/token distance=<distance>`

(no default)

Определяет расстояние между центрами лексем при их размещении опцией `children are tokens`.

```
\begin{tikzpicture}
\node[place,label=above:$p_3$]{}
[children are tokens,token distance=1.1ex]
child {node [token] {}} child {node [token,red] {}}
child {node [token,blue] {}} child {node [token] {}};
\end{tikzpicture}
```



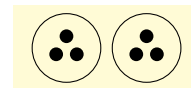
Однако опция `children are tokens` несколько громоздка. Поэтому определены специальные опции для стандартных ситуаций, использующие `children are tokens`. Таким образом, любое изменение в, скажем, стиле `every tokens` будет отражаться на том, как эти опции изображают лексем.

`/tikz/tokens=<number>`

(no default)

Опция передается узлу `place`, а не узлу `tokens`, определяя число `<number>` лексем для места, использующего стиль `tokens`. Как результат, следующие два `tikz`-выражения делают одно и то же:

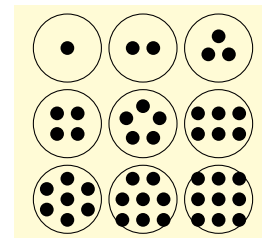
```
\tikz \node[place] {}
  [children are tokens] child {node [token] {}}
  child {node [token] {}} child {node [token] {}};
```



```
\tikz \node[place,tokens=3] {};
```

Можно сказать и `tokens=0`, но в этом случае лексем не рисуются. Опция `tokens` не сможет правильно обрабатывать более девяти лексем.

```
\begin{tikzpicture}
[every place/.style={minimum size=9mm}]
\foreach \x/\y/\tokennumber in
  {0/2/1,1/2/2,2/2/3,
   0/1/4,1/1/5,2/1/6,
   0/0/7,1/0/8,2/0/9}
  \node [place,tokens=\tokennumber] at (\x,\y) {};
\end{tikzpicture}
```

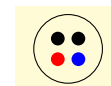


`/tikz/colored tokens=<color list>`

(no default)

Опция должна задаться, когда создается узел места, который получает в качестве параметра список цветов. Тогда она добавит в узел, столько лексем, сколько элементов в списке, рисуя каждую соответствующим цветом.

```
\tikz \node[place,colored tokens={
  black,black,red,blue}] {};
```

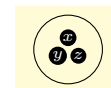


`/tikz/structured tokens=<token texts>`

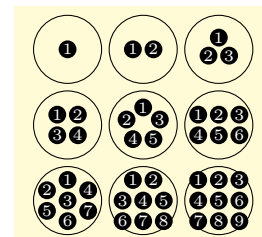
(no default)

Опция также должна передаваться месту, получая список текстов для лексем. Для каждого текста в место будет добавляться новая лексема.

```
\tikz \node[place,structured tokens={
  $x$, $y$, $z$}] {};
```



```
\begin{tikzpicture}
[every place/.style={minimum size=9mm}]
\foreach \x/\y/\tokennumber in
  {0/2/1,1/2/2,2/2/3,
   0/1/4,1/1/5,2/1/6,
   0/0/7,1/0/8,2/0/9}
  \node [place,structured tokens={
    1,...,\tokennumber}] at (\x,\y) {};
\end{tikzpicture}
```

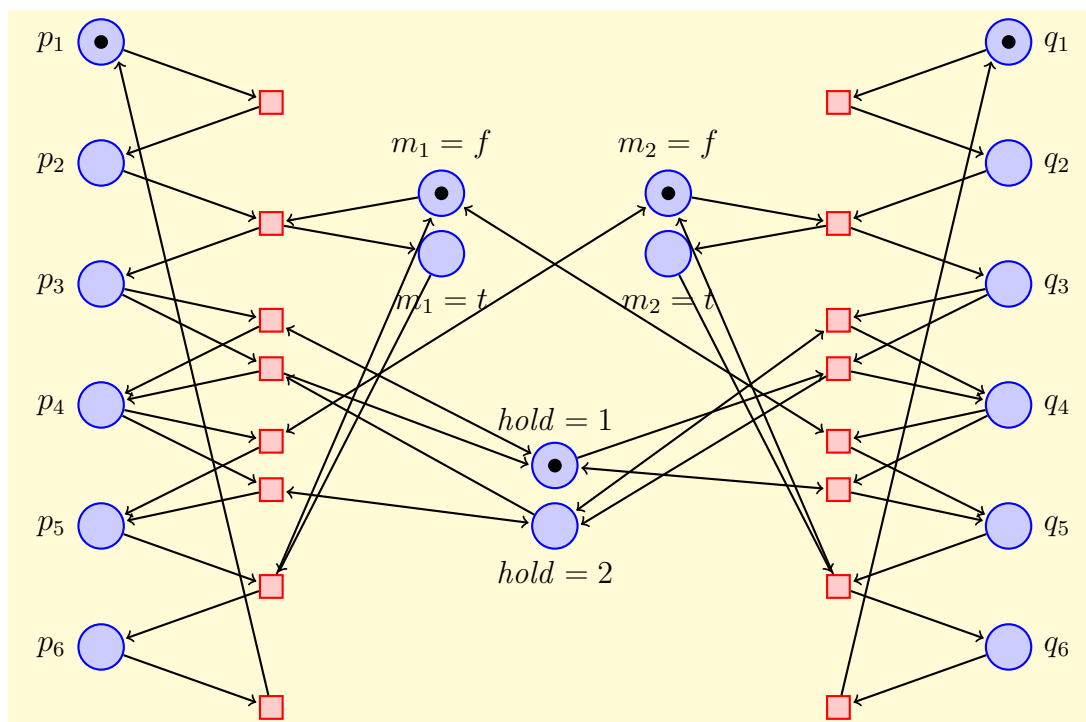


14.4 Примеры

```

\begin{tikzpicture} [yscale=-1.6,xscale=1.5,thick,
  every transition/.style={draw=red,fill=red!20,minimum size=3mm},
  every place/.style={draw=blue,fill=blue!20,minimum size=6mm}]
\foreach \i in {1,...,6}{
  \node[place,label=left:$p_{\i}$] (p\i) at (0,\i) {};
  \node[place,label=right:$q_{\i}$] (q\i) at (8,\i) {};
}
\foreach \name/\var/\vala/\valb/\height/\x in
  {m1/m_1/f/t/2.25/3,m2/m_2/f/t/2.25/5,h/\mathit{hold}/1/2/4.5/4} {
  \node[place,label=above:{$\var=\vala$}] (\name\vala) at (\x,\height) {};
  \node[place,yshift=-8mm,label=below:{$\var=\valb$}]
    (\name\valb) at (\x,\height) {};
}
\node[token] at (p1) {}; \node[token] at (q1) {};
\node[token] at (m1f) {}; \node[token] at (m2f) {};
\node[token] at (h1) {};
\node[transition] at (1.5,1.5) {} edge [pre] (p1) edge [post] (p2);
\node[transition] at (1.5,2.5) {} edge [pre] (p2) edge [pre] (m1f)
  edge[post] (p3) edge [post] (m1t);
\node[transition] at (1.5,3.3) {} edge [pre] (p3) edge [post] (p4)
  edge [pre and post] (h1);
\node[transition] at (1.5,3.7) {} edge [pre] (p3) edge [pre] (h2)
  edge [post](p4) edge [post] (h1.west);
\node[transition] at (1.5,4.3) {} edge [pre] (p4) edge [post] (p5)
  edge [pre and post] (m2f);
\node[transition] at (1.5,4.7) {} edge [pre] (p4) edge [post] (p5)
  edge [pre and post] (h2);
\node[transition] at (1.5,5.5) {} edge [pre] (p5) edge [pre] (m1t)
  edge [post](p6) edge [post] (m1f);
\node[transition] at (1.5,6.5) {} edge [pre] (p6) edge [post]
  (p1.south east);
\node[transition] at (6.5,1.5) {} edge [pre] (q1) edge [post] (q2);
\node[transition] at (6.5,2.5) {} edge [pre] (q2) edge [pre] (m2f)
  edge [post](q3) edge [post] (m2t);
\node[transition] at (6.5,3.3) {} edge [pre] (q3) edge [post] (q4)
  edge [pre and post] (h2);
\node[transition] at (6.5,3.7) {} edge [pre] (q3) edge [pre] (h1)
  edge [post](q4) edge [post] (h2.east);
\node[transition] at (6.5,4.3) {} edge [pre] (q4) edge [post] (q5)
  edge [pre and post] (m1f);
\node[transition] at (6.5,4.7) {} edge [pre] (q4) edge [post] (q5)
  edge [pre and post] (h1);
\node[transition] at (6.5,5.5) {} edge [pre] (q5) edge [pre] (m2t)
  edge [post](q6) edge [post] (m2f);
\node[transition] at (6.5,6.5) {} edge [pre] (q6) edge [post]
  (q1.south west);
\end{tikzpicture}

```



Теперь та же самая сеть еще раз, но с измененными стилями:

```

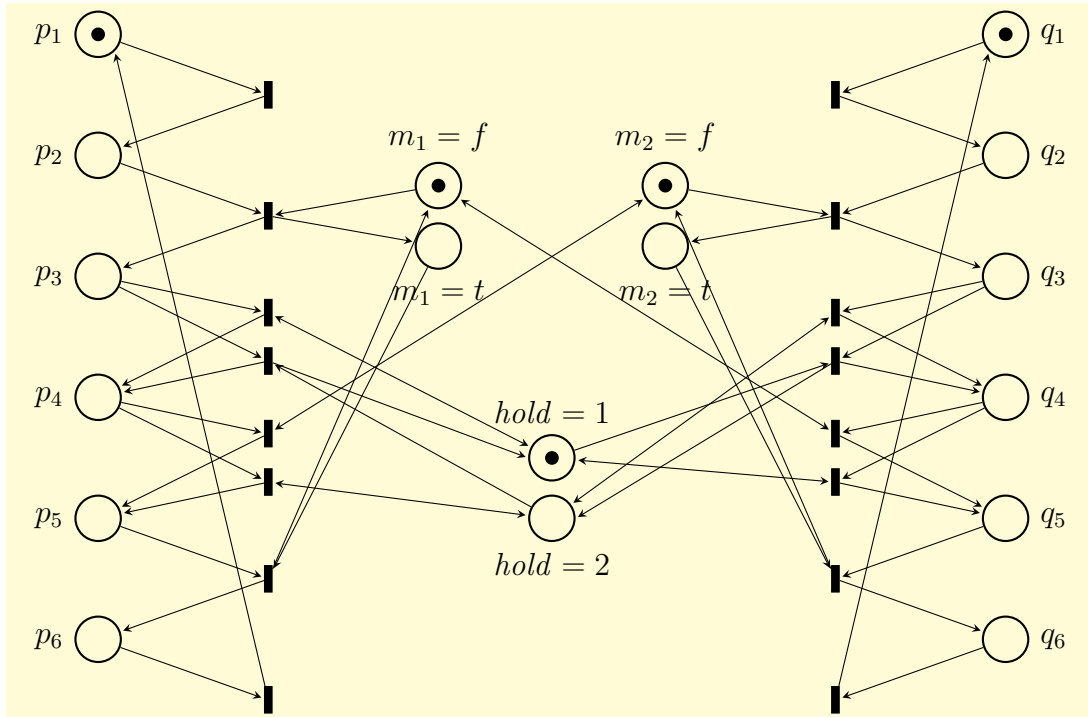
\begin{tikzpicture}
  [yscale=-1.6,xscale=1.5,thick,thin,>=stealth,
  every transition/.style={fill,minimum width=1mm,minimum height=3.5mm},
  every place/.style={draw,thick,minimum size=6mm}]
  \foreach \i in {1,...,6}
    {\node[place,label=left:$p_{\i}$] (p\i) at (0,\i) {};}
    \node[place,label=right:$q_{\i}$] (q\i) at (8,\i) {};}
  \foreach \name/\var/\vala/\valb/\height/\x in
    {m1/m_1/f/t/2.25/3,m2/m_2/f/t/2.25/5,h/\mathit{hold}/1/2/4.5/4}
    {\node[place,label=above:{$\var$}] (\name\vala) at (\x,\height) {};}
    \node[place,yshift=-8mm,label=below:{$\var$}]
      (\name\valb) at (\x,\height) {};}
  \node[token] at (p1) {};\node[token] at (q1) {};\node[token] at (m1f) {};\node[token] at (m2f) {};\node[token] at (h1) {};\node[transition] at (1.5,1.5) {} edge [pre] (p1) edge [post] (p2);
  \node[transition] at (1.5,2.5) {} edge[pre] (p2) edge[pre] (m1f)
    edge[post] (p3) edge[post] (m1t);
  \node[transition] at (1.5,3.3) {} edge [pre] (p3) edge [post] (p4)
    edge [pre and post] (h1);
  \node[transition] at (1.5,3.7) {} edge [pre] (p3) edge [pre] (h2)
    edge [post](p4) edge [post] (h1.west);
  \node[transition] at (1.5,4.3) {} edge [pre] (p4) edge [post] (p5)
    edge [pre and post] (m2f);
  \node[transition] at (1.5,4.7) {} edge [pre] (p4) edge [post] (p5)
    edge [pre and post] (h2);
  \node[transition] at (1.5,5.5) {} edge [pre] (p5) edge [pre] (m1t)
    edge [post](p6) edge [post] (m1f);

```

```

\node[transition] at (1.5,6.5) {} edge [pre] (p6) edge [post]
                                         (p1.south east);
\node[transition] at (6.5,1.5) {} edge [pre] (q1) edge [post] (q2);
\node[transition] at (6.5,2.5) {} edge [pre] (q2) edge [pre] (m2f)
                                         edge [post](q3) edge [post] (m2t);
\node[transition] at (6.5,3.3) {} edge [pre] (q3) edge [post] (q4)
                                         edge [pre and post] (h2);
\node[transition] at (6.5,3.7) {} edge [pre] (q3) edge [pre] (h1)
                                         edge [post](q4) edge [post] (h2.east);
\node[transition] at (6.5,4.3) {} edge [pre] (q4) edge [post] (q5)
                                         edge [pre and post] (m1f);
\node[transition] at (6.5,4.7) {} edge [pre] (q4) edge [post] (q5)
                                         edge [pre and post] (h1);
\node[transition] at (6.5,5.5) {} edge [pre] (q5) edge [pre] (m2t)
                                         edge [post](q6) edge [post] (m2f);
\node[transition] at (6.5,6.5) {} edge [pre] (q6) edge [post]
                                         (q1.south west);
\end{tikzpicture}

```



Глава 15

Библиотека plotmarks

Библиотека `plotmarks` определяет дополнительные графические метки, которые описываются ниже. Если для каждой указанной в таблице метки выполнить код вида

```
\tikz \draw[black!40]
  plot[mark=<mark name>,mark options={color=blue}]
  coordinates {(0,0) (1, 0.5) (2,0) (3,0.5)};
```

где вместо `<mark name>` поставить имя метки, указанное в первом столбце таблицы, то получим результат, представленный во втором столбце таблицы.

Имя метки	Пример	Имя метки	Пример
-			
o		asterisk	
star		10-pointed star	
oplus		oplus*	
otimes		otimes*	
square		square*	
triangle		triangle*	
diamond		diamond*	
halfdiamond*		halfsquare*	
halfsquare right*		halfsquare left*	
pentagon		pentagon*	
Mercedes star		Mercedes star flipped	
halfcircle		halfcircle*	
heart		text	

Каждая из меток может вращаться посредством опции `mark options`, например, `mark options={rotate=90}` или `every mark/.append style={rotate=90}`.

`/pgf/mark color={<color>}` (no default, initially empty)

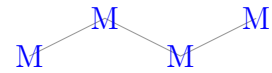
Определяет дополнительный цвет заполнения для меток `halfcircle`, `halfcircle*`, `halfdiamond*`, `halfsquare*`. Значение `empty` устанавливает белый цвет (который является начальным). Специальное значение `none` отключает заполнение соответствующих частей.

Заметим, что метка `halfsquare` заполняется цветом, который определяет опция `mark color`, а в варианте `halfsquare*` половина будет заполняться цветом `mark color`, а половина фактическим цветом заполнения `fill`.

`/pgf/text mark={<text>}` (no default, initially p)

Изменяет на `<text>` текст в метке `mark=text`.

```
\tikz \draw[black!40]
  plot[mark=text,text mark=M,
        mark options={blue}]
  coordinates {(0,0) (1,0.5) (2,0) (3,0.5)};
```



Нет ограничений на число символов в тексте или на что-то еще. Фактически, любой материал \TeX 'а может быть вставлен, как `<text>`, включая изображения.

`/pgf/text mark as node=true|false` (no default, initially false)

Управляет тем, как будет рисоваться метка `mark=text`: либо как `\node`, либо как `\pgftext` (см. ниже). Первый выбор очень гибкий, но, может быть медленным, второй — очень быстрый и обычно достаточен для достижения желаемого результата.

`/pgf/text mark style={<options for mark=text>}` (no default)

Определяет ряд опций, которые управляют внешним видом метки `mark=text`.

Если `/pgf/text mark as node=false` (значение по умолчанию), то `{<options>}` передается как параметр в `\pgftext`, который обеспечивает только некоторые основные ключи, такие как `left`, `right`, `top`, `bottom`, `base`, `rotate`.

Если `/pgf/text mark as node=true`, то `{<options>}` передается как параметр в `\node`. Это означает, что можно использовать очень мощный набор опций, включая `anchor`, `scale`, `fill`, `draw`, `rounded corners` и так далее.

Глава 16

Библиотека positioning

Библиотека `positioning` переопределяет стандартные опции размещения узлов, предоставляя больше возможностей по управлению их размещением.

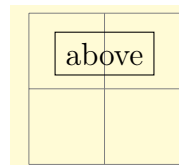
`/tikz/above=<specification>` (default 0pt)

Вместо параметра `<option>` в виде числа, можно использовать более сложный параметр `<specification>`, который имеет следующую общую форму: он начинается с части сдвига `<shifting part>`, которая сопровождается дополнительной частью `<of part>`.

Часть `<shifting part>` может иметь три формы.

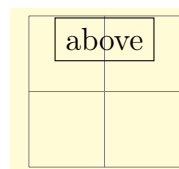
1. Часть `<shifting part>` может быть размерным числом `<dimension>` (или математическим выражением, возвращающим размерную величину) таким, как `2cm` или `3cm/2+4cm`. Тогда выбирается якорь узла `south` и узел сдвигается вверх на величину `<dimension>` (если библиотека `positioning` не загружена, опции `above` работает так).

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\node at (1,1) [above=2pt+3pt,draw] {above};
\end{tikzpicture}
```



2. Часть `<shifting part>` может быть безразмерным числом `<number>` (или любым математическим выражением, которое не включает `pt` или `cm`). Например, `2` или `3 + \sin 60`. В этом случае, также выбирается якорь `south` и узел вертикально сдвигается на вертикальную составляющую точки `(0, <number>)`.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\node at (1,1) [above=.4,draw] {above};
% южная граница узла на 4mm выше точки (1,1)
\end{tikzpicture}
```



3. Часть `<shifting part>` может иметь форму

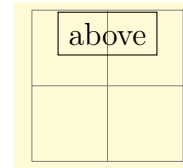
`<number or dimension 1>` and `<number or dimension 2>`.

Такая форма не имела бы смысла для опции `above` (она полезна для опций типа `above left`), но теперь `above` автоматически работает следующим образом:

(1) вычисляется точка `(<number or dimension 2>, <number or dimension 1>)` по обычным правилам вычисления координат (Внимание! Порядок изменен!);

- (2) узел сдвигается на вертикальную составляющую этой точки;
- (3) якорь устанавливается на юг (`south`).

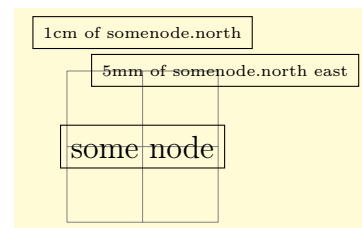
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\node at (1,1) [above=.4 and 1mm,draw] {above};
% южная граница узла на 4мм выше точки (1,1)
\end{tikzpicture}
```



Часть `<shifting part>` может сопровождаться частью `<of part>`, которая имеет одну из следующих форм.

1. Часть `<of-part>` объявляется в виде `of <coordinate>`, при этом `<coordinate>` не заключается в круглые скобки, и это не просто имя узла. Например, могло бы быть `of somenode.north` или `of 2,3`. В этом случае сначала параметр `at` узла устанавливается равным `<coordinate>`. Затем, узел сдвигается согласно части `<shift-part>`. И, наконец, якорь устанавливается на `south`.

```
\begin{tikzpicture}[every node/.style=draw]
\draw[help lines] (0,0) grid (2,2);
\node (somenode) at (1,1) {some node};
\node [above=5mm of somenode.north east]
{\tiny 5mm of somenode.north east};
\node [above=1cm of somenode.north]
{\tiny 1cm of somenode.north};
\end{tikzpicture}
```



Как можно заметить, опция `above=5mm of somenode.north east` размещает узел на 5мм выше северо-восточного якоря узла `somenode`. Того же можно добиться, если написать `above=5mm at=(somenode.north east)`. Если часть `<shift-part>` опущена, сдвиг не всегда равен нулю: используется значение ключа `node distance` (см. ниже).

2. Часть `<of-part>` может иметь вид `of <node name>`. Тогда

- (1) якорь устанавливается на `south`;
- (2) узел сдвигается согласно части `<shifting part>` или, если она опущена, согласно значению ключа `node distance`;
- (3) параметр узла `at` устанавливается равным `<node name>.north`.

Как результат, новый узел будет размещен так, что расстояние между его южной границей и северной границей узла `<node name>` будет равно заданному расстоянию.

`/tikz/below=<specification>` (no default)

Работает как ключ `above`, но перемещение происходит вниз.

`/tikz/left=<specification>` (no default)

Работает как ключ `above`, но перемещение происходит влево по горизонтали.

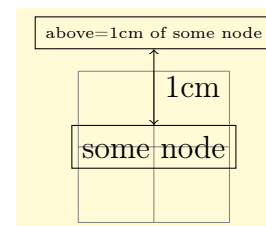
`/tikz/right=<specification>` (no default)

Работает как ключ `above`, но перемещение происходит вправо по горизонтали.

```

\begin{tikzpicture}[every node/.style=draw]
  \draw[help lines] (0,0) grid (2,2);
  \node (some node) at (1,1) {some node};
  \node (other node)[above=1cm of some node]
    {\tiny above=1cm of some node};
  \draw [<->](some node.north) -- (other node.south)
    node [midway,right,draw=none] {1cm};
\end{tikzpicture}

```



Можно значительно изменить поведение `<specification>`, используя ключ

`/tikz/on grid=<boolean>` (no default, initially false)

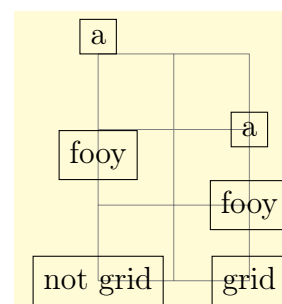
Когда этот ключ получает значение `true`, часть `<of-part>` текущей формы ведет себя по другому: якоря установленные для текущего узла, так же как и якорь, используемый для узла `<node name>`, устанавливаются как `center`. В результате, когда говорят `above=1cm somenode` и присутствует ключ `on grid` со значением `true`, новый узел будет размещаться так, что его центр будет на 1cm выше центра узла `somenode`. Неоднократное размещение узлов таким образом приведет к узлам, центрированным по точкам сетки (отсюда и название ключа).

```

\begin{tikzpicture}[every node/.style=draw]
  \draw[help lines] (0,0) grid (2,3);
  \node (a1) at (0,0) {not grid}; % не по сетке
  \node (b1) [above=1cm of a1] {fooy};
  \node (c1) [above=1cm of b1] {a};

  \node (a2) at (2,0) {grid}; % по сетке
  \node (b2) [on grid,above=1cm of a2] {fooy};
  \node (c2) [on grid,above=1cm of b2] {a};
\end{tikzpicture}

```



`/tikz/node distance=<shifting part>` (no default, initially 1cm and 1cm)

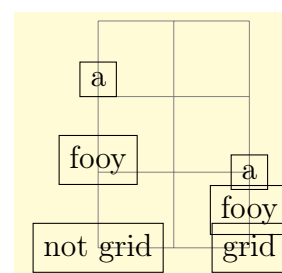
Значение этого ключа используется как `<shifting part>`, тогда и только тогда, когда часть `<of-part>` присутствует, а часть `<shifting part>` отсутствует.

```

\begin{tikzpicture}[every node/.style=draw,node distance=5mm]
  \draw[help lines] (0,0) grid (2,3);
  \node (a1) at (0,0) {not grid}; % не по сетке
  \node (b1) [above=of a1] {fooy};
  \node (c1) [above=of b1] {a};

  \begin{scope}[on grid] % по сетке
    \node (a2) at (2,0) {grid};
    \node (b2) [above=of a2] {fooy};
    \node (c2) [above=of b2] {a};
  \end{scope}
\end{tikzpicture}

```



`/tikz/above left=<specification>` (no default)

Работает как ключ `above`, но поведение части `<shifting part>` сложнее:

1. Когда `<shifting part>` имеет форму

`<number or dimension> and <number or dimension>`

то на первую координату `<number or dimension>` происходит смещение узла вертикально вверх, и на вторую — влево (это то, что ожидается, кроме, возможно, той ситуации, когда используются опции `x` и `y`, чтобы изменить систему xy -координат так, чтобы единичные векторы больше не указывали ожидаемые направления).

2. Когда `<shifting part>` имеет форму `<number or dimension>`, узел сдвигается на `<number or dimension>` в направлении 135° . Это означает, что есть различие между `<shifting part>` вида `1cm` и вида `1cm and 1cm`: во втором случае узел сдвигается на `1cm` вверх и на `1cm` влево, а в первом случае он сдвигается на $\sqrt{2}/2$ вверх и влево.

Следующие опции работают так же, как и опция `above left`.

`/tikz/below left=<specification>` (no default)

`/tikz/above right=<specification>` (no default)

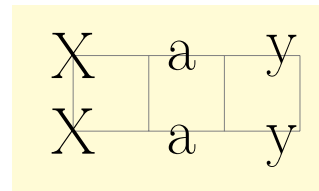
`/tikz/below right=<specification>` (no default)

Библиотека `positioning` также вводит следующие новые ключи размещения.

`/tikz/base left=<specification>` (no default)

Работает как ключ `left`, только вместо якоря `east` используется якорь `base east`, и, когда используется вторая форма `<of-part>`, соответственно, используется якорь `base west`. Этот ключ полезен для формирования цепочки узлов так, чтобы выровнились их базовые линии.

