

Ti*k*Z
L!KΣ

&
&

PGF
BGE

Manual for Version 3.1.5b

MANUAL for Version 3.1.5b

```
\begin{tikzpicture}
\coordinate (front) at (0,0);
\coordinate (horizon) at (0,.31\paperheight);
\coordinate (bottom) at (0,-.6\paperheight);
\coordinate (sky) at (0,.57\paperheight);
\coordinate (left) at (-.51\paperwidth,0);
\coordinate (right) at (.51\paperwidth,0);

\shade [bottom color=white,
        top color=blue!30!black!50]
        ([yshift=-5mm]horizon -| left)
        rectangle (sky -| right);

\shade [bottom color=black!70!green!25,
        top color=black!70!green!10]
        (front -| left) -- (horizon -| left)
        decorate [decoration=random steps] {
            -- (horizon -| right) }
        -- (front -| right) -- cycle;

\shade [top color=black!70!green!25,
        bottom color=black!25]
        ([yshift=-5mm-1pt]front -| left)
        rectangle ([yshift=1pt]front -| right);

\fill [black!25]
        (bottom -| left)
        rectangle ([yshift=-5mm]front -| right);

\def\nodeshadowed[#1]#2;{
\node[scale=2,above,#1]{
\global\setbox\mybox=\hbox{#2}
```

```
\nodeshadowed [at={(-5,8 )},yslant=0.05]
{\Huge Ti\textcolor{orange}{\emph{k}}Z};
\nodeshadowed [at={( 0,8.3)}]
{\huge \textcolor{green!50!black!50}{\&}};
\nodeshadowed [at={( 5,8 )},yslant=-0.05]
{\Huge \textsc{PGF}};
\nodeshadowed [at={( 0,5 )}]
{Manual for Version \pgftypesetversion};

\foreach \where in {-9cm,9cm} {
\nodeshadowed [at={(\where,5cm)}] {\tikz
\draw [green!20!black, rotate=90,
1-system={rule set={F -> FF-[-F+F]+[+F-F]},
axiom=F, order=4,step=2pt,
randomize step percent=50, angle=30,
randomize angle percent=5}] 1-system; }}

\foreach \i in {0.5,0.6,...,2}
\fill
[white,opacity=\i/2,
decoration=Koch snowflake,
shift=(horizon),shift={(\rand*11,rnd*7)},
scale=\i,double copy shadow={
opacity=0.2,shadow xshift=0pt,
shadow yshift=3*\i pt,fill=white,draw=none}]
decorate {
decorate {
decorate {
(0,0)- ++(60:1) -- ++(-60:1) -- cycle
} } };

\node (left text) ...
```

Für meinen Vater, damit er noch viele schöne T_EX-Graphiken erschaffen kann.
For my father, so that he can still create lots of nice T_EX graphics.
为了我父亲，这样他仍然可以创建很多漂亮的 T_EX 图形。

Till

Copyright 2007 to 2013 by Till Tantau

Permission is granted to copy, distribute and/or modify *the documentation* under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

Permission is granted to copy, distribute and/or modify *the code of the package* under the terms of the GNU Public License, Version 2 or any later version published by the Free Software Foundation. A copy of the license is included in the section entitled GNU Public License.

Permission is also granted to distribute and/or modify *both the documentation and the code* under the conditions of the LaTeX Project Public License, either version 1.3 of this license or (at your option) any later version. A copy of the license is included in the section entitled L^AT_EX Project Public License.

Translation Copyright Copyright © 2020 BY WANG HAILIN¹. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

¹Tongji University, School of Civil Engineering, Department of Geotechnical Engineering, E-mail: 2011133@tongji.edu.cn, GitHub Homepage: <https://github.com/Hailin-Jing/>

The TikZ and PGF Packages

Manual for version 3.1.5b

<https://github.com/pgf-tikz/pgf>*

Till Tantau[†]

Institut für Theoretische Informatik

Universität zu Lübeck

November 21, 2020

目录

1	简介	4
1.1	TikZ 系统下的层级	4
1.2	与其它图形宏包的比较	5
1.3	工具包	5
1.4	如何阅读本手册	6
1.5	作者与致谢	6
1.6	获取帮助	6
I	教程和准则	7
2	教程：卡尔学生的图片	8
2.1	问题描述	8
2.2	配置环境	8
2.2.1	在 L ^A T _E X 中配置环境	8
2.2.2	在 Plain T _E X 中配置环境	9
2.2.3	在 ConT _E Xt 中配置环境	10
2.3	直线路径的创建	10
2.4	曲线路径的创建	10
2.5	圆路径的创建	11
2.6	矩形路径的创建	12
2.7	网格路径的创建	12
2.8	增加一点风格	13
2.9	绘图选项	13
2.10	弧线路径的创建	14
2.11	剪切路径	15
2.12	抛物线和正弦线路径构造	15
2.13	填充和绘图	16

*本翻译文档的项目地址：<https://github.com/Hailin-Jing/PGFManual-zh-cn>

[†]原文档的编辑。文档的部分内容由其他作者撰写，如在该处进行了标注以及如1.5节所示的部分或章节。

2.14	阴影	16
2.15	指定坐标	17
2.16	相交路径	18
2.17	添加箭头	19
2.18	分组	20
2.19	坐标变换	20
2.20	重复工作: For 循环	21
2.21	添加文本	22
2.22	Pics: 再访 Angle	26
3	教程: 哈根的 Petri 网	27
3.1	问题陈述	27
3.2	配置环境	27
3.2.1	在 L ^A T _E X 中配置环境	27
3.2.2	在 Plain T _E X 中配置环境	28
3.2.3	在 ConT _E Xt 中配置环境	28
3.3	节点简介	28
3.4	使用 At 语法放置节点	29
3.5	使用样式	29
3.6	节点尺寸	30
3.7	节点命名	30
3.8	使用相对位置放置节点	31
3.9	在节点旁添加标签	31
3.10	连接节点	33
3.11	在线旁添加标签	35
3.12	添加隐藏线和多行文本	36
3.13	使用图层: 背景矩形	37
3.14	完整的代码	37
4	教程: 欧几里得的琥珀版本的几何原本	40
4.1	书 I, 命题 I	40
4.1.1	配置环境	40
4.1.2	线段 AB	41
4.1.3	以点 A 为圆心的圆	41
4.1.4	圆的交集	43
4.1.5	完整代码	44
4.2	书 I, 命题 II	45
4.2.1	使用分割运算进行点 D 的构造	46
4.2.2	线段与圆相交	47
4.2.3	完整代码	48
5	教程: 简单的图表	50
5.1	为节点设置样式	50
5.2	使用定位选项对齐节点	52
5.3	使用矩阵对齐节点	54
5.4	作为图形的图解	56
5.4.1	连接已经定位的节点	56
5.4.2	使用 Graph 命令创建节点	57



6	教程：约翰内斯的演讲图	61
6.1	问题描述	61
6.2	树状图简介	61
6.3	创建讲义图	64
6.4	添加讲义注释	69
6.5	添加背景	71
6.6	添加日历	72
6.7	完整代码	73
7	绘图准则	78
7.1	规划创建的图形所需要的时间	78
7.2	创建图形的工作流程	78
7.3	将图形与正文紧密结合	79
7.4	图形和文本之间的一致性	79
7.5	图形中的标签	80
7.6	图表	80
7.7	注意力和注意力分散	83
II	安装与配置	85
8	安装	87
8.1	宏包和驱动程序版本	87
8.2	安装预捆绑的软件包	87
8.2.1	Debian	88
8.2.2	MiKTeX	88
8.3	在 texmf 目录中安装	88
8.3.1	可以将所有内容整合在一起的安装	88
8.3.2	适用于 TDS 的安装	88
8.4	更新安装	88
9	许可证和版权	89
9.1	必需遵守哪些许可证?	89
9.2	GNU 通用公共许可证, V2	89
9.2.1	前言	89
9.2.2	复制、分发与修改的条款与条件	90
9.2.3	无担保声明	92
9.3	L ^A T _E X 项目公共许可证 (LPPL), V1.3c 2006-05-20	92
9.3.1	导言	92
9.3.2	定义	92
9.3.3	分发和修改的条件	93
9.3.4	无担保声明	94
9.3.5	作品的维护	94
9.3.6	该许可证允许下能否和如何分发	94
9.3.7	选择这个或其它许可证	95
9.3.8	不进行分发时的修改的建议	95
9.3.9	如何使用该许可证	95
9.3.10	不可替代的衍生作品	95

9.3.11	重要的建议	96
9.4	GNU 自由文档许可证, V1.2, 2002 年 11 月	96
9.4.1	引言	96
9.4.2	效力与定义	96
9.4.3	逐字的复制	97
9.4.4	大量地复制	97
9.4.5	修改	98
9.4.6	组合文件	99
9.4.7	文件的收集	99
9.4.8	独立作品的聚集	99
9.4.9	翻译	99
9.4.10	终止	99
9.4.11	本授权的未来改版	100
9.4.12	授权附录: 如何使用本授权用于你的文件	100
10	支持的格式	101
10.1	支持的输入格式: L ^A T _E X, Plain T _E X, ConT _E Xt	101
10.1.1	使用 L ^A T _E X 格式	101
10.1.2	使用 Plain T _E X 格式	101
10.1.3	使用 ConT _E Xt 格式	101
10.2	支持的输出格式	102
10.2.1	选择后端驱动	102
10.2.2	输出 PDF 格式的文件	102
10.2.3	输出 PostScript 格式的文件	103
10.2.4	输出 SVG 格式的文件	104
10.2.5	输出完美便携式的 DVI 文件	105
III	TikZ 不是绘图程序	106
11	设计原则	107
11.1	用于指定点的特殊语法	108
11.2	用于指定路径的句法	108
11.3	路径上的动作	108
11.4	图形参数的键值语法	108
11.5	用于指定 <code>node</code> 的特殊语法	109
11.6	用于指定 <code>tree</code> 的特殊语法	109
11.7	用于指定 <code>graph</code> 的特殊语法	110
11.8	图形参数的分组	110
11.9	坐标转换系统	111
12	层次结构: 宏包, 环境, 分组和样式	112
12.1	加载宏包和库	112
12.2	创建图片	112
12.2.1	使用环境创建图片	112
12.2.2	使用命令创建图片	114
12.2.3	处理类别代号和 Babel 宏包	115
12.2.4	添加背景	115

12.3	使用分组构造图片	115
12.3.1	Scope 环境	115
12.3.2	Scope 环境的简写形式	116
12.3.3	单命令分组	117
12.3.4	在路径内部使用分组	117
12.4	使用图形选项	117
12.4.1	图形选项是如何处理的	117
12.4.2	使用样式来管理图片的外观	118
IV	库	120
13	思维导图图形库	122
13.1	概述	122
13.2	思维导图样式	122
13.3	概念节点	123
13.3.1	孤立的概念	123
13.3.2	树中的概念	125
13.4	连接概念	127
13.4.1	简单连接	127
13.4.2	圆形带连接	128
13.4.3	圆形连接带的 To-Path	129
13.4.4	树的边	130
13.5	添加注释	132
V	数据可视化	133
14	数据可视化简介	134
14.1	概念：数据点	134
14.2	概念：可视化管道	134

1 简介

欢迎使用 TikZ 和底层 PGF 系统的文档。TikZ 最初是一种小型 L^AT_EX 样式，可以直接用 pdfL^AT_EX 在我 (Till Tantau) 的博士论文中创建图形，现在已经发展成为一种功能强大的图形语言，其手册超过一千页。TikZ 提供的丰富选择通常令初学者望而却步；但是幸运的是，本文档随附了一些节奏缓慢的教程，这些教程将教您几乎所有有关 TikZ 的知识，而无需阅读其余内容。

我希望首先从“什么是 TikZ？”这个问题开始，它基本上只定义了许多绘制图形的 T_EX 命令。例如，代码 `\tikz\draw(0pt,0pt) --(20pt,6pt);` 产生线条 ，代码 `\tikz \fill[orange] (1ex,1ex) circle (1ex);` 产生圆 。从某种意义上说，当您使用 TikZ 时，您将对图形进行“编程”，使用 T_EX 时就像对文档进行“编程”一样。这也解释了名称：TikZ 是“textsc gnu's Not Unix”传统的递归首字母缩写，意思是“TikZ ist *kein* Zeichenprogramm”，翻译为“TikZ 不是一种绘图程序”，提醒读者注意什么。使用 TikZ，您可以获得图形的“T_EX 排版设置方法”的所有优势：快速创建简单图形，精确定位，宏的使用通常为了更为出色的排版。您还会继承所有缺点：学习曲线陡峭，没有 WYSIWYG，小的更改需要很长的重新编译时间，并且代码并没有真正“展示”事物的外观。

现在我们知道 TikZ 是什么了，那么“PGF”呢？如前所述，TikZ 最初是一个实现 T_EX 图形宏的项目，该宏既可以与 pdf L^AT_EX 一起使用，也可以与经典的（基于 PostScript 的）L^AT_EX 一起使用。换句话说，我想为 T_EX 实现一种“便携式图形格式”，因此命名为 PGF。这些早期的宏仍然存在，它们构成了本手册中描述的系统的“基础层”，但是作者如今的大部分交互都是与 TikZ 交互的，它已成为其自身的完整语言。

1.1 TikZ 系统下的层级

事实上 TikZ 系统下有 2 个层级：

系统层：该层提供了“驱动程序”中正在发生的事情的完整抽象。驱动程序是 dvips 或 dvipdfm 之类的程序，需要一个 .dvi 文件作为输入并生成 .ps 或 .pdf 文件。（pdf_{tex} 程序也算作驱动程序，即使它没有输入 .dvi 文件。也没关系。）每个驱动程序都有自己的语法来生成图形，这使每个想要以可移植方式创建图形的人都感到头疼。PGF 的系统层“抽象”了这些差异。例如，系统命令 `\pgfsys@lineto{10pt}{10pt}` 将当前路径扩展到当前 `{pgfpicture}` 的坐标 10pt,10pt。使用 dvips, dvipdfm 或 pdf_{tex} 处理文档时，系统命令将转换为不同的 `\special` 命令。由于每个附加命令将使 PGF 移植到新驱动程序的工作量更大，因此系统层应尽可能“简化”。作为用户，您不会直接使用系统层。

基础层：基础层提供了一组基本命令，这些命令使您可以比直接使用系统层更容易地生成复杂的图形。例如，系统层不提供用于创建圆的命令，因为圆可以由更基本的贝塞尔曲线组成（当然，基本是这样）。但是，作为用户，您将需要一个简单的命令来创建圆（至少我要这样做），而不必写下半页的贝塞尔曲线来生成坐标。因此，基础层提供了命令 `\pgfpathcircle` 为您生成必要的曲线坐标。

基础层由一个 *core* 组成，该 *core* 由几个只能相互加载的相互依赖的包 *en bloc* 组成，另外的 *modules* 通过诸如节点管理之类的更多专用命令扩展了核心或绘图接头。例如，BEAMER 包仅使用核心，而不使用 *shapes* 模块。