```
\begin{tikzpicture}[node distance=1ex]
\draw[help lines] (0,0) grid (3,1);
\huge \node (X) at (0,1) {X};
      \node (a) [right=of X] {a};
      \node (y) [right=of a] {y};
\huge \node (X) at (0,0) {X};
      \node (a) [base right=of X] {a};
      \node (y) [base right=of a] {y};
\end{tikzpicture}
```



Следующие ключи работают аналогично опция `base left`.

`/tikz/base right=<specification>` (no default)

`/tikz/mid left=<specification>` (no default)

Работает как `base left`, но вместо якорей `base east` и `base west` использует якоря `mid east` и `mid west`, соответственно.

`/tikz/mid right=<specification>` (no default)

Приведем примеры, которые иллюстрируют различия в применении перечисленных выше опций. Но прежде сделаем одно замечание. Даже с библиотекой `positioning` для размещения большого числа узлов простые опции, типа `above`, `above left` и их аналоги, не всегда удобны. Для таких ситуаций существуют две библиотеки, упрощающие позиционирование: библиотека матриц `matrix` и библиотека цепочек `chains`. Эти библиотеки, описаны в главах 10 и 4.

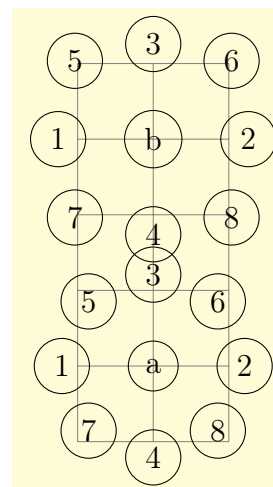
```

\begin{tikzpicture}
  [every node/.style={draw,circle}]
\draw[help lines] (0,0) grid (2,5);

\begin{scope}[node distance=5mm]
  \node(a) at (1,1){a}; \node[left=of a] {1};
  \node[right=of a]{2}; \node[above=of a] {3};
  \node[below=of a]{4}; \node[above left=of a]{5};
  \node[above right=of a] {6};
  \node[below left=of a] {7};
  \node[below right=of a] {8};
\end{scope}

\begin{scope}[node distance=5mm and 5mm]
  \node(b) at (1,4){b}; \node[left=of b] {1};
  \node[right=of b]{2}; \node[above=of b] {3};
  \node[below=of b]{4}; \node[above left=of b]{5};
  \node[above right=of b] {6};
  \node[below left=of b] {7};
  \node[below right=of b] {8};
\end{scope}
\end{tikzpicture}

```



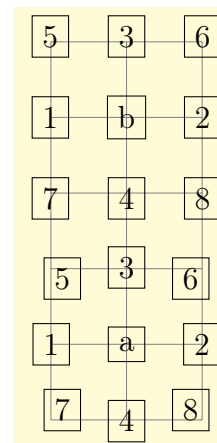
```

\begin{tikzpicture}
  [every node/.style={draw,rectangle}]
\draw[help lines] (0,0) grid (2,5);

\begin{scope}[node distance=5mm]
  \node(a) at (1,1){a}; \node[left=of a] {1};
  \node[right=of a]{2}; \node[above=of a] {3};
  \node[below=of a]{4}; \node[above left=of a]{5};
  \node[above right=of a] {6};
  \node[below left=of a] {7};
  \node[below right=of a] {8};
\end{scope}

\begin{scope}[node distance=5mm and 5mm]
  \node(b) at (1,4){b}; \node[left=of b] {1};
  \node[right=of b]{2}; \node[above=of b] {3};
  \node[below=of b]{4}; \node[above left=of b]{5};
  \node[above right=of b] {6};
  \node[below left=of b] {7};
  \node[below right=of b] {8};
\end{scope}
\end{tikzpicture}

```



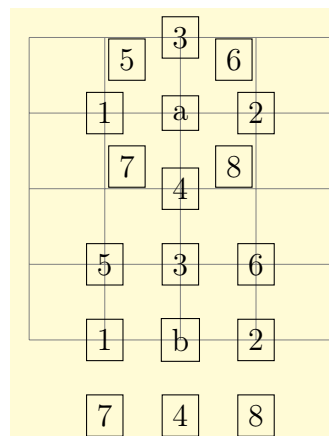
```

\begin{tikzpicture}
  [every node/.style={draw,rectangle},on grid]
\draw[help lines] (0,0) grid (4,4);

\begin{scope}[node distance=1]
  \node(a) at (2,3){a}; \node[left=of a] {1};
  \node[right=of a]{2}; \node[above=of a] {3};
  \node[below=of a]{4}; \node[above left=of a]{5};
  \node[above right=of a] {6};
  \node[below left=of a] {7};
  \node[below right=of a] {8};
\end{scope}

\begin{scope}[node distance=1 and 1]
  \node(b) at (2,0){b}; \node[left=of b] {1};
  \node[right=of b]{2}; \node[above=of b] {3};
  \node[below=of b]{4}; \node[above left=of b]{5};
  \node[above right=of b] {6};
  \node[below left=of b] (7);
  \node[below right=of b] {8};
\end{scope}
\end{tikzpicture}

```



Глава 17

Библиотека shadings

Библиотека `shadings`, помимо способов растушевывания `axis`, `ball`, `radial`, доступных независимо от того, загружена библиотека или нет, определяет новые. Растушевывания перечисляются в алфавитном (английском) порядке. Цветами некоторых из этих способов можно управлять, используя специальные опции (например, `left color=`).

Растушевывание `axis` всегда доступно, цвета изменяется постепенно между тремя горизонтальными линиями. Верхняя линия — верхняя (высшая) точка пути, средняя линия находится в середине пути, нижняя линия — нижняя точка пути.

`/tikz/top color=<color>` (no default)

Опция предписывает цвет, который используется наверху в растушевывании `axis`. Когда опция задана, происходит следующее:

1. Выбирается опция `shade`.
2. Выбирается опция `shading=axis`.
3. Средний цвет растушевывания выбирается как среднее заданного цвета для верхней части (`<color>`) и цвета, который в настоящее время выбран для нижней части.
4. Угол вращения растушевывания устанавливается равным 0.

```
\tikz \draw[top color=red]
(0,0) rectangle (2,1);
```



`/tikz/bottom color=<color>` (no default)

Работает аналогично опции `top color`, но для нижнего цвета растушевывания `axis`.

`/tikz/middle color=<color>` (no default)

Определяет цвет для середины растушевывания `axis`. Так как задание верхнего и нижнего цвета изменяют цвет посередине, эта опция должна задаваться последней, если нужно задать все три опции:

```
\tikz \draw[top color=white,bottom color=black,
middle color=red](0,0) rectangle (2,1);
```



`/tikz/left color=<color>` (no default)

Делает то же, что и опция `top color`, но с углом растушевывания равным 90°.

`/tikz/right color=<color>` (no default)

Работает аналогично опции `left color`.

```
\tikz \draw[left color=yellow,right color=green!50]
(0,0) rectangle (2,1);
```

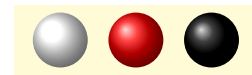


Растушевывание ball всегда доступно, заполняет путь так, что он становится похожим на шар. Значение по умолчанию для цвета шара — `blue`.

```
/tikz/ball color=<color> (no default)
```

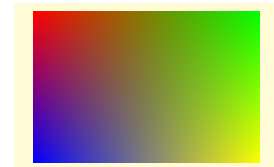
Устанавливает цвет, используемый для растушевывания шара, а также опции `shade` и `shading=ball`. Весь шар никогда не будет иметь цвет `<color>`. В подсветке его центра присутствует определенное количество белого цвета, а ближе к границе — черного. По этой причине, можно использовать и код `color=white`, и код `color=black`.

```
\begin{tikzpicture}
\shade[ball color=white] (0,0) circle (2ex);
\shade[ball color=red] (1,0) circle (2ex);
\shade[ball color=black] (2,0) circle (2ex);
\end{tikzpicture}
```



Растушевывание bilinear interpolation заполняет прямоугольник посредством двухлинейной интерполяции между цветами, заданными в четырех углах прямоугольника. Эти четыре цвета имеют имена (это же и имена опций, их устанавливающих): `lower left`, `lower right`, `upper left`, `upper right`. Изменяя цвета, можно изменить видимый результат растушевывания.

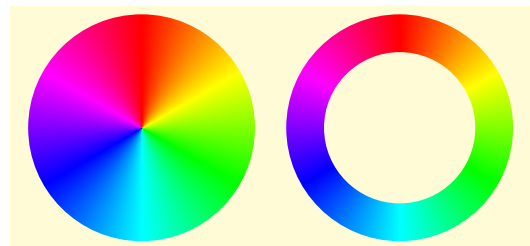
```
\tikz
\shade[upper left=red,upper right=green,
lower left=blue,lower right=yellow]
(0,0) rectangle (3,2);
```



Растушевывание color wheel заполняет круг, создавая иллюзию вращающегося колеса. Чтобы получить цветное кольцо, следует «выбросить» некоторый внутренний круг, для чего используется правило `even odd rule` для определения внутренней точки (см. [1, 15.4.2]).

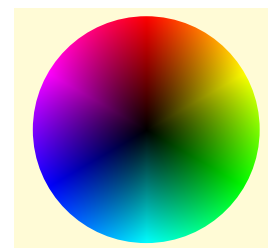
```
\tikz \shade[shading=color wheel]
(0,0) circle (1.5);

\tikz \shade[shading=color wheel]
[even odd rule]
(0,0) circle (1.5)
(0,0) circle (1);
```



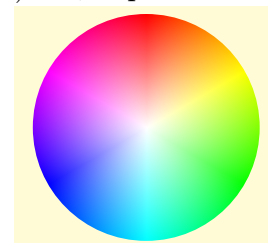
Растушевывание color wheel black center очень похоже на растушевывание `color wheel`, но яркость снижается до нуля (до черного цвета) в центре круга.


```
\tikz \shade[shading=color wheel black center]
(0,0) circle (1.5);
```



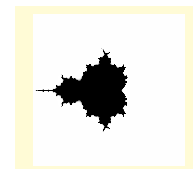
Растушевывание **color wheel white center** очень похоже на растушевывание **color wheel**, но насыщенность снижается до нуля (до белого цвета) в центре.

```
\tikz \shade[shading=color wheel white center]
(0,0) circle (1.5);
```



Растушевывание **Mandelbrot set** — только для развлечения: путь заполняется масштабируемым множеством Мандельбройта (Mandelbrot set).

```
\tikz \shade[shading=Mandelbrot set]
(0,0) rectangle (2,2);
```



Растушевывание **radial** всегда доступно, путь заполняется постепенным размыванием определенного цвета в середине до другого цвета на границе. Если путь — круг, то внешний цвет будет достигнут точно на границе. Если путь растушевывания не круг, то внешний цвет немного продолжится в углы. По умолчанию внутренний цвет серый, внешний цвет белый.

Пользуясь опциями **inner color** и **outer color** можно переопределить внутренний и внешний цвет этого растушевывания. Когда используется опция **inner color**, автоматически устанавливаются опции **shade** и **shading=radial**

```
\tikz \draw[inner color=red]
(0,0) rectangle (2,1);
```

```
\tikz \draw[outer color=red,
inner color=white]
(0,0) rectangle (2,1);
```



Глава 18

Библиотека shadows

Библиотека `shadows` определяет новые стили, которые помогают добавить в путь или узел (частично) прозрачные тени. Тень — обычно, черная или серая область, рисуемая позади пути или узла так, чтобы добавить изображению визуальную глубину. Опции из библиотеки теней основаны на опции `preaction`, что позволяет использовать путь дважды: один раз, чтобы нарисовать тень (немного сдвинутую), и второй раз, чтобы нарисовать сам путь. Тень можно добавить к пути, а не ко всему окружению.

В дополнение к общей опции `shadow`, существуют специальные опции, например, `circular shadow`, которые можно использоваться только со специальным видом пути (`circular shadow` — с кругом). У таких теней есть то преимущество, что они визуально более естественны, поскольку более гладко сопрягают тень с фоном. Специальные тени используют фединг, который поддерживают только некоторые принтеры.

18.1 Опция `general shadow`

Тени создаются единственной опцией — `general shadow`. Ее отличие других опций, таких как `drop shadow`, `copy shadow`, только в предварительно устанавливаемых командах. Обычно опция `general shadow` не должна использоваться напрямую.

```
/tikz/general shadow=<shadow options> (default empty)
```

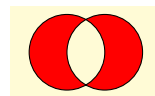
Опцию можно задавать только в `\path` или `node`. Параметр `<shadow options>` содержит опции, позволяющие создать нужную тень. После установки этих опций можно применять к пути преобразования холста: масштабирование `shadow scale` (с началом масштабирования в центре пути), и сдвиги `shadow xshift` и `shadow yshift`. Поскольку эти преобразования выполняются, используя преобразования холста, тени не рассматриваются при вычислении ограничивающего прямоугольника рисунка.

```
/tikz/shadow scale=<factor> (no default, initially 1)
```

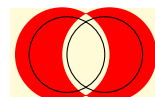
```
/tikz/shadow xshift=<factor> (no default, initially 0pt)
```

```
/tikz/shadow yshift=<factor> (no default, initially 0pt)
```

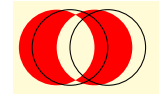
```
\tikz[even odd rule] \draw[general shadow={fill=red}]
(0,0) circle (.5) (0.5,0) circle (.5);
```



```
\tikz[even odd rule]
\draw[general shadow={fill=red, shadow scale=1.25}]
(0,0) circle (.5) (0.5,0) circle (.5);
```



```
\tikz[even odd rule]
\draw[general shadow={fill=red,shadow xshift=-5pt}]
(0,0) circle (.5) (0.5,0) circle (.5);
```



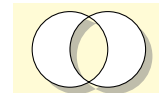
18.2 Тени для произвольных путей и форм

18.2.1 Опция `drop shadow`

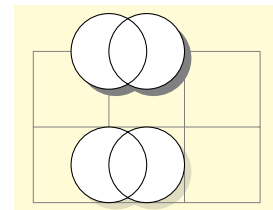
`/tikz/drop shadow=<shadow options>` (default empty)

Добавляет тень к пути `\path` или узлу `node`, используя опцию `general shadow` и передавая этой опции параметры из `<shadow options>`, и, дополнительно, перед ними, следующие опции: `shadow scale=1`, `shadow xshift=.5ex`, `shadow yshift=-.5ex`, `opacity=.5`, `fill=black!50`, `every shadow`.

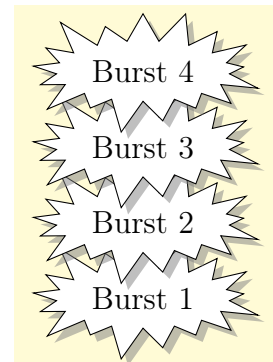
```
\tikz [even odd rule]
\filldraw [drop shadow,fill=white]
(0,0) circle (.5) (0.5,0) circle (.5);
```



```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\filldraw [drop shadow={opacity=0.25},fill=white]
(1,.5) circle (.5) (1.5,.5) circle (.5);
\filldraw [drop shadow={opacity=1},fill=white]
(1,2) circle (.5) (1.5,2) circle (.5);
\end{tikzpicture}
```



```
\begin{tikzpicture}
\foreach \i in {1,...,4}
\node[starburst,drop shadow,fill=white,draw]
at (0,\i) {Burst \i};
\end{tikzpicture}
```

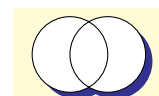


`/tikz/every shadow`

(style, initially empty)

Стиль выполняется в дополнение к любым параметрам из `<shadow options>` для каждой тени. Изменяя этот стиль, можно изменить способ отображения тени.

```
\begin{tikzpicture}[every shadow/.style={
opacity=.8,fill=blue!50!black}]
\filldraw [drop shadow,fill=white]
(0,0) circle (.5) (0.5,0) circle (.5);
\end{tikzpicture}
```

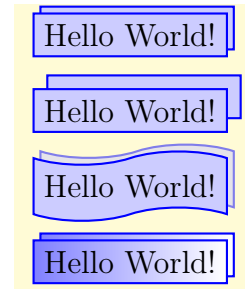


18.2.2 Опция `copy shadow`

`/tikz/copy shadow=<shadow options>` (default empty)

Опция не создает реальной тени, а создает копию пути, которая располагается с небольшим сдвигом позади исходного, тем самым, имитируя тень. Таких имитаций может быть несколько. Опция по умолчанию устанавливает ключи: `shadow scale=1`, `shadow xshift=.5ex`, `shadow yshift=-.5ex`, `every shadow`. Еще устанавливаются опции `fill=<fill color>` и `draw=<draw color>`, используемые для заполнения и прорисовки основного пути.

```
\begin{tikzpicture}
\node [copy shadow,fill=blue!20,draw=blue,thick]
      {Hello World!};
\node at (0,-1) [copy shadow={shadow xshift=1ex,
      shadow yshift=1ex},fill=blue!20,draw=blue,thick]
      {Hello World!};
\node at (0,-2) [copy shadow={opacity=.5},tape,
      fill=blue!20,draw=blue,thick] {Hello World!};
% Нужно указать левый цвет, так как растушевывание
% не применяется автоматически при создании тени
\node at (0,-3) [copy shadow={left color=blue!50},
      left color=blue!50,draw=blue,thick]{Hello World!};
\end{tikzpicture}
```



`/tikz/double copy shadow=<shadow options>` (default empty)

Аналогична предыдущей опции, но число копий удваивается и каждая последовательно рисуется с указанными сдвигами.

```
\begin{tikzpicture}
\node [double copy shadow,fill=blue!20,
      draw=blue,thick] {Hello World!};
\node at (0,-1) [double copy shadow={
      shadow xshift=1ex,shadow yshift=1ex},
      fill=blue!20,draw=blue,thick] {Hello World!};
\node at (0,-2) [double copy shadow={opacity=.5},
      tape,fill=blue!20,draw=blue,thick] {Hello World!};
\node at (0,-3) [double copy shadow={left color=blue!50},
      left color=blue!50,draw=blue,thick] {Hello World!};
\end{tikzpicture}
```



18.3 Тени для специальных путей и узлов

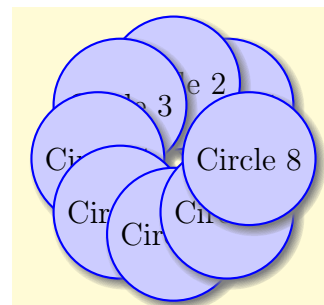
Тени этого раздела должны применяться только к путям, имеющим специальную форму, иначе они будут выглядеть весьма странно.

`/tikz/circular drop shadow=<shadow options>` (no default)

Опция работает как `drop shadow`, но добавляет фединг, постепенно изменяя по кругу цвет тени так, что тень постепенно исчезнет к границе. Для этой тени предварительно

устанавливаются опции: `fill=black`, `shadow scale=1.1`, `shadow xshift=.3ex`, `shadow yshift=-.3ex`, `every shadow`, `path fading={circle with fuzzy edge 15 percent}`.

```
\begin{tikzpicture}
\foreach \i in {1,...,8}
  \node[circle,circular drop shadow,
        draw=blue,fill=blue!20,thick]
        at (\i*45:1) {Circle \i};
\end{tikzpicture}
```

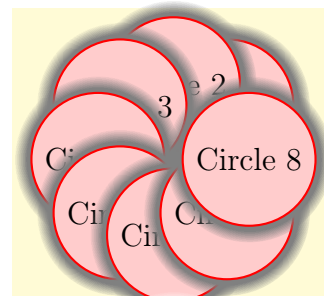


`/tikz/circular glow=<shadow options>`

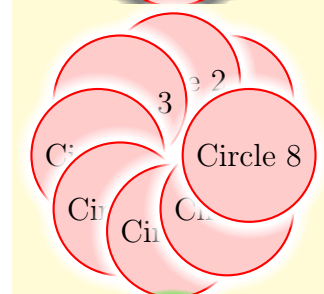
(no default)

Опция работает аналогично предыдущей, только тень не сдвигается. Это создает визуальный эффект свечения позади круга. Для этой тени предварительно устанавливаются опции: `shadow scale=1.25`, `shadow xshift=0pt`, `shadow yshift=0pt`, `fill=black`, `path fading={circle with fuzzy edge 15 percent}`, `every shadow`.

```
\begin{tikzpicture}
\foreach \i in {1,...,8}
  \node[circle,circular glow,fill=red!20,
        draw=red,thick]
        at (\i*45:1) {Circle \i};
\end{tikzpicture}
```



```
\begin{tikzpicture}
\foreach \i in {1,...,8}
  \node[circle,circular glow={fill=white},
        fill=red!20,draw=red,thick]
        at (\i*45:1) {Circle \i};
\end{tikzpicture}
```

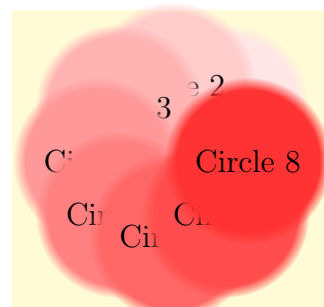


```
\begin{tikzpicture}
\foreach \i in {1,...,8}
  \node[circle,circular glow={fill=green},
        fill=black,text=green!50!black]
        at (\i*45:1) {Circle \i};
\end{tikzpicture}
```



Особенно интересный эффект получается, если использовать опцию `circular glow` и не использовать опцию `fill` для заполнения пути:

```
\begin{tikzpicture}
\foreach \i in {1,...,8}
  \node[circle,circular glow={fill=red!\i0}]
        at (\i*45:1) {Circle \i};
\end{tikzpicture}
```



Глава 19

Библиотеки форм

19.1 Предопределенные формы

В Tikz определены три стандартные формы: прямоугольник (`rectangle`), круг (`circle`) и точка (`coordinate`). Две первые формы имеют стандартный набор якорей, а форма `coordinate` — только якорь центра (`center`). Существуют много дополнительных форм, определенных в разных библиотеках. Большинство этих форм разработаны Марком Виброу (Mark Wibrow). Библиотека `shapes` введена только для совместимости. Надо загружать непосредственно только ее подбиблиотеки `shapes.geometric`, `shapes.misc`. На внешний вид форм влияет много параметров, которые перечислены в [1, раздел 16.2].

19.2 Геометрические формы

Геометрические формы, соответствующие основным геометрическим объектам, становятся доступными после загрузки библиотеки `shapes.geometric`.

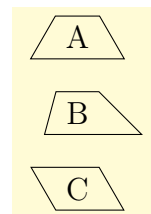
▭ **Форма `trapezium`** (трапеция) — четырехугольник с двумя непараллельными и двумя параллельными сторонами. Трапеция поддерживает вращение.

Нижние внутренние углы трапеции определяются ключами

```
/pgf/trapezium left angle=<angle> (no default, initially 60)  
/pgf/trapezium right angle=<angle> (no default, initially 60)  
/pgf/trapezium angle=<angle> (no default)
```

Ключ просто устанавливает значения двух предыдущих ключей равными `<angle>`.

```
\begin{tikzpicture}  
  \tikzstyle{every node}=[trapezium, draw]  
  \node at (0,2) {A};  
  \node[trapezium left angle=75, trapezium right angle=45pt]  
    at (0,1) {B};  
  \node[trapezium left angle=120, trapezium right angle=60pt]  
    at (0,0) {C};  
\end{tikzpicture}
```



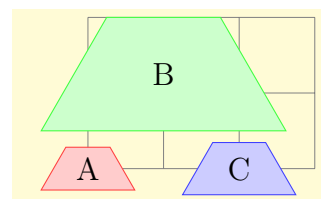
```
/pgf/trapezium stretches=<boolean> (default false)
```

Ключ управляет тем, будет ли ширина и высота трапеции изменяться независимо, когда рассматриваются любые минимальные требования к размерам. Устанавливая

`<boolean>` равным `true`, трапецию можно независимо растягивать по горизонтали и вертикали. При установке `<boolean>` равным `false` (по умолчанию), гарантируется, что растяжение по горизонтали и вертикали всегда будет одинаковым.

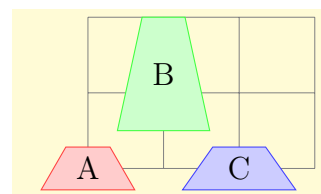
```
\tikzset{my node/.style={trapezium,
    fill=#1!20,draw=#1!75, text=black}}

\begin{tikzpicture}
\draw [help lines] grid (3,2);
\node [my node=red] {A};
\node [my node=green, minimum height=1.5cm]
    at (1, 1.25) {B};
\node [my node=blue, minimum width=1.5cm]
    at (2, 0) {C};
\end{tikzpicture}
```



```
\tikzset{my node/.style={trapezium,
    fill=#1!20, draw=#1!75, text=black}}

\begin{tikzpicture}
\tikzset{trapezium stretches=true}
\draw [help lines] grid (3,2);
\node [my node=red] {A};
\node [my node=green,minimum height=1.5cm]
    at (1, 1.25) {B};
\node [my node=blue,minimum width=1.5cm]
    at (2, 0) {C};
\end{tikzpicture}
```

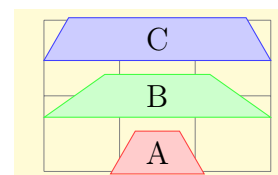


`/pgf/trapezium stretches body=<boolean>` (default false)

Подобна опции `trapezium stretches`, но, когда параметр `<boolean>` равен `true`, увеличивается только тело трапеции, когда применяется минимальная ширина.

```
\tikzset{my node/.style={trapezium,
    fill=#1!20, draw=#1!75, text=black}}

\begin{tikzpicture}
\draw[help lines] grid (3,2);
\node[my node=red] at (1.5,.25) {A};
\node[my node=green,minimum width=3cm,trapezium stretches]
    at (1.5,1) {B};
\node[my node=blue,minimum width=3cm,trapezium stretches body]
    at (1.5,1.75) {C};
\end{tikzpicture}
```

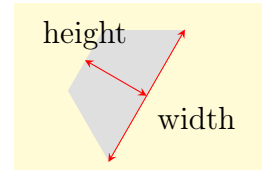


Независимо от вращения трапеции, ее ширина и высота остаются неизменными.