从理论上讲，TikZ 本身只是几种可能的“前端”之一。是一类命令集或特殊语法，使得使用基础层更加容易。直接使用基础层的问题是为编写代码通常太“冗长”。例如，要绘制一个简单的三角形，使用基础层时可能需要多达五个命令：一个用于在三角形的第一个角点处开始绘制路径，一个用于将路径扩展到第二个角点处，一个用于将路径扩展到第二个角点。第三个命令用于闭合路径，另一个用于实际绘制三角形（而不是填充三角形）。使用 TikZ 前端，所有这些都归结为一个简单的类似于 METAFONT 的命令：

```
\draw(0,0)--(1,0)--(1,1)--cycle;
```

实际上，TikZ 是 PGF 的唯一“正经”的前端。它使您可以访问 PGF 的所有功能，但旨在使其易于使用。它的语法是 METAFONT 和 PSTricks 以及我自己的一些想法的混合体。事实上也有 TikZ 之外还有其他前

端，但它们更多地被用作“技术研究”，而不是 TikZ 的正经替代品。特别是 `pgfpict2e` 前端重新实现标准 \LaTeX `{picture}` 环境和命令，例如 `\line` 或 `\vector` 使用 PGF 基础层。由于 `pict2e.sty` 宏包在重新实现上至少与 `{picture}` 环境表现得同样出色，这一层并不是真正的“必需”的。而是，此程序包背后的想法是简单演示如何实现前端。

由于大多数用户只会使用 TikZ，几乎没有人会直接去使用系统层，因此本手册的第一部分主要涉及 TikZ；最后再来说明基础层和系统层。

1.2 与其它图形宏包的比较

TikZ 并不是 \TeX 的唯一图形宏包。下面我将尝试将 TikZ 与其他宏包进行合理的比较。

1. 标准 \LaTeX `{picture}` 环境允许您创建简单的图形，但仅此而已。这当然不是由于 \LaTeX 设计师缺乏知识或想象力造成的。相反，这是为 `{picture}` 环境的可移植性付出的代价：它与所有后端驱动程序一起使用。
2. `pstricks` 软件包肯定足够强大，可以创建任何可能的图形，但是它并不是真正可移植的。最重要的是，它不适用于 `pdftex` 也不使用任何其他产生 PostScript 代码以外的驱动程序。与 TikZ 相比，`pstricks` 具有类似的基础支持。在过去十年中，用户为特殊用途提供了许多不错的额外程序包。TikZ 语法比 `pstricks` 更一致。像 TikZ 这样的语法是“以更集中的方式开发的”，并且“在心里清楚 `pstricks` 的缺点”。
3. `xypic` 宏包是用于创建图形的较旧的宏包。然而，由于语法和文档有点晦涩难懂，因此使用和学习起来更加困难。
4. `dratex` 宏包包是用于创建图形的小型图形软件包。与包括 TikZ 在内的其他宏包相比，它非常小，可能有优势也可能没有优势。
5. `metapost` 程序是 TikZ 的强大替代方案。它曾经是一个外部程序，并且带来了很多问题，但现在已内置在 \LuaTeX 中。`metapost` 的障碍是包含标签。。它比使用 PGF 更容易实现。
6. 对于不希望使用 TikZ 或上面的其他宏包包进行“编程”的用户，`xfig` 程序是 TikZ 的一种重要替代方案。有一个转换程序可以将 `xfig` 图形转换为 TikZ。

1.3 工具包

PGF 宏包附带了许多实用程序包，这些实用程序包实际上与创建图形无关，并且可以独立于 PGF 使用。但是，它们与 PGF 捆绑在一起，部分是出于方便，部分是因为其功能与 PGF 紧密相关。这些实用程序包是：

1. `pgfkeys` 软件包定义了强大的密钥管理工具。它可以完全独立于 PGF 来使用。
2. `pgffor` 包定义了有用的 `\foreach` 声明。
3. `pgfcalendar` 包定义用于创建日历的宏。通常，这些日历将使用 PGF 的图形引擎呈现，但是您可以使用 `pgfcalendar` 还可以使用普通文本排版日历。该软件包还定义了用于“处理”日期的命令。
4. `pgfpages` 包用于将多个页面组合成一个页面。它提供了用于将多个“虚拟页面”组合成一个“物理页面”的命令。这个想法是，每当 \TeX 有一个页面准备好“发货”时，`pgfpages` 中断此发货，而是将要发货的页面存储在特殊框中。当以这种方式积累了足够的“虚拟页面”时，它们会按比例缩小并排列在“物理页面”上，然后真正运出。这种机制允许您直接在 \LaTeX 内部创建文档的“一页一页”文档，而无需使用任何外部程序。但是，`pgfpages` 可以做的还不止这些。您可以使用它在页面上放置徽标和水印，在一页上最多打印 16 页，在页面上添加边框等等。

1.4 如何阅读本手册

本手册介绍了 TikZ 的设计及其用法。该组织大致是根据“用户友好性”来确定的。首先介绍最简单和最常用的命令和子包，然后再讨论更底层和深奥的功能。

如果尚未安装 TikZ，请先阅读安装说明。其次，阅读教程部分可能是一个好主意。最后，您可能希望略过 TikZ 的描述。通常，您无需阅读基础层上的章节。如果您打算编写自己的前端，或者希望将 PGF 移植到新驱动程序，则只需阅读系统层上的该部分。

全文中介绍了系统提供的“公共”命令和环境。在每个这样的描述中，所描述的命令，环境或选项均以红色打印。用绿色显示的文本是可选的，可以省略。

1.5 作者与致谢

PGF 系统及其文档的大部分由 Till Tantau 编写。主团队的另一位成员是 Mark Wibrow，他负责如 PGF 数学引擎，许多形状，装饰引擎和矩阵的编写。第三个成员是 Christian Feuersänger，他在手册中贡献了浮点库，图像外部化，扩展键处理和自动超链接。

此外，Christophe Jorssen, Jin-Hwan Cho, Olivier Binda, Matthias Schulz, Renée Ahrens, Stephan Schuster 和 Thomas Neumann 也做出了贡献。

此外，许多人通过写电子邮件，发现错误或发送库和补丁程序为 PGF 系统做出了贡献。非常感谢所有人这些人，这些人太多了，无法为所有人标注姓名！

1.6 获取帮助

当您需要有关 PGF 和 TikZ 的帮助时，请执行以下操作：

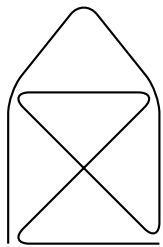
1. 阅读手册，至少是与您的问题有关的部分。
2. 如果那不能解决问题，请尝试查看 GitHub 上 PGF 和 TikZ 的开发页面（请参见本文档的标题）。也许有人已经报告了类似的问题，并且有人找到了解决方案。
3. 在网站上，您会找到许多论坛以获得帮助。在这里，您可以编写帮助讨论，文件错误报告，加入邮件列表等等。
4. 在提交错误报告（尤其是有关安装的错误报告）之前，请确保这确实是一个错误。特别要看一下你用 TeX 编译生成的 .log 文件。这个 .log 文件应显示所有正确的文件均已从正确的目录中加载。通过查看 .log 文件，几乎可以解决所有安装问题。
5. 作为最后的手段您可以尝试给我（Till Tantau）发送电子邮件，或者，如果问题与数学引擎有关，请发送电子邮件给 Mark Wibrow。我不介意收到电子邮件，我只是收到太多了。因此，我不能保证您的电子邮件会及时得到答复，甚至根本不会得到答复。如果您将邮件邮寄到 PGF 邮件列表，则解决问题的机会会更高（自然，我会阅读此列表并在有时间的时候回答问题）。

Part I

教程和准则

by Till Tantau

为了帮助您开始使用 TikZ，而不是冗长的安装和配置部分，本手册从教程开始。它们解释了系统的所有基本和一些更高级的功能，而没有涉及所有细节。本部分还包含一些有关使用 TikZ 创建图形时应如何进行操作



```
\tikz\draw[thick,rounded corners=8pt]
(0,0)--(0,2)--(1,3.25)--(2,2)--(2,0)--(0,2)--(2,2)--(0,0)--(2,0);
```

2 教程：卡尔学生的图片

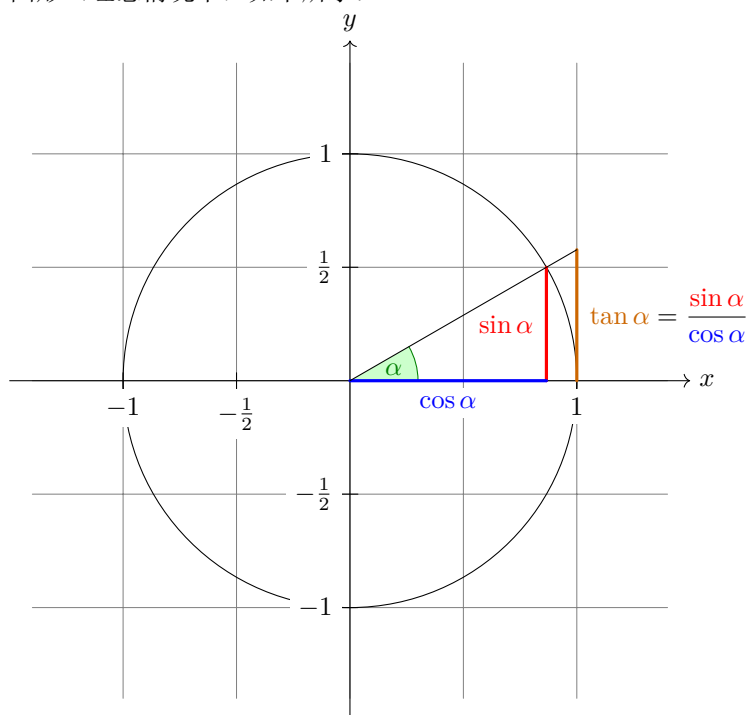
本教程适用于 TikZ 的新用户。它没有详尽介绍 TikZ 的所有功能，只是您可能立即使用的那些功能。

卡尔 (Karl) 是一位数学和化学高中老师。他曾经使用 \LaTeX 的 `{picture}` 环境在练习题和考试中创建图形。虽然结果是可以接受的，但是创建图形往往是一个漫长的过程。另外，角度略有错误的直线也容易出现，圆也似乎难以正确。自然，他的学生们并不在乎这些线是否具有正确的直角，并且无论画得多么好，他们都觉得卡尔的考试太难了。但是卡尔从未对结果完全满意。

卡尔的儿子对这样的结果并不满意 (毕竟他不必参加考试)，他告诉卡尔，他可能希望尝试一种用于制作图形的新的宏包。有点令人困惑的是，该宏包似乎有两个名称：首先，卡尔必须下载并安装一个名为 PGF 的宏包。然后事实证明，在该程序包中还有另一个名为 TikZ 的宏包，它应该表示 “TikZ ist *kein* Zeichenprogramm (TikZ 不是一个绘图程序)”。卡尔发现这一切都有些奇怪，TikZ 似乎表明该宏包无法满足他的需要。但是，使用 GNU 软件已经有一段时间了，并且 “GNU 不是 Unix”，似乎还有希望。他的儿子向他保证，TikZ 的名称旨在警告人们，TikZ 不是可用于使用鼠标或平板电脑绘制图形的程序。相反，它更像是一种 “图形语言”。

2.1 问题描述

卡尔希望在下次练习题中为他的学生添加一张图解。他目前正在向学生教授正弦和余弦。他想绘制的图形 (理想情况下) 如下所示：



The angle α is 30° in the example ($\pi/6$ in radians). The sine of α , which is the height of the red line, is

$$\sin \alpha = 1/2.$$

By the Theorem of Pythagoras we have $\cos^2 \alpha + \sin^2 \alpha = 1$. Thus the length of the blue line, which is the cosine of α , must be

$$\cos \alpha = \sqrt{1 - 1/4} = \frac{1}{2}\sqrt{3}.$$

This shows that $\tan \alpha$, which is the height of the orange line, is

$$\tan \alpha = \frac{\sin \alpha}{\cos \alpha} = 1/\sqrt{3}.$$

2.2 配置环境

在 TikZ 中绘制图片，在图片的开头，您需要告诉 \TeX 或 \LaTeX 您要开始绘制图片。在 \LaTeX 中，这是使用 `{tikzpicture}` 环境完成的，在 Plain \TeX 中，您只需使用 `\tikzpicture` 开始绘制图片以及使用 `\endtikzpicture` 结束绘制。

2.2.1 在 \LaTeX 中配置环境

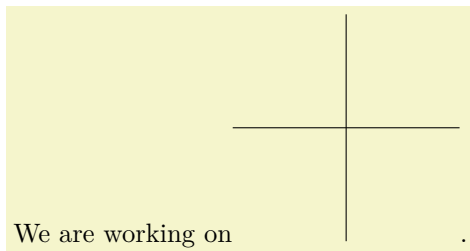
卡尔作为一个 \LaTeX 用户，他的文件如下：

```

\documentclass{article}%say
\usepackage{tikz}
\begin{document}
Weareworkingon
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\end{tikzpicture}.
\end{document}

```

执行后，即通过运行 `pdflatex` 或通过运行 `latex` 然后运行 `dvips`，结果将包含如下内容：



```

Weareworkingon
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\end{tikzpicture}.

```

诚然，这还不是全部，但我们确实已经确定了坐标轴。好吧，虽然不完全，但是我们有构成绘制坐标轴的线。卡尔突然感到那张图解还差得远。

让我们更详细地看一下代码。首先，包 `tikz` 已加载。该软件包是基本 PGF 系统的所谓的“前端”。本手册中也介绍了基础层，它更基础一些，因此也更难使用。前端通过提供更简单的语法使事情变得更容易。

在环境内部有两个 `\draw` 命令。它们的意思是：“应该绘制从命令到分号为止指定的路径。”第一个路径指定为 `(-1.5,0) -- (0,1.5)`，表示“从位置 `(-1.5,0)` 到位置 `(0,1.5)` 的直线”。在此，位置是在特殊坐标系中指定的，初始条件下，一个单位为 1 厘米。

卡尔非常高兴地注意到环境会自动保留足够的空间来容纳图片。

2.2.2 在 Plain T_EX 中配置环境

卡尔的妻子格达（Gerda）恰好也是一名数学老师，她不是 L^AT_EX 用户，而使用 Plain T_EX，因为她更喜欢以“旧方式”做事。她还可以使用 TikZ。她必须使用 `\input tikz.tex` 代替 `\usepackage{tikz}`，使用 `\tikzpicture` 代替 `\begin{tikzpicture}` 以及使用 `\endtikzpicture` 代替 `\end{tikzpicture}`。

因此，她将使用：

```

%%PlainTeXfile
\input tikz.tex
\baselineskip=12pt
\hsize=6.3truein
\vsize=8.7truein
Weareworkingon
\tikzpicture
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\endtikzpicture.
\bye

```

格达可以使用 `pdftex` 或者 `tex` 和 `dvips` 来排版这个文件。TikZ 将自动识别她正在使用哪个驱动程序。如果她希望同时使用 `dvipdfm` 和 `tex`，那么她需要修改文件 `pgf.cfg` 或可以在她输入 `tikz.tex` 或 `pgf.tex` 的位置之前添加 `\def\pgfsysdriver{pgfsys-dvipdfm.def}`。

2.2.3 在 ConTeXt 中配置环境

卡尔的叔叔汉森(Hans)使用 ConTeXt。与格达一样,汉森也可以使用 TikZ。他将使用 `\usemodule[tikz]` 代替 `\usepackage{tikz}`。使用 `\starttikzpicture` 代替 `\begin{tikzpicture}`, 使用 `\stoptikzpicture` 代替 `\end{tikzpicture}`。

他的示例版本如下所示:

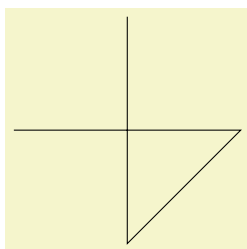
```
%%ConTeXtfile
\usemodule[tikz]

\starttext
Weareworkingon
\starttikzpicture
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\stoptikzpicture.
\stoptext
```

汉森现在将使用 `texexec` 或 `context`, 并且以通常的方式排版此文件。

2.3 直线路径的创作

TikZ 中所有图片的基本构造块是路径。一个路径是一系列相连的直线和曲线(这不是全部,但让我们暂时忽略复杂性)。通过将起始位置的坐标指定为方括号中的点来启动路径,如 $(0,0)$ 中所示。接下来是一系列“路径扩展操作”。最简单的是 `--`, 我们已经用过了。它必须跟随着另一个坐标,它将路径以直线延伸到这个新位置。例如,如果我们将坐标轴的两条路径变换为一条路径,会得到如下结果:



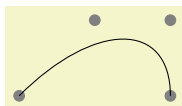
```
\tikz\draw(-1.5,0)--(1.5,0)--(0,-1.5)--(0,1.5);
```

卡尔对这里没有 `{tikzpicture}` 环境这一事实感到有点困惑。相反,使用小命令 `\tikz`。这个命令要么接受一个参数(以 `\tikz{\draw(0,0)--(1.5,0)}` 中的括号开始,这会产生_____),要么收集所有内容,直到下一个分号,并将其放到 `{tikzpicture}` 环境中。根据经验,所有 TikZ 图形绘制命令都必须作为 `\tikz` 的参数出现,或者出现在 `{tikzpicture}` 环境中。幸运的是,命令 `\draw` 将只在这个环境中定义,因此您不太可能在这里意外地出错。

2.4 曲线路径的创作

卡尔想做的下一件事是画圆。对于这个问题,直线显然是不行的。相反,我们需要一些绘制曲线的方法。为此,TikZ 提供了特殊的语法。需要一两个“控制点”。它们背后的数学原理并不简单,但基本的思想是:假设您在点 x , 第一个控制点是 y 。然后曲线将开始朝着 y 在 x 的方向,也就是说,曲线在 x 处的切线将指向 y 。接下来,假设曲线应该结束于 z , 第二个控制点是 w 。曲线会在 z 点结束曲线在 z 点的切线会经过 w 点。

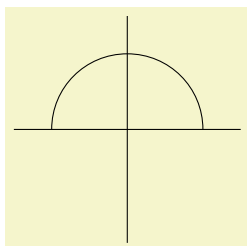
这是一个示例(为清楚起见添加了控制点):



```
\begin{tikzpicture}
\filldraw[gray] (0,0) circle[radius=2pt]
(1,1) circle[radius=2pt]
(2,1) circle[radius=2pt]
(2,0) circle[radius=2pt];
\draw(0,0)..controls(1,1)and(2,1)..(2,0);
\end{tikzpicture}
```

以“弯曲”方式扩展路径的一般语法是 `.. controls <第一个控制点> and <第二个控制点> ..`。您可以省略 `and` (<第二个控制点>)，这将使第一个控制点被使用两次。

因此，卡尔现在可以将前半圆添加到图片中：



```
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(-1,0)..controls(-1,0.555)and(-0.555,1)..(0,1)
..controls(0.555,1)and(1,0.555)..(1,0);
\end{tikzpicture}
```

卡尔对结果感到满意，但是发现以这种方式指定圆非常尴尬。幸运的是，有一种更简单的方法。

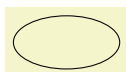
2.5 圆路径的创作

为了画一个圆，可以使用路径构建运算符 `circ`。此运算符后跟方括号中的半径，如以下示例所示：（请注意，运算符之前的位置用作圆的圆心。）




```
\tikz\draw(0,0)circle[radius=10pt];
```

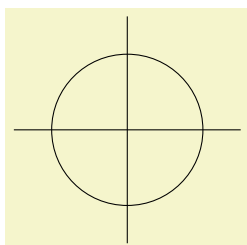
您也可以使用 `ellipse` 操作添加椭圆路径。您可以指定两个，而不是单个半径：



```
\tikz\draw(0,0)ellipse[x radius=20pt,y radius=10pt];
```

要绘制的轴线不是水平和竖直的，而是指向任意方向的椭圆（一个像这样旋转后的椭圆 ），您可以使用旋转，稍后将对此进行解释。顺便提一下，小椭圆的代码是 `\tikz \draw[rotate=30] (0,0) ellipse [x radius=6pt, y radius=3pt];`。

那么，回到卡尔的问题，他可以写 `\draw (0,0) circle [radius=1cm];` 画圆：

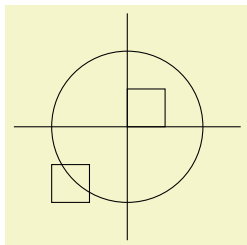


```
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\end{tikzpicture}
```

在这一点上，卡尔有点惊慌，圆是如此之小，他想要最后的图片大得多。他很高兴地了解到 `TikZ` 具有强大的变换选项，并且很容易将所有内容放大到原来的三倍。但是为了节省一些空间，让我们暂时保持大小不变。

2.6 矩形路径的创作

接下来我们想要的是背景中的网格。有几种方法可以产生它。例如，有人可能会画很多矩形。由于矩形非常常见，因此它们有一种特殊的语法：要向当前路径添加矩形，我们使用 `rectangle` 路径构造操作。该操作之后应该跟着另一个坐标，并将一个矩形添加到路径中，使前一个坐标和下一个坐标都是矩形的角点。所以，我们添加两个矩形的图片：




```
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\draw(0,0)rectangle(0.5,0.5);
\draw(-0.5,-0.5)rectangle(-1,-1);
\end{tikzpicture}
```

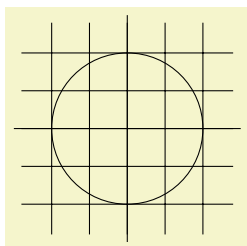
虽然这在其他情况下可能很好，但这并没有真正解决卡尔的问题：首先，我们需要大量的矩形，然后边界没有“封闭”。

因此，当卡尔得知有一个 `grid` 路径构造操作时，他打算使用漂亮的 `\draw` 命令简单地绘制四条垂直和四条水平线。

2.7 网格路径的创作

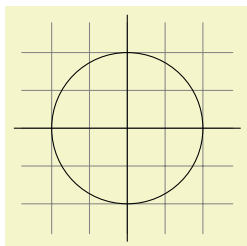
`grid` 路径操作向当前路径添加一个网格。它将添加组成网格的线，填充矩形，矩形的一个角点是当前点，另一个角点是紧跟 `grid` 操作的点。例如，代码 `\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` 生成 。请注意，可以使用 `\draw` 的可选参数来指定网格宽度（还有 `xstep` 和 `ystep` 来分别定义步长）。卡尔很快就会了解到，使用这些选项可以影响到很多事情。

对于卡尔来说，可以使用以下代码：



```
\begin{tikzpicture}
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\draw[step=.5cm](-1.4,-1.4)grid(1.4,1.4);
\end{tikzpicture}
```

再次查看所需的图片后，卡尔注意到，使网格的颜色更浅一点会很好。（他的儿子告诉他，如果不使网格的颜色变浅，网格往往会分散注意力。）为了使网格的颜色变浅，卡尔在 `\draw` 命令中添加了两个选项。首先，他使用 `gray` 颜色表示网格线。其次，他将线宽减小到 `very thin`。最后，他交换命令的顺序，以便首先绘制网格，然后再绘制其他所有内容。



```
\begin{tikzpicture}
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\end{tikzpicture}
```


2.8 增加一点风格

除了使用选项 `gray,very thin`, 卡尔也可以声明 `help lines`。Styles 是预定义的选项集, 可用于组织图形的绘制方式。通过声明 `help lines` 相当于您说“使用我(或其他人)设置的样式来绘制辅助线”。如果卡尔在以后的某个时刻决定应该绘制网格, 例如使用 `blue!50` 颜色代替 `gray`, 他可以在某处提供以下选项:

```
helplines/.style={color=blue!50,verythin}
```

这种“样式设置器”的作用是, 在当前分组或环境中 `help lines` 选项与 `color=blue!50,very thin` 的效果相同。

使用样式可使您的图形代码更加灵活。您可以以一致的方式更改事物看起来的方式。通常, 样式是在图片的开头定义的。但是, 有时您可能希望全局定义样式, 以便文档的所有图片都可以使用此样式。然后, 您可以通过更改一种样式轻松更改所有图形的外观。在这种情况下, 您可以在文档开头使用 `\tikzset` 命令, 如

```
\tikzset{help lines/.style=very thin}
```

要建立样式的层级结构, 您可以让一种样式使用另一种样式。因此, 为了定义基于 `grid` 风格的样式 `Karl's grid` 卡尔可以这样声明

```
\tikzset{Karl's grid/.style={help lines,color=blue!50}}
...
\draw[Karl's grid](0,0)grid(5,5);
```

通过参数化, 样式变得更加强大。这意味着, 与其他选项一样, 样式也可以与参数一起使用。例如, 卡尔可以对网格进行参数化, 以便默认情况下为蓝色, 但是他也可以使用其他颜色。

```
\begin{tikzpicture}
[Karl's grid/.style={help lines,color=#1!50},
Karl's grid/.default=blue]

\draw[Karl's grid](0,0)grid(1.5,2);
\draw[Karl's grid=red](2,0)grid(3.5,2);
\end{tikzpicture}
```

在此示例中, 样式 `Karl's grid` 的定义用作 `{tikzpicture}` 环境的可选参数。其他元素的附加样式将紧跟在逗号之后。在使用多种样式的情况下, 环境的可选参数可能很容易长于实际内容。

2.9 绘图选项

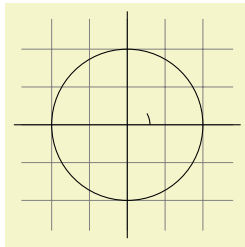
卡尔想知道还有什么其他选项可以影响路径的绘制。他已经看到可以使用 `color=<color>` 选项来设置线条的颜色。选项 `draw=<color>` 做的几乎是一样的, 只是它只设置了线条的颜色, 可以使用不同的颜色填充(卡尔填充圆弧的角度会用到这个选项)。

他看到样式 `very thin` 产生非常细的线。卡尔对此并不感到惊讶, 也不会惊讶于 `thin` 产生细线, `thick` 产生的粗线, `very thick` 产生非常粗的线, `ultra thick` 产生非常非常粗的线以及 `ultra thin` 产生非常非常细的线, 以至于低分辨率的打印机和显示器将很难显示它们。他想知道是什么使线条具有“正常”的宽度。事实证明 `thin` 是正确的选择, 因为它的厚度与 \TeX 的 `\hrule` 命令相同。但是, 卡尔想知道 `thin` 和 `thick` 之间是否有“中间”的宽度。有: `semithick`。

对线条的另一种有用的处理是虚线或点线。为此, 可以使用 `dashed` 和 `dotted` 两种样式, 产生 `---` 和 `.....`。这两个选项也存在松散和密集的版本, 称为 `loosely dashed`, `densely dashed`, `loosely dotted`, `densely dotted`。如果他真的真的需要, 卡尔也可以使用 `dash pattern` 选项定义更复杂的虚线样式, 但他的儿子坚持认为, 在使用时要格外小心, 而且大多会分散注意力。卡尔的儿子声称复杂的虚线图案是邪恶的。卡尔的学生不关心虚线的样式。

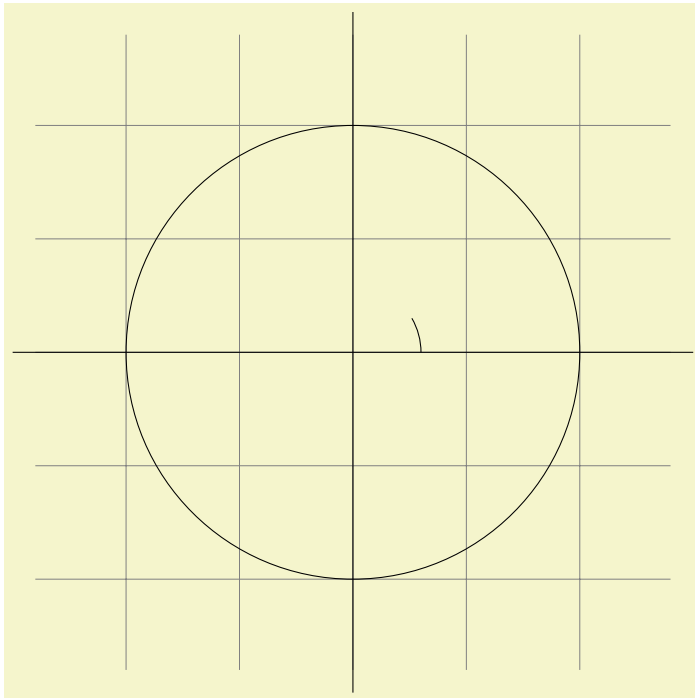
2.10 弧线路径的创建

我们的下一个障碍是绘制该角度的弧线。为此，`arc` 路径构造操作非常有用，它可以绘制一部分圆或椭圆。`arc` 操作之后的括号指定圆弧的选项。一个示例是 `arc [start angle=10, end angle=80, radius=10pt`，其含义就像它说的那样。卡尔显然需要从 0° 到 30° 的弧线。半径应相对较小，可能约为圆半径的三分之一。当使用弧形路径构造操作时，指定的弧形将以其起始点添加到当前位置。因此，我们首先必须“到达那里”。



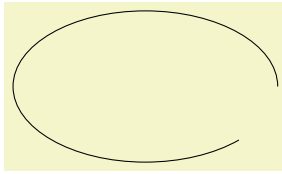
```
\begin{tikzpicture}
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\draw(3mm,0mm)arc[start angle=0,end angle=30,radius=3mm];
\end{tikzpicture}
```

卡尔认为这确实有点小，除非学会了缩放方法，否则他无法继续。为此，他可以添加 `[scale=3]` 选项。他可以将此选项添加到每个 `\draw` 命令中，但那样做非常麻烦。相反，他将其添加到整个环境中，这使得该选项应用于其中的所有内容。



```
\begin{tikzpicture}[scale=3]
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\draw(3mm,0mm)arc[start angle=0,end angle=30,radius=3mm];
\end{tikzpicture}
```

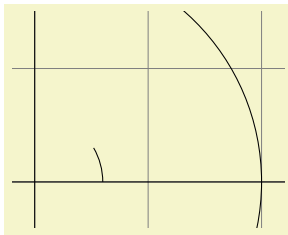
对于圆而言，您可以指定“两个”半径以获得椭圆弧。



```
\tikz\draw(0,0)
arc[start angle=0,end angle=315,
x radius=1.75cm,y radius=1cm];
```