```

\begin{tikzpicture}
[>stealth, every node/.style={text=black},
 shape border uses incircle, shape border rotate=60]
\node [trapezium, fill=gray!25, minimum width=2cm] (t) {};
\draw [red, <->] (t.top side) -- (t.bottom side)
node [at start, above] {height};
\draw [red, <->] (t.bottom left corner) -- (t.bottom right corner)
node [midway, below right] {width};
% Независимо от вращения трапеции, ее ширина и высота не меняются.
\end{tikzpicture}

```



◆ **Форма diamond** (алмаз) хорошо подходит для размещения текста. Отношение между шириной и высотой формы по умолчанию равно 1, но может быть изменено с помощью опции `aspect=<value>`, которая вызывает макрос `\pgfsetshapeaspect`.

○ **Форма ellipse** (эллипс) также хорошо подходит для размещения текста, если нет внутреннего разделения формы.

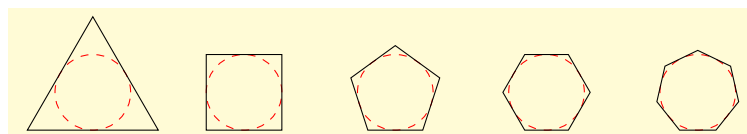
◐ **Форма semicircle** (полукруг) поддерживает вращение границы формы.

◓ **Форма regular polygon** (правильный многоугольник) по умолчанию, рисуется так, чтобы внизу была сторона (а не угол). Форма поддерживает вращение, но граница многоугольника всегда создается, используя вписанную окружность, радиус которой вычисляется так, чтобы многоугольник плотно охватывал содержимое узла (включая `inner sep`).

```

\begin{tikzpicture}
\foreach \a in {3,...,7}{\draw[red, dashed] (\a*2,0) circle(0.5cm);
\node[regular polygon, regular polygon sides=\a, draw,
inner sep=0.3535cm] at (\a*2,0) {};}
\end{tikzpicture}

```

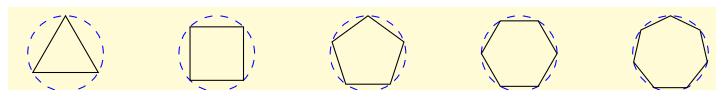


Если узел увеличивается до некоторого указанного минимального размера, размер интерпретируется как диаметр описанной окружности, то есть, окружности, которая проходит через все углы границы многоугольника.

```

\begin{tikzpicture}
\foreach \a in {3,...,7}{\draw[blue, dashed] (\a*2,0) circle(0.5cm);
\node[regular polygon, regular polygon sides=\a,
minimum size=1cm, draw] at (\a*2,0) {};}
\end{tikzpicture}

```



Как ясно из приведенных примеров, есть ключ, позволяющий установить число сторон правильного многоугольника.

`/pgf/regular polygon sides=<integer>` (no default, initially 5)

☆ **Форма `star`** (звезда) по умолчанию (без преобразований) рисуется с одним обращенным вверх лучом. Форма поддерживает вращение, а граница звезды всегда создается, используя вписанную окружность. Звезду можно представлять, как набор внутренних и внешних точек. Внутренние точки границы лежат на границе круга такого радиуса, который содержит узел, а внешние точки — на описанной окружности. Любое определение минимального размера, ширины или высоты, интерпретируется как диаметр описанной окружности.

Для изменения вида звезды можно использовать следующие ключи.

`/pgf/star points=<integer>` (no default, initially 5)

Устанавливает число точек (лучей) звезды.

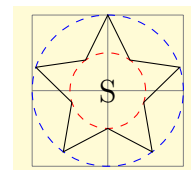
`/pgf/star point height=<distance>` (no default, initially .5cm)

Устанавливает расстояние между внутренней и внешней окружностями.

`/pgf/star point ratio=<number>` (no default, initially 1.5)

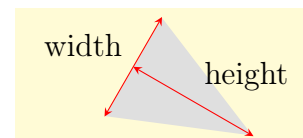
Устанавливает отношение между внутренним и внешним радиусами окружностей. Это отношение сохраняется при масштабировании.

```
\begin{tikzpicture}
  \draw [help lines] (0,0) grid (2,2);
  \draw [blue, dashed] (1,1) circle(1cm);
  \draw [red, dashed] (1,1) circle(.5cm);
  \node [star, star point height=.5cm,
        minimum size=2cm, draw] at (1,1) {S};
\end{tikzpicture}
```



▷ **Форма `isosceles triangle`** (равнобедренный треугольник) поддерживает вращение формы. Угол поворота определяет направление, в котором располагается вершина треугольника. Однако, независимо от вращения формы, ширину и высоту равнобедренного треугольника всегда рассматривают следующим образом:

```
\begin{tikzpicture}
[>=stealth, every node/.style={text=black},
  shape border uses incircle,
  shape border rotate=-30]
\node[isosceles triangle, fill=gray!25,
      minimum width=1.5cm] (t) {};
\draw[red, <->] (t.left corner) -- (t.right corner)
  node [midway, above left] {width};
\draw[red, <->] (t.apex) -- (t.lower side)
  node [midway, above right] {height};
\end{tikzpicture}
```



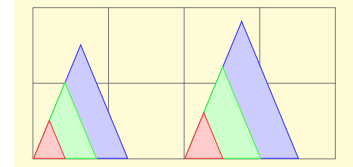
`/pgf/isosceles triangle apex angle` (no default, initially 45)

Устанавливает угол в основании равнобедренного треугольника.

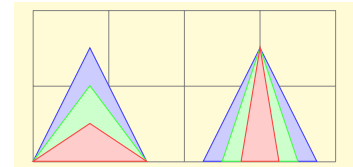
`/pgf/isosceles triangle stretches=<boolean>` (default `true`)

По умолчанию `<boolean>` — ложь. Это означает, любое требование минимальной ширины или минимальной высоты будет выполнено так, чтобы сохранить углы при основании. Если же `<boolean>` — истина, опции установки минимальной ширины и высоты могут применяться независимо.

```
\begin{tikzpicture}
[paint/.style={draw=#1!75, fill=#1!20}]
\tikzset{every node/.style={
  isosceles triangle, draw, inner sep=0pt,
  anchor=left corner, shape border rotate=90}}
\draw[help lines] grid(4,2);
\foreach \a/\c in {1.5/blue, 1/green, 0.5/red}{
  \node[paint=\c, minimum height=\a cm] at (0,0) {};
  \node[paint=\c, minimum width=\a cm] at (2,0) {};}
\end{tikzpicture}
```



```
\begin{tikzpicture}
[paint/.style={draw=#1!75, fill=#1!20}]
\tikzset{every node/.style={isosceles triangle,
  draw, inner sep=0pt, anchor=south,
  shape border rotate=90,
  isosceles triangle stretches}}
\draw[help lines] grid(4,2);
\foreach \a/\c in {1.5/blue, 1/green, 0.5/red}{
  \node[paint=\c, minimum height=\a cm, minimum width=1.5cm]
    at (0.75,0) {};
  \node[paint=\c, minimum width=\a cm, minimum height=1.5cm]
    at (3,0) {};}
\end{tikzpicture}
```



◊ **Форма kite** (воздушный змей) поддерживает вращение границы формы. Есть ключи, позволяющие определить верхние и нижние углы воздушного змея.

`/pgf/kite upper vertex angle=<angle>` (no default, initially 120)

Устанавливает верхний внутренний угол воздушного змея.

`/pgf/kite lower vertex angle=<angle>` (no default, initially 60)

Устанавливает нижний внутренний угол воздушного змея.

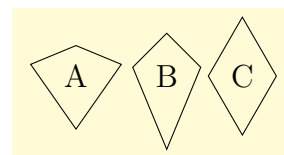
`/pgf/kite vertex angles=<angle specification>` (no default)

Устанавливает и верхний и нижний углы. Параметр `<angle specification>` может быть парой углов в виде `<upper angle>` and `<lower angle>`, или единственный угол, но тогда оба угла будут одинаковыми.

```

\begin{tikzpicture}
\tikzstyle{every node}=[kite, draw]
\node[kite upper vertex angle=135,
kite lower vertex angle=70] at (0,0) {A};
\node[kite vertex angles=90 and 45] at (1.2,0) {B};
\node[kite vertex angles=60] at (2.2,0) {C};
\end{tikzpicture}

```

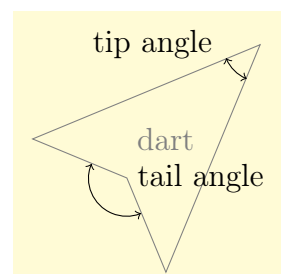


➤ **Форма `dart`** (стрела, дротик), которую также иногда называют наконечником стрелы или вогнутым воздушным змеем, поддерживает вращение формы. Угол вращения определяет направление, в котором располагается наконечник стрелы (если не были применены другие преобразования).

```

\begin{tikzpicture}
\node[dart, draw, gray, shape border uses incircle,
shape border rotate=45] (d) {dart};
\draw [<->]
(d.tip)++(202.5:.5cm) arc(202.5:247.5:.5cm);
\node [left=.5cm] at (d.tip) {tip angle};
\draw [<->]
(d.tail center)++(157.5:.5cm) arc(157.5:292.5:.5cm);
\node [right] at (d.tail center) {tail angle};
\end{tikzpicture}

```



`/pgf/dart tip angle=<angle>` (no default, initially 45)

Устанавливает внутренний угол в вершине стрелы.

`/pgf/dart tail angle=<angle>` (no default, initially 135)

Устанавливает внешний угол в основании стрелы.

➤ **Форма `circular sector`** (круговой сектор или клин) поддерживает вращение формы. Угол вращения определяет направление, в котором располагается вершина сектора (если не были применены другие преобразования).

`/pgf/circular sector angle=<angle>` (no default, initially 60)

Устанавливает центральный угол сектора.

⊖ **Форма `cylinder`** (цилиндр, точнее, двумерное представление цилиндра) поддерживает вращение формы.

```

\begin{tikzpicture}
\node[cylinder, draw, shape aspect=.5] {ABC};
\end{tikzpicture}

```

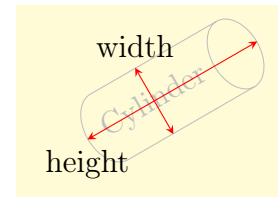


Независимо от вращения, высота — всегда расстояние между верхней и нижней дугами, а ширина — всегда расстояние между прямыми сторонами.

```

\begin{tikzpicture}[>=stealth]
\node [cylinder, gray!50, rotate=30, draw,
       minimum height=2cm, minimum width=1cm]
      (c) {Cylinder};
\draw[red, <->] (c.top) -- (c.bottom)
      node [at end, below, black] {height};
\draw[red, <->] (c.north) -- (c.south)
      node [at start, above, black] {width};
\end{tikzpicture}

```

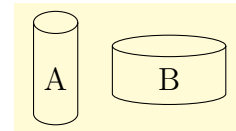


Изменение формы до некоторой минимальной высоты вытягивает только тело цилиндра. В отличие от этого, изменение формы до некоторой минимальной ширины изменит концевые кривые.

```

\begin{tikzpicture}[>=stealth, shape aspect=.5]
\tikzset{every node/.style={cylinder,
                             shape border rotate=90, draw}}
\node [minimum height=1.5cm] {A};
\node [minimum width=1.5cm] at (1.5,0) {B};
\end{tikzpicture}

```



`/pgf/aspect=<value>`

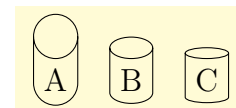
(no default, initially 1.0)

Аспект — рекомендация для радиусов кривых, завершающих цилиндр. Он может игнорироваться, если форма увеличивается до некоторой минимальной ширины.

```

\begin{tikzpicture}[>=stealth]
\tikzset{every node/.style={cylinder,shape border rotate=90, draw}}
\node [aspect=1.0] {A};
\node [aspect=0.5] at (1,0) {B};
\node [aspect=0.25] at (2,0) {C};
\end{tikzpicture}

```



`/pgf/cylinder uses custom fill=<boolean>`

(default true)

Допускает (или нет) заполнение пользователем тела и вершины цилиндра. Фоновый путь для формы не должен заполняться (например, в TikZ опция `fill` для узла должна косвенно или явно устанавливаться в `none`).

`/pgf/cylinder end fill=<color>`

(no default, initially white)

Устанавливает цвет вершины цилиндра.

`/pgf/cylinder body fill=<color>`

(no default, initially white)

Устанавливает цвет тела цилиндра.

```

\begin{tikzpicture}[>=stealth, aspect=0.5]
\node [cylinder, cylinder uses custom fill,
       cylinder end fill=red!50,cylinder body fill=red!25] {Cylinder};
\end{tikzpicture}

```



19.3 Символьные формы

Символьные формы доступны после загрузки библиотеки `shapes.symbols`.

⊘ **Форма `forbidden sign`** (знак запрета) помещает узел в перечеркнутый круг. Круг — часть фона, диагональ — часть пути переднего плана; таким образом, диагональная линия находится выше текста. Форма наследует все якоря от формы `circle`.

```
\begin{tikzpicture}
\node [forbidden sign,line width=1ex,draw=red,
      fill=white] {Smoking};
\end{tikzpicture}
```



🔍 **Форма `magnifying glass`** (лупа) помещает узел в круг с ручкой, прикрепленной к узлу. Угол расположения ручки лупы и ее длину можно изменять. Форма наследует все якоря от формы `circle`.

`/pgf/magnifying glass handle angle fill=<degree>` (default -45)

Устанавливает угол поворота для ручки лупы.

`/pgf/magnifying glass handle angle aspect=<factor>` (default 1.5)

Устанавливает длину ручки лупы как кратное от радиуса круга.

```
\begin{tikzpicture}
\node [magnifying glass,line width=1ex,draw]
      {huge};
\end{tikzpicture}
```



☁ **Форма `cloud`** (облако) рисуется растянутой настолько, чтобы охватить содержимое узла (строго говоря, используя эллипс, содержащий узел, и включая `inner sep`).

```
\begin{tikzpicture}
\node[cloud,draw, fill=gray!20,aspect=2] {ABC};
\node[cloud,draw,fill=gray!20] at (1.5,0) {D};
\end{tikzpicture}
```



Облако можно рассматривать как эллипс, граница которого неравномерно «раздута».

`/pgf/cloud puffs=<integer>` (no default, initially 10)

Устанавливает число «вздутий» на облаке.

`/pgf/cloud puff arc=<angle>` (no default, initially 135)

Устанавливает длину дуги вздутия (в градусах).

Как и форма `diamond`, форма `cloud` также использует ключ `aspect`, определяющий отношение ширины и высоты облака. Но при некоторых обстоятельствах может быть нежелательным постоянно указывать это отношение для облака. Поэтому реализован ключ

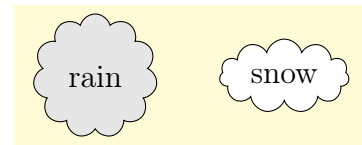
`/pgf/cloud ignores aspect=<boolean>` (default true)

Определяет игнорировать или нет ключ `aspect`. Начальное значение — false (ложь).

```

\begin{tikzpicture}
  [aspect=1, every node/.style={cloud,
    cloud puffs=11, draw}]
  \node [fill=gray!20] {rain};
  \node [cloud ignores aspect, fill=white]
    at (2.5,0) {snow};
\end{tikzpicture}

```

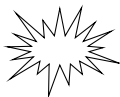
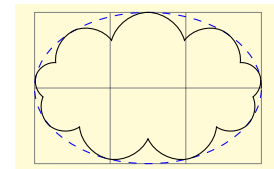


Любые требования к минимальному размеру применяются к эллипсу, который проходит через точки середины всех дуг вздутий. Эти требования рассматриваются только после применения любых других спецификаций аспекта.

```

\begin{tikzpicture}
\draw [help lines] grid (3,2);
\draw [blue, dashed]
  (1.5, 1) ellipse (1.5cm and 1cm);
\node [cloud,cloud puffs=9,draw,
  minimum width=3cm,minimum height=2cm] at (1.5, 1) {};
\end{tikzpicture}

```



Форма **starburst** представляет собой случайный взрыв эллиптической звезды, поддерживает вращение формы.

```

\begin{tikzpicture}
\node[starburst, fill=yellow, draw=red,
  line width=2pt] {\bf BANG!};
\end{tikzpicture}

```



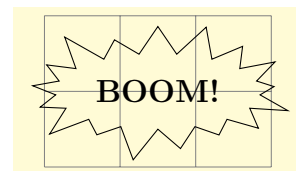
Как и форма `star`, форма `starburst` должны рассматриваться как имеющая набор внутренних точек и внешних точек. Внутренние точки лежат на эллипсе, который плотно облегает содержимое узла (включая `inner sep`). Используя значение опции `starburst point height` внешние точки генерируются случайным образом, получая высоту как случайное число между этим значением и одной четвертой этого значения. Для заданной формы `starburst` угол для каждой внешней точки фиксирован, и определяется числом точек, заданных для формы.

Важно отметить, что, в то время как возможная максимальная высота точки используется для расчета требуемой минимальной ширины или высоты, внешние точки генерируются случайным образом, так что (к сожалению) нет гарантий, что все подобные требования будут полностью удовлетворены.

```

\begin{tikzpicture}
\draw[help lines] grid(3,2);
\node[starburst,draw,minimum width=3cm,
      minimum height=2cm]
      at (1.5, 1) {\bf BOOM!};
\end{tikzpicture}

```



`/pgf/starburst points=<integer>` (no default, initially 17)

Устанавливает число точек для формы `starburst`.

`/pgf/starburst point height=<length>` (no default, initially .5cm)

Устанавливает максимальное расстояние между радиусом для внутренних точек и радиусом для внешних точек.

`/pgf/random starburst=<integer>` (no default, initially 100)

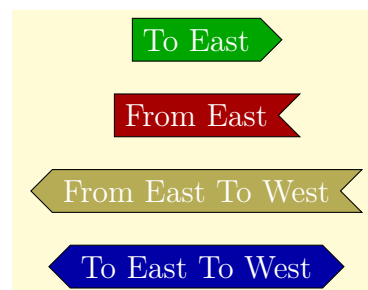
Устанавливает начальное число для генератора случайных чисел, чтобы создать форму `starburst`. Максимальное значение для `<integer>` — 16383. Если `<integer>`=0, генератор случайных чисел не используется, и для всех внешних точек используется максимальная высота. Если `<integer>` будет отсутствовать, то начальное число будет выбрано случайным образом.

□ **Форма `signal`** (сигнал) — прямоугольник, с произвольно выбираемой формой сторон. Сигнал может указывать (`to`) куда-то наружу «в этом направлении». Он может также указывать (`from`) внутрь «от этого направления».

```

\begin{tikzpicture}
[every node/.style={signal,draw,text=white,
                    signal to=nowhere}]
\node[fill=green!65!black,signal to=east]
      at (0,1) {To East};
\node[fill=red!65!black,signal from=east]
      at (0,0) {From East};
\node[fill=yellow!65!black, signal from=east,
      signal to=west]
      at (0,-1) {From East To West};
\node[fill=blue!65!black,
      signal to=east and west]
      at (0,-2) {To East To West};
\end{tikzpicture}

```



`/pgf/signal pointer angle=<angle>` (no default, initially 90)

Устанавливает угол для указывающей стороны формы. Этот угол поддерживается, при указании любого минимального размера, таким образом, любая настройка ширины формы будет воздействовать на ее высоту, и наоборот.


`/pgf/signal from=<direction> and <opposite direction>` (no default, initially nowhere)

Устанавливает, какие стороны принимают указатель внутрь (указывают на центр формы). Возможные значения `<direction>` и `<opposite direction>` — `north`, `south`, `east`, `west` (или `above`, `below`, `right`, `left`). Дополнительное ключевое слово `nowhere` (никуда) используется для того, чтобы очистить стороны от указателей.

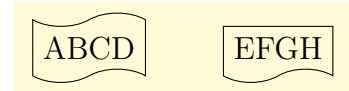
`/pgf/signal to=<direction> and <opposite direction>` (no default, initially east)

Устанавливает, какие стороны принимают указатель, направленный наружу.

Отметим, что pgf игнорирует любую команду, которая использует не противоположные друг другу направления (в том числе и для опций `signal to` и `signal from`). Так, например, нельзя получить сигнал с точкой, направленной наружу влево, и внутрь снизу.

 **Форма `tape`** (лента) — развивающийся на ветру прямоугольник, который плотно охватывает содержимое узла (включая `inner sep`).

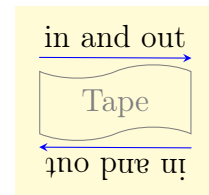
```
\begin{tikzpicture}
\node[tape, draw]{ABCD};
\node[tape, draw, tape bend top=none]
      at (2.5, 0) {EFGH};
\end{tikzpicture}
```



`/pgf/tape bend top=<bend style>` (no default, initially in and out)

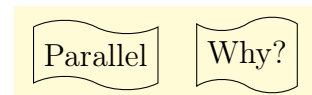
Определяет, как изгибается верхняя сторона. Параметр `<bend style>` может принимать значения `in and out`, `out and in` или `none` (то есть, прямая). Изгибающиеся стороны рисуются по часовой стрелке (верх, правая боковая сторона, низ, левая боковая сторона), используя (по умолчанию) стиль изгиба `in and out`, то есть сторона сначала изгибается внутрь, а затем изгибается во вне.

```
\begin{tikzpicture}[-stealth]
\node [tape,draw,gray,minimum width=2cm] (t){Tape};
\draw [blue] ([yshift=5pt] t.north west) --
           ([yshift=5pt]t.north east)
           node[midway,above, black] {in and out};
\draw [blue]([yshift=-5pt] t.south east) --
           ([yshift=-5pt]t.south west)
           node[sloped,allow upside down,midway, above,black] {in and out};
\end{tikzpicture}
```



Чтобы изгибающиеся стороны были параллельны, стороны должны использовать один и тот же стиль изгиба. Хотя можно для верхней и нижней стороны выбирать противоположные стили изгиба, но стоит ли это делать?

```
\begin{tikzpicture}
\tikzstyle{every node}=[tape, draw]
\node [tape bend top=out and in,
       tape bend bottom=out and in] {Parallel};
\node at (2,0) [tape bend bottom=out and in] {Why?};
\end{tikzpicture}
```



`/pgf/tape bend bottom=<bend style>` (no default, initially in and out)

Определяет, как изгибается нижняя сторона.

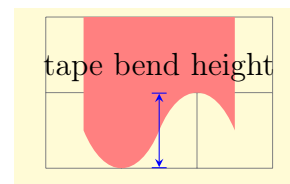
`/pgf//tape bend height=<length>` (no default, initially 5pt)

Устанавливает полную высоту для стороны с изгибом.


```

\begin{tikzpicture}[>=stealth]
\draw [help lines] grid(3,2);
\node [tape, fill,minimum size=2cm,red!50,
      tape bend top=none,tape bend height=1cm
      at (1.5,1.5) (t) {}];
\draw [|<->,blue] (1.5,0) -- (1.5,1)
      node [at end, above, black] {tape bend height};
\end{tikzpicture}

```



19.4 Формы в виде стрелок

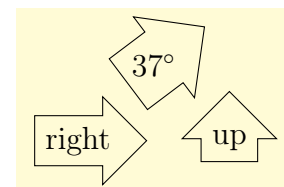
Библиотека `shapes.arrows` определяет формы в виде стрелок. Эти формы напоминают наконечники для стрелок, но сильно отличаются от (нормальных) наконечников. В частности, их *нельзя* использовать в качестве наконечников.

➡ **Форма `single arrow`** (одиночная стрелка) окружает содержимое узла (включая `inner sep`), поддерживает вращение. Угол вращения определяет направление, в котором указывает стрелка (если не применялись другие преобразования вращения).

```

\begin{tikzpicture}
[every node/.style={single arrow, draw},
 rotate border/.style={shape border rotate=#1,
                       shape border uses incircle}]
\node {right};
\node at (2,0) [shape border rotate=90]{up};
\node at (1,1) [rotate border=37,inner sep=0pt] {$37^\circ$};
\end{tikzpicture}

```

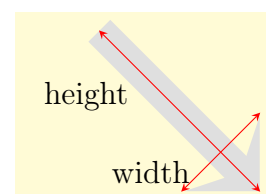


Независимо от вращения стрелки, ширина измеряется между выступами в голове стрелки, а высота измеряется от острия стрелки до конца хвоста стрелки.

```

\begin{tikzpicture} [>=stealth,rotate border/.style={
  shape border uses incircle,shape border rotate=#1}]
\node[rotate border=-45,fill=gray!25,single arrow,
      minimum height=3cm,single arrow head extend=.5cm,
      single arrow head indent=.25cm] (arrow) {};
\draw[red,<->] arrow.before tip) -- (arrow.after tip)
      node [near end,left,black] {width};
\draw[red,<->] (arrow.tip) -- (arrow.tail)
      node [near end, below left, black] {height};
\end{tikzpicture}

```



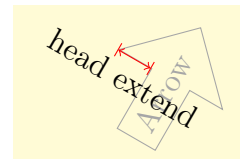
`/pgf/single arrow tip angle=<angle>` (no default, initially 90)

Устанавливает угол для внешнего выступа головы стрелки. Расширение стрелки до некоторой минимальной ширины может увеличить высоту формы, чтобы сохранить этот угол.

`/pgf/single arrow head extend=<length>` (no default, initially .5cm)

Устанавливает расстояние между хвостом стрелки и внешним выступом головы стрелки. Оно может измениться, если форма изменит свои размеры.

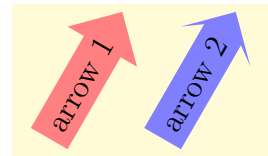
```
\begin{tikzpicture}
\node[single arrow, draw,single arrow head extend=.5cm,
      gray!70, rotate=60] (a) {Arrow};
\draw[red,|<->|] (a.before tip) -- (a.before head)
      node [midway, below, sloped, black] {head extend};
\end{tikzpicture}
```



`/pgf/single arrow head indent=<length>` (no default, initially 0cm)

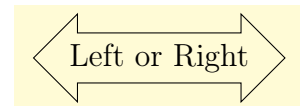
Перемещает на указанную длину вперед точку, в которой голова стрелки присоединяется к хвосту стрелки.