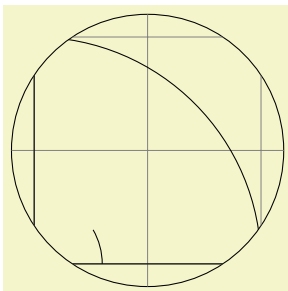
2.11 剪切路径

为了节省空间，最好裁剪一下卡尔的图形，以便我们专注于“有趣的”部分。TikZ 中的剪切非常容易。您可以使用 `\clip` 命令剪切所有后续图形。它的工作方式类似于 `\draw`，只是它不绘制任何内容，而是使用给定的路径剪切所有的图形。



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,0.75);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\draw(3mm,0mm)arc[start angle=0,end angle=30,radius=3mm];
\end{tikzpicture}
```

你也可以同时做这两件事：绘制和剪切一个路径。为此，使用 `\draw` 命令并添加 `clip` 选项。（这并不是全部：您还可以使用 `\clip` 命令并添加 `draw` 选项。实际上，`\draw` 只是 `\path[draw]` 的简写而 `\clip` 是 `\path[clip]` 的简写，而且你也可以这样使用 `\path[draw,clip]`。下面是一个例子：



```
\begin{tikzpicture}[scale=3]
\clip[draw](0.5,0.5)circle(.6cm);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\draw(3mm,0mm)arc[start angle=0,end angle=30,radius=3mm];
\end{tikzpicture}
```

2.12 抛物线和正弦线路径构造

虽然卡尔在他的图中不需要它们，但是他很高兴地知道有 `parabola` 和 `sin` 以及 `cos` 路径操作，可以在当前路径上添加抛物线和正弦和余弦曲线。对于 `parabola` 操作，当前点将位于抛物线上，以及在抛物线操作后给定的点。考虑下面的例子：



```
\tikz\draw(0,0)rectangle(1,1)(0,0)parabola(1,1);
```

也可以将拐弯放置在其他位置：



```
\tikz\draw[x=1pt,y=1pt](0,0)parabolabend(4,16)(6,12);
```

操作 `sin` 和 `cos` 在区间 $[0, \pi/2]$ 中添加一条正弦或余弦曲线，前一个点在曲线的起点，曲线在给定的终点结束。这里有两个例子：

A sine / curve.

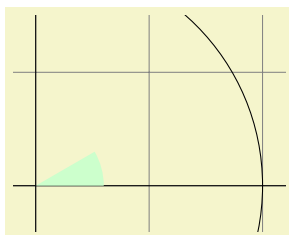
```
Asine\tikz\draw[x=1ex,y=1ex](0,0)sin(1.57,1);curve.
```



```
\tikz\draw[x=1.57ex,y=1ex](0,0)sin(1,1)cos(2,0)sin(3,-1)cos(4,0)
(0,1)cos(1,0)sin(2,-1)cos(3,0)sin(4,1);
```

2.13 填充和绘图

回到这张图片，卡尔现在想用一个非常浅的绿色来“填充”这个角度。他使用 `\fill` 而不是 `\draw`。卡尔是这样做的：



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,0.75);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\fill[green!20!white](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--(0,0);
\end{tikzpicture}
```

颜色 `green!20!white` 表示 20% 的绿色和 80% 的白色混合在一起。这样的颜色表达式是可能的，因为 TikZ 使用 Uwe Kern 编写的 `xcolor` 包，请参阅该包的文档以获得关于颜色表达式的详细信息。

如果卡尔没有使用 `--(0,0)`“闭合”路径在最后会发生什么？在本例中，路径会自动闭合，因此可以省略它。事实上，写以下代码甚至更好：

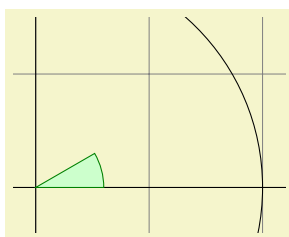
```
\fill[green!20!white](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
```

`--cycle` 通过平滑地连接第一个点和最后一个点，使当前路径闭合（事实上是当前路径的当前部分）。要了解差异，请考虑以下示例：



```
\begin{tikzpicture}[line width=5pt]
\draw(0,0)--(1,0)--(1,1)--(0,0);
\draw(2,0)--(3,0)--(3,1)--cycle;
\useasboundingbox(0,1.5);%makeboundingboxhigher
\end{tikzpicture}
```

您还可以同时使用 `\filldraw` 命令来填充和绘制路径。这将首先绘制路径，然后填充它。这看起来可能不是很有用，但是你可以指定不同的颜色来填充和描边。这些被指定为可选参数，像这样：



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,0.75);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\filldraw[fill=green!20!white,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\end{tikzpicture}
```

2.14 阴影

卡尔简要地考虑了通过“阴影”使角度“更精致”的可能性。不使用统一的颜色填充区域，而是使用不同颜色之间的平滑过渡。为此，`\shade` 和 `\shadedraw` 可以同时添加阴影和进行绘图操作。



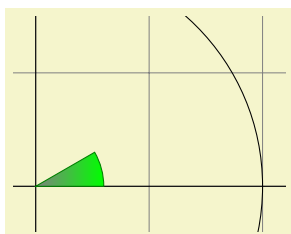
```
\tikz\shade(0,0)rectangle(2,1)(3,0.5)circle(.5cm);
```

默认的阴影是从灰色到白色的平滑过渡。要指定不同的颜色，您可以使用选项：



```
\begin{tikzpicture}[rounded corners,ultra thick]
\shade[top color=yellow,bottom color=black](0,0)rectangle+(2,1);
\shade[left color=yellow,right color=black](3,0)rectangle+(2,1);
\shadedraw[inner color=yellow,outer color=black,draw=yellow](6,0)rectangle+(2,1);
\shade[ball color=green](9,.5)circle(.5cm);
\end{tikzpicture}
```

对于卡尔来说，以下代码可能是合适的：



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,0.75);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\shadedraw[left color=gray,right color=green,draw=green!50!black]
(0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\end{tikzpicture}
```

然而，他明智地决定阴影通常只会分散注意力，而没有给图片添加任何东西。

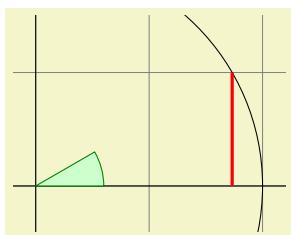
2.15 指定坐标

卡尔现在想要加上正弦和余弦线。他已经知道可以使用 `color=` 选项来设置线条的颜色。那么，指定坐标的最好方法是什么？

有不同的方式来指定坐标。最简单的方法是像 $(10\text{pt}, 2\text{cm})$ 这样。这意味着在 x 方向上的坐标为 10pt ，在 y 方向上的坐标为 2cm 。或者，您也可以省略单位 $(1, 2)$ ，这意味着“当前 x 向量的 1 倍以及当前 y 向量的 2 倍”。向量默认在 x 方向上为 1cm ，在 y 方向上为 1cm 。

为了在极坐标中指定点，使用符号 $(30:1\text{cm})$ ，意思是在 30 度方向上且极径为 1cm 。这显然是非常有用的，以“得到圆上的点 $(\cos 30^\circ, \sin 30^\circ)$ ”。

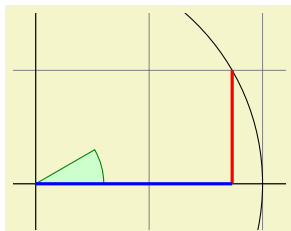
您可以在坐标前添加一个或两个 $+$ 符号，如 $+(0\text{cm}, 1\text{cm})$ 或 $++(2\text{cm}, 0\text{cm})$ 。这些坐标的解释是不同的：第一种形式表示“上一指定位置上方 1cm ”，第二种形式表示“上一指定位置右方 2cm ，使其成为新的指定位置”。例如，我们可以画出正弦线如下：



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,0.75);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\filldraw[fill=green!20,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[red,very thick](30:1cm)--+(0,-0.5);
\end{tikzpicture}
```

卡尔使用了 $\sin 30^\circ = 1/2$ 这个事实。然而，他很怀疑他的学生是否知道这一点，所以如果有一种方法来指定点从 $(30:1\text{cm})$ 径直向下绘制到在 x 轴上的点就好了。这确实是可能的，使用一种特殊的语法：Karl 可以用 $(30:1\text{cm}|0,0)$ 。一般来说， $(\langle p \rangle | - \langle q \rangle)$ 是“通过 p 的竖直线和通过 q 的水平线的交点”。

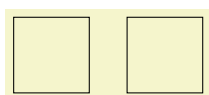
接下来，我们来画余弦线。一种方法是 $(30:1\text{cm}|0,0)--(0,0)$ 。另一种方法是这样的：我们从正弦结束的地方“继续”：



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,0.75);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw(-1.5,0)--(1.5,0);
\draw(0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\filldraw[fill=green!20,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[red,very thick](30:1cm)--+(0,-0.5);
\draw[blue,very thick](30:1cm)++(0,-0.5)--(0,0);
\end{tikzpicture}
```

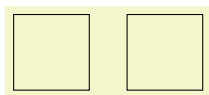
注意，在 $(30:1\text{cm})$ 和 $++(0,-0.5)$ 之间不存在 $--$ 。具体来说，这个路径解释如下：“首先， $(30:1\text{cm})$ 告诉我将笔移动到 $(\cos 30^\circ, 1/2)$ 。接下来，指定了另一个坐标点，所以我移动我的笔，但不绘制任何东西。这个新点比上一个位置下降了半个单位，因此它在 $(\cos 30^\circ, 0)$ 。最后，我把笔移到原点，但这次画一些东西（因为用到了 $--$ ）。”

要理解 $+$ 和 $++$ 的区别，请考虑下面的例子：



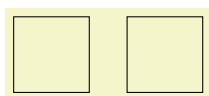
```
\begin{tikzpicture}
\def\rectanglepath{--++(1cm,0cm)--++(0cm,1cm)--++(-1cm,0cm)--cycle}
\draw(0,0)\rectanglepath;
\draw(1.5,0)\rectanglepath;
\end{tikzpicture}
```

通过比较，当使用单个 $+$ 号时，坐标是不同的：



```
\begin{tikzpicture}
\def\rectanglepath{--+(1cm,0cm)--+(1cm,1cm)--+(0cm,1cm)--cycle}
\draw(0,0)\rectanglepath;
\draw(1.5,0)\rectanglepath;
\end{tikzpicture}
```

当然，所有这些都可以写成更清晰更经济的形式（一个或两个 $+$ 号）：



```
\tikz\draw(0,0)rectangle+(1,1)(1.5,0)rectangle+(1,1);
```

2.16 相交路径

留给卡尔的是 $\tan \alpha$ 这一条线，这似乎很难使用变换和极坐标来指定。他能做的第一个也是最简单的事情就是使用坐标 $(1, \{\tan(30)\})$ ，因为 TikZ 的数学引擎知道如何计算像 $\tan(30)$ 这样的东西。注意添加的大括号，否则 TikZ 的解析器会认为第一个右括号结束了坐标（通常，当坐标中包含括号时，需要坐标周围添加大括号）。

但是，卡尔还可以使用更复杂但更“几何”的方式来计算橙色线的长度：他可以将路径的交点指定为坐标。 $\tan \alpha$ 的线从 $(1,0)$ 开始，并向上到达一个点，该点是这条“向上”的线和一条线从原点到 $(30:1\text{cm})$ 线的交点。通过 `intersections` 库可以得到这样的计算。

卡尔必须做的是创建两条“看不见的”路径，在感兴趣的位置相交。可以使用 `\path` 命令创建其他方式看不到的路径，而不需要任何如 `draw` 或 `fill` 的选项。然后，卡尔可以将 `name path` 选项添加到该路径以供以后参考。一旦构造好了路径，卡尔就可以使用 `name intersections` 为坐标分配名称，以供以后参考。

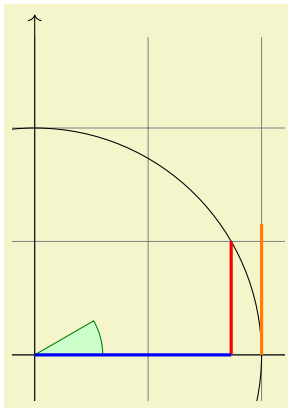
```
\path[name path=upwardline](1,0)--(1,1);
\path[name path=slopedline](0,0)--(30:1.5cm);%abitlonger,sothatthereisanintersection

%(add`\usetikzlibrary{intersections}'afterloadingtikzinthepreamble)
\draw[name intersections={of=upwardlineandslopedline,by=x}]
[very thick,orange](1,0)--(x);
```

2.17 添加箭头

卡尔现在想在轴的末端添加小箭头。他注意到在许多情节中，甚至在科学期刊中，这些箭头似乎都消失了，大概是因为生成程序无法生成它们。卡尔认为箭头尖端位于轴的末端。他的儿子同意。他的学生不关心箭头。

事实证明，添加箭头非常容易：卡尔在绘图命令中为坐标轴添加 `->` 选项：

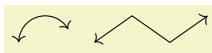


```
\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,1.51);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\draw[->](-1.5,0)--(1.5,0);
\draw[->](0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];
\filldraw[fill=green!20,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[red,very thick](30:1cm)--+(0,-0.5);
\draw[blue,very thick](30:1cm)++(0,-0.5)--(0,0);

\path[name path=upwardline](1,0)--(1,1);
\path[name path=slopedline](0,0)--(30:1.5cm);
\draw[name intersections={of=upwardlineandslopedline,by=x}]
[very thick,orange](1,0)--(x);
\end{tikzpicture}
```

如果卡尔使用选项 `<-` 而不是 `->`，箭头就会被放在路径的开始处。选项 `<->` 在路径的两端都放置箭头。

对于可以添加箭头的路径类型，存在某些限制。根据经验，您只能将箭头添加到一条开放的“线”中。例如，您不能将箭头添加到例如矩形或圆形。但是，您可以向弯曲路径和具有多个线段的路径添加箭头，如下示例所示：



```
\begin{tikzpicture}
\draw[<->](0,0)arc[start angle=180,end angle=30,radius=10pt];
\draw[<->](1,0)--(1.5cm,10pt)--(2cm,0pt)--(2.5cm,10pt);
\end{tikzpicture}
```

卡尔详细观察发现 TikZ 把箭头放在了末端。当他放大时看起来像这样：→。形状似乎模糊不清，实际上，这恰好是 $\text{T}_{\text{E}}\text{X}$ 的标准箭头的末尾，用在 $f: A \rightarrow B$ 之类的东西中。

卡尔喜欢这个箭头，特别是因为它不是“像许多其他软件包提供的箭头”那样宽。然而，他希望，有时他可能需要使用一些其他种类的箭头。为此，卡尔可以用 `>=` (末端箭头的类型)，其中 (末端箭头的类型) 是一个特殊的箭头规范。例如，如果卡尔用 `>=Stealth`，那么他告诉 TikZ 他想要“像隐形战斗机一样”的箭头：



```
\usetikzlibrary {arrows.meta}
\begin{tikzpicture}[>=Stealth]
\draw[->](0,0)arc[start angle=180,end angle=30,radius=10pt];
\draw[<<- ,very thick](1,0)--(1.5cm,10pt)--(2cm,0pt)--(2.5cm,10pt);
\end{tikzpicture}
```


卡尔想知道是否真的需要使用这种军事名称来表示箭头。当儿子告诉他微软的 PowerPoint 也使用这样的名称时，他并没有真正感到高兴。他决定让他的学生在某个时候对这个问题进行讨论。

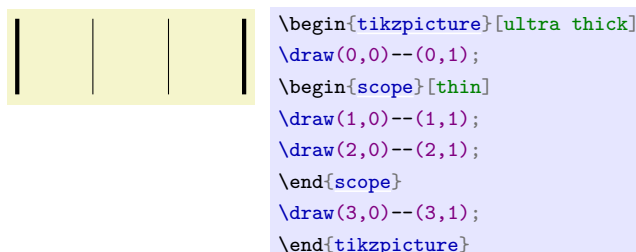
除了 **Stealth** 卡尔还可以选择其他几种预定义的箭头，请参见??节。此外，如果需要新的箭头类型，他可以自己定义箭头类型。

2.18 分组

卡尔已经看到，有许多图形选项会影响路径的渲染方式。通常，他想将某些选项应用于整套图形命令。例如，卡尔可能希望使用 **thick** 笔绘制三个路径。但希望其他所有东西都能“正常”绘制。

如果卡尔想为整个图片设置某个图形选项，他可以简单地将该选项传递给 `\tikz` 命令或 `{tikzpicture}` 环境（格达将把这些选项传递给 `\tikzpicture`，汉森将它们传递给 `\starttikzpicture`）。但是，如果卡尔希望将图形选项应用到本地组，他可以将这些命令放在 `{scope}` 环境中（格达使用 `\scope` 和 `\endscope`，汉森使用 `\startscope` 和 `\stopscope`）。该环境将图形选项作为可选参数，这些选项应用于范围内的所有内容，但不应用于范围外的任何内容。

这是一个案例：



分组还有另一个有趣的作用：裁剪区域的任何更改都是该分组的局部更改。因此，如果您用 `\clip` 在某个分组内的某个地方，`\clip` 命令的效果将在分组的末尾结束。这是有用的，因为没有其他方式“扩大”剪切区域。

卡尔也已经看到给像 `\draw` 这样的命令提供选项只适用于该命令。事实证明，情况要稍微复杂一些。首先，像 `\draw` 这样的命令的选项实际上并不是该命令的选项，但它们是“路径选项”，可以在路径的任何位置提供。因此，除了用 `\draw[thin] (0,0) --(1,0);` 还可以用 `\draw (0,0) [thin] --(1,0);` 或 `\draw (0,0) --(1,0) [thin];`；所有这些都会产生同样的效果。这看起来可能有些奇怪，因为在上一种情况下，看起来 **thin** 应该只在从 (0,0) 到 (1,0) 这一行的“之后”才生效。但是，大多数图形选项只适用于整个路径。事实上，如果你将 **thin** 和 **thick** 用在同一条路径上，给出的最后一个选项将会“赢得胜利”。

阅读以上内容时，卡尔注意到只有“大多数”图形选项适用于整个路径。确实，所有变换选项都不会适用于整个路径，而仅适用于“路径上遵循它们的所有内容”。稍后我们将对此进行更详细的介绍。但是，在路径构建过程中给出的所有选项仅适用于该路径。

2.19 坐标变换

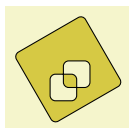
当您指定 (1cm,1cm) 之类的坐标时，该坐标在页面上的什么位置？为了确定位置，TikZ，TeX 和 PDF 或 PostScript 都会使用一种确定的坐标变换，以确定其在页面上的最终位置。

TikZ 提供了许多选项，允许您在 TikZ 的私有坐标系统中变换坐标。例如，**xshift** 选项允许您将所有后续点移动一定距离：

```
\tikz\draw(0,0)--(0,0.5) [xshift=2pt] (0,0)--(0,0.5);
```

需要注意的是，您可以在“路径的中间”坐标变换，PDF 或 PostScript 不支持这一特性。原因是 TikZ 会持续跟踪自己的坐标变换矩阵。

这是一个更复杂的示例：



```
\begin{tikzpicture}[even odd rule,rounded corners=2pt,x=10pt,y=10pt]
\filldraw[fill=yellow!80!black](0,0)rectangle(1,1)
[xshift=5pt,yshift=5pt](0,0)rectangle(1,1)
[rotate=30](-1,-1)rectangle(2,2);
\end{tikzpicture}
```

最有用的坐标变换是用于平移的 `xshift` 和 `yshift` 变换, `shift` 变换将指定图形平移 `shift={(1,0)}` 或 `shift={+(0,0)}` 中给定的距离 (大括号是必需的, 以便 $\text{T}_{\text{E}}\text{X}$ 不会将逗号分隔为选项), `rotate` 变换将指定图形旋转指定的角度 (还有一个 `rotate around` 用于绕给定点旋转), `xscale` 和 `yscale` 用于将指定图形在 x 或 y 方向上缩放指定的倍数, `xslant` 和 `yslant` 用于图形的倾斜。如果这些变换以及我没有提到的变换都不足够, 那么 `cm` 选项允许您应用任意变换矩阵。顺便说一下, 卡尔的学生不知道什么是变换矩阵。

2.20 重复工作: For 循环

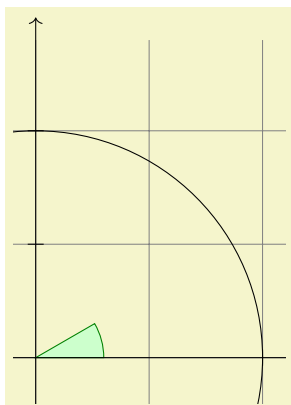
卡尔的下一个目标是在 -1 , $-1/2$, $1/2$ 和 1 的位置的坐标轴上添加小刻度。为此, 最好使用某种 ‘循环’, 特别是因为他希望在这些位置上做同样的事情。可以使用不同的包来完成这项工作。 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 对此有自己的内部命令, `psstricks` 带有强大的 `\multido` 命令。所有这些都可以与 `TikZ` 一起使用, 所以如果您熟悉它们, 可以随意使用它们。`TikZ` 引入了另一个命令 `\foreach`, 我引入这个命令是因为我永远记不住其他宏包的语法。`\foreach` 命令定义在宏包 `pgffor` 中, 可以独立于 `TikZ` 使用, 但 `TikZ` 会自动包含它。

`\foreach` 命令的基本形式很容易使用:

```
x = 1, x = 2, x = 3, \foreach\xin{1,2,3}{\x=\x$,}
```

一般语法是 `\foreach <变量> in {(值清单)} <命令>`。在 (命令) 内部, `<变量>` 将分配给不同的值。如果 (命令) 不是以大括号开头, 则直到下一个分号的所有内容都将用作 (命令)。

对于卡尔, 他可以使用以下代码绘制坐标轴上的刻度:



```
\begin{tikzpicture}[scale=3]
\clip(-0.1,-0.2)rectangle(1.1,1.51);
\draw[step=.5cm,gray,very thin](-1.4,-1.4)grid(1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[->](-1.5,0)--(1.5,0);
\draw[->](0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];

\foreach\xin{-1cm,-0.5cm,1cm}
\draw(\x,-1pt)--(\x,1pt);
\foreach\yin{-1cm,-0.5cm,0.5cm,1cm}
\draw(-1pt,\y)--(1pt,\y);
\end{tikzpicture}
```

实际上, 创建刻度线有很多不同的方法。例如, 卡尔可以将 `\draw ...`; 放在大括号内。他还可以用, 比如说, 这样

```
\foreach\xin{-1,-0.5,1}
\draw[xshift=\x cm](0pt,-1pt)--(0pt,1pt);
```

卡尔很好奇在更复杂的情况下会发生什么, 比如说, 20 个刻度。为每个 `\foreach` 明确地提到集合中的所有数字似乎很麻烦。事实上, 可以在 `\foreach` 声明中使用 `...` 产生大量迭代值 (但是, 它必须是无量纲的实数), 如下示例所示:



```
\tikz\foreach\xin{1,...,10}
\draw(\x,0)circle(0.4cm);
```

如果在 ... 之前提供 2 个数字, 则 \foreach 语句根据它们的差值进行步进:

```
\tikz\foreach\xin{-1,-0.5,...,1}
\draw(\xcm,-1pt)--(\xcm,1pt);
```

我们还可以使用嵌套循环以产生有趣的效果:

1,5	2,5	3,5	4,5	5,5	7,5	8,5	9,5	10,5	11,5	12,5
1,4	2,4	3,4	4,4	5,4	7,4	8,4	9,4	10,4	11,4	12,4
1,3	2,3	3,3	4,3	5,3	7,3	8,3	9,3	10,3	11,3	12,3
1,2	2,2	3,2	4,2	5,2	7,2	8,2	9,2	10,2	11,2	12,2
1,1	2,1	3,1	4,1	5,1	7,1	8,1	9,1	10,1	11,1	12,1

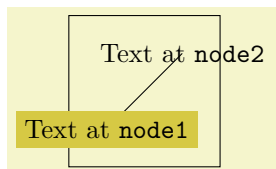
```
\begin{tikzpicture}
\foreach\xin{1,2,...,5,7,8,...,12}
\foreach\yin{1,...,5}
{
\draw(\x,\y)+(-.5,-.5)rectangle++(.5,.5);
\draw(\x,\y)node{\x,\y};
}
\end{tikzpicture}
```

\foreach 语句甚至可以做更棘手的事情, 但是上面已经给出了这个想法。

2.21 添加文本

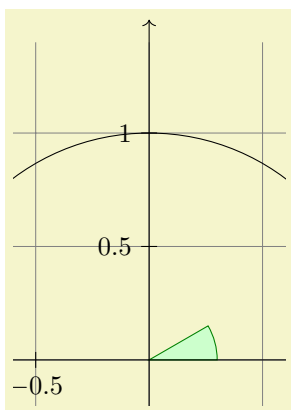
到目前为止, 卡尔对图片非常满意。但是, 最重要的部分, 即文字标签, 仍然缺失!

TikZ 提供了一个易于使用且功能强大的系统, 用于在特定位置向图片添加文本和复杂形状。其基本思想如下: 当 TikZ 构造路径并在路径中间遇到关键字 **node** 时, 它读取节点详述。关键字 **node** 后面通常跟着一些选项以及放在花括号之间的文本。这段文字放在普通的 $\text{T}_\text{E}\text{X}$ 盒子中 (如果节点规范直接遵循坐标, 并且通常就是这种情况, TikZ 可以执行一些魔法般的操作, 从而甚至可以在框中使用逐字记录文本), 然后放在当前位置, 即最后指定的位置 (根据给定的选项, 可能会移动一点)。但是, 仅在完全绘制/填充/阴影/剪切或任何路径之后才绘制所有节点。



```
\begin{tikzpicture}
\draw(0,0)rectangle(2,2);
\draw(0.5,0.5)node[fill=yellow!80!black]
{Textat\verb!node1!}
--(1.5,1.5)node{Textat\verb!node2!};
\end{tikzpicture}
```

显然, 卡尔不仅希望将节点放置在最后指定的位置, 而是希望将其放置在这些位置的左侧或右侧。为此, 你在图片中放置的每个节点对象都配备了多个锚点。例如, **north** 锚点位于形状上端的中间, **south** 的锚点位于形状的底部, **north east** 的锚点位于形状的右上角。当提供选项 **anchor=north** 时, 将放置文本, 使得该北锚位于当前位置, 因此该文本位于当前位置的下方。卡尔使用此方法绘制刻度线, 如下所示:



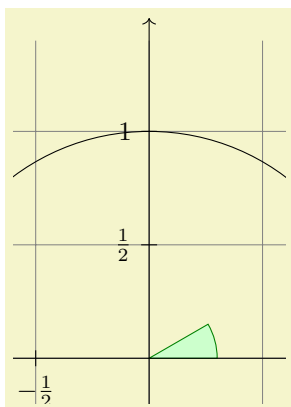
```
\begin{tikzpicture}[scale=3]
\clip(-0.6,-0.2)rectangle(0.6,1.51);
\draw[step=.5cm,help lines](-1.4,-1.4)grid(1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[->](-1.5,0)--(1.5,0);\draw[->](0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];

\foreach\xin{-1,-0.5,1}
\draw(\x cm,1pt)--(\x cm,-1pt)node[anchor=north]{\x$};
\foreach\yin{-1,-0.5,0.5,1}
\draw(1pt,\y cm)--(-1pt,\y cm)node[anchor=west]{\y$};
\end{tikzpicture}
```

这已经很不错了。使用这些锚点，卡尔现在可以添加大多数的其他文本元素。然而，卡尔认为，虽然“正确”，但这是违反直觉的，为了把某些东西放在一个给定的点下面，他必须使用 `north` 锚点。因此，有一个 `below` 的选项，它的作用与 `anchor=north` 相同。类似地，`above right` 与 `anchor=south west` 的作用相同。另外，`below` 选项有一个可选的尺寸参数。如果给定，该形状将额外向下移动给定的数量。因此，`below=1pt` 可以用来把一个文本标签放在某个点下面，另外，将它向下移动 1pt。

卡尔对刻度不太满意。他希望显示 $\frac{1}{2}$ 或 $\frac{1}{2}$ 而不是 0.5，部分原因是为了展示 TeX 和 TikZ 的出色功能，部分原因是对于 $\frac{1}{3}$ 或 π 这样的刻度，在上面放置“数学”的刻度而不是仅“数字的”刻度当然是更加可取的。另一方面，他的学生更喜欢 0.5 而不是 $\frac{1}{2}$ ，因为他们一般不太喜欢分数。

卡尔现在面临一个问题：对于 `\foreach` 声明，位置 `\x` 仍应为 0.5，因为 TikZ 不知道应该是 $\frac{1}{2}$ 。但另一方面，排版文本实际上应该是 $\frac{1}{2}$ 。要解决此问题，`\foreach` 提供了一种特殊的语法：与其使用一个变量 `\x`，卡尔可以指定两个（或更多）用斜杠分隔的变量 `\x / \xtext`。然后，集合中的元素使用 `\foreach` 迭代也必须采用 `\langle first \rangle / \langle second \rangle` 的形式。在每次迭代中，`\x` 将设置为 `\langle first \rangle`，`\xtext` 将设置为 `\langle second \rangle`。如果没有给出 `\langle second \rangle`，将再次使用 `\langle first \rangle`。因此，这是刻度的新代码：

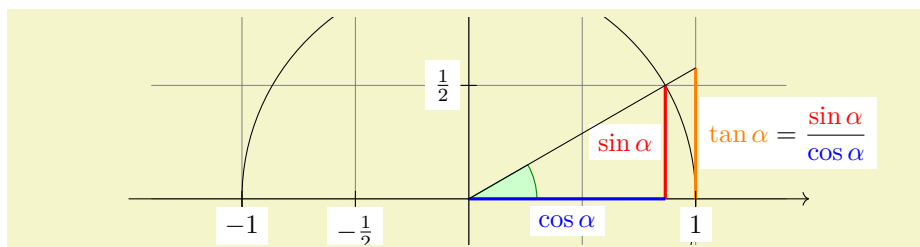


```
\begin{tikzpicture}[scale=3]
\clip(-0.6,-0.2)rectangle(0.6,1.51);
\draw[step=.5cm,help lines](-1.4,-1.4)grid(1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black](0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[->](-1.5,0)--(1.5,0);\draw[->](0,-1.5)--(0,1.5);
\draw(0,0)circle[radius=1cm];

\foreach\x/\xtextin{-1,-0.5/-\frac{1}{2},1}
\draw(\x cm,1pt)--(\x cm,-1pt)node[anchor=north]{\xtext$};
\foreach\y/\ytextin{-1,-0.5/-\frac{1}{2},0.5/\frac{1}{2},1}
\draw(1pt,\y cm)--(-1pt,\y cm)node[anchor=west]{\ytext$};
\end{tikzpicture}
```