```
\begin{tikzpicture} [every node/.style={single arrow,
                                      draw=none, rotate=60}]
\node [fill=red!50] {arrow 1};
\node [fill=blue!50,
      single arrow head indent=1ex] at (1.5,0) {arrow 2};
\end{tikzpicture}
```



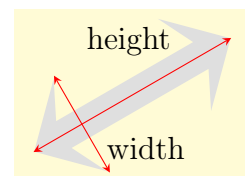
Форма `double arrow` (двойная стрелка) плотно окружает содержимое узла (включая `inner sep`) и поддерживает вращение формы.

```
\begin{tikzpicture}
  [every node/.style={double arrow, draw}]
\node [double arrow, draw] {Left or Right};
\end{tikzpicture}
```



Двойная стрелка ведет себя так же, как одиночная стрелка. Ее ширина равна расстоянию между внешними выступами головы стрелки, а высота — расстоянию между остриями.

```
\begin{tikzpicture}[>=stealth, rotate border/.style={
  shape border uses incircle, shape border rotate=#1}]
\node[rotate border=210, fill=gray!25,
      minimum height=3cm, double arrow,
      double arrow head extend=.5cm,
      double arrow head indent=.25cm] (arrow) {};
\draw[red,<->]
      (arrow.before tip 1) -- (arrow.after tip 1)
      node [near start, right, black] {width};
\draw[red, <->] (arrow.tip 1) -- (arrow.tip 2)
      node [near end, above left, black] {height};
\end{tikzpicture}
```



`/pgf/double arrow tip angle=<angle>` (no default, initially 90)

Устанавливает угол для внешнего выступа головы стрелки. Расширение стрелки до некоторой минимальной ширины может увеличить высоту, чтобы сохранить заданный угол.

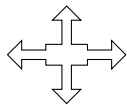
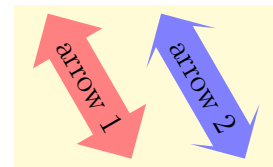
`/pgf/double arrow head extend=<length>` (no default, initially .5cm)

Устанавливает расстояние между хвостом стрелки и внешним выступом головы стрелки. Это расстояние может измениться, если форма изменит свои размеры.

`/pgf/double arrow head indent=<length>` (no default, initially 0cm)

Перемещает точку присоединения головы стрелки к хвосту стрелки вперед на расстояние `<length>`.

```
\begin{tikzpicture}[every node/.style={
  double arrow, draw=none, rotate=-60}]
\node [fill=red!50] {arrow 1};
\node [double arrow head indent=1ex,
  fill=blue!50] at (1.5,0) {arrow 2};
\end{tikzpicture}
```

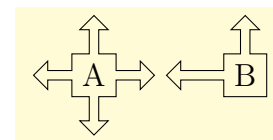


Форма **arrow box** представляет собой прямоугольник со стрелками, которые выходят из его четырех сторон.

`/pgf/arrow box arrows={<list>}` (no default)

Устанавливает расстояния, на которые от узла простираются все стрелки. Параметр `<list>` должен состоять из четырех ключевых слов `north`, `south`, `east`, `west`, отделенных запятыми (поэтому список должен помещаться в фигурные скобки!). Расстояния определяются для каждой стороны после двоеточия (например, `north:1cm`, или `west:5cm from center`). Если расстояние для элемента не определено, используется расстояние, определенное ранее (в начале обработки списка предполагается, что это 0cm, таким образом, для первого элемента в списке всегда должно определяться расстояние). Любые неопределенные стрелки не рисуются.

```
\begin{tikzpicture}
\node[arrow box, draw] {A};
\node[arrow box, draw,
  arrow box arrows={north:.5cm,west:0.75cm}]
  at (2,0) {B};
\end{tikzpicture}
```

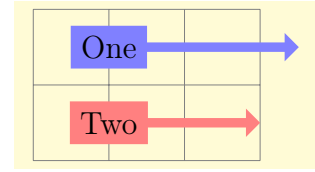


Любое требование минимального размера применяется только к основному прямоугольнику. Длину каждой из стрелок можно определять независимо или от соответствующей границы прямоугольника (значение по умолчанию), или от центра прямоугольника.

```

\begin{tikzpicture}
\tikzset{box/.style={arrow box, fill=#1}}
\draw [help lines] grid(3,2);
\node[box=blue!50, arrow box arrows={east:2cm}]
      at (1,1.5){One};
\node[box=red!50,
      arrow box arrows={east:2cm from center}] at (1,0.5){Two};
\end{tikzpicture}

```



`/pgf/arrow box tip angle=<angle>` (no default, initially 90)

Устанавливает угол для внешнего выступа наконечника всех четырех стрелок.

`/pgf/arrow box head extend=<length>` (no default, initially .125cm)

Устанавливает расстояние от тела стрелки до внешнего выступа головы стрелки, которое используется всеми стрелками.

`/pgf/arrow box head indent=<length>` (no default, initially 0cm)

Перемещает точку присоединения головы стрелки к телу стрелки, которое используется всеми стрелками.

`/pgf/arrow box shaft width=<length>` (no default, initially .125cm)

Устанавливает ширину тела для всех стрелок.

`/pgf/arrow box north arrow=<distance>` (no default, initially .5cm)

Устанавливает расстояние, на которое северная стрелка простирается от узла. По умолчанию — от границы формы, но при использовании ключа `from center`, расстояние будет измеряться от центра формы. Если `<distance>` равно 0pt или отрицательному числу, стрелка не рисуется.

Аналогично работают и следующие три опции.


`/pgf/arrow box south arrow=<distance>` (no default, initially .5cm)

`/pgf/arrow box east arrow=<distance>` (no default, initially .5cm)

`/pgf/arrow box west arrow=<distance>` (no default, initially .5cm)

19.5 Формы для нескольких текстовых частей

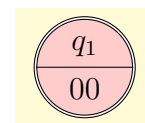
Библиотека `shapes.multipart` определяет общие формы, содержащие несколько (текстовых) частей.

 Форма **circle split** состоит из круга с горизонтальной разделяющей линией по середине. Верхняя часть — основная часть (часть `text`), нижняя часть — часть `lower`.

```

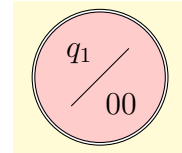
\begin{tikzpicture}
\node [circle split,draw,double,fill=red!20]
      { $q_1$ \nodepart{lower} $00$ };
\end{tikzpicture}


```



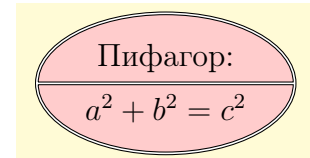
 Форма **circle solidus** подобна форме `circle split`, но две текстовые части разделяются диагональной линией.


```
\begin{tikzpicture}
\node [circle solidus,draw,double,fill=red!20]
  { $q_1$ \nodepart{lower} $00$ };
\end{tikzpicture}
```



 Форма **ellipse split** состоит из эллипса с горизонтальной разделяющей линией по середине. Верхняя часть — основная часть (`text`), нижняя часть — `lower`.

```
\begin{tikzpicture}
\node [ellipse split,draw,double,fill=red!20]
  { Пифагор: \nodepart{lower} $a^2+b^2=c^2$ };
\end{tikzpicture}
```



 Форма **rectangle split** — прямоугольник, который можно разбить по горизонтали или вертикали на несколько частей (по умолчанию на 4). Примеры ниже дают представление о некоторых основных опциях (см. детали в [1, р. 450–452]) этой формы.

```
\begin{tikzpicture}
[every text node part/.style={align=center}]
\node[rectangle split,rectangle split parts=3,
      draw, text width=3.2cm]
  {Student
   \nodepart{second} age:int\\ name:String
   \nodepart{third} getAge():int\\ getName():String};
\end{tikzpicture}
```

Student
age:int name:String
getAge():int getName():String

```
\begin{tikzpicture}
[every node/.style={draw, anchor=text,
  rectangle split, rectangle split parts=3}]
\node{text\nodepart{second}\nodepart{third}third};
\node[rectangle split ignore empty parts] at (2,0)
  {text \nodepart{second} \nodepart{third}third};
\end{tikzpicture}
```

text	text
	third
third	

```
\def\x{one \nodepart{second} 2
  \nodepart{third} 3 \nodepart{fourth} 4}
```

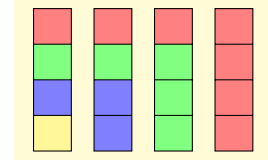
```
\begin{tikzpicture}[every node/.style={
rectangle split, rectangle split parts=4,draw}]
\node[rectangle split part align={
  center,left,right}] at (0,0) {\x};
\node[rectangle split part align={center,left}] at (1.3,0) {\x};
\node[rectangle split part align={center}] at (2.6,0) {\x};
\end{tikzpicture}
```

one	one	one
2	2	2
3	3	3
4	4	4

```

\begin{tikzpicture}
\tikzset{every node/.style={rectangle split,
draw, minimum width=.5cm}}
\node[rectangle split part fill={red!50,
green!50, blue!50, yellow!50}] {};
\node[rectangle split part fill={red!50,
green!50, blue!50}] at (0.8,0) {};
\node[rectangle split part fill={red!50, green!50}] at (1.6,0) {};
\node[rectangle split part fill={red!50}]
at (2.4,0) {};
\end{tikzpicture}

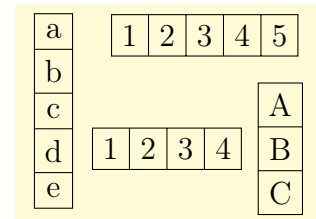
```



```

\begin{tikzpicture}
[my shape/.style={rectangle split,
rectangle split parts=#1,draw,anchor=center}]
\node [my shape=5] at (0,1)
{a\nodepart{two}b\nodepart{three}%
c\nodepart{four}d\nodepart{five}e};
\node [my shape=5,rectangle split horizontal]
at (2,2)
{1\nodepart{two}2\nodepart{three}%
3\nodepart{four}4\nodepart{five}5};
\node [my shape=3] at (3,0.5) {A\nodepart{two}B\nodepart{three}C};
\node[my shape=4,rectangle split horizontal] at (1.5,0.5)
{1\nodepart{two}2\nodepart{three}3\nodepart{four}4};
\end{tikzpicture}

```



19.6 Формы для меток-идентификаторов

Формы для меток-идентификаторов (выносные формы) становятся доступными после загрузки библиотеки `shapes.callouts`. После этого подсказки, указатели и другие информационные сообщения создать очень легко: надо только создать узел, а затем нарисовать пути от границы узла до нужной точки.

Формы такого вида состоят из основной формы (разного вида), и указателя (который является частью формы), указывающего на нечто в рисунке (или вне рисунка). Позиция на границе основной формы, с которой связан указатель, определяется автоматически. Однако, указатель игнорируется при вычислении минимального размера формы, а также при позиционировании якоря.

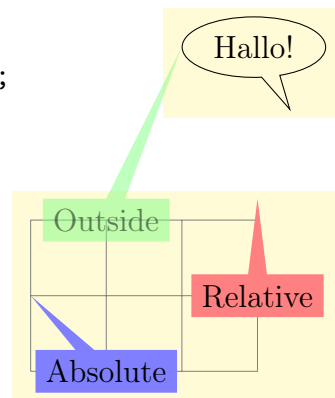
Есть два вида указателей: относительный указатель и абсолютный указатель. Относительный указатель вычисляет угол, определяемый координатой относительно центра основной формы, находит точку на границе, которой соответствует этот угол, а затем прибавляет координату к этой точке. Это означает, что предположения о размере основной формы не существенны: относительный указатель всегда будет снаружи. С другой стороны, абсолютный указатель, намного проще: он указывает в конкретную координату (и может даже указать на именованную координату в другом рисунке).

```

\begin{tikzpicture}[remember picture]
  \node[ellipse callout, draw] (hallo) {Hallo!};
\end{tikzpicture}

\begin{tikzpicture}[remember picture,
note/.style={rectangle callout, fill=#1}]
\draw [help lines] grid(3,2);
\node [note=red!50,
      callout relative pointer={(0,1)}]
      at (3,1) {Relative};
\node [note=blue!50,
      callout absolute pointer={(0,1)}] at (1,0) {Absolute};
\node [note=green!50,opacity=.5, overlay,
      callout absolute pointer={hallo.south}] at (1,2) {Outside};
\end{tikzpicture}

```



Следующие опции применимы, как правило, ко всем выносным формам. Но всегда нужно помнить, что опции со словами `relative` (относительный) `absolute` (абсолютный) имеют разный формат данных, в зависимости от того, используются ли они в `pgf` или `TikZ`.

`/pgf/callout relative pointer=<coordinate>` (no default, initially)

Устанавливает вектор указателя метки 'относительно' формы метки.

`/pgf/callout absolute pointer=<coordinate>` (no default)

Устанавливает вектор указателя метки абсолютно в указанную точку рисунка.

`/tikz/callout relative pointer=<coordinate>` (no default, initially (315:.5cm))

TikZ-версия опции `callout relative pointer`. Точка `coordinate` может определяться, используя TikZ-формат для задания координат.

`/tikz/callout absolute pointer=<coordinate>` (no default)

TikZ-версия опции `callout absolute pointer`. Точка `coordinate` может определяться, используя TikZ-формат для задания координат.

Можно сократить указатель на некоторую величину, используя следующую опцию:

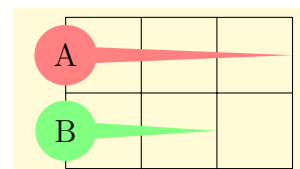
`/pgf/callout pointer shorten=<distance>` (no default, initially 0cm)

Переместить указатель метки к центру основной формы, используя `<distance>`.

```

\begin{tikzpicture}
\tikzset{callout/.style={ellipse callout,
      callout pointer arc=30,
      callout absolute pointer={#1}}}
\draw (0,0) grid (3,2);
\node[callout={(3,1.5)}, fill=red!50] at (0,1.5) {A};
\node[callout={(3,.5)}, fill=green!50,
      callout pointer shorten=1cm] at (0,.5) {B};
\end{tikzpicture}

```





Форма **rectangle callout** состоит из основной формы, которая является прямоугольником, плотно охватывающим содержимое узла (включая `inner sep`). Она поддерживает опции, описанные выше, и, дополнительно, опцию

`/pgf/callout pointer width=<length>` (no default, initially .25cm)

Устанавливает ширину указателя на границе прямоугольника.



Форма **ellipse callout** имеет основной формой эллипс, который плотно охватывает содержимое узла (включая `inner sep`). Она поддерживает опции

`absolute callout pointer`, `relative callout pointer`, `callout pointer shorten`,

описанные выше, и, дополнительно, следующую опцию:

`/pgf/callout pointer arc=<angle>` (no default, initially 15)

Устанавливает ширину указателя на границе эллипса равной дуге `<angle>`.



Форма **cloud callout** имеет основной формой облако, которое и содержит узел. Указатель сегментирован и состоит из нескольких уменьшающихся по размеру эллипсов (чтобы нарисовать форму облака, нужна библиотека `shapes.symbols`).

```
\begin{tikzpicture}
\node[cloud callout, cloud puffs=15,
      aspect=2.5, cloud puff arc=120,
      shading=ball,text=white] {\bf Imagine...};
\end{tikzpicture}
```



Форма `cloud callout` поддерживает опции

`absolute callout pointer`, `relative callout pointer`, `callout pointer shorten`,

описанные выше. Основная форма может изменяться, используя опции для формы `cloud`. Поддерживаются еще и следующие опции:

`/pgf/callout pointer start size=<value>` (no default, initially .2 of callout)

Устанавливает размер первого сегмента в указателе (то есть, сегмента, самого близкого к основной форме — облаку). Для `<value>` возможны три формы:

- единственная размерность (например, 5pt), в этом случае первый эллипс будет иметь диаметры в 5pt (будет кругом);
- две размерности (например, 10pt и 2.5pt), которые устанавливают *x*- и *y*- диаметры первого эллипса;
- десятичная дробь (например, .2 of callout), тогда *x*- и *y*- диаметры первого эллипса будут установлены как части ширины и высоты основной формы. Ключевое слово `of callout` нельзя опускать.

`/pgf/callout pointer end size=<value>` (no default, initially .1 of callout)

Устанавливает размер последнего эллипса в указателе.

`/pgf/callout pointer segments=<number>` (no default, initially 2)

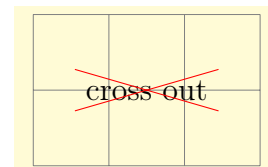
Устанавливает число сегментов в указателе. Если их определить много, pgf наложит их один на другой.

19.7 Библиотека специальных форм

Библиотека `shapes.misc` определяет универсальные формы для разных целей, которые не попали в предыдущие категории.

✂ **Форма `cross out`** перечеркивает узел. Ее передний план — две диагональные линии между углами ограничивающего прямоугольника узла. Форма `cross out` наследует все якоря от формы `rectangle`.

```
\begin{tikzpicture}
\draw [help lines] (0,0) grid (3,2);
\node [cross out,draw=red] at (1.5,1)
      {cross out};
\end{tikzpicture}
```



Полезное приложение — внутренний текст в следующем примере:

```
Cross \tikz[baseline]
      \node [cross out,draw,anchor=text] {me}; out!
```

Cross ~~me~~ out!

↘ **Форма `strike out`** идентична форме `cross out`, но состоит из одной линии от нижнего левого угла в верхний правый угол.

```
Strike \tikz[baseline]
       \node [strike out,draw,anchor=text] {me}; out!
```

Strike ~~me~~ out!

◻ **Форма `rounded rectangle`** — прямоугольник с произвольно скругленными сторонами. Есть ключи, позволяющие определить, как скругляются стороны.

```
\begin{tikzpicture}
\node[rounded rectangle, draw, fill=red!20] {Hallo};
\end{tikzpicture}
```

Hallo

`/pgf/rounded rectangle arc length=<angle>` (no default, initially 180)

Устанавливает длину дуг (в градусах) для скругления. Рекомендуемые значения для `<angle>` — между 90 и 180.

```
\begin{tikzpicture}
\matrix[row sep=5pt,
        every node/.style={draw, rounded rectangle}]{
\node[rounded rectangle arc length=180]{180}; \\
\node[rounded rectangle arc length=120]{120}; \\
\node[rounded rectangle arc length=90]{90}; \\
};
\end{tikzpicture}
```

180

120

90

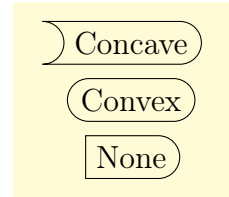
`/pgf/rounded rectangle west arc=<arc type>` (no default, initially convex)

Устанавливает стиль скругления для западной (левой) стороны. Разрешенные значения для параметра `<arc type>`: `concave`, `convex`, `none`.

```

\begin{tikzpicture}
\matrix[row sep=5pt,
  every node/.style={draw, rounded rectangle}]{
\node[rounded rectangle west arc=concave]{Concave}; \\
\node[rounded rectangle west arc=convex]{Convex}; \\
\node[rounded rectangle left arc=none]{None}; \\
}
\end{tikzpicture}

```



`/pgf/rounded rectangle left arc=<arc type>` (style, no default)


Альтернативная опция для определения левой (западной) дуги.

`/pgf/rounded rectangle east arc=<arc type>` (no default, initially convex)

Устанавливает стиль округления для восточной (правой) стороны.

`/pgf/rounded rectangle right arc=<arc type>` (style, no default)

Альтернативная опция для определения восточной дуги.

 Форма **chamfered rectangle** — прямоугольник с произвольно срезанными углами. Есть ключи, позволяющие определить, как их срезать и нарисовать.

```

\begin{tikzpicture}
\node[chamfered rectangle, white, fill=red,
  double=red, draw, very thick] {\bf STOP!};
\end{tikzpicture}

```



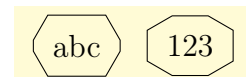
`/pgf/chamfered rectangle angle=<angle>` (no default, initially 45)

Устанавливает угол от вертикали для срезки углов.

```

\begin{tikzpicture}\tikzset{every node/.style={
  chamfered rectangle, draw}}
\node[chamfered rectangle angle=30]{abc};
\node[chamfered rectangle angle=60] at (1.5,0){123};
\end{tikzpicture}

```



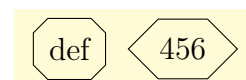
`/pgf/chamfered rectangle xsep=<length>` (no default, initially .666ex)

Устанавливает расстояние от угла узла (включая `inner sep`) по горизонтали до срезки. Если расстояние больше половины длины срезаемой стороны, оно игнорируется, а срезы проводятся так, чтобы они встретились в середине.

```

\begin{tikzpicture}\tikzset{every node/.style={
  chamfered rectangle, draw}}
\node[chamfered rectangle xsep=2pt]{def};
\node[chamfered rectangle xsep=2cm] at (1.5,0){456};
\end{tikzpicture}

```



`/pgf/chamfered rectangle ysep=<length>` (no default, initially .666ex)

Устанавливает расстояние от угла узла (включая `inner sep`) по вертикали до срезки. В остальном — аналогична предыдущей опции.

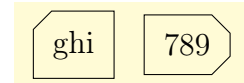
`/pgf/chamfered rectangle sep=<length>` (no default, initially `.666ex`)

Устанавливает расстояние от угла узла (включая `inner sep`) по вертикали и горизонтали до срезки. В остальном — аналогична предыдущим двум опциям.

`/pgf/chamfered rectangle corners={<list>}` (no default, initially `chamfer all`)

Определяет, какие углы срезать. Углы определяются по направлениям (`north east`, `north west`, `south west`, `south east`) и должны отделяться запятыми (так как углов несколько, список должен помещаться в фигурные скобки). Любые углы не указанные в списке автоматически не обрезаются. Разрешаются еще два дополнительных значения: `chamfer all` и `chamfer none`.

```
\begin{tikzpicture}\tikzset{every node/.style={
    chamfered rectangle, draw}}
\node[chamfered rectangle corners=north west] {ghi};
\node[chamfered rectangle corners={north east,
    south east}] at (1.5,0) {789};
\end{tikzpicture}
```



Глава 20

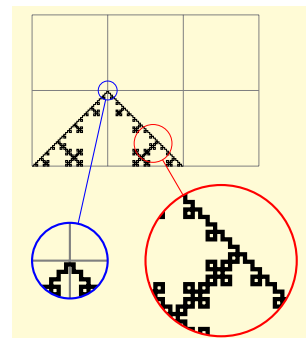
Библиотека `spy`

Библиотека определяет опции для создания изображений, в которых некоторая часть изображения повторяется в другой области в увеличенном виде (как будто ее просматривается через подзорную трубу (`spyglass`), отсюда и название библиотеки).

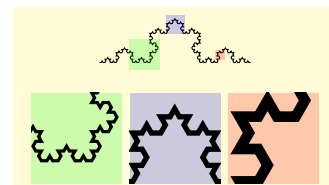
20.1 Увеличение части изображения

Идея библиотеки `spy` — легко создавать рисунки, в которых некоторая важная часть повторена в другом месте, но в увеличенном виде:

```
\begin{tikzpicture}[spy using outlines={circle,
    magnification=4, size=2cm, connect spies}]
\draw [help lines] (0,0) grid (3,2);
\draw [decoration=Koch curve type 1]
    decorate {decorate{decorate{decorate{
        (0,0) -- (2,0) }}}};
\spy [red] on (1.6,0.3)
    in node [left] at (3.5,-1.25);
\spy [blue, size=1cm] on (1,1)
    in node [right] at (0,-1.25);
\end{tikzpicture}
```



```
\begin{tikzpicture}
    [spy using overlays={size=12mm}]
\draw [decoration=Koch snowflake]
    decorate {decorate{decorate{decorate{
        (0,0) -- (2,0) }}}};
\spy [green,magnification=3] on (0.6,0.1) in node at (-0.3,-1);
\spy [blue,magnification=5] on (1,0.5) in node at (1,-1);
\spy [red,magnification=10] on (1.6,0.1) in node at (2.3,-1);
\end{tikzpicture}
```



Увеличение использует так называемое преобразование холста: увеличивается все, включая ширину линий и текст. Для того, чтобы сработало увеличение, рисунок должен рисоваться первый раз в нормальном виде, а второй раз — в увеличенном. Это позволяют сделать несколько опций и команд.

- Нужно дать знать TikZ, что рисунок (или только окружение) должен увеличиваться, добавляя в окружения `{scope}` и `{tikzpicture}` опцию `spy scope`, которая устанавливает другие опции, такие как `spy using outlines`.
- В таких окружениях можно использовать команду `\spy`, которая имеет специальный синтаксис и (в некоторый момент) создает два узла: один узел, отображающий увеличенный рисунок (называемый `spy-in node`) и второй узел, показывающей какая часть первоначального рисунка увеличивается (называемый `spy-on node`). Узел `spy-in node` — обычный узел, который может иметь любую форму или границу, и в нем можно использовать все возможности TikZ. Единственное его отличие от нормального узла то, что вместо некоторого текста он содержит увеличенную часть некоторого рисунка.

Команда `\spy` не создает узлы сразу. Создание узлов откладывается до конца того окружения `spy scope`, в котором используется команда `\spy`. Так сделано потому, что повторить целое окружение в узле, содержащем увеличенную версию целого рисунка, можно только тогда, когда этот узел уже создан.

Основной вопрос при увеличении части рисунка сводится к следующему: как определить, какую часть рисунка нужно увеличить (`spy-on node`) и где эту увеличенную часть нужно расположить (`spy-in node`). Есть два возможных пути:

1. Определить размер и позицию узла `spy-on node`, затем размер узла `spy-in node` определяется по размеру узла `spy-on node` и коэффициенту увеличения в этом случае можно определить, где разместить узел `spy-in node`, но нельзя узнать его размер).

2. Альтернативный вариант состоит в том, чтобы определить размер и позицию узла `spy-in node`. Затем, подобно первому случаю, неявно (автоматически) определяется размер узла `spy-on node`, и можно только указать, где должен размещаться узел `spy-on node`, но не его размер. Именно этот метод использует библиотека `spy`.

20.2 Окружение `spy scope`

`/tikz/spy scope=<options>` (default empty)

Опция используется только в окружении `{scope}` или в любой среде, создающей такое окружение (например, `{tikzpicture}`). Последствия применения опции таковы:

- сбрасывается множество графических параметров (цвет, тип линии, ...);
- сообщается TikZ, что содержимое окружения нужно сохранить внутри специального поля;
- определяется команда `\spy` так, что ее можно использовать в окружении.
- в конце окружения, создаются узлы, принадлежащие командам `\spy`, используемым в окружении;
- параметры `<options>` сохраняются внутренним образом. Каждый раз, когда используется команда `\spy`, будут использоваться параметры `<options>`;
- определяются три ключа, которые в окружении с `spy scope` являются просто полезными сокращениями:

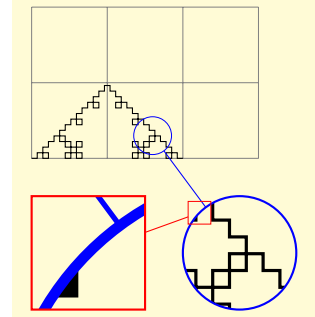
`/tikz/size=<dimension>` — сокращение для `minimum size`;

`/tikz/height=<dimension>` — сокращение для `minimum height`;

`/tikz/width=<dimension>` — сокращение для `minimum width`.

Допустимо вкладывать окружения с `spy scope`. В этом случае, все команды `\spy` во внутреннем окружении работают только на материале этого окружения, тогда как команды `\spy` вне внутренних окружений, но расположенные во внешнем окружении, позволяют работать с материалом обоих окружений — увеличивать увеличенное (вложенные `spy`-окружения — это вложенные области видимости).