卡尔对结果感到非常满意，但是他的儿子指出这仍然不能令人满意：网格和圆会干扰数字并降低其可读性。卡尔对此并不十分担心（他的学生甚至没有注意到），但是他的儿子坚持认为有一个简单的解决方案：卡尔可以添加 `[fill=white]` 选项，用白色填充文本形状的背景。

卡尔接下来要做的就是添加 $\sin \alpha$ 之类的标签。为此，他想在线的中间放置一个标签。这样做，而不是指定标签 `node {\sin\alpha$}` 直接在直线的一个端点之后（即将标签放置在端点处），卡尔可以在 `--` 之后，在坐标之前直接给出标签。缺省情况下，这会将标签放在行的中间，但是 `pos=` 选项可以用来修改它。另外，`near start` 和 `near end` 可以用来修改这个位置：



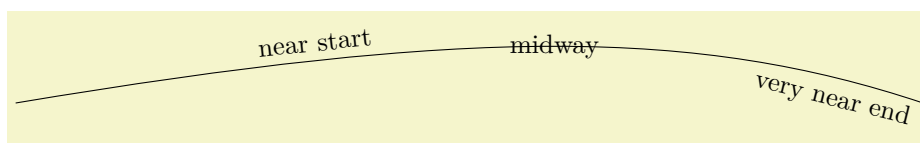
```
\usetikzlibrary {intersections}
\begin{tikzpicture} [scale=3]
\clip(-2,-0.2)rectangle(2,0.8);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4)grid(1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0)--(3mm,0mm)
arc[start angle=0,end angle=30,radius=3mm]--cycle;
\draw[>-] (-1.5,0)--(1.5,0)coordinate(xaxis);
\draw[>-] (0,-1.5)--(0,1.5)coordinate(yaxis);
\draw(0,0)circle[radius=1cm];

\draw[very thick,red]
(30:1cm)--node[left=1pt,fill=white]{\sin\alpha}(30:1cm|-xaxis);
\draw[very thick,blue]
(30:1cm|-xaxis)--node[below=2pt,fill=white]{\cos\alpha}(0,0);
\path[name path=upwardline](1,0)--(1,1);
\path[name path=slopedline](0,0)--(30:1.5cm);
\draw[name intersections={of=upwardlineandslopedline,by=t}]
[very thick,orange](1,0)--node[right=1pt,fill=white]
{\displaystyle\tan\alpha\color{black}=
\frac{{\color{red}\sin\alpha}}{{\color{blue}\cos\alpha}}}(t);

\draw(0,0)--(t);

\foreach\x/\xtextin{-1,-0.5/-\frac{1}{2},1}
\draw(\xcm,1pt)--(\xcm,-1pt)node[anchor=north,fill=white]{\xtext};
\foreach\y/\ytextin{-1,-0.5/-\frac{1}{2},0.5/\frac{1}{2},1}
\draw(1pt,\ycm)--(-1pt,\ycm)node[anchor=west,fill=white]{\ytext};
\end{tikzpicture}
```

您也可以在曲线上放置标签，并通过添加 `sloped` 选项，使其旋转以使其与直线的斜率匹配。这是一个例子：



```
\begin{tikzpicture}
\draw(0,0)..controls(6,1)and(9,1)..
node[near start,sloped,above]{nearstart}
node[midway]
node[very near end,sloped,below]{verynearend}(12,0);
\end{tikzpicture}
```

现在仍然需要在图片右侧绘制说明文字。这里的主要困难在于限制文本“标签”的宽度，该宽度相当长，因为使用了换行符。幸运的是，卡尔可以使用选项 `text width=6cm`，以获得理想的效果。因此，这是完整的代码：

```

\begin{tikzpicture}
[scale=3, line cap=round,
%Styles
axes/.style=,
important line/.style={very thick},
information text/.style={rounded corners,fill=red!10,inner sep=1ex}]

%Colors
\colorlet{anglecolor}{green!50!black}
\colorlet{sincolor}{red}
\colorlet{tancolor}{orange!80!black}
\colorlet{coscolor}{blue}

%Thegraphic
\draw[help lines,step=0.5cm](-1.4,-1.4)grid(1.4,1.4);

\draw(0,0)circle[radius=1cm];

\begin{scope}[axes]
\draw[>->](-1.5,0)--(1.5,0)node[right]{ $x$ }coordinate(xaxis);
\draw[>->](0,-1.5)--(0,1.5)node[above]{ $y$ }coordinate(yaxis);

\foreach\x/\xtextin{-1,-.5/-\frac{1}{2},1}
\draw[xshift=\x cm](0pt,1pt)--(0pt,-1pt)node[below,fill=white]{ $\xtext$ };

\foreach\y/\ytextin{-1,-.5/-\frac{1}{2},.5/\frac{1}{2},1}
\draw[yshift=\y cm](1pt,0pt)--(-1pt,0pt)node[left,fill=white]{ $\ytext$ };
\end{scope}

\filldraw[fill=green!20,draw=anglecolor](0,0)--(3mm,0pt)
arc[start angle=0,end angle=30,radius=3mm];
\draw(15:2mm)node[anglecolor]{ $\alpha$ };

\draw[important line,sincolor]
(30:1cm)--node[left=1pt,fill=white]{ $\sin\alpha$ }(30:1cm|-xaxis);

\draw[important line,coscolor]
(30:1cm|-xaxis)--node[below=2pt,fill=white]{ $\cos\alpha$ }(0,0);

\path[name path=upwardline](1,0)--(1,1);
\path[name path=slopedline](0,0)--(30:1.5cm);
\draw[name intersections={of=upwardlineandslopedline,by=t}]
[very thick,orange](1,0)--node[right=1pt,fill=white]
{ $\displaystyle\tan\alpha$ }\color{black}=
\frac{\color{red}\sin\alpha}{\color{blue}\cos\alpha}(t);

\draw(0,0)--(t);

\draw[xshift=1.85cm]
node[right,text width=6cm,information text]
{
The\color{anglecolor}angle $\alpha$ is $30^\circ$ inthe
example( $\pi/6$ inradians).The\color{sincolor}sineof
 $\alpha$ ,whichistheheightoftheredline,is
\[\color{sincolor}\sin\alpha=1/2.\]
BytheTheoremofPythagoras...
};
\end{tikzpicture}

```

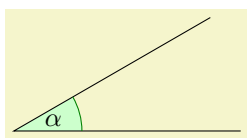
2.22 Pics: 再访 Angle

卡尔希望他创建的图片某些部分的代码可能会非常有用，以至于他将来希望重用它们。自然的做法是创建 \TeX 宏来存储他希望重用的代码。但是， $\text{\textit{TikZ}}$ 提供了另一种直接集成到其解析器中的方式：图片！

“pic”是一种“不是很完整的图片”，因此我们使用简称。这个想法是，图片只是一些代码，您可以使用 `pic` 命令添加到图片的不同位置，它的语法几乎与 `node` 命令相同。主要区别在于，您可以指定应该显示的预定义图片的名称，而不是在花括号中指定一些文本。

定义新的 pics 很容易，请参见??节，但是现在我们只想使用一个这样的预定义的 pic: `angle` pic。顾名思义，这是一个由一张由一个楔形和一段弧以及一些文本组成的小图形（在以下示例中，卡尔需要加载 `angles` 和 `quotes` 库）。pic 很有用的原因是，楔形的大小会被自动计算。

`angle` pic 在 BA 和 BC 两条线之间绘制一个角度，其中 A 、 B 和 C 是三个坐标。在我们的例子中， B 是原点， A 在 x 轴的某处， C 在 30° 这条线上的某处。



```
\usetikzlibrary {angles,quotes}
\begin{tikzpicture}[scale=3]
\coordinate(A)at(1,0);
\coordinate(B)at(0,0);
\coordinate(C)at(30:1cm);

\draw(A)--(B)--(C)
pic[draw=green!50!black,fill=green!20,angle radius=9mm,
"$\alpha$"]{angle=A--B--C};
\end{tikzpicture}
```

让我们来看看，这里发生了什么。首先，我们使用 `\coordinate` 命令指定了三个坐标点。它允许我们在图片中对一个特定的坐标命名。然后是一个普通的 `\draw` 命令，然后是 `pic` 命令。这个命令有很多选项，在花括号中是最重要的一点：我们指定要添加一个 `angle` pic，这个角应该位于我们命名的 A 、 B 和 C 之间（我们也可以使用其他名称）。注意，我们希望在 `pic` 中显示的文本是在 `pic` 选项中的引号中指定的，而不是在花括号中。

要了解有关 pics 的更多信息，请参见??节。

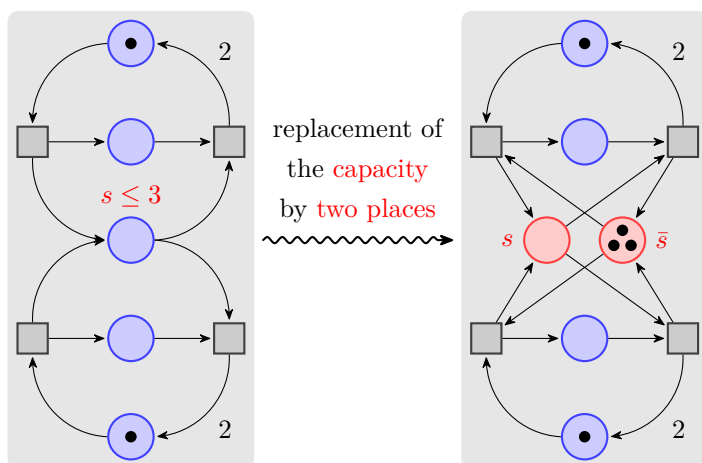
3 教程：哈根的 Petri 网

在第二篇教程中，我们探讨 TikZ 和 PGF 节点的原理。

哈根（Hagen）明天需要就他最喜欢的分布式系统形式：Petri 网发表演讲。哈根以前常常用黑板发表演讲，每个人似乎对此都很满意。不幸的是，他的听众最近迷恋上了基于投影仪的奇特演示，似乎还有一定的同行压力，要求他的 Petri 网的图形也应使用图形程序绘制。为此，他所在学院的一位教授推荐了 TikZ，哈根决定尝试一下。

3.1 问题陈述

对于他的演讲，哈根希望创建一个图形来演示如何通过无库所容量的网络模拟有库所容量的网络。图形在理想情况下应该看起来像这样：??



3.2 配置环境

为了绘制图片哈根将需要加载 TikZ 包，就像在以前的教程中卡尔做的那样。然而，哈根也将需要加载一些卡尔不需要的额外的库包。这些库宏包包含额外的定义，比如图片中通常不需要的额外箭头提示，需要显式加载。

哈根将需要加载几个库：`arrow.meta` 库用于图形中使用的特殊箭头，`decorations.pathmorphing` 库用于图形中间的“蛇形线”，`backgrounds` 库用于添加两个矩形区域后面的背景图片，`fit` 库用来轻松计算这些矩形的大小，以及 `positioning` 库将节点放置在相对于其它节点的位置上。

3.2.1 在 L^AT_EX 中配置环境

使用 L^AT_EX 时，请使用：

```
\documentclass{article}%say

\usepackage{tikz}
\usetikzlibrary{arrows.meta,decorations.pathmorphing,backgrounds,positioning,fit,petri}

\begin{document}
\begin{tikzpicture}
\draw(0,0)--(1,1);
\end{tikzpicture}
\end{document}
```


3.2.2 在 Plain TeX 中配置环境

使用 Plain TeX 时，请使用：

```
%%PlainTeXfile
\inputtikz.tex
\usetikzlibrary{arrows.meta,decorations.pathmorphing,backgrounds,positioning,fit,petri}
\baselineskip=12pt
\hsize=6.3truein
\vsizer=8.7truein
\tikzpicture
\draw(0,0)--(1,1);
\endtikzpicture
\bye
```

3.2.3 在 ConTeXt 中配置环境

使用 ConTeXt 时，请使用：

```
%%ConTeXtfile
\usemodule[tikz]
\usetikzlibrary[arrows,decorations.pathmorphing,backgrounds,positioning,fit,petri]

\starttext
\starttikzpicture
\draw(0,0)--(1,1);
\stoptikzpicture
\stoptext
```

3.3 节点简介

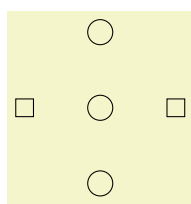
原则上，我们已经知道如何创建哈根想要的图形（也许除了蛇形线之外，我们将介绍到该图形）：我们从大的浅灰色矩形开始，然后添加许多圆形和小矩形以及一些箭头。

但是，这种方法有许多缺点：首先，很难在以后更改任何内容。例如，如果我们决定向 Petri 网添加更多的库所（在 Petri 网理论中圆形节点被称为库所），则所有坐标都会改变，因此我们需要重新计算所有内容。其次，很难读取 Petri 网的代码，因为它只是一长串复杂的坐标和绘制命令列表——Petri 网的底层结构丢失了。

幸运的是，TikZ 提供了一种避免上述问题的强大机制：节点。在上一教程中，我们已经遇到了节点，我们使用它们在卡尔的图形中添加标签。在本教程中，我们将看到节点功能更加强大。

节点只是图片的一小部分。创建节点后，将提供一个应绘制该节点的位置和一个形状。`circle` 形状的节点将被绘制为圆形，`rectangle` 形状的节点将被绘制为矩形等。一个节点可能还包含一些文本，这就是卡尔可以使用节点显示文本的原因。最后，节点还可以取一个名字供以后参考。

在哈根的图片中，我们将使用节点作为 Petri 网的库所和变迁（库所为圆形，变迁为矩形）。让我们从左边的 Petri 网的上半部分开始。在上半部分，我们有 3 个库所和 2 个变迁。我们没有画三个圆和两个矩形，而是用了三个形状为 `circ` 的节点和两个形状为 `rectangle` 的节点。



```
\begin{tikzpicture}
\path(0,2)node[shape=circle,draw]{}
(0,1)node[shape=circle,draw]{}
(0,0)node[shape=circle,draw]{}
(1,1)node[shape=rectangle,draw]{}
(-1,1)node[shape=rectangle,draw]{};
\end{tikzpicture}
```

哈根指出，这看起来不太像是最终的图片，但似乎是一个不错的开始。

让我们更详细地查看代码。整个图形由一条路径组成。忽略 `node` 操作，在这条路径上没有发生太多事情：它只是一个坐标序列，它们之间没有“发生”任何事。实际上，即使出现了像 `line-to` 或 `curve-to` 之类的命令，`\path` 命令也不会对生成的路径“做”任何事。因此，所有的魔法都存在于 `node` 命令中。

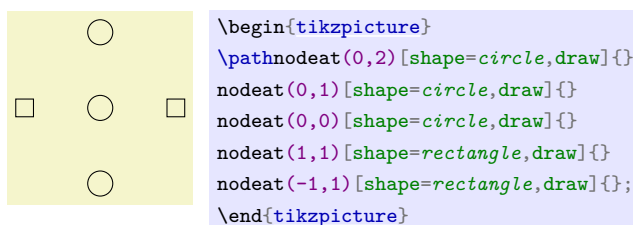
在上一个教程中，我们了解到 `node` 将在最后一个坐标处添加一段文本。因此，五个节点中的每个节点的文本都将添加在不同的位置。在上面的代码中，该文本为空（由于 `{}` 为空）。那么，为什么我们什么都看不到？答案是 `node` 操作的 `draw` 选项：它使“形状周围的文本”被绘制。

因此，代码 `(0,2) node [shape=circle,draw] {}` 表示：“在主路径中，首先将笔移动到坐标为 $(0,2)$ 的位置。然后，在创建节点时临时中止主路径的创建。该节点是绘制在一个空文本周围的 `circ` 节点。该圆将被绘出，但并没有被填充或进行其它操作。创建整个节点后，将保存该节点，直到完成主路径为止。”接下来的 `(0,1) node [shape=circle,draw] {}` 类似地产生以下效果：“通过将笔移至坐标为 $(0,1)$ 的位置后继续绘制主路径。然后在此位置也构造一个节点。主路径完成后也会显示该节点。”以此类推。

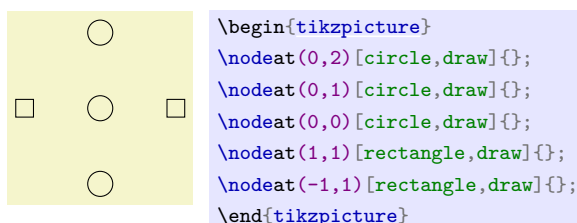
3.4 使用 `At` 语法放置节点

哈根现在了解 `node` 操作将节点添加到路径，但是使用 `\path` 操作创建路径似乎有点愚蠢。包括许多多余的 `move-to` 操作，仅用于放置节点。他很高兴得知有一些方法可以以更明智的方式添加节点。

首先，`node` 操作允许添加 `at` ($\langle coordinate \rangle$)，以便直接指定节点的位置，避免节点放置在最后一个坐标上。哈根然后可以据此编写以下内容：



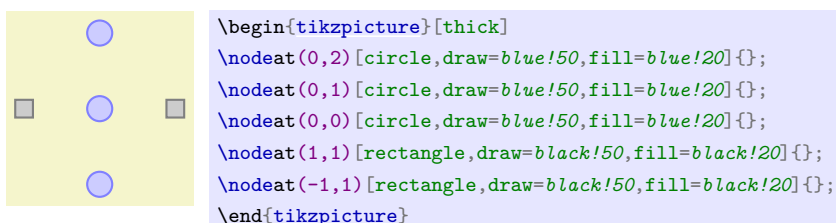
现在，哈根仍然只剩下一条空的路径，但至少该路径不再包含奇怪的 `move-to` 路线。事实证明，这可以进一步改进：`\node` 命令是 `\path node` 的缩写，这允许哈根这样编写：



相比于之前，哈根更喜欢现在这种语法。注意，哈根还省略了 `shape=`，与 `color=` 一样，TikZ 允许您在不引起混乱的情况下省略 `shape=`。

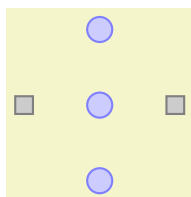
3.5 使用样式

出于冒险的感觉，哈根试图让节点看起来更好。在最终的图片中，圆和矩形需要用不同的颜色填充，从而产生以下代码：



虽然这在图片中看起来更好，但是代码开始变得有些难看。理想情况下，我们希望代码传递“有三个库所和两个变迁”的消息，而不是要使用哪种填充颜色。

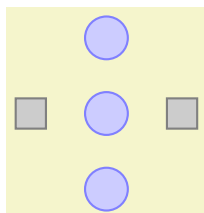
为了解决此问题，哈根使用了样式。他为库所定义了一种样式，为变迁定义了另一种样式：



```
\begin{tikzpicture}
[place/.style={circle,draw=blue!50,fill=blue!20,thick},
transition/.style={rectangle,draw=black!50,fill=black!20,thick}]
\nodeat(0,2)[place]{};
\nodeat(0,1)[place]{};
\nodeat(0,0)[place]{};
\nodeat(1,1)[transition]{};
\nodeat(-1,1)[transition]{};
\end{tikzpicture}
```

3.6 节点尺寸

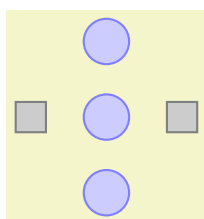
在哈根开始命名和连接节点之前，让我们首先确定节点具有最终外观。它们还有点太小。的确，哈根想知道为什么它们根本没有任何尺寸，毕竟文本是空的。原因是 TikZ 会在文本周围自动添加一些空格。空格的数量是使用选项 `inner sep` 设置的。因此，为了增加节点的大小，哈根可以这样写：



```
\begin{tikzpicture}
[inner sep=2mm,
place/.style={circle,draw=blue!50,fill=blue!20,thick},
transition/.style={rectangle,draw=black!50,fill=black!20,thick}]
\nodeat(0,2)[place]{};
\nodeat(0,1)[place]{};
\nodeat(0,0)[place]{};
\nodeat(1,1)[transition]{};
\nodeat(-1,1)[transition]{};
\end{tikzpicture}
```

但是，这实际上并不是达到预期效果的最佳方法。最好使用 `minimum size` 选项。该选项允许哈根指定节点应具有的最小尺寸。如果由于文本较长而实际上需要增大该节点，则该节点将变大，但是如果文本为空，则该节点将具有 `minimum size`。这个选项对于确保包含不同长度的文本的几个节点具有相同的大小也很有用。`minimum height` 和 `minimum width` 选项还允许您独立指定最小高度和最小宽度。

因此，哈根需要做的就是给节点提供 `minimum size`。为了安全起见，他还设置了 `inner sep=0pt`。这样可以确保节点真正具有 `minimum size`。而对于设置的非常小的最小尺寸，节点并不是具有这个最小尺寸，而是包含自动添加的空间所需的最小尺寸。



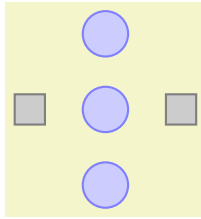
```
\begin{tikzpicture}
[place/.style={circle,draw=blue!50,fill=blue!20,thick,
inner sep=0pt,minimum size=6mm},
transition/.style={rectangle,draw=black!50,fill=black!20,thick,
inner sep=0pt,minimum size=4mm}]
\nodeat(0,2)[place]{};
\nodeat(0,1)[place]{};
\nodeat(0,0)[place]{};
\nodeat(1,1)[transition]{};
\nodeat(-1,1)[transition]{};
\end{tikzpicture}
```

3.7 节点命名

哈根的下一个目标是使用箭头连接节点。这似乎是一项棘手的工作，因为箭头不应从节点的中间开始，而是边界上的某个地方，哈根非常希望避免手动计算这些位置。

幸运的是，PGF 将为他执行所有必要的计算。但是，他首先必须为节点分配名称，以便以后可以引用它们。

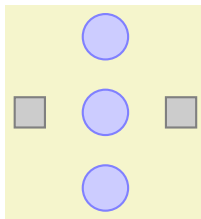
命名节点有两种方法。第一种是使用 `name=` 选项。第二种方法是在 `node` 操作之后的括号中写入节点的名称。哈根认为第二种方法似乎很奇怪，但是他很快就会改变看法。



```
%...setupstyles
\begin{tikzpicture}
\node(waiting1)at(0,2)[place]{};
\node(critical1)at(0,1)[place]{};
\node(semaphore)at(0,0)[place]{};
\node(leavecritical)at(1,1)[transition]{};
\node(entercritical)at(-1,1)[transition]{};
\end{tikzpicture}
```

哈根很高兴地注意到，这些名称有助于理解代码。节点的名称可以是任意的，但它们不应包含逗号，句点，括号，冒号和其他一些特殊字符。但是，它们可以包含下划线和连字符。

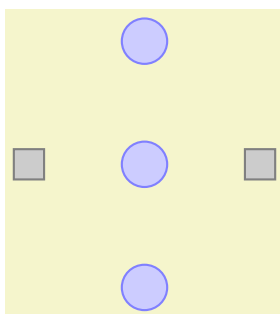
`node` 操作的语法在节点名称、`at` 和必须提供的选项的顺序方面是相当自由的。实际上，你甚至可以在 `node` 和花括号文本之间有多个选项块，它们会累积。你可以随意地重新排列它们，也许下面的方法更可取：



```
\begin{tikzpicture}
\node[place](waiting1)at(0,2){};
\node[place](critical1)at(0,1){};
\node[place](semaphore)at(0,0){};
\node[transition](leavecritical)at(1,1){};
\node[transition](entercritical)at(-1,1){};
\end{tikzpicture}
```

3.8 使用相对位置放置节点

尽管哈根仍然希望连接节点，但他首先希望再次解决另一个问题：节点的放置。尽管他喜欢 `at` 语法，在这种情况下，他宁愿将节点“彼此相对”地进行放置。因此，哈根想说 `critical 1` 节点应在 `waiting 1` 节点的下方，无论 `waiting 1` 节点可能在哪个位置。有多种方法可以实现此目的，但是在哈根的案例中，最好的方法是使用 `belloe` 选项：



```
\usetikzlibrary{positioning}
\begin{tikzpicture}
\node[place](waiting){};
\node[place](critical)[below=of waiting]{};
\node[place](semaphore)[below=of critical]{};
\node[transition](leavecritical)[right=of critical]{};
\node[transition](entercritical)[left=of critical]{};
\end{tikzpicture}
```

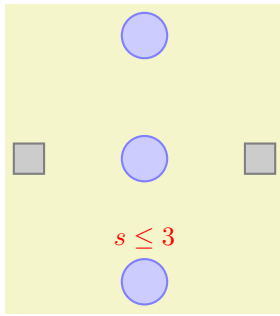
加载 `positioning` 库后，当 `belloe` 选项后面跟着 `of` 时，则将节点的位置移动到指定方向，使其置于与指定节点的距离为 `node distance` 的位置。`node distance` 要么是节点中心之间的距离（当 `on grid` 选项被设置为 `true` 时），要么是边界之间的距离（当 `on grid` 选项被设置为 `false` 时，这是默认值）。

即使上面的代码与早期的代码具有相同的效果，哈根仍可以将其传递给他的同事，他们甚至可以不必看图片就可以阅读和理解它。

3.9 在节点旁添加标签

在查看哈根如何连接节点之前，让我们将容量 $s \leq 3$ 添加到底部节点。为此，可以采用两种方法：

1. 哈根只需要在 `semaphore` 节点的 `north` 锚点上方添加一个新节点。

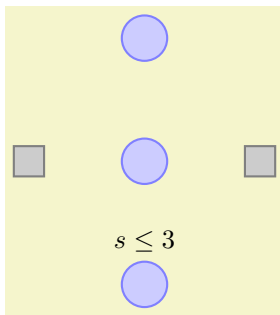


```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};

\node[red,above] at (semaphore.north) {$s \leq 3$};
\end{tikzpicture}
```

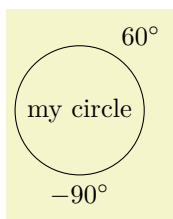
这是一种将“始终有效”的通用方法。

2. 哈根还可以使用特殊的 `label` 选项。该选项被赋给 `node`, 使另一个节点被添加到给出该选项的节点旁边。这里是一个想法: 当创建 `semaphore` 节点时, 我们希望表明我们想要另一个位置高于它的节点。为此, 我们使用选项 `label=above:$s \leq 3$`。此选项的解释如下: 我们希望在 `semaphore` 上方有一个节点, 此节点应显示为“ $s \leq 3$ ”。除了 `above` 我们还可以在冒号或在 60 之类的数字之前使用 `below left`。



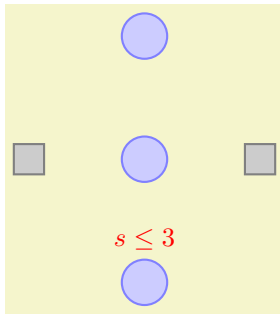
```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical,
label=above:$s \leq 3$] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\end{tikzpicture}
```

也可以给多个 `label` 选项, 将会绘制多个标签。



```
\tikz
\node[circle,draw,label=60:$60^\circ$,label=below:$-90^\circ$] {mycircle};
```

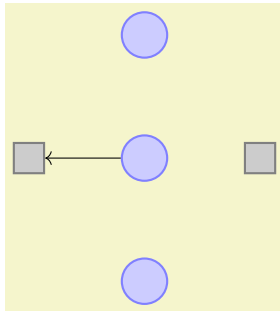
哈根对 `label` 选项并不完全满意, 因为标签不是红色的。为了实现这一点, 他有两个选择: 首先, 他可以重新定义 `every label` 的风格。其次, 他可以向标签的节点中添加选项。这些选项是按照 `label=` 给出的, 所以他在上面写 `label=[red]above:$s \leq 3$`。然而, 这并不能完全工作, 因为 `TEX` 认为 `]` 结束了 `semaphore` 节点的整个选项列表。因此, 哈根必须添加括号这样使用 `label={ [red]above:$s \leq 3$ }`。由于这看起来有点丑, 哈根决定重新定义 `every label` 风格。



```
\usetikzlibrary {positioning}
\begin{tikzpicture}[every label/.style={red}]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical,
label=above:$s\leq 3$] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\end{tikzpicture}
```

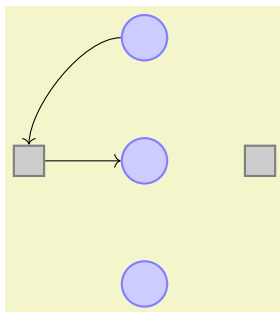
3.10 连接节点

现在是连接节点的时候了。让我们先从简单的事情开始，即从直线的 `enter critical` 到 `critical`。我们希望此直线从 `enter critical` 的右侧开始，并在 `critical` 的左侧结束。为此，我们可以使用节点的锚点。每个节点都定义了一大堆锚点，这些锚点位于其边界上或内部。例如，`center` 锚点位于节点的中心，`west` 锚点位于节点的左侧，依此类推。要访问节点的坐标，我们使用一个坐标，该坐标包含节点名称，后跟一句点，再后跟锚点名称：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\draw[->] (critical.west)--(entercritical.east);
\end{tikzpicture}
```

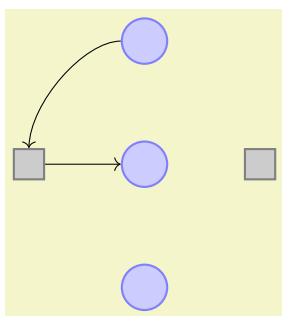
接下来，让我们处理从 `waiting` 到 `enter critical` 的曲线。可以使用曲线和控件来指定：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\draw[->] (entercritical.east)--(critical.west);
\draw[->] (waiting.west)..controls+(left:5mm)and+(up:5mm)
..(entercritical.north);
\end{tikzpicture}
```