```
\begin{tikzpicture}
[spy using outlines={rectangle, red,
  magnification=5,size=1.5cm, connect spies}]
\begin{scope}
[spy using outlines={circle, blue,
  magnification=3, size=1.5cm, connect spies}]
\draw [help lines] (0,0) grid (3,2);
\draw [decoration=Koch curve type 1]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy on (1.6,0.3) in node (zoom) [left] at (3.5,-1.25);
\end{scope}
\spy on (zoom.north west) in node [right] at (0,-1.25);
\end{tikzpicture}
```



20.3 Команда `spy`

`\spy[<options>]` on <coordinate> in node <node options>;

Используется только в окружении `spy scope`. Особенности ее синтаксиса таковы:

- команда `spy` не является частным случаем команды `\path`. Скорее, это небольшой синтаксический анализатор;
- после необязательного параметра <options> обязательно должно стоять `on`, сопровождаемое координатой. Эта координата определяет центр области, которая должна быть увеличена;
- после <coordinate> обязательно должно стоять `in node`, сопровождаемое некоторыми опциями <node options>. Синтаксис этих опций тот же, что и для обычной команды пути `node` (таких как `[left]` или `(foo) [red] at (bar)`). Однако параметры в <node options> не сопровождаются фигурными скобками: они должны следовать непосредственно за `in node` и заканчиваться точкой с запятой.

Результаты команды `spy`: до окончания текущего окружения `spy scope` параметры <options>, <coordinate> и <node options> сохраняются внутри окружения. Это означает, что, в частности, можно сослаться на любой узел в этом окружении, даже если он еще не был определен в момент задания команды `spy`. В конце текущего окружения `spy scope` создаются два узла `spy-in node` и `spy-on node`, причем узел `spy-in node` создается после узла `spy-on node` (и, следовательно, перекроет узел `spy-on node`, если они накладываются).

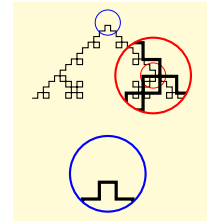
Когда создан узел `spy-in node`, в дополнение к действиям, вызванным параметрами <options> и опциями окружения `{spy scope}`, используются <node options>. Дополнительно, с каждым узлом `spy-in node` используется следующий стиль:

`/tikz/every spy in node`

(style, no value)

Позиция узла (опция `at`) устанавливается в `<coordinate>` по умолчанию так, чтобы он покрывал увеличиваемую область (но можно переопределить опцию `at`):

```
\begin{tikzpicture}[spy using outlines={
    circle, magnification=3, size=1cm}]
\draw [decoration=Koch curve type 1]
    decorate{decorate{decorate{(0,0) -- (2,0)}}};
\spy [red] on (1.6,0.3) in node;
\spy [blue] on (1,1) in node at (1,-1);
\end{tikzpicture}
```



Узел отображает не текст, а рисунок из текущего окружения `spy scope`, но преобразованный холстом согласно следующему ключу:

`/tikz/lens=<options>` (no default)

Параметр `<options>` должен содержать команды преобразования, такие как `scale` или `rotate`. Эти преобразования применяются к изображению, когда оно показывается в узле `spy-on node`. Так как самое общее преобразование — простое масштабирование, для него есть специальный стиль:

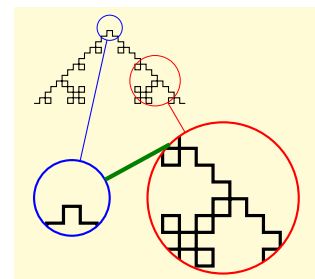
`/tikz/magnification=<number>` (no default)

Делает то же, что и `lens={scale=<number>}`.

Обычно размер узла определяются так, чтобы охватывать «текст» узла. Но для узла `spy-on node` это плохо, так как «текст» этого узла содержит целый рисунок. А TikZ действует так, как будто «текст» узла имеет нулевой размер. Нужно использовать опцию, такую как `minimum size`, чтобы заставить узел иметь определенный размер. Напомним, что опция `size` — сокращение для опции `minimum size` в окружении `spy scope`.

Узлу `spy-in node` можно обычным образом дать имя. Узел `spy-in node` всегда именуется автоматически, как `tikzspyinnode`. В `spy scope` можно использовать этот узел так же, как и любой другой:

```
\begin{tikzpicture}
\begin{scope}[spy using outlines={circle,
    magnification=3, size=2cm, connect spies}]
\draw [decoration=Koch curve type 1]
    decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy [red] on (1.6,0.3) in node (a) [left]
    at (3.5,-1.25);
\spy [blue, size=1cm] on (1,1)
    in node (b) [right] at (0,-1.25);
\end{scope}
\draw [ultra thick, green!50!black] (b) -- (a.north west);
\end{tikzpicture}
```



Как только оба узла созданы, чтобы связать их, используется текущее значение следующего ключа:

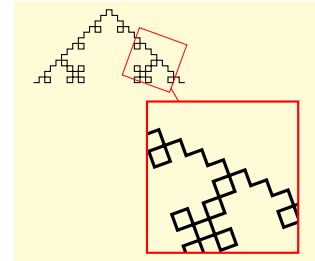
`/tikz/spy connection path=<code>` (no default, initially empty)

В `<code>` к этим узлам можно обращаться как к `tikzspyinnode` и `tikzspyonnode`. Например, стиль `connect spies` (см. далее) устанавливает следующую команду

```
\draw[thin] (tikzspyonnode) -- (tikzspyinnode);
```

Возвратимся к узлу `spy-in node`: этот узел центрируется по `<coordinate>` (якорь узла устанавливается в `center`, а опция `at` — в `<coordinate>`). Его размер и форма первоначально определены такими же, как размер и форма узла `spy-on node` (если, конечно, они явно не переопределены). Затем, дополнительно, применяется преобразование инвертирования, определенное опцией `lens`, в результате приводя узел к размеру и форме точно соответствующей области рисунка, которая отображается в узле `spy-on node`.

```
\begin{tikzpicture}[spy using outlines={lens={
  scale=3,rotate=20}, size=2cm, connect spies}]
\draw [decoration=Koch curve type 1]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy [red] on (1.6,0.3) in node at (2.5,-1.25);
\end{tikzpicture}
```



Как и для узлов `spy-in node`, есть стиль и для узлов `spy-on node` (его имя по умолчанию, как уже отмечалось, `tikzspyonnode`):

```
/tikz/every spy on node (style, no value)
```

Если есть много узлов `spy-on node` и нужно к ним обращаться, следует использовать опцию `name` внутри стиля `every spy on node`. Параметры `inner sep` и `outer sep` в узлах `spy-in node` и `spy-on node` устанавливаются равными `0pt`.

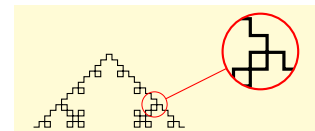
20.4 Предопределенные spy-стили

Есть предопределенные `spy`-стили, которые делают использование библиотеки `spy` проще. Следующие два стиля могут использоваться вместо `spy scope`, но реально передают параметры `<options>` непосредственно `spy scope`. Они дополнительно устанавливают нужные графические стили, используемые в обоих узлах.

```
/tikz/spy using outlines=<options> (default empty)
```

Создает окружение `spy scope`, в котором узел `spy-in node` рисуется, используя толстую линию, но не заполняется; а узел `spy-on node` рисуется, используя очень тонкую линию, и тоже не заполняется.

```
\begin{tikzpicture}[spy using outlines={circle,
  magnification=3, size=1cm, connect spies}]
\draw [decoration=Koch curve type 1]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy [red] on (1.6,0.3) in node at (3,1);
\end{tikzpicture}
```



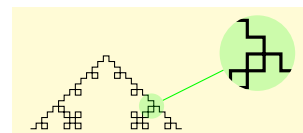
```
/tikz/spy using overlays=<options> (default empty)
```

Создает окружение `spy scope`, в котором узлы заполняются, но заполнение непрозрачно на 20%.

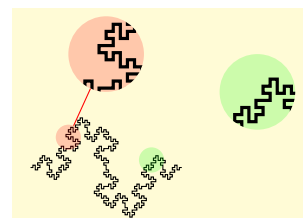
```
/tikz/connect spies (no value))
```

Стиль позволяет связать оба узла тонкой прямой линией:


```
\begin{tikzpicture}[spy using overlays={circle,
  magnification=3, size=1cm, connect spies}]
\draw [decoration=Koch curve type 1]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy [green] on (1.6,0.3) in node at (3,1);
\end{tikzpicture}
```



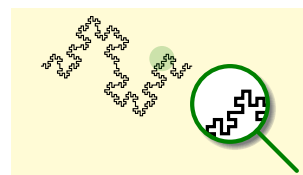
```
\begin{tikzpicture}[spy using overlays={circle,
  magnification=3, size=1cm}]
\draw [decoration=Koch curve type 2]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy [green] on (1.6,0.1) in node at (3,1);
\spy [red,connect spies] on (0.5,0.4)
  in node at (1,1.5);
\end{tikzpicture}
```



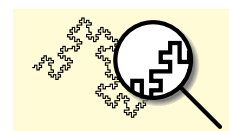
20.5 Примеры

Обычно, узлы `spy-in node` и `spy-on node` должны иметь одну и ту же форму. Однако можно использовать и разные, например, форму `circle` для узла `spy-on node` и `magnifying glass` для узла `spy-in node`, можно даже поместить лупу на рисунок:

```
\tikzset{spy using mag glass/.style={spy scope={
  every spy on node/.style={circle,fill,
    fill opacity=0.2, text opacity=1},
  every spy in node/.style={magnifying glass,
    circular drop shadow,fill=white, draw,
    ultra thick, cap=round}, #1}}}
\begin{tikzpicture}[spy using mag glass={magnification=3,size=1cm}]
\draw [decoration=Koch curve type 2]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy [green!50!black] on (1.6,0.1) in node at (2.5,-0.5);
\end{tikzpicture}
```



```
\begin{tikzpicture}
[spy scope={magnification=4,size=1cm},
  every spy in node/.style={magnifying glass,
  ultra thick,fill=white,circular drop shadow,
  draw, cap=round}]
\draw [decoration=Koch curve type 2]
  decorate{decorate{decorate{(0,0) -- (2,0) }}};
\spy on (1.6,0.1) in node;
\end{tikzpicture}
```



Глава 21

Библиотека `topaths`

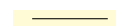
Библиотека `topaths` определяет операции `to`, которые используются в пути. Например, можно сказать `to [loop]`, чтобы добавить цикл в узел. Библиотека загружается TikZ автоматически, поэтому ее не нужно загружать явно в преамбуле документа.

21.1 Прямые линии

Следующий стиль устанавливает путь в виде прямой линии от начальной координаты до целевой (конечной). Он может использоваться не только с операцией `to`, но и с операцией `edge`.

`/tikz/line to` (style, no value)

```
\tikz {\draw (0,0) to[line to] (1,0);}
```

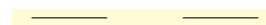


21.2 Перемещение

Следующий стиль устанавливает путь, который просто «перепрыгивает» в целевую точку. Он также может использоваться не только с операцией `to`, но и с `edge`.

`/tikz/move to` (style, no value)

```
\tikz \draw (0,0) to[line to] (1,0)
to[move to] (2,0) to[line to] (3,0);
```



21.3 Кривые

Стиль `curve to` рисует путь в виде кривой. Точный вид кривой определяют опции.

`/tikz/curve to` (style, no value)

Кривая покидает начальную координату под углом, который определяется опцией `out`, и достигает целевой координаты под другим углом, который определяется опцией `in`. Контрольные точки кривой устанавливаются на расстоянии, которое вычисляется различными способами в зависимости от определенных опций.

Все следующие опции неявно вызывают стиль `curve to`.

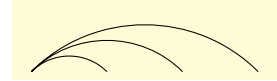
`/tikz/out=<angle>` (no default)

Определяет угол, под которым кривая покидает начальную координату. Если начальная координата — узел, начальная координата — точка на границе узла, расположенная в заданном `<angle>` направлении. Контрольная точка будет, таким образом, лежать на определенном расстоянии в направлении `<angle>` от начальной координаты.

`/tikz/in=<angle>` (no default)

Определяет угол, под которым кривая достигает целевой координаты.

```
\begin{tikzpicture}[out=45,in=135]
  \draw (0,0) to (1,0) (0,0) to (2,0) (0,0) to (3,0);
\end{tikzpicture}
```

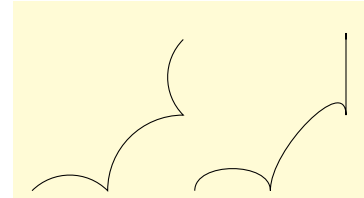


`/tikz/relative=<true or false>` (default true)

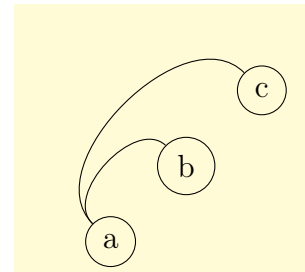
Указывает TikZ, должны ли углы `in` и `out` рассматриваться как абсолютные или как относительные. Абсолютный угол, угол по отношению к краю бумаги (если, конечно, не были установлены другие преобразования). Относительный угол, измеряется относительно прямой линии, проходящей от начальной в целевую координату.

```
\begin{tikzpicture}[out=45,in=135,relative]
  \draw (0,0) to (1,0) to (2,1) to (2,2);
```

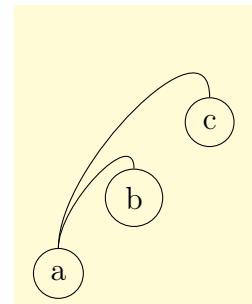
```
\begin{tikzpicture}[out=90,in=90,relative=false]
  \draw (0,0) to (1,0) to (2,1) to (2,2);
\end{tikzpicture}
```



```
\begin{tikzpicture}[out=90,in=90,relative]
  \node [circle,draw] (a) at (0,0) {a};
  \node [circle,draw] (b) at (1,1) {b};
  \node [circle,draw] (c) at (2,2) {c};
  \path (a) edge (b) edge (c);
\end{tikzpicture}
```



```
\begin{tikzpicture}[out=90,in=90,relative=false]
  \node [circle,draw] (a) at (0,0) {a};
  \node [circle,draw] (b) at (1,1) {b};
  \node [circle,draw] (c) at (2,2) {c};
  \path (a) edge (b) edge (c);
\end{tikzpicture}
```



`/tikz/bend left=<angle>` (default last value)

Устанавливает `out=<angle>`, `in=180 - <angle>` как относительные углы. Если угол `<angle>` не задается, используется последний заданный угол в опциях `bend left` или `bend right`.

`/tikz/bend right=<angle>` (default last value)

Работает как и опция `bend left`, только для другой стороны.

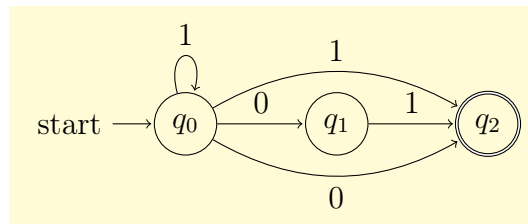
`/tikz/looseness=<number>` (no default, initially 1)

Определяет число, которое характеризует степень «свободы» («провисания») кривой. Происходит следующее: TikZ вычисляет расстояние между начальной и целевой координатами (если начальная и/или целевая координата — узел, расстояние вычисляется между точками на их границе). Это расстояние затем умножается на фиксированный множитель и на коэффициент `<number>`. Результирующее расстояние, скажем d , используется как расстояние до контрольных точек от начальной и целевой координаты. Фиксированный множитель выбирается так, чтобы, если `<number>` равно 1 и если углы `in` и `out` отличаются на 90° , то результат — четверть круга.

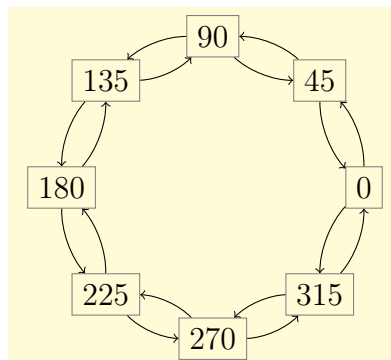
```
\tikz \draw (0,0) to [out=0,in=-90,looseness=1] (1,1);
\tikz \draw (0,0) to [out=0,in=-90,looseness=0.5] (1,1);
\tikz \draw (0,0) to [out=0,in=-90,looseness=0.2] (1,1);
```



```
\begin{tikzpicture}[shorten >=1pt,node distance=2cm,on grid]
  \node[state,initial] (q_0) {$q_0$};
  \node[state] (q_1) [right=of q_0] {$q_1$};
  \node[state,accepting] (q_2) [right=of q_1] {$q_2$};
  \path[->] (q_0) edge node [above] {0} (q_1)
             edge [loop above] node {1} ()
             edge [bend left] node [above] {1} (q_2)
             edge [bend right] node [below] {0} (q_2)
             (q_1) edge node [above] {1} (q_2);
\end{tikzpicture}
```



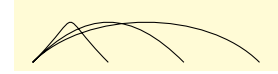
```
\begin{tikzpicture}
\foreach \angle in {0,45,...,315}
  \node [rectangle,draw=black!50] (\angle) at (\angle:2) {\angle};
\foreach \from/\to in {0/45,45/90,90/135,135/180,
  180/225,225/270,270/315,315/0}
  \path (\from) edge [->,bend right=22,looseness=0.8] (\to)
          edge [<-,bend left=22,looseness=0.8] (\to);
\end{tikzpicture}
```



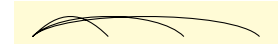
`/tikz/distance=<distance>` (no default)

Устанавливает расстояние d до контрольных точек равным `<distance>`. Это означает, что любое другое значение расстояния игнорируется.

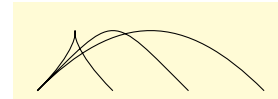
```
\begin{tikzpicture}[out=45,in=135, distance=1cm]
\draw (0,0) to (1,0) (0,0) to (2,0) (0,0) to (3,0);
\end{tikzpicture}
```



```
\begin{tikzpicture}[out=45,in=135, distance=.5cm]
\draw (0,0) to (1,0) (0,0) to (2,0) (0,0) to (3,0);
\end{tikzpicture}
```



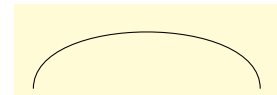
```
\begin{tikzpicture}[out=45,in=135, distance=1.5cm]
\draw (0,0) to (1,0) (0,0) to (2,0) (0,0) to (3,0);
\end{tikzpicture}
```



`/tikz/control=<coordinate>and<coordinate>` (no default)

Заставляет использовать в качестве контрольных точек две указанные координаты.

```
\tikz \draw (0,0) to
[controls=+(90:1) and +(90:1)] (3,0);
```



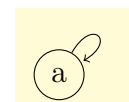
Библиотека определяет множество других ключей, которые позволяют устанавливать контрольные точки, расстояния до них, степень провисания кривой только для стартовой или целевой точки кривой (см. [1, pp. 471–472]).

21.4 Циклы

`/tikz/loop` (style, no value)

Подобен стилю `curve to`, но отличается следующим: первое, указанная целевая координата игнорируется, и начальная координата используется как целевая; второе, опция `looseness` устанавливается равной 8, а `min distance` — 5mm. Эти параметры приводят к хорошим циклам, когда угол, равный разности между углами `in` и `out`, составляет 30° .

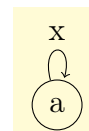
```
\begin{tikzpicture}
\node[circle,draw] {a} edge[in=30,out=60,loop] ();
\end{tikzpicture}
```



`/tikz/loop above` (style, no value)

Устанавливает стиль `loop` и устанавливает углы `in` и `out` так, что цикл располагается выше узла, и, кроме того, правильно располагает метку узла.

```
\begin{tikzpicture}
\node[circle,draw] {a} edge[loop above] node {x} ();
\end{tikzpicture}
```

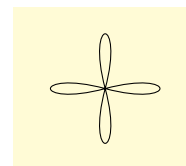


Аналогично предыдущему стилю работают следующие стили.

<code>/tikz/loop below</code>	(style, no value)
<code>/tikz/loop left</code>	(style, no value)
<code>/tikz/loop right</code>	(style, no value)
<code>/tikz/every loop</code>	(style, initially ->,shorten >=1pt)

Последний из перечисленных стилей устанавливается в начале каждого цикла.

```
\begin{tikzpicture}
[every loop/.style={}, scale=2]
\draw (0,0) to [loop above] () to [loop right] ()
to [loop below] () to [loop left] ();
\end{tikzpicture}
```



Глава 22

Библиотека `through`

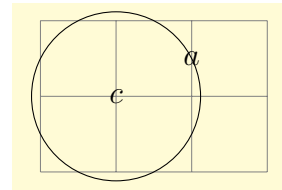
Библиотека `through` определяет опции для создания форм, проходящих через заданные точки.

`/tikz/circle through=<coordinate>` (no default)

Когда эта опция задается как опция узла, происходит следующее:

1. Параметры `inner sep` и `outer sep` устанавливаются равными нулю.
2. Форма определяется как `circle`.
3. Параметр `minimum size` устанавливается таким, что окружность вблизи центра узла (которая определяется, используя `at`), проходит через `<coordinate>`.

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\node (a) at (2,1.5) {$a$};
\node [draw] at (1,1)
  [circle through={(a)}] {$c$};
\end{tikzpicture}
```



Глава 23

Библиотека trees

Библиотека `trees` определяет новые стили и функции роста, полезные при создании деревьев.

23.1 Функции роста

`/tikz/grow via three points=one child at (<x>) and two children at (<y>) and (<z>)`

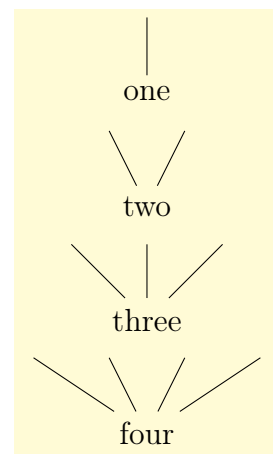
Устанавливает функцию роста, которая работает следующим образом. Если родительский узел имеет только одного наследника, то этот наследник размещается в (<x>). Если родительский узел имеет двух наследников, они размещаются в (<y>) и (<z>). Если родительский узел имеет более двух наследников, наследники размещаются в точках, которые линейно экстраполируются из трех точек (<x>), (<y>) и (<z>):

$$x + \frac{n-1}{2}(y-x) + (c-1)(z-y),$$

где n — число наследников, а c — номер текущего наследника (начиная с 1).

Если предполагается линейное расположение узлов, то, используя эту функцию роста, можно получить разумное размещение любого числа наследников. Приведем несколько примеров размещения, основанные на этой функции роста дерева.

```
\begin{tikzpicture}[grow via three points={%
  one child at (0,1) and two children at (-.5,1)
  and (.5,1)}]
\node at (0,0) {one} child;
\node at (0,-1.5) {two} child child;
\node at (0,-3) {three} child child child;
\node at (0,-4.5) {four} child child child child;
\end{tikzpicture}
```

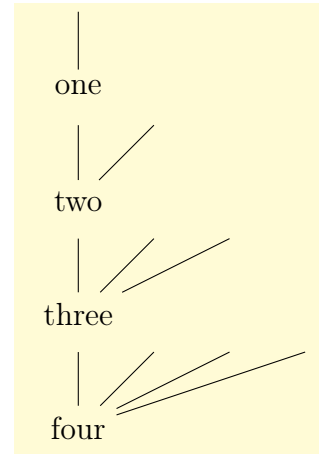


В следующем примере наследники размещаются выше, а дерево растет вправо.


```

\begin{tikzpicture}[grow via three points={%
  one child at (0,1) and two children at (0,1)
  and (1,1)}]
\node at (0,0) {one} child;
\node at (0,-1.5) {two} child child;
\node at (0,-3) {three} child child child;
\node at (0,-4.5) {four} child child
  child child;
\end{tikzpicture}

```

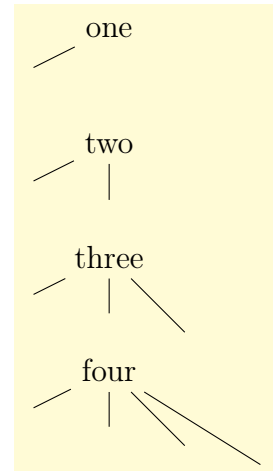


В последнем примере наследники размещаются по линии, уходящей вниз и вправо.

```

\begin{tikzpicture}[grow via three points={%
  one child at (-1,-.5)
  and two children at (-1,-.5) and (0,-.75)}]
\node at (0,0) {one} child;
\node at (0,-1.5) {two} child child;
\node at (0,-3) {three} child child child;
\node at (0,-4.5) {four} child child child child;
\end{tikzpicture}

```



Эти примеры должны прояснить, как можно создать новые стили, чтобы упорядочить дочерние узлы по линиям.

`/tikz/grow cyclic`

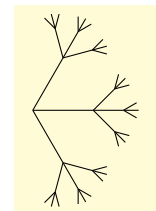
(style, no value)

Размещает наследников в точках окружности на расстоянии `\tikzleveldistance` от коренного узла. Вместо того чтобы использовать для размещения узлов одного уровня расстояние (опцию `sibling distance`), используется опция `/tikz/sibling angle=<angle>`, которая определяет угол между двумя наследниками одного уровня.

```

\begin{tikzpicture}[grow cyclic,
  level 1/.style={level distance=8mm,sibling angle=60},
  level 2/.style={level distance=4mm,sibling angle=45},
  level 3/.style={level distance=2mm,sibling angle=30}]
\coordinate [rotate=-90] % вниз
  child foreach \x in {1,2,3}{child foreach \x in {1,2,3}
    {child foreach \x in {1,2,3}}};
\end{tikzpicture}

```



`/tikz/clockwise from=<angle>`

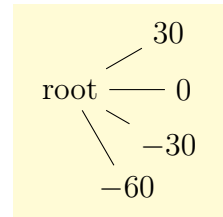
(no default)

Размещает наследников по окружности по правилу: первый размещается на расстоянии `\tikzleveldistance` от корневого узла под углом `<angle>`, второй — на том же расстоянии под углом `<angle>` - `\tikzsiblingangle`, третий — еще смещается на угол `\tikzsiblingangle` по часовой стрелке,

```

\begin{tikzpicture}
\node {root}[clockwise from=30,sibling angle=30]
  child {node {$30$}} child {node {$0$}}
  child {node {$-30$}} child {node {$-60$}};
\end{tikzpicture}

```



`/tikz/counterclockwise from=<angle>` (no default)

Работает аналогично опции `clockwise from`, но производит размещение наследников против часовой стрелки, то есть углы прибавляются, а не вычитаются.

23.2 Дуги от корневого узла

Следующие стили могут использоваться для того, чтобы изменить то, как рисуются дуги, идущие от корневого узла.

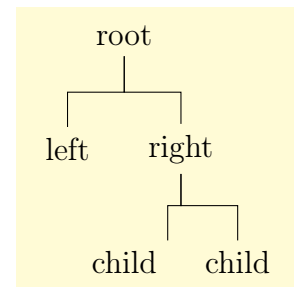
`/tikz/edge from parent fork down` (style, no value)

Чертит линию от корневого узла вниз (до половины расстояния), а затем до наследников, используя только горизонтальные и вертикальные линии.

```

\begin{tikzpicture}
\node {root}[edge from parent fork down]
  child {node {left}}
  child {node {right}}
  child[child anchor=north east] {node {child}}
  child {node {child}}};
\end{tikzpicture}

```



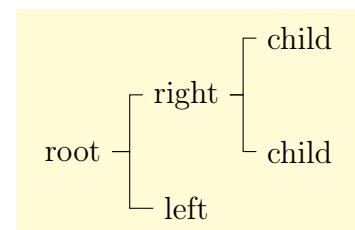
`/tikz/edge from parent fork right` (style, no value)

Стиль ведет себя так же, как предыдущий, но рисует дерево слева направо.

```

\begin{tikzpicture}
\node {root}
[edge from parent fork right, grow=right]
  child {node {left}}
  child {node {right}}
  child {node {child}}
  child {node {child}}};
\end{tikzpicture}

```



Следующие два стили аналогичны предыдущим.

`/tikz/edge from parent fork left` (style, no value)

`/tikz/edge from parent fork up` (style, no value)

Глава 24

Библиотека turtle

Небольшая библиотека `turtle` определяет опции, позволяющие создавать простую черепашую графику в традициях языка программирования Logo. Эти команды главным образом предназначены для развлечений, но могут использоваться и для дела.

```
\tikz[turtle/distance=6mm]
\draw [turtle={home,forward,right,forward,
              left,forward,left,forward}];
```



Для команд `forward`, `back`, `left`, `right` существуют соответствующие сокращения: `fd`, `bk`, `lt`, `rt`, при этом повороты совершаются на 90° .

Ключи для `turtle` очень похожи на опции, но внутренне они используют опцию `insert path`, чтобы построить путь.

Основная модель рисования, скрывающаяся за черепашей графикой с ее командами очень проста: есть (виртуальная) черепаха, которая ползает по странице и таким образом рисует путь. Черепаха всегда повернута головой в определенном направлении. Когда черепаха перемещается вперед, путь расширяется в этом направлении; повороты черепахи только изменяют направление.

Черепаха всегда двигается относительно последней текущей точки пути, и можно смешивать обычные команды пути с командами для черепахи. Однако направление перемещения черепахи управляется независимо от других команд пути.

`/tikz/turtle=<keys>` (no default)

Заставляет черепаху выполнить ключи из `<keys>`.

`/tikz/turtle/home` (no value)

Размещает черепаху в начале координат с направлением вверх.

`/tikz/turtle/forward=<distance>`

Заставляет черепаху продвинуться на заданное расстояние `<distance>`. Если расстояние не определено, используется текущее значение ключа

`/tikz/turtle/distance=<distance>` (no default, initially 1cm)

Определяет расстояние по умолчанию, на которое перемещается черепаха.

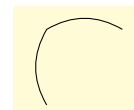
`/tikz/turtle/back=<distance>`

Аналогична команде `forward=-<distance>`.

`/tikz/turtle/how` (style, no value)

Стиль может устанавливаться в пути, используемом черепахой. Изменяя этот стиль, можно изменить вид пути.

```
\tikz \draw [turtle={how/.style={bend left},
home,forward,right,forward}];
```



```
/tikz/turtle/left=<angle>
```

(default 90)

Заставляет черепаху повернуть налево на заданный угол.

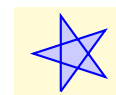
```
/tikz/turtle/right=<angle>
```

(default 90)

Заставляет черепаху повернуть направо на заданный угол.

Черепахью графику можно сочетать с утверждением `\foreach`:

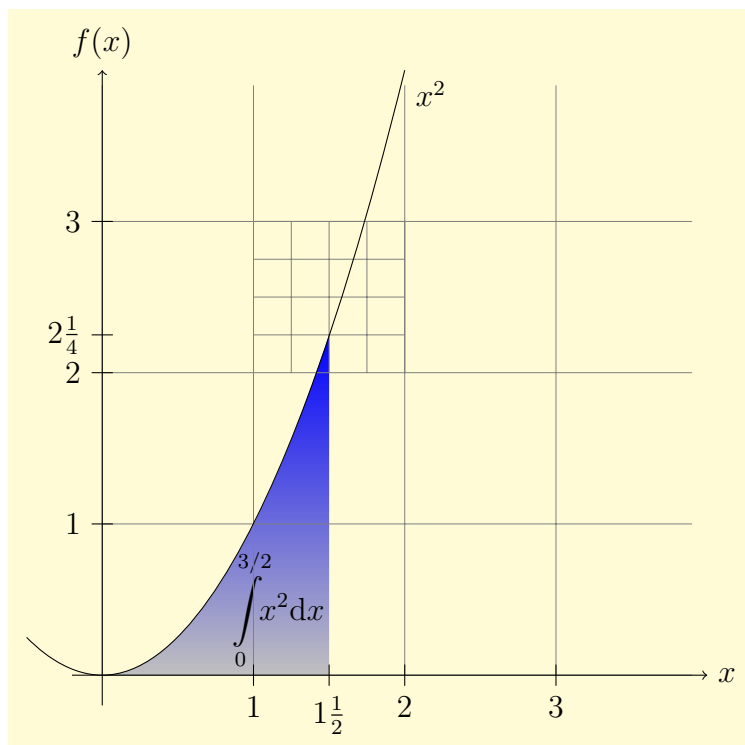
```
\tikz \filldraw [thick,blue,fill=blue!20]
[turtle=home]
\foreach \i in {1,...,5}
{[turtle={forward,right=144}]};
```



Часть II

Некоторые утилиты

Сервисные пакеты (утилиты) не используются напрямую при создании графики, но могут оказаться полезными и сами по себе. Все они либо напрямую зависят от PGF, либо предназначены для совместной работы с PGF и TikZ, даже если могут использоваться автономно.



Глава 25

Пакет pgffor

Пакет `pgffor` загружается автоматически TikZ, но не pgf, где его нужно загрузить явно (`\usepackage{pgffor}`). В нем определены две команды: `\foreach` и `\breakforeach`.
`\foreach`<variables>[<options>] in <list> <commands>

Синтаксис этой команды сложен. Рассмотрим его постепенно. В самом простом случае, <variables> — единственная TeX-команда, такая как `\x` или `\point`, а <options> может отсутствовать. В рассматриваемом самом простом случае, <list> — или любой разделяемый запятыми список значений, окруженных фигурными скобками, или это имя макроса, который содержит такой список. В качестве значений может использоваться все что угодно, но наиболее вероятно числовая информация. Наконец, <commands> — TeX-текст, заключенный в фигурные скобки.

При всех этих предположениях, выражение `\foreach` будет выполнять команду <commands> многократно, по одному разу для каждого элемента из <list>. Каждый раз, когда <commands> выполняется, переменная <variable> будет принимать текущее значение элемента из списка.

```
\foreach \x in {1,2,3,0} {[\x]}
```

```
[1][2][3][0]
```

```
\def\mylist{1,2,3,0}  
\foreach \x in \mylist {[\x]}
```

```
[1][2][3][0]
```

Заметим, что при каждом выполнении <commands>, <commands> помещается в TeX-группу. Это означает, что локальные изменения в счетчиках внутри <commands> не сохраняются до следующей итерации. Например, если добавить 1 к счетчику внутри <commands> локально, то в следующей итерации, счетчик будет иметь то же самое значение, которое он имел в начале первой итерации. Следует добавить опцию `\global`, если нужно сохранять изменения от итерации к итерации.

Синтаксис команд. Рассмотрим более сложную ситуацию. Первое усложнение происходит тогда, когда <commands> не является некоторым текстом в фигурных скобках. Если выражение `\foreach` не сталкивается с открывающей фигурной скобкой, оно вместо этого прочитывает все до следующей точки с запятой и использует прочитанное в качестве <commands>. Это полезно в ситуации, подобной следующей:

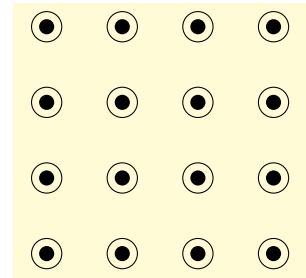
```
\tikz \foreach \x in {0,1,2,3}  
      \draw (\x,0) circle (0.2cm);
```

```
○ ○ ○ ○
```

Однако, «прочтение до следующей точки с запятой», не вся правда. Есть другое

правило: если выражение `\foreach` непосредственно сопровождается вторым выражением `\foreach`, второе выражение собирается как `<commands>`. Это позволяет написать следующий код:

```
\begin{tikzpicture}
\foreach \x in {0,1,2,3}
  \foreach \y in {0,1,2,3}
    { \draw (\x,\y) circle (0.2cm);
      \fill (\x,\y) circle (0.1cm); }
\end{tikzpicture}
```



Точечная нотация. Второе усложнение связано со списком `<list>`. Если этот список содержит в качестве элемента списка многоточие `...`, этот элемент заменяется «предполагаемыми значениями». Более точно, вот что происходит.

Обычно, когда обрабатывается элемент списка `...`, уже должны быть заданы два элемента списка, являющиеся числами. Назовем эти числа x и y и пусть $d := y - x$. Затем, должно быть еще одно число после многоточия, назовем его z . В этом случае, список вида x, y, \dots, z , меняется на список $x, x + d, x + 2d, x + 3d, \dots, x + md$, где m — наибольшее число, для которого $x + md \leq z$, если $d > 0$, или $x + md \geq z$, если $d < 0$.

Вот несколько поясняющих примеров:

```
\foreach \x in {1,2,...,6} {\x, }      → 1, 2, 3, 4, 5, 6,
\foreach \x in {2,4,...,10} {\x, }     → 2, 4, 6, 8, 10,
\foreach \x in {1,3,...,11} {\x, }     → 1, 3, 5, 7, 9, 11,
\foreach \x in {1,3,...,10} {\x, }     → 1, 3, 5, 7, 9,
\foreach \x in {1,1.1,...,1.5} {\x, }  → 1, 1.1, 1.20001, 1.30002, 1.40002,
\foreach \x in {a,b,3,...,1,2,2.135,...,2.5} {\x, }
                                          → a, b, 3, 2, 2.135, 2.26999, 2.40498,
```

Заметим, что в случае дробного шага, при некоторой малой величине d , могут возникать погрешности округления, что приведет к последовательности чисел, отличной от теоретической. Таким образом, в предпоследнем примере, 1.5 следует поменять на 1.501. Есть один частный случай для `...`: если `...` предшествует только одно число (отсутствует y , как в последнем примере), то полагается $d = 1$, если $x < z$, и $d = -1$, если $x > z$.

```
\foreach \x in {1,...,6} {\x, }      → 1, 2, 3, 4, 5, 6, (d = 1),
\foreach \x in {9,...,3.5} {\x, }    → 9, 8, 7, 6, 5, 4, (d = -1).
```

Еще один частный случай для `...` возникает, когда x , $[y]$ и z буквы:

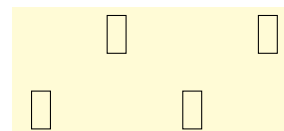
```
\foreach \x in {a,...,e} {\x, }      → a, b, c, d, e,
\foreach \x in {Z,X,...,M} {\x, }    → Z, X, V, T, R, P, N,
```

Следующий частный случай для `...` — контекстная замена. Если `...` используется в некотором контексте, например, `\sin(...)`, этот контекст интерпретируется правильно, при условии, что элементы списка до `...` имеют одну и ту же модель, за исключением того, что, вместо точек, они содержат число или букву:

```
\foreach \x in {2^1,2^...,2^5} {\x$, } → 2^1, 2^2, 2^3, 2^4, 2^5,
\foreach \x in {0\pi,0.5\pi,...\pi,2\pi} {\x$, } → 0\pi, 0.5\pi, 1\pi, 1.5\pi, 2\pi,
\foreach \x in {A_1,..._1,E_1} {\x$, } → A_1, B_1, C_1, D_1, E_1,
```

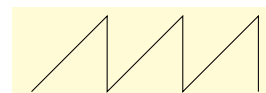

Специальная обработка пар. Различные элементы списка отделяются запятыми. Однако, это вызывает проблемы, когда элементы списка сами содержат запятые, как, например, пары $(0,1)$. В этом случае, нужно поместить элементы, содержащие запятые, в фигурные скобки: $\{(0,1)\}$. Однако, поскольку пары — естественный и часто встречающийся случай, есть специальная трактовка выражения для `\foreach`: если элемент списка начинается с открывающей круглой скобки $($, все до следующей закрывающей круглой скобки $)$ рассматривается как элемент списка. Таким образом, можно написать:

```
\tikz
\foreach \position in
  {(0,0), (1,1), (2,0), (3,1)}
\draw \position rectangle +(.25,.5);
```

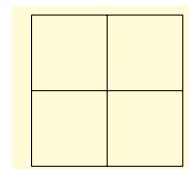


Использование `\foreach` в путях. TikZ позволяет использовать `\foreach` внутри конструкции пути. В этом случае, `<commands>` должен содержать команды конструирования пути. Вот два примера:

```
\tikz
\draw (0,0) \foreach \x in {1,...,3}
  { -- (\x,1) -- (\x,0) };
```



```
\tikz \draw
  \foreach \p in {1,...,3}
    {(\p,1)--(\p,3) (1,\p)--(3,\p)};
```



Множество переменных. Часто нужно выполнить итерации одновременно по двум переменным. Так как `\foreach`-циклы можно вкладывать, решение является прямым. Однако, иногда две переменные в итерации должны использоваться совместно. Например, задан список дуг, которые соединяют по две точки, и желательно выполнить итерацию по этим дугам. При этом, хотелось бы, чтобы начальная и конечная точки дуги устанавливались в две различные переменные.

Чтобы добиться этого, можно использовать следующий синтаксис: `<variables>` может содержать не одну TeX-переменную, а список переменных, отделенных слэшами $(/)$. В этом случае элементы списка могут также быть списками значений, разделенных слэшами. Предполагая, что `<variables>` и элементы списка являются списками значений, каждый раз, когда выполняется `<commands>`, каждая из переменных в `<variables>` пробегает «свою» часть списка. Вот поясняющий пример:

```
\foreach \x/\y in {1/2,a/b}{''\x\ and \y''\ } → "1 and 2" "a and b"
```

Если некоторый слэш-список в `<list>` не имеет достаточного числа слэшей (и потому элементов), то будет повторяться последний элемент из слэш-списка:

```
\tikz \foreach \x/\y in {0,1/a,2,3/b}
  \draw (\x,0) node{\y};
```

0 a 2 b

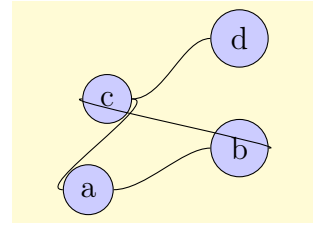
```
\tikz \foreach \x/\xtext in {0,...,3,2.72/e}
  \draw (\x,0) node{\$xtext$};
```

0 1 2 e3

```

\begin{tikzpicture}
% Определение узлов
\path[nodes={circle,fill=blue!20,draw}]
  (0,0)      node(a) {a}
  (2,0.55)   node(b) {b}
  (.25,1.2)  node(c) {c}
  (2,2)      node(d) {d};
% Рисование связей
\foreach \source/\target in {a/b, b/c, c/a, c/d}
  \draw (\source) .. controls +(.75cm,0pt) and
    +(-.75cm,0pt)..(\target);
\end{tikzpicture}

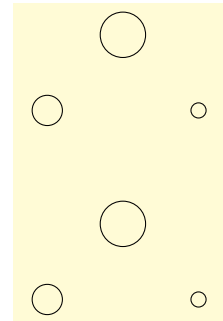
```



```

\begin{tikzpicture}
% Нарисовать окружности
\foreach \x/\y/\diameter in
  {0/0/2mm, 1/1/3mm, 2/0/1mm}
  \draw (\x,\y) circle (\diameter);
% Тот же результат
\foreach \center/\diameter in
  {(0,0)/2mm},{(1,1)/3mm},{(2,0)/1mm}
\draw[yshift=2.5cm] \center circle (\diameter);
\end{tikzpicture}

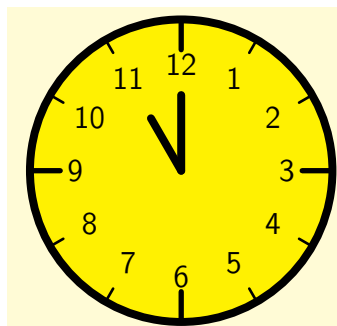
```



```

\begin{tikzpicture}[line cap=round,line width=3pt]
  \filldraw [fill=yellow] (0,0) circle (2cm);
\foreach \angle / \label in
  { 0/3, 30/2, 60/1, 90/12, 120/11, 150/10,
    180/9, 210/8, 240/7, 270/6, 300/5, 330/4 }
{ \draw[line width=1pt] (\angle:1.8cm) -- (\angle:2cm);
  \draw (\angle:1.4cm) node{\textsf{\label}}; }
\foreach \angle in {0,90,180,270}
  \draw[line width=2pt] (\angle:1.6cm) -- (\angle:2cm);
\draw (0,0) -- (120:0.8cm); % Часовая стрелка
\draw (0,0) -- (90:1cm);   % Минутная стрелка
\end{tikzpicture}%

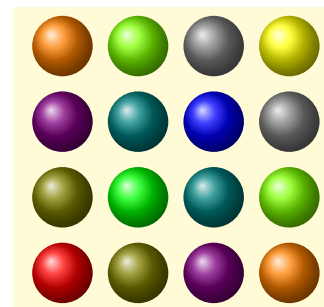
```



```

\tikz[shading=ball]
\foreach \x/\cola in
    {0/red,1/green,2/blue,3/yellow}
\foreach \y/\colb in
    {0/red,1/green,2/blue,3/yellow}
\shade[ball color=\cola!50!\colb]
    (\x,\y) circle (0.4cm);

```



Опции для настройки `\foreach`. Ключи, описанные ниже, могут использоваться в параметре `<options>` команды `\foreach`.

`/pgf/foreach/var=<variable>` (no default)

Этот ключ предоставляет альтернативный способ определения переменных, и должен использоваться перед всеми другими ключами:

```
\foreach [var=\x,var=\y] ~ \foreach \x/\y.
```

`/pgf/foreach/evaluate=<variable>as<macro>using<formula>` (no default)

По умолчанию элементы списка `<list>` не вычисляются. Этот ключ позволяет вычислить переменную, используя механизм математики. Переменная должна быть определена либо используя ключ `var`, либо в параметре `<variables>` команды `\foreach`. По умолчанию, результат вычисления сохраняется в `<variable>`. Однако, можно использовать выражение `as <macro>`, чтобы сохранить результат в макросе `<macro>`.

```
\foreach \x [evaluate=\x] in {2^1,2^...,2^8}{\x$, }
```

2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0, 256.0,

```
\foreach \x [evaluate=\x as \xeval] in {2^1,2^...,2^4}{\x=\xeval$, }
```

$2^1 = 2.0$, $2^2 = 4.0$, $2^3 = 8.0$, $2^4 = 16.0$, $2^5 = 32.0$, $2^6 = 64.0$, $2^7 = 128.0$, $2^8 = 256.0$,

Дополнительное использование выражения `using <formula>` означает, что нужное вычисление не должно явно формулироваться перед каждым элементом из `<list>`, а `<formula>` должна содержать, по крайней мере, одну ссылку на `<variable>`.

```

\tikz\foreach \x [evaluate=\x as \shade using \x*10] in {0,1,...,10}
\node [fill=red!\shade!yellow, minimum size=0.65cm] at (\x,0) {\x};

```

0 1 2 3 4 5 6 7 8 9 10

`/pgf/foreach/remember=<variable> as <macro> (initially <value>)` (no default)

Позволяет значение элемента, сохраненное в `<variable>`, запомнить на время следующей итерации в макросе `<macro>`. Если переменная вычисляется, то запоминается результат этого вычисления. По умолчанию значение `<variable>` для первой итерации равно нулю, однако, опция `initially <value>` позволяет изначально определить `<macro>` как `<value>`.

```

\foreach \x [remember=\x as \lastx (initially A)] in
    {B,...,H}{\overrightarrow{\lastx\x}$, }

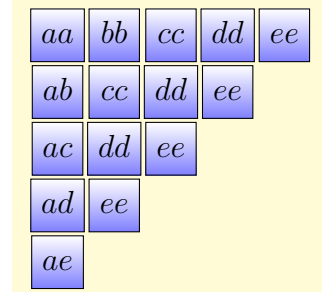
```

$$\overrightarrow{AB}, \overrightarrow{BC}, \overrightarrow{CD}, \overrightarrow{DE}, \overrightarrow{EF}, \overrightarrow{FG}, \overrightarrow{GH},$$

`/pgf/foreach/count`=<macro> from <value> (no default)

Позволяет макросу <macro> сохранить позицию текущего элемента в списке. Дополнительное выражение from <value> позволяет начать счет с <value>.

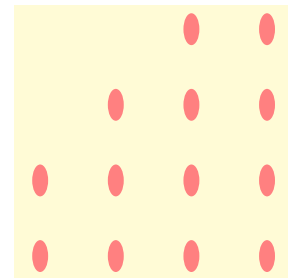
```
\tikz[x=0.75cm,y=0.75cm]
  \foreach \x [count=\xi] in {a,...,e}
    \foreach \y [count=\yi] in {\x,...,e}
      \node [draw, top color=white,
            bottom color=blue!50,
            minimum size=0.666cm]
            at (\xi,-\yi) {$\mathstrut\x\y$};
```



`\breakforeach`

Если команда `\breakforeach` задается внутри команды `\foreach`, дальнейшее выполнение команд <commands> прекращается. Однако текущее выполнение <commands> продолжается до их завершения, поэтому лучше всего использовать эту команду в конце команды `\foreach`.

```
\begin{tikzpicture}
  \foreach \x in {1,...,4}
    \foreach \y in {1,...,4}
      {\fill[red!50] (\x,\y)
        ellipse (3pt and 6pt);
       \ifnum \x<\y \breakforeach \fi }
\end{tikzpicture}
```



Глава 26

Даты и календари

Пакет `pgfcalendar`, который, как и пакет `pgffor`, может использоваться независимо от `pgf`, решает две задачи.

1. Обеспечивает функции для работы с датами. Можно преобразовать дату в формате ISO (например, 1975-12-26) в день так называемого Юлианского календаря. День Юлианского календаря — целое число дней, протекших с полудня по Гринвичу понедельника 1 января 4713 года до н.э. Обратное, пакет содержит функцию, позволяющую преобразовать день Юлианского календаря в дату формата ISO. Юлианские даты позволяют существенно упростить работу с датами. Например, день недели данной даты можно легко вычислить, вычисляя остаток от деления Юлианской даты на 7.

2. Обеспечивает макрос для набора календарей. Он едва ли будет явно использоваться, но на его основе созданы простые и понятные команды для набора календарей.

26.1 Обработка дат

26.1.1 Преобразования между типами дат

`\pgfcalendaratetojulian`{<date>}{<counter>}

Макрос преобразует дату в формате Год-Месяц-День в дату Юлианского календаря. Для текущих года, месяца, дня, нужно использовать ключевые слова `\year`, `\month` и `\day`. Ключевое слово `last` обращается к последнему дню месяца. Можно использовать `<+integer>` или `<-integer>`, чтобы прибавить или отнять от даты указанное число дней.

2006-01-01 — ссылается на первое января 2006 года.

2006-02-last — ссылается на 28 февраля 2006 года.

`\year-\month-\day` — ссылается на сегодняшний день.

2006-01-01+2 — ссылается на 3-е января 2006 года.

`\year-\month-\day+1` — ссылается на завтрашний день.

`\year-\month-\day -1` — ссылается на вчерашний день.

Выражение `\pgfcalendaratetojulian{2007-01-14}{\mycount}` установит число 2454115 в макрос `\mycount`.

`\pgfcalendarjuliantodate`{<Julian day>} {<year macro>}{<month macro>}{<day macro>}

Преобразует Юлианскую дату в дату формата ISO. Параметр `<Julian day>` должен быть числом или счетчиком Т_ЭX'a, `<year macro>`, `<month macro>` и `<day macro>` должны быть именами макросов Т_ЭX'a, которые будут хранить год, месяц и день Юлианской даты в Григорианском календаре. Макрос `<year macro>` будет хранить год без начальных нулей. Отметим,

что этот макрос может произвести год 0 (в противоположность другим календарям, где год 0 не существует). Макрос `<month macro>` хранит число из двух цифр, представляющее месяц (с начальным нулем, если это необходимо). Макрос `<day macro>` также хранит число из двух цифр, представляющее день месяца (снова с возможным начальным нулем).

Чтобы преобразовать Юлианскую дату в дату стандарта ISO, следует использовать код следующего вида:

```
\pgfcalendardatetojulian{2454115}{\myyear}{\mymonth}{\myday}
\edef\isodate{\myyear-\mymonth-\myday}
```

Этот код установит в `\isodate` дату 2007-01-14.

```
\pgfcalendarjuliantoweekday{<Julian day>}{<week day counter>}
```

Преобразует Юлианскую дату в день недели. Параметр `<week day counter>` должен быть счетчиком TEX'a, который будет хранить 0 для понедельника, 1 для вторника, и так далее. Например, код `\pgfcalendarjuliantoweekday {2454115}{\mycount}` установит в `\mycount` число 6.

26.1.2 Проверка дат

```
\pgfcalendarifdate{<date>}{<tests>}{<code>}{<else code>}
```

Позволяет выполнить код, основываясь на свойствах даты `<date>`, которая должна представляться датой в ISO-формате. Для этой даты выполняются тесты `<tests>` и, если хотя бы один проходит, выполняется `<code>`. Если же ни один из тестов не проходит, выполняется `<else code>`. Например:

```
\pgfcalendarifdate{2007-02-07}{Wednesday}
      {Is a Wednesday}
      {Is not a Wednesday}
```

вернет `Is a Wednesday`.

Тесты `<tests>` — разделенный запятыми список пар «ключ–значение». Следующие тесты определены по умолчанию: `all` (его проходят все даты), `Monday` (проходят даты, представляющие понедельники), `Tuesday` (вторники), `Wednesday` (среды), `Thursday` (четверги), `Friday` (пятницы), `Saturday` (субботы), `Sunday` (воскресенья), `workday` (проходят все, кроме суббот и воскресений), `weekend` (проходят только субботы и воскресенья), `equals=<reference>`.

В тесте `equals=<reference>`, параметр `<reference>` может представляться в одной из двух форм: даты в полном ISO-формате (например, 2007-01-01) или год может отсутствовать (например, 12-31). Например, тест `equals=2007-01-10` пройдет только указанная дата. Тест `equals=05-01` пройдет любая дата, указывающая на первое мая любого года.

Помимо этих простых тестов, определены более специальные тесты.

`at least=<reference>` — дата проходит тест, если `<date>` меньше или равна `<reference>`. Снова, `<reference>` может быть полной датой или ее короткой версией. Тест `least=07-01` пройдут все даты второй половины любого года.

`at most=<reference>` — дата проходит тест, если `<date>` больше или равна `<reference>`.

`between<start reference> and <end reference>` — дата проходит тест, если она находится между указанными датами: тест `between=2007-01-01 and 2007-02-28` проходят даты января и февраля 2007 года, а тест `between=05-01 and 05-07` — даты первой недели мая любого года.

`day of month=<number>` — проходят даты, соответствующие указанному дню `<number>` каждого месяца. Например, тест `day of month=1` проходят даты первого дня каждого месяца.

`end of month=<number>` — проходят даты, соответствующие указанному дню `<number>` с конца каждого месяца. Например, тест `end of month=1` пройдет последний день каждого месяца,

а тест `end of month=2` — предпоследний день каждого месяца. Если параметр `<number>` отсутствует, предполагается, что он равен 1.

В дополнение к приведенным тестам можно определять новые (см. детали [1, p. 511]).

26.1.3 Вывод дат

`\pgfcalendarweekdayname{<week day number>}`

Команда вернет текстовое представление дня недели по числу `<week day number>`. Например, `\pgfcalendarweekdayname{0}` вернет `Monday` (если текущий язык английский).

`\pgfcalendarweekdayshortname{<week day number>}`

Команда работает подобно предыдущей, только вернет сокращенную версию имени дня недели. Например, код `\pgfcalendarweekdayshortname{2}` вернет `Wed`.

`\pgfcalendarmonthname{<month number>}`

Команда вернет текстовое представление месяца по числу `<month number>`. Например, код `\pgfcalendarmonthname{12}` вернет `December`.

`\pgfcalendarmonthshortname{<month number>}`

Аналогична предыдущей, только вернет сокращенную версию имени месяца. Например, код `\pgfcalendarmonthshortname{12}` вернет `Dec`.

26.1.4 Локализация

Текстовые представления дней недели или месяцев обернуты в команду `\translate` из пакета `translator` (если он не загружен, перевод не выполняется). Если пакет `translator` загружен, пакет `pgfcalendar` будет пытаться загрузить словарь `translator-months-dictionary`. Как результат, все даты будут представляться на текущем языке, установленном в пакете `translator` (см. документацию по этому пакету).

26.2 Вывод календарей

`\pgfcalendar{<prefix>}{<start date>}{<end date>}{<rendering code>}`

Позволяет набрать календарь. Это общая команда, и фактическая работа делается заданием грамотно реализованного кода `<rendering code>`. Отметим, что этот макрос не обязательно вызывать в среде `{pgfpicture}`, его можно использовать для набора календарей в обычном `TeX`'е.

Процесс набора любого календаря сводится к следующему: параметры `<start date>` и `<end date>` определяют диапазон дат; для каждой даты из этого диапазона выполняется код `<rendering code>`, который, как правило, помещает узлы в рисунок, но может делать и другие вещи. Одна из задач кода `<rendering code>` — правильно разместить календарь.

В коде `<rendering code>` доступны следующие макросы:

<code>\pgfcalendarprefix</code>	<code>\pgfcalendarbeginiso</code>
<code>\pgfcalendarbeginjulian</code>	<code>\pgfcalendarendiso</code>
<code>\pgfcalendarendjulian</code>	<code>\pgfcalendarcurrentjulian</code>
<code>\pgfcalendarcurrentweekday</code>	<code>\pgfcalendarcurrentyear</code>
<code>\pgfcalendarcurrentmonth</code>	<code>\pgfcalendarcurrentday</code>

Кроме них, в `\pgfcalendar` локально доступен макрос

`\ifdate {<tests>}{<code>}{<else code>}`

Дает тот же результат, что и запрос `\pgfcalendarifdate` для текущей даты.

Рассмотрим несколько очень простых календарей. Первым создадим календарь, который только перечисляет даты из определенного диапазона.

```
\pgfcalendar{cal}{2007-01-20}{2007-02-10}{\pgfcalendarcurrentday\ }
```

```
20 21 22 23 24 25 26 27 28 29 30 31 01 02 03 04 05 06 07 08 09 10
```

Более интересный пример: добавим переход на новую строку после каждого воскресенья.

```
\pgfcalendar{cal}
{2007-01-20}{2007-02-10}
{ \pgfcalendarcurrentday\
  \ifdate{Sunday}{\par}{} }

```

```
20 21
22 23 24 25 26 27 28
29 30 31 01 02 03 04
05 06 07 08 09 10
```

Но хотелось бы, чтобы все понедельники располагались в столбце. Для этого, следует позиционировать каждую дату по горизонтально, используя пробел.

```
\pgfcalendar{cal}{2007-01-20}{2007-02-10}
{\leavevmode \hbox to0pt{\hskip%
  \pgfcalendarcurrentweekday cm%
  \pgfcalendarcurrentday\hss}%
  \ifdate{Sunday}{\par}{}}

```

```
20 21
22 23 24 25 26 27 28
29 30 31 01 02 03 04
05 06 07 08 09 10
```

В следующем примере отобразим целый месяц и добавим его имя.

```
\pgfcalendar{cal}{2007-01-01}{2007-31-01}{%
  \ifdate{day of month=1}
  { \par\bigskip\hbox to7.5cm{\itshape\hss\pgfcalendarshorthand mt,
    \pgfcalendarshorthand y0\hss}\par }{}}%
\leavevmode{ \ifdate{weekend}{\color{black!50}}{\color{black}}%
  \hbox to0pt{ \hskip\pgfcalendarcurrentweekday cm%
    \hbox to1cm{\hss\pgfcalendarshorthand d=}\hss% }%
  } \ifdate{Sunday}{\par}{} }

```

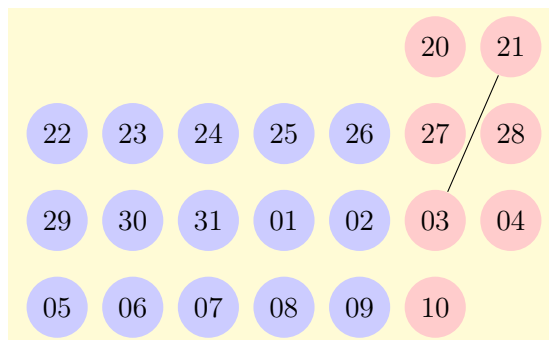
```

          January, 2007
    1      2      3      4      5      6      7
    8      9     10     11     12     13     14
   15     16     17     18     19     20     21
   22     23     24     25     26     27     28
   29     30     31

```

В последнем примере используем окружение `{tikzpicture}`.

```
\begin{tikzpicture}
\pgfcalendar{cal}{2007-01-20}{2007-02-10}{%
  \ifdate{workday} {\tikzset{filling/.style={fill=blue!20}}}
                   {\tikzset{filling/.style={fill=red!20}}}
  \node (\pgfcalendarsuggestedname) at (\pgfcalendarcurrentweekday,0)
    [anchor=base,circle,filling] {\pgfcalendarcurrentday};
  \ifdate{Sunday}{\pgftransformmyshift{-3em}}{} }
\draw (cal-2007-01-21) -- (cal-2007-02-03);
\end{tikzpicture}
```

`\pgfcalendarshorthand{<kind>}{<representation>}`

Команда может использоваться в `\pgfcalendar`, чтобы представить текущий день, месяц, год или день недели, в зависимости от того, является ли `<kind>` одним из символов `d`, `m`, `y`, `w`. Параметр `<representation>` может быть одним из символов `-`, `=`, `0`, `.`, `t`.

- Знак «минус» выбирает самое короткое возможное числовое представление (без начальных нулей).
- Знак «=» выбирает самое короткое числовое представление, но добавляет пробел к единственной цифре дня и месяца (гарантируя, что они будут иметь ту же длину, что и другие даты).
- Цифра «0» выбирает числовое представление с двумя цифрами для дней и месяцев.
- Символ «t» выбирает текстовое представление.
- Точка (.) выбирает сокращенное текстовое представление.

Обычно, можно локально написать выражение `\let\%=\pgfcalendarshorthand`, что позволит потом писать `\%wt` вместо более громоздкого `\pgfcalendarshorthand{w}{t}`.

```
\let\%=\pgfcalendarshorthand \pgfcalendar{cal}{2007-01-20}{2007-01-20}
  {ISO form: \%y0-\%m0-\%d0, long form: \%wt, \%mt \%d-, \%y0}
```

ISO form: 2007-01-20, long form: Saturday, January 20, 2007

`\pgfcalendarsuggestedname`

Этот макрос расширяется в рекомендованное имя узлов, представляющих дни в календаре. Если `<prefix>` (параметр в `\pgfcalendar`) пуст, макрос расширяется до пустой строки, иначе он расширяется до значения `<prefix>` календаря, которое сопровождается дефисом и ISO-форматом даты. Таким образом, когда дата 2007-01-01 набрана в календаре с префиксом `<prefix>` равным `mycalendar`, макрос расширяется до `mycalendar-2007-01-01`.

Глава 27

Управление страницами

Пакет `pgfpages`, хотя и не связан с созданием рисунков, но полезен при формировании страниц, поскольку обеспечивает простой и гибкий способ размещения нескольких страниц на одной странице \TeX 'а. В настоящее время, `pgfpages` работает только с \LaTeX 'ом, поддерживается и `latex`, и `pdflatex`.

Однако, использование пакета `pgfpages` разрушит существующие гиперссылки (они будут присутствовать, но будут осуществлять переход в неправильные позиции).

27.1 Основное использование

Внутренняя организация `pgfpages` сложна, так как пакет решает множество сложных задач. По этой причине предопределены так называемые схемы размещения (layouts), которые устанавливают должным образом все требуемые опции. Схемы размещения определяются командой `\pgfpagesuselayout` и используются следующим образом:

```
\documentclass{article}
\usepackage{pgfpages}
\pgfpagesuselayout{2 on 1}[a4paper,landscape,border shrink=5mm]
\begin{document}
  This text is shown on the left.
\clearpage
  This text is shown on the right.
\end{document}
```

Схема размещения `2 on 1` помещает две страницы на одну. Опция `a4paper` говорит о том, что используется физическая страница формата A4, и что страница должна использоваться поперек (опция `landscape`), что в этом случае естественно. Обычно, логические страницы, то есть, страницы, о которых \TeX «думает», что он их набирает, `pgfpages` автоматически сокращает до таких размеров, чтобы они легли на страницу формата A4. Опция `border shrink` просит `pgfpages` добавить дополнительно 5mm так, чтобы появился зазор шириной 5mm вокруг получающихся логических страниц. Как второй пример, поместим две страницы, созданные классом `beamer` на одной странице:

```
\documentclass{beamer}
\usepackage{pgfpages}
\pgfpagesuselayout{2 on 1}[a4paper,border shrink=5mm]
\begin{document}
\begin{frame}
  This text is shown at the top.
\end{frame}
\end{document}
```

```
\begin{frame}
  This text is shown at the bottom.
\end{frame}
\end{document}
```

Отметим, что опция `landscape` не используется, так как логические страницы `beamer` уже располагаются поперек страницы, определяя для этого опцию `landscape`. Однако, если нужно использовать схемы размещения `4 on 1` и `16 on 1`, то следует добавить опцию `landscape`. Используя схемы размещения `8 on 1` и `32 on 1`, эту опцию добавлять не нужно.

Предупреждение: использование `pgfpages` порождает неправильные номера страниц в аух-файле. Причина в том, что `TeX` записывает номера страниц в аух-файле только тогда, когда сформируется физическая страница. К счастью, эта проблема просто решается: сначала следует собрать документ обычным образом, не используя команду `pgfpagesuselayout` (помещая перед ней `%`). Затем, повторно запустить `TeX` с включенной командой `pgfpagesuselayout`, добавляя в преамбулу еще и команду `\nofiles`, которая гарантирует что аух-файл не будет меняться, а это то, что нужно.

Заключительный пример использует схему размещения `resize to` (она работает подобно гипотетической схеме размещения `1 on 1`). Это размещение изменяет размеры логической страницы так, что они соответствуют конкретным физическим размерам. Так как эта схема не изменяет номера страниц, ничего не происходит и с аух-файлом. Например, строки в преамбуле

```
\usepackage{pgfpages}
\pgfpagesuselayout{resize to}[a4paper]
```

гарантируют, что физический вывод будет происходить на бумагу формата `A4`. Это может оказаться полезным, когда нужно обработать много страниц, которые набраны для формата `letter`, а доступен принтер для бумаги формата `A4` или наоборот. Например, следующая статья будет подходить для печати на бумаге формата `letter`:

```
\documentclass[a4paper]{article}
% A4 - текущий логический и физический размеры
\usepackage{pgfpages}
\pgfpagesuselayout{resize to}[letterpaper]
% A4 остается логическим размером, а letterpaper - физическим
\begin{document}
\title{My Great Article}
\dotsc \dotsc \dotsc \dotsc \dotsc \dotsc
\end{document}
```

27.2 Предопределенные схемы размещения

Схема размещения выбирается, используя команду

```
\pgfpagesuselayout{<layout>}[<options>]
```

Устанавливает конкретную схему размещения `<layout>` с заданным набором параметров `<options>`. Если эта команда вызывается несколько раз, работает только последняя (схемы размещения не накапливаются!).

```
\pgfpagesuselayout{resize to}[<options>]
```

Размещение используется, чтобы изменить размеры каждой логической страницы на конкретный физический размер. Чтобы определить целевой размер, можно задавать следующие опции:

`physical paper height=<size>` — устанавливает высоту физического размера страницы равной `<size>`.

`physical paper width=<size>` — устанавливает ширину физического размера страницы равной `<size>`.

`a0paper` — устанавливает физический размер страницы равным A0.

`a1paper` — устанавливает физический размер страницы равным A1.

`a2paper` — устанавливает физический размер страницы равным A2.

`a3paper` — устанавливает физический размер страницы равным A3.

`a4paper` — устанавливает физический размер страницы равным A4 (21.0 × 29.7 см).

`a5paper` — устанавливает физический размер страницы равным A5.

`a6paper` — устанавливает физический размер страницы равным A6.

`letterpaper` — устанавливает физический размер страницы равным размеру американский почтовой бумаги (25.4 × 40.7 см).

`legalpaper` — устанавливает физический размер страницы равным размеру американский юридический страницы (33 × 40,6 см).

`executivepaper` — устанавливает физический размер страницы равным размеру американского исполнительного листа.

`landscape` — меняет высоту и ширину физической страницы местами.

`border shrink=<size>` — дополнительно уменьшает размер логической страницы на физической странице на `<size>`.

`\pgfpagesuselayout{2 on 1}[<options>]`

Помещает две логических страницы вдоль друг друга на каждой физической странице, если логическая высота больше, чем логическая ширина (логические страницы находятся в режиме `portrait`). Иначе, помещает две логические страницы одна над другой (логические страницы находятся в режиме `landscape`). Используя это размещение, желательно использовать команду `\nofiles` (автоматически это не происходит).

Могут использоваться те же опции, что и для схемы размещения `resize to`, плюс опция `odd numbered pages right`, которая помещает первую страницу справа.

`\pgfpagesuselayout{4 on 1}[<options>]`

Помещает четыре логических страницы на одну физическую страницу. Могут использоваться те же опции, что и для схемы размещения `resize to`.

`\pgfpagesuselayout{8 on 1}[<options>]`

Помещает восемь логических страниц на одну физическую страницу. Как и в случае схемы `2 on 1`, ориентация зависит от того, размещаются ли логические страницы в режиме `portrait` или в режиме `landscape`.

`\pgfpagesuselayout{16 on 1}[<options>]`

Помещает шестнадцать логических страниц на одну физическую страницу.

`\pgfpagesuselayout{rounded corners}[<options>]`

Размещение добавляет на каждой странице закругленные углы, что, возможно, хорошо выглядит при использовании проектора во время демонстрации презентации. В дополнение к опциям, задаваемым для размещения `resize to`, чтобы изменить размеры можно задавать опцию `corner width=<size>`, которая определяет размер угла.

```
\documentclass{beamer}
\usepackage{pgfpages}
\pgfpagesuselayout{rounded corners}[corner width=5pt]
\begin{document}
\dots \dots \dots \dots \dots \dots \dots \dots
\end{document}
```

```
\pgfpagesuselayout{two screens with lagging second}[<options>]
```

Размещение помещает две логических страницы рядом друг с другом. Вторая страница всегда показывает то, что главная страница показывала на предыдущей физической странице. Это может оказаться полезным, когда проекторы, связанные с компьютером, могут показывать разные части физической страницы на разных проекторах. Для размещения страниц можно использовать опции `second right`, `second left`, `second bottom`, `second top`.

```
\pgfpagesuselayout{two screens with optional second}[<options>]
```

Размещение работает аналогично предыдущему. Различие в том, что содержание второго экрана изменяется только тогда, когда вызывается одна из команд

```
\pgfshipoutlogicalpage{2}{<box>} или \pgfcurrentpagewillbellogicalpage{2}
```

Первая команда помещает заданный бокс `<box>` на второй странице. Вторая определяет, что текущая страница должна быть помещена там, где она заканчивается. Могут использоваться те же опции, что и для предыдущей схемы размещения.

Если ни одна из предопределенных схем размещения не решает возникших проблем, можно создать собственную схему размещения и собственное содержание логических страниц. Детали того, как это сделать см. в [1, p.518–521].

Глава 28

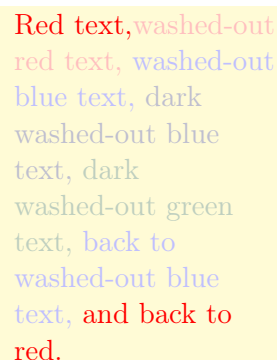
Расширенная поддержка цвета

Пакет `xxcolor` в настоящее время является частью `pgf` и расширяет возможности пакета `xcolor`, написанного Уве Керном (Uwe Kern), который, в свою очередь, расширяет возможности пакета `color`. Основная цель пакета `xxcolor` состоит в том, чтобы обеспечить среду, внутри которой все цвета можно сделать «размытыми» и «приглушенными», что требуется во многих ситуациях, но обычно достигается окольными путями, если такой среды нет.

```
\begin{colormixin}{<mix-in specification>}
  <environment contents>
\end{colormixin}
```

Опция `<mix-in specification>` применяется ко всем цветам среды. В начале среды опция применяется к текущему цвету, то есть, к тому цвету, который был перед запуском среды. Опция `<mix-in specification>` — это число между 0 и 100, сопровождаемое восклицательным знаком и именем цвета. Когда команда `\color` используется в среде `{colormixin}`, это число определяет, какой процент от указанного цвета должен использоваться. Остальная часть (до 100%) заполняется цветом, указанным в `<mix-in specification>`. Таким образом, значение опции `<mix-in specification>` равно `90!blue`, подмешает 90% синего к 10% любого другого цвета, значение равно `25!white` делает все светлее.

```
\begin{minipage}{3.5cm}\raggedright
\color{red}Red text,%
\begin{colormixin}{25!white}
  washed-out red text,
  \color{blue} washed-out blue text,
  \begin{colormixin}{25!black}
    dark washed-out blue text,
    \color{green} dark washed-out green text,%
  \end{colormixin}
  back to washed-out blue text,%
\end{colormixin}
and back to red.
\end{minipage}%
```



Отметим, что среда только изменяет цвета, которые были установлены, используя стандартную L^AT_EX-команду `\color`. В частности, цвета в рисунках не изменяются. Есть, однако, некоторая поддержка, предлагаемая в отношении команд `\pgfuseimage` и `\pgfuseshading`. Если первая команда вызвана в среде `colormixin` с параметром, скажем, `{50!black}` на рисунке с именем `foo`, команда сначала проверит, есть ли рисунок с именем `foo.50!black`. Если есть, используется этот рисунок. Это позволяет определить разные рисунки для каждого случая. Если среды `colormixin` вкладываются, все определения `<mix-in specification>` объединяют-

ся. Например, во внутренней среде вышеупомянутого примера, команда `\pgfuseimage{foo}` сначала проверит, существует ли рисунок с именем `foo`. `!50!white!25!black`.

`\colorcurrentmixin`

Расширяется до текущего накопленного параметра средой `colormixin`. Каждое вложение новой среды `colormixin` добавляет новый элемент к этому списку.

```
\begin{minipage}{\linewidth-6pt}\raggedright
\begin{colormixin}{75!white}
\colorcurrentmixin\ should be “!75!white”\par
\begin{colormixin}{75!black}
\colorcurrentmixin\ should be “!75!black!75!white”\par
\begin{colormixin}{50!white}
\colorcurrentmixin\ should be “!50!white!75!black!75!white”\par
\end{colormixin}
\end{colormixin}
\end{colormixin}
\end{minipage}
```

`!75!white` should be “!75!white”

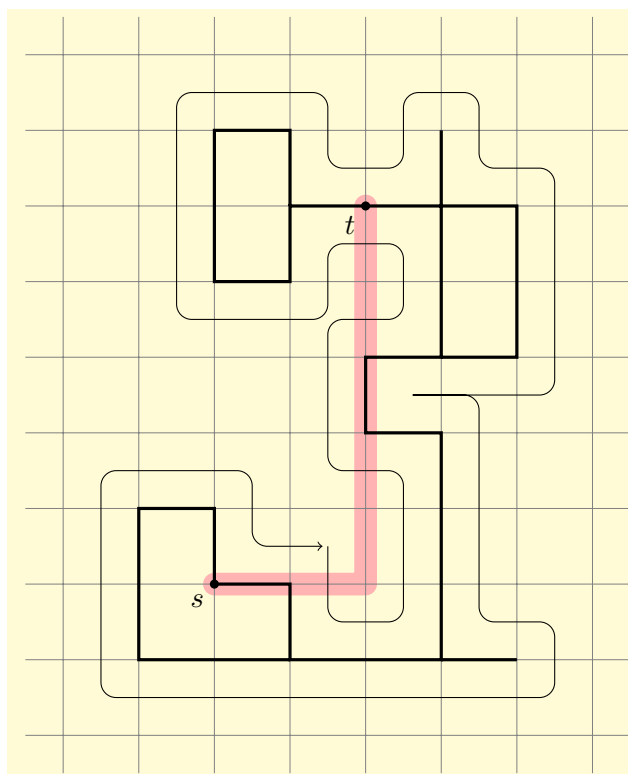
`!75!black!75!white` should be “!75!black!75!white”

`!50!white!75!black!75!white` should be “!50!white!75!black!75!white”

Часть III

Объектно-ориентированное
программирование

Модуль `oo`, который надо загрузить в преамбуле документа (`\usepgfmodule{oo}`), определяет относительно небольшое число `TeX`-команд для определения классов, методов, атрибутов (переменных) и объектов в смысле объектно-ориентированного программирования. Предполагается, что читатель знакомы с основами хотя бы одного объектно-ориентированного языка программирования: Java, C++, Eiffel, Smalltalk, CLOS, Haskell.



Глава 29

ООП и Т_ЕX

29.1 Краткий обзор

Изначально Т_ЕX не поддерживал объектно-ориентированное программирование видимо потому, что был написан в то время, когда такой стиль еще не был в моде. Когда в Т_ЕX используется объектно-ориентированный стиль, некоторые его конструкции программирования кажутся усложненными. Модуль `oo` в `pgf` может помочь упростить их. Он написан, полностью используя макросы Т_ЕX'а и, таким образом, совершенно переносим. Это также означает, что он не очень быстрый (но и не слишком медленный), и потому его не следует использовать для критичных по времени программ.

В основном, `oo`-система поддерживает классы (в объектно-ориентированном смысле, но это не имеет никакого отношения к классам Л^AT_ЕX), методы, конструкторы, атрибуты, объекты и тождественность объектов. Пока не поддерживается наследование, перегрузка, деструкторы, вложение классов.

Первый шаг — определение класс макросом `\pgfoclass` (все нормальные макросы в объектно-ориентированной `pgf`-системе начинаются с `\pgfoo`). Этот макрос получает имя класса и в его теле определяются методы, используя макрос `\method` (который может использоваться только в классе). Такие методы немного похожи на методы, определяемые в, скажем, Java. Атрибуты объекта объявляются, используя команду `\attribute`, которая также может использоваться только в определении класса.

Как только класс определен, можно создавать объекты (экземпляры) этого класса, используя `\pgfoonew`. Такой объект имеет много объектных особенностей в нормальном объектно-ориентированном языке программирования: каждый объект уникален, то есть, когда создается другой объект, он полностью отличен от всех других объектов. Каждый объект имеет ряд частных атрибутов, которые могут изменяться. Предположим, что есть класс `point`. Тогда создание нового объекта (называемого экземпляром) этого класса обычно имеет x -атрибут и y -атрибут. Создание другого экземпляра класса `point` создаст другой объект с собственными x - и y -атрибутами.

Указывая объект, можно вызвать метод для этого объекта. В методе можно получить доступ к атрибутам того объекта, для которого вызван метод. «Жизнь» объекта всегда заканчивается вместе с областью видимости Т_ЕX'а, в которой он создан. Значение атрибута сохраняется до конца жизни объекта или пока оно не будет изменено.

29.2 Пример: класс `stamp`

Как пример разработаем класс `stamp` (оттиск) и `stamp`-объекты. Идея состоит в том, что `stamp`-объект в состоянии «что-то тиснуть» на рисунке. Это означает, что `stamp`-объект имеет атрибут, который хранит текст, который надо «отштамповать», и есть метод, который просит объект поместить его текст куда-нибудь на холст. Метод можно вызывать многократно и мо-

жет существовать несколько различных `stamp`-объектов, каждый из которых «штампует» свой текст. Экземпляры класса могут создаваться динамически, когда необходимы, или библиотека может определить необходимое их число во внешней области видимости.

Такие оттиски подобны многим другим сущностям в `pgf`, например, наконечникам стрел, шаблонам, и их можно было бы реализовать таким объектно-ориентированным способом (что, возможно, было бы и лучше, но объектно-ориентированная подсистема — новое дополнение к `pgf`).

29.3 Классы

Начнем с определения класса `stamp`, используя макрос `\pgfooclass`:

```
\pgfooclass{<class name>}{<body>}
```

Определяет класс с именем `<class name>`. Имя класса может содержать пробелы и большинство других символов, но не точки. Так, допустимы имена классов `MyClass`, или `my class`, или `Class_C++_emulation??1`. Тело `<body>` фактически только выполняется, таким образом, здесь допустим любой нормальный `TEX`-код. Однако, в то время как `<body>` выполняется, макросы `\method` и `\attribute` устанавливаются так, чтобы они могли использоваться при определении методов и атрибутов для этого класса. Определение класса является локальным для той области видимости, где класс определяется.

```
\pgfooclass{stamp}
{ % Это класс stamp
  \attribute text;
  \attribute rotation angle=20;
  \method stamp(#1) { ... .. %конструктор}
  \method apply(#1,#2) {... .. %отобразить оттиск в точке (#1,#2)}
  ... ..
}
```

Тело `<body>` класса часто состоит только из вызовов макросов `\attribute` и `\method`, которые подробно обсуждаются в последующих разделах.

29.4 Объекты

Как только класс объявлен, можно создавать объекты (экземпляры) этого класса, используя команду `\pgfoonew`, которая имеет специфический синтаксис:

```
\pgfoonewobject <object handle or attribute>=
new<class name>(<constructor arguments>)
```

Создает новый экземпляр. Класс объекта с именем `<class name>` должен быть уже объявлен, используя `\pgfooclass`. Когда экземпляр создан, вызывается метод-конструктор экземпляра со списком параметров в `<constructor arguments>`. Получающийся объект сохраняется, и его жизнь закончится точно в конце существования текущей области видимости (нет деструкторов). Вот пример, создающий три экземпляра класса `stamp`.

```
\pgfoonew \firststamp = new stamp()
\pgfoonew \secondstamp = new stamp()
{\pgfoonew \thirdstamp = new stamp() ... ..}
% Экземпляр \thirdstamp более не существует, \firststamp, \secondstamp
% существуют. Если попытаться сохранить объект \thirdstamp в
% глобальной переменной, попытка доступа к нему приведет к ошибке.
```

Параметр `<object handle or attribute>` может быть или атрибутом `<attribute>` или дескриптором объекта `<object handle>`. Когда задается дескриптор объекта `<object handle>`, он должен быть нормальным именем макроса Т_ЕX'a, который будет "указывать" на объект (см. раздел 29.7). Можно использовать этот макрос, чтобы вызвать методы объекта (см. раздел 29.5). Когда задается `<attribute>`, его нужно задавать в фигурных скобках, которые используются для обнаружения присутствия атрибута. В этом случае, дескриптор (handle) только что созданного объект сохраняется в этом атрибуте.

```
\pgfooclass{foo}
{ \attribute stamp obj;
  \attribute another object;
  \method \foo() { \pgfoonew{stamp obj}=new stamp()
                  \pgfoonew{another object}=new bar() }
  ... .. }
}
```

`\pgfoogc`

Макрос запускает сборщика мусора, освобождая память от глобальных Т_ЕX-макросов, которые используются уже не существующими объектами. Этот макрос вызывается автоматически после каждой области видимости, в которой был создан объект, и явно вызываться не должен.

29.5 Методы

Методы определяются в теле класса, используя команду

```
\method<method name>( <parameter list> ){ <method body> }
```

Определяет в классе новый метод с именем `<method name>`. Имя метода может содержать пробелы и другие символы, таким образом, допустимы такие имена, как `put_stamp_here` или `put stamp here`.

Три имени для методов являются специальными. Первое: метод, имеющий то же имя, что и класс, называют конструктором; такой метод должен быть обязательно, даже если его тело пусто. В настоящее время деструктора не существует: объект просто становится неопределенным при завершении его области видимости. Другие два метода называются `get id`, и `get handle`, всегда создаются автоматически и их нельзя переопределять (см. раздел 29.7).

Перегрузка методов не возможна, то есть, нельзя иметь в одном классе два метода с одним и тем же именем (даже при разных списках параметров). Однако, два разных класса могут содержать одноименные методы, то есть, каждый класс формирует свое пространство имен (namespace) для методов.

За именем метода `<method name>` должен следовать `<parameter list>` — список параметров, заключенный в круглые скобки, которые должны присутствовать даже тогда, когда этот список пуст. Список `<parameter list>` — нормальный список параметров Т_ЕX'a, который должен соответствовать списку аргументов в круглых скобках при вызове метода и, таким образом, может содержать `#n`.

Когда вызывается метод, выполняется тело метода `<body>`. Главное отличие от обычного макроса состоит в том, что, когда выполняется тело метода, специальный макрос с именем `\pgfoothis` ссылается на тот объект, который вызвал этот метод. Для того чтобы вызвать метод для объекта, нужно создать этот объект и воспользоваться его дескриптором. Для этого используется специальный синтаксис:

```
<object handle>.<method name>( <parameters> )
```

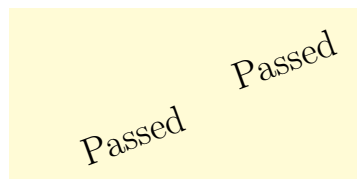
Метод определяется в классе объекта, список аргументов `<parameters>` соответствуют списку параметров метода и, если это так, тело метода выполняется для объекта с дескриптором `<object handle>`. Выполнение тела метода происходит вне области видимости, поэтому результаты выполнения тела метода продолжают существовать.

```

\pgfooclass{stamp}
{ \method stamp() {} % Конструктор
  \method apply(#1,#2)
    {\node [rotate=20,font=\large] at (#1,#2){Passed};} % рисует оттиск }
  \pgfoonew \mystamp=new stamp()

  \begin{tikzpicture}
    \mystamp.apply(1,2)
    \mystamp.apply(3,3)
  \end{tikzpicture}

```



В методе можно вызывать другие методы. Если есть дескриптор другого объекта, его можно просто вызвать способом, описанным выше. Чтобы вызвать метод для текущего объекта, можно использовать специальный дескриптор такого объекта `\pgfoothis`, доступный только при выполнении метода. Там он указывает на объект, для которого вызван этот метод, и который позволяет вызывать другой метод для этого же объекта.

```

\pgfooclass{stamp}
{\method stamp() {}
  \method apply(#1,#2)
    {\pgfoothis.shift origin(#1,#2) \node[rotate=20,font=\huge] {Passed};}
  \method shift origin(#1,#2){\tikzset{xshift=#1,yshift=#2}} %Частный метод
}

```

29.6 Атрибуты

Каждый объект имеет ряд атрибутов, которые могут изменяться с течением времени. Атрибуты объявляются командой `\attribute`, которая, как и команда `\method`, может использоваться только в области видимости `\pgfooclass`. Атрибуты могут изменяться (только) методами. Когда атрибут изменяется, это изменение не является локальным к текущей Т_ЕX-группе, а сохраняется до конца жизни объекта или пока атрибут не измениться еще раз.

`\attribute<attribute name>=<initial value>;`

Объявляет атрибут с именем `<attribute name>`. Это имя, как имя метода или класса, может быть произвольным, но не должно содержать точек. Для атрибута можно определить `<initial value>`; если ничего не задается, автоматически используется пустая строка. Начальное значение — это то значение, которое атрибут будет иметь только после того, как объект создан, но до вызова конструктора.

```

\pgfooclass{stamp}
{ \attribute text;
  \attribute rotation angle = 20;
  \method stamp(#1) { \pgfooset{text}{#1} }
  \method apply(#1,#2) { \pgfoothis.shift origin(#1,#2)
    \node [rotate=\pgfoovalueof{rotation angle},font=\huge]
    {\pgfoovalueof{text}}; % Поставить оттиск }
  \method shift origin(#1,#2) {..... }
  \method set rotation (#1) { \pgfooset{rotation angle}{#1} }
}

```

Атрибуты могут устанавливаться и читаться только внутренними методами, этого нельзя сделать, используя дескриптор объекта. Говоря в терминах традиционного объектно-ориентированного программирования, атрибуты всегда являются частными. Нужен метод установки (set-метод) и чтения атрибута (get-метод).

Для установки атрибута используется специальный макрос, который может использоваться только внутри тела метода.

`\pgfooset{<attribute>}{<value>}`

Устанавливает `<attribute>` текущего объекта равным `<value>`.

```
\method set rotation (#1){\pgfooset{rotation angle}{#1}}
```

`\pgfoolet{<attribute>}{<macro>}`

Устанавливает, используя T_EX-команду `\let`, `<attribute>` текущего объекта равным текущему значению макроса `<macro>`.

```
\method foo () {\pgfoolet{my func}\myfunc}
```

% Последующее изменение `\myfunc` не влияет на значение атрибута `my func`.

`\pgfoovalueof{<attribute>}`

Расширяется (в конечном счете) до текущего значения `<attribute>` текущего объекта.

```
\method apply(#1,#2) {\pgfoothis.shift origin(#1,#2)
  \node [rotate=\pgfoovalueof{rotation angle},font=\huge]
  {\pgfoovalueof{text}}; }
```

`\pgfooget{<attribute>}{<macro>}`

Читает текущее значение `<attribute>` и сохраняет результат в `<macro>`.

```
... \method get rotation (#1) {\pgfooget{rotation angle}{#1}}
... \mystamp.get rotation(\therotation)
    “\therotation” is now “20” (or whatever).
```

29.7 Тожественность объектов

Каждый объект имеет уникальную идентичность, которая определяется просто целым числом — идентификатором. Можно восстановить идентификатор объекта (`<id>`), используя метод `get id` (см. ниже), но обычно в этом нет необходимости, поскольку сам `id` не может использоваться для того, чтобы получить доступ к объекту. Доступ к объектам следует получать через их методы, а они, в свою очередь, могут вызываться через дескриптор объекта.

Дескриптор объекта можно создать четырьмя способами:

1. Вызов `\pgfoonew<object handle>= ...` делает `<object handle>` дескриптором вновь созданного объекта.

2. Используя `\let`, чтобы создать псевдоним дескриптора существующего объекта: если `\mystamp` — дескриптор, говоря `\let\myotherstamp =\mystamp`, можно создать второй дескриптор для того же объекта.

3. Может использоваться `\pgfooobj{<id>}` как дескриптор для объекта с заданным `<id>`.

4. Используя метод `get handle` (см. далее), создать дескриптор для данного объекта.

Рассмотрим последние два метода.

`\pgfooobj{<id>}`

При условии, что `<id>` — идентификатор существующего объекта (еще живого объекта), вызов этой команды возвращает его дескриптор. После этого дескриптор может использоваться, чтобы вызывать методы:

```

\pgfoonew \mystamp=new stamp() % Создать новый объект.
\mystamp.get id(\myid)          % Получить id объекта и сохранить в \myid:
% Следующие два запроса дают тот же результат:
\mystamp.apply(1,1)
\pgfooobj{\myid}.apply(1,1)

```

Метод `get id` может использоваться, чтобы восстановить идентификатор объекта. Этот метод переопределяется для каждого класса, но не следует пробовать определять метод с таким именем самостоятельно.

get id < macro >

Вызов `<obj.get id (<macro>)` сохранит идентификатор объекта `<id>` в макросе `<macro>`.

Единственный способ использовать позже восстановленный идентификатор объекта состоит в том, чтобы вызвать `\pgfooobj`.

Разные «живые» объекты (которые еще находятся в пределах области видимости, в которой были созданы) будут всегда иметь разные идентификаторы, поэтому идентификаторы можно использовать для проверки равенства объектов. Однако, после разрушения объекта, тот же `<id>` может использоваться для другого вновь созданного объекта.

Вот типичное приложение, в котором следует вызывать этот метод: нужно собрать список объектов, для которых время от времени надо вызывать конкретный метод. Для процесса сбора объектов можно использовать макрос `\addtoobjectlist`, который принимает дескриптор объекта в качестве параметра. Дескриптор легко где-то сохранить, но дескриптор — это всего лишь дескриптор. Как правило, после вызова `\addtoobjectlist` дескриптор или не будет доступен, или не будет существовать, даже при том, что объект все еще существует. В этом случае, желательно сохранить где-нибудь еще идентификатор объекта вместо дескриптора. Таким образом, для объекта, переданного методу `\addtoobjectlist` надо еще вызвать метод `get id` и сохранить полученный в результате идентификатор `<id>`, а не дескриптор.

Чтобы создать дескриптор объекта, также может использоваться еще один предопределенный метод

get handle({<macro name>})

Вызов этого метода для объекта заставит `<macro name>` стать дескриптором для данного объекта. Для любого дескриптора объекта `\obj` кроме `\pgfoothis` следующие два выражения дадут один и тот же результат:

(1) `\let<macro name> =\obj` (2) `\obj.get handle(<macro name>)`

Первый метод проще и быстрее. Однако, у `\pgfoothis` есть отличие: вызов

```
\pgfoothis.get handle(<macro name>)
```

заставит `<macro name>` стать дескриптором для текущего объекта и сохранится, чтобы быть им даже после того, как метод выполнен. Для сравнения, вызов

```
\let<macro name>=\pgfoothis
```

заставит `\obj` быть тем же, что и специальный макрос `\pgfoothis`, таким образом, `\obj` будет всегда обращаться к текущему объекту, который может изменяться с течением времени.

29.8 Класс signal

Объектно-ориентированный модуль определяет, в дополнение к основному механизму для определения и использования классов и объектов, один класс: `signal` (сигнал), который реализует так называемый механизм «сигнал-слот». Идея состоит в следующем: время от времени происходят вещи, о которых нужно сообщать множеству других объектов. Могут происходить

разные вещи и ими будут интересоваться разные объекты. Объект `signal` может использоваться для того, чтобы сообщать о том, что произошла такая специальная вещь. Например, объект `signal` мог бы использоваться для того, чтобы сообщать о том, что «страница была послана».

Объекты могут настраиваться на сигнал. Они делают это, связывая один из своих методов (называемый слотом) с сигналом. После этого, всякий раз, когда испускается сигнал, вызывается связанный с ним метод объекта (ов).

Разные объекты могут связывать с одним и тем же сигналом разные слоты. В настоящее время, нельзя разорвать связь слота с сигналом, то есть, как только объект имеет связь с сигналом, он продолжает получать сообщение от этого сигнала до конца жизни сигнала. Это верно даже тогда, когда объект уже не существует, а сигнал существует. Таким образом, сигнальные объекты всегда должны создаваться перед объектами, которые их слушают.

Метод-конструктор сигнала `signal()` ничего не делает. Метод

`connect(<object handle>, <method name>)`

получает в качестве параметров дескриптор объекта и имя метода этого объекта. Он будет поставлен в очередь пар «объект-метод» во внутреннем списке и каждый раз, когда что-то испускает сигнал, вызывается указанный метод объекта. Внимание: нельзя передавать `\pgfoothis` в качестве дескриптора объекта. Это заставит объект-сигнал соединиться с самим собой. Если нужно соединить сигнал с методом текущего объекта, сначала следует создать псевдоним, используя метод `get handle`:

```
\pgfooclass{some class}
  { \method some class() {\pgfoothis.get handle(\me)
    \somesignal.connect(\me,foo)
    \anothersignal.connect(\me,bar) }

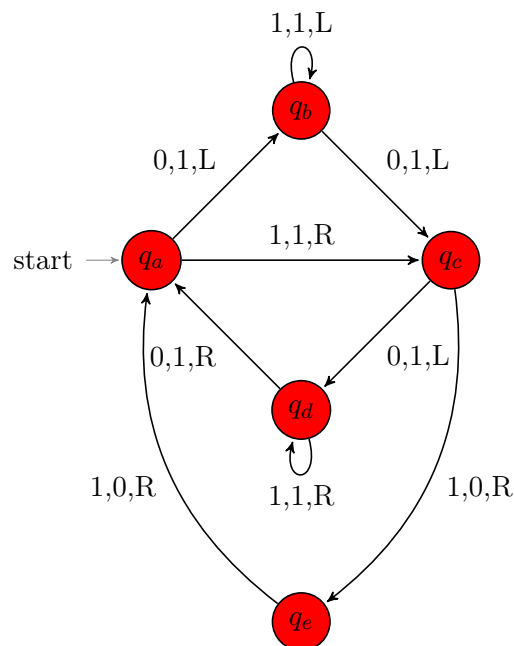
    \method foo () {}
    \method bar (#1,#2) {} }
\pgfoonew \objA=new some class()
\pgfoonew \objB=new some class()
```

Метод `emit(<arguments>)` испускает сигнал ко всем подключенным слотам. Это означает, что для всех объектов, которые были связаны через вызов `connect`, метод (слот), который был определен во время этого вызова, вызывается с заданными аргументами `<arguments>`.

```
\anothersignal.emit(1,2) % вызовет \objA.bar(1,2) и \objB.bar(1,2)
```


Литература

- [1] Tantau T. The TikZ and PGF Packages. Manual for version 2.10 — Institut für Theoretische Informatik, Universität zu Lübeck. — October 25, 2010 (<http://sourceforge.net/projects/pgf>).
- [2] Кнут Д. Е. Все про $\text{T}_\text{E}\text{X}$. — Протвино: РД $\text{T}_\text{E}\text{X}$, 1993. — 592 с.: ил.
- [3] Гуссенс, М. Путеводитель по пакету $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ и его расширению $\text{L}_\text{A}\text{T}_\text{E}\text{X}2_\epsilon$. Пер. с англ. — М.: «Мир», 1999. — 606 с., илл.
- [4] Львовский С. М. Набор и вёрстка в системе $\text{L}_\text{A}\text{T}_\text{E}\text{X}$, 3-е изд., испр. и доп. — М.: МЦНМО, 2003. — 448 с.
- [5] Котельников И. А., Чеботаев П. З. $\text{L}_\text{A}\text{T}_\text{E}\text{X}$ по-русски, 3-е изд., перераб. и доп. — Новосибирск: «Сибирский хронограф», 2004. — 496 с.



Оглавление

I	Основные библиотеки TikZ&PGF	3
1	Библиотека automata	5
1.1	Рисунок конечного автомата	5
1.2	Состояния с выводом и без вывода данных	6
1.3	Начальные и поглощающие состояния	7
1.4	Примеры	8
2	Библиотека backgrounds	11
3	Библиотека calendar	15
3.1	Команда <code>\calendar</code>	15
3.1.1	Создание простого списка дней	21
3.1.2	Добавление метки месяца	21
3.1.3	Создание понедельного списка	21
3.1.4	Создание помесячного списка дней	22
3.2	Размещение дней календаря	22
3.3	Метки месяцев	24
3.4	Примеры	26
4	Библиотека chains	29
4.1	Начало и продолжение цепочки	29
4.2	Узлы в цепочке	31
4.3	Соединение узлов в цепочке	34
4.4	Ветви	34
5	Библиотеки схем и цепей	36
5.1	Введение	36
5.1.1	Первый пример	36
5.1.2	Символы	37
5.1.3	Графика символов	37
5.1.4	Аннотации	39
5.2	Основная библиотека для схем	40
5.2.1	Размер символа	40
5.2.2	Объявление новых символов	41
5.2.3	Размещение символов в правильном направлении	41
5.2.4	Информационные метки	42
5.2.5	Объявление и использование аннотаций	44
5.2.6	Графический образ символа	45
5.3	Логические схемы	46
5.3.1	Библиотека форм логических вентилях	49
5.3.2	Библиотека форм американских логических вентилях	51
5.3.3	Библиотека форм логических ИЕС-вентилей	51
5.4	Электротехнические схемы	53

5.4.1	Основные библиотеки	53
5.4.2	Предопределенные ee-символы библиотеки IES	56
6	Библиотеки декорирования	57
6.1	Обзор библиотек и общие опции	57
6.2	Декорации трансформации пути	58
6.2.1	Декорации, использующие только прямые	58
6.2.2	Декорации, использующие кривые	59
6.3	Декорации, заменяющие путь	61
6.4	Маркировка декораций	63
6.4.1	Метки общего характера	63
6.4.2	Метки в виде стрелок	68
6.4.3	Метки в виде следа	68
6.4.4	Формы фоновых меток	70
6.5	Текстовые декорации	75
6.6	Рекурсивные декорации	79
7	Библиотека для ER-диаграмм	80
7.1	Сущности	80
7.2	Связи	81
7.3	Атрибуты	81
8	Библиотека fit	83
9	Библиотека lindenmeyersystems	86
9.1	Объявление L-системы	87
9.2	Использование L-систем	89
9.2.1	L-системы в PGF	89
9.2.2	L-системы в TikZ	90
10	Библиотека matrix	92
10.1	Матрица узлов	92
10.2	Особенности команды <code>\</code>	94
10.3	Разделители	95
11	Библиотека mindmap	97
11.1	Стиль для понятийных диаграмм	97
11.2	Понятийные узлы	98
11.2.1	Изолированные понятия	98
11.2.2	Понятия в деревьях	99
11.3	Связи между понятиями	101
11.3.1	Простые связи	101
11.3.2	Декорирование линии связи	102
11.3.3	Дуги дерева понятий	104
11.4	Добавление аннотаций	105
12	Библиотека folding	107
13	Библиотека patterns	110
13.1	Шаблоны, содержащие только форму	110
13.2	Окрашенные шаблоны	111

14 Библиотека petri	112
14.1 Места	112
14.2 Переходы	112
14.3 Лексемы	113
14.4 Примеры	116
15 Библиотека plotmarks	119
16 Библиотека positioning	121
17 Библиотека shadings	127
18 Библиотека shadows	130
18.1 Опция general shadow	130
18.2 Тени для произвольных путей и форм	131
18.2.1 Опция drop shadow	131
18.2.2 Опция copy shadow	132
18.3 Тени для специальных путей и узлов	132
19 Библиотеки форм	134
19.1 Предопределенные формы	134
19.2 Геометрические формы	134
19.3 Символьные формы	141
19.4 Формы в виде стрелок	145
19.5 Формы для нескольких текстовых частей	148
19.6 Формы для меток-идентификаторов	150
19.7 Библиотека специальных форм	153
20 Библиотека spy	156
20.1 Увеличение части изображения	156
20.2 Окружение spy score	157
20.3 Команда spy	158
20.4 Предопределенные spy-стили	160
20.5 Примеры	161
21 Библиотека topaths	162
21.1 Прямые линии	162
21.2 Перемещение	162
21.3 Кривые	162
21.4 Циклы	165
22 Библиотека through	167
23 Библиотека trees	168
23.1 Функции роста	168
23.2 Дуги от корневого узла	170
24 Библиотека turtle	171
II Некоторые утилиты	173
25 Пакет pgffor	175

26	Даты и календари	181
26.1	Обработка дат	181
26.1.1	Преобразования между типами дат	181
26.1.2	Проверка дат	182
26.1.3	Вывод дат	183
26.1.4	Локализация	183
26.2	Вывод календарей	183
27	Управление страницами	186
27.1	Основное использование	186
27.2	Предопределенные схемы размещения	187
28	Расширенная поддержка цвета	190
III	Объектно-ориентированное программирование	192
29	ООП и ТЭХ	194
29.1	Краткий обзор	194
29.2	Пример: класс <code>stamp</code>	194
29.3	Классы	195
29.4	Объекты	195
29.5	Методы	196
29.6	Атрибуты	197
29.7	Тождественность объектов	198
29.8	Класс <code>signal</code>	199
	Литература	201