哈根看到了他现在如何添加他所有的边，但是整个过程看起来很笨拙，不是很灵活。再次，代码似乎使图形的结构变得晦涩，而不是展示图形是如何绘制而来的。

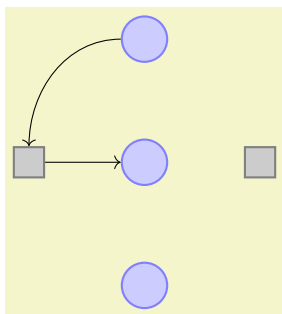
因此，让我们开始改进边的代码。首先，哈根可以省略锚点：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\draw[->] (entercritical) -- (critical);
\draw[->] (waiting) .. controls+(left:8mm) and+(up:8mm)
.. (entercritical);
\end{tikzpicture}
```

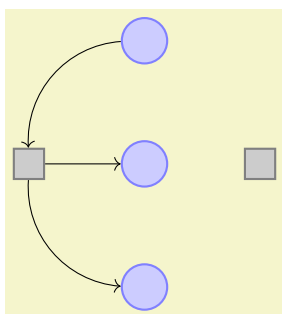
哈根有点惊讶，这是可行的。毕竟，TikZ 怎么知道从 `enter critical` 到 `critical` 的直线实际上应该从边界开始呢？每当 TikZ 遇到作为“坐标”的整个节点名时，它就会尝试“聪明地”为该节点选择锚点。根据接下来发生的情况，TikZ 将选择位于节点边界上的锚点，该锚点位于到下一个坐标或控制点的直线上。确切的规则有点复杂，但选择的点通常是正确的，如果不是这样，哈根仍然可以手工指定所需的锚点。

哈根现在想以某种方式简化曲线操作。事实证明，这可以使用特殊的路径操作来实现：`to` 操作。此操作有很多选项（您甚至可以自己定义新选项）。这组选项对哈根很有用：`in` 和 `out` 组。这些选项采用曲线应离开或到达起始坐标或目标坐标的角度。如果没有这些选项，则会绘制一条直线：



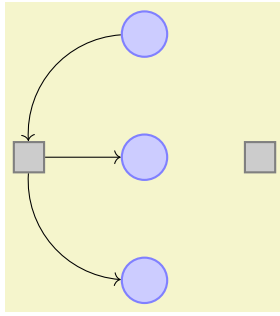
```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\draw[->] (entercritical) to (critical);
\draw[->] (waiting) to[out=180,in=90] (entercritical);
\end{tikzpicture}
```

`to` 操作还有另一种选项，它甚至更适合于哈根的问题：`bend right` 选项。此选项也需要一个角度，但是此角度仅指定曲线向右弯曲的角度：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
\draw[->] (entercritical) to (critical);
\draw[->] (waiting) to[bend right=45] (entercritical);
\draw[->] (entercritical) to[bend right=45] (semaphore);
\end{tikzpicture}
```

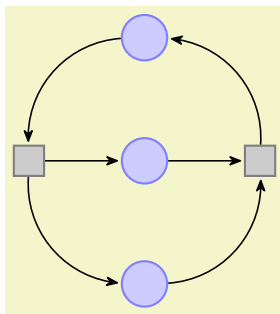
现在是时候让哈根学习指定边的另一种方法了：使用 `edge` 路径操作。此操作与 `to` 非常相似。操作，但是有一个重要的区别：与节点一样，`edge` 操作生成的边不是主路径的一部分，而是仅在以后添加。这看似不太重要，但是会带来一些不错的效果。例如，每条边可以有自己的箭头提示和颜色，依此类推，而且所有边都可以在同一路径上给出。这使哈根可以编写以下内容：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};
\node[transition] (leavecritical) [right=of critical] {};
\node[transition] (entercritical) [left=of critical] {};
edge[->] (critical)
edge[<- ,bend left=45] (waiting)
edge[-> ,bend right=45] (semaphore);
\end{tikzpicture}
```

每个 edge 都会绘制一条新路径，每条路径的命令都由 enter critical 节点和 edge 命令之后的节点组成，节点之间采用 to 连接。

最后要介绍的是 pre 和 post 两种样式，并使用 bend angle=45 选项一劳永逸地设置弯曲角度：



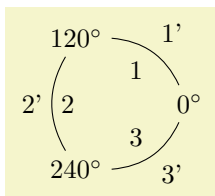
```
\usetikzlibrary {arrows.meta,positioning}
%Stylesplaceandtransitionasbefore
\begin{tikzpicture}
[bend angle=45,
pre/.style={<- ,shorten <=1pt,>={Stealth[round]},semithick},
post/.style={-> ,shorten >=1pt,>={Stealth[round]},semithick}]

\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};

\node[transition] (leavecritical) [right=of critical] {}
edge[pre] (critical)
edge[post,bend right] (waiting)
edge[pre,bend left] (semaphore);
\node[transition] (entercritical) [left=of critical] {}
edge[post] (critical)
edge[pre,bend left] (waiting)
edge[post,bend right] (semaphore);
\end{tikzpicture}
```

3.11 在线旁添加标签

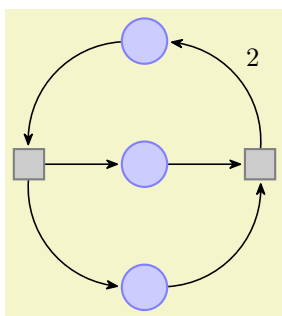
哈根接下来需要添加的是弧线上的“2”。为此，哈根可以使用 TikZ 的自动节点放置：通过添加选项 auto，TikZ 可以将节点定位在曲线和线上，使得它们不在曲线上而是位于曲线旁边。添加 swap 将对该线的标签进行镜像。这是一个一般示例：



```
\begin{tikzpicture}[auto,bend right]
\node(a)at(0:1){$0^\circ$};
\node(b)at(120:1){$120^\circ$};
\node(c)at(240:1){$240^\circ$};

\draw(a)tonode{1}node[swap]{1'}(b)
(b)tonode{2}node[swap]{2'}(c)
(c)tonode{3}node[swap]{3'}(a);
\end{tikzpicture}
```

这里发生了什么？节点以某种方式在 to 操作中给定！完成此操作后，节点被放置在 to 操作创建的曲线或直线的中间。然后，auto 选项导致节点以这样一种方式进行移动，即它不在曲线上，而是在曲线旁边。在本例中，我们为每个 to 操作提供两个节点。对于哈根来说，auto 选项并不是真正必要的，因为两个“2”标签也可以很容易地“手动”放置。然而，在一个有许多边的复杂情形中，自动放置可能是一件好事。



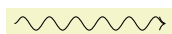
```
\usetikzlibrary {arrows.meta,positioning}
%Stylesasbefore
\begin{tikzpicture}[bend angle=45]
\node[place] (waiting){};
\node[place] (critical) [below=of waiting]{};
\node[place] (semaphore) [below=of critical]{};

\node[transition] (leavecritical) [right=of critical]{}
edge[pre] (critical)
edge[post,bend right]node[auto,swap]{2}(waiting)
edge[pre,bend left] (semaphore);
\node[transition] (entercritical) [left=of critical]{}
edge[post] (critical)
edge[pre,bend left] (waiting)
edge[post,bend right] (semaphore);
\end{tikzpicture}
```

3.12 添加隐藏线和多行文本

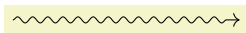
利用节点机制，哈根现在可以轻松创建两个 Petri 网络。他不确定的是如何在网之间创建蛇形线。

为此，他可以使用 *decoration* 库。要绘制蛇形线，哈根只需要为路径设置两个选项 `decoration=snake` 和 `decorate`。这将导致路径的所有线都被蛇形替换。也可以仅在路径的某些部分使用蛇形，但是哈根不需要。



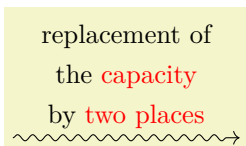
```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
\draw[->,decorate,decoration=snake](0,0)--(2,0);
\end{tikzpicture}
```

好吧，这看起来还不太正确。问题是蛇形恰好在箭头开始的位置结束。幸运的是，这里有一个选项可以帮助您。同样，蛇形应该更小一些，这可能会受到更多选择的影响。



```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
\draw[->,decorate,
decoration={snake,amplitude=.4mm,segmentlength=2mm,postlength=1mm}]
(0,0)--(3,0);
\end{tikzpicture}
```

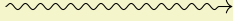
现在，哈根需要在蛇形上方添加文本。该文本是多行文本，因此具有一定的挑战性。哈根对此有两种选择：首先，他可以指定 `align=center`。然后使用 `\\` 命令以在所需位置强制换行。



```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
\draw[->,decorate,
decoration={snake,amplitude=.4mm,segmentlength=2mm,postlength=1mm}]
(0,0)--(3,0)
node[above,align=center,midway]
{
replacementof\\
the\textcolor{red}{capacity}\\
by\textcolor{red}{two places}
};
\end{tikzpicture}
```

哈根除了可以手动指定换行符之外，还可以为文本指定宽度并让 $\text{T}_{\text{E}}\text{X}$ 为他自动换行：

replacement of
the **capacity**
by **two places**



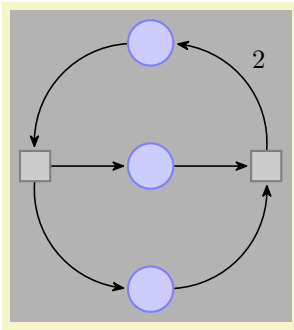
```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
\draw[->,decorate,
decoration={snake,amplitude=.4mm,segmentlength=2mm,postlength=1mm}]
(0,0)--(3,0)
\node[above,text width=3cm,align=center,midway]
{
replacementofthe\textcolor{red}{capacity}by
\textcolor{red}{twoplaces}
};
\end{tikzpicture}
```

3.13 使用图层：背景矩形

哈根仍然需要添加背景矩形。这些有点棘手：哈根希望在 Petri 网绘制完成之后再绘制矩形。原因是只有这样，他才能方便地得到矩形角点的坐标。如果哈根首先画出这个矩形，那么他需要知道 Petri 网的确切大小，但他并不知道。

解决方案是使用图层。backgrounds 库已加载，哈根可以将其图片的一部分放置在具有“背景层”的范围内。然后，图片的该部分成为该环境的参数所指定的图层的一部分。当 {tikzpicture} 在环境结束时，各层从背景层开始相互重叠。这导致在背景层上绘制的所有内容都位于主文本的后面。

下一个棘手的问题是，矩形应该有多大？当然，哈根可以“手工”计算大小，也可以使用一些聪明的方法来计算节点的 x 和 y 的坐标，但如果使用 TikZ 计算一个所有节点都适合的矩形会更好。为此，可以使用 fit 库。它定义了 fit 选项，当将该选项提供给节点时，将对该节点的大小进行调整和移动，以便准确地覆盖作为 fit 选项参数提供的所有节点和坐标。



```
\usetikzlibrary {arrows.meta,backgrounds,fit,positioning}
%Stylesasbefore
\begin{tikzpicture}[bend angle=45]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};

\node[transition] (leavecritical) [right=of critical] {}
edge[pre] (critical)
edge[post,bend right] node[auto,swap] {2} (waiting)
edge[pre,bend left] (semaphore);
\node[transition] (entercritical) [left=of critical] {}
edge[post] (critical)
edge[pre,bend left] (waiting)
edge[post,bend right] (semaphore);

\begin{scope}[on background layer]
\node[fill=black!30,fit=(waiting)(critical)(semaphore)
(leavecritical)(entercritical)] {};
\end{scope}
\end{tikzpicture}
```

3.14 完整的代码

哈根现在终于把所有东西都放在一起了。这时，他才知道已经有了一个绘制 Petri 网的库！事实证明，该库主要提供与哈根相同的定义。例如，它定义了 place，与哈根类似。调整代码以便使用该库，缩短了哈根的代码，如下所示。

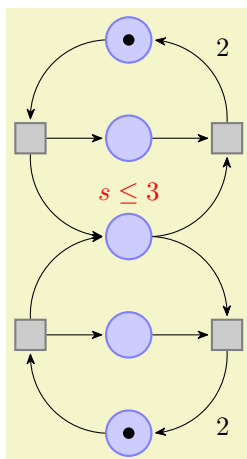
首先，哈根需要较少的样式定义，但他仍然需要指定库所和变迁的颜色。

```

\begin{tikzpicture}
[node distance=1.3cm,on grid,>={Stealth[round]},bend angle=45,auto,
every place/.style={minimum size=6mm,thick,draw=blue!75,fill=blue!20},
every transition/.style={thick,draw=black!75,fill=black!20},
red place/.style={place,draw=red!75,fill=red!20},
every label/.style={red}]

```

现在我们得到了网络的代码：

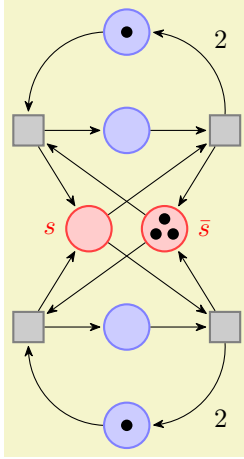


```

\usetikzlibrary {arrows.meta,petri,positioning}
\node[place,tokens=1](w1){};
\node[place](c1)[below=of w1]{};
\node[place](s)[below=of c1,label=above:$s\leq 3$]{};
\node[place](c2)[below=of s]{};
\node[place,tokens=1](w2)[below=of c2]{};

\node[transition](e1)[left=of c1]{}
edge[pre,bend left](w1)
edge[post,bend right](s)
edge[post](c1);
\node[transition](e2)[left=of c2]{}
edge[pre,bend right](w2)
edge[post,bend left](s)
edge[post](c2);
\node[transition](l1)[right=of c1]{}
edge[pre](c1)
edge[pre,bend left](s)
edge[post,bend right]node[swap]{2}(w1);
\node[transition](l2)[right=of c2]{}
edge[pre](c2)
edge[pre,bend right](s)
edge[post,bend left]node{2}(w2);

```



```
\usetikzlibrary {arrows.meta,petri,positioning}
\begin{scope}[xshift=6cm]
\node[place,tokens=1](w1'){};
\node[place](c1')[below=of w1']{};
\node[red place](s1')[below=of c1',xshift=-5mm]
[label=left:$s$]{};
\node[red place,tokens=3](s2')[below=of c1',xshift=5mm]
[label=right:$\bar{s}$]{};
\node[place](c2')[below=of s1',xshift=5mm]{};
\node[place,tokens=1](w2')[below=of c2']{};

\node[transition](e1')[left=of c1']{}
edge[pre,bend left](w1')
edge[post](s1')
edge[pre](s2')
edge[post](c1');
\node[transition](e2')[left=of c2']{}
edge[pre,bend right](w2')
edge[post](s1')
edge[pre](s2')
edge[post](c2');
\node[transition](l1')[right=of c1']{}
edge[pre](c1')
edge[pre](s1')
edge[post](s2')
edge[post,bend right]node[swap]{2}(w1');
\node[transition](l2')[right=of c2']{}
edge[pre](c2')
edge[pre](s1')
edge[post](s2')
edge[post,bend left]node{2}(w2');
\end{scope}
```

背景和蛇形的代码如下：

```
\begin{scope}[on background layer]
\node(r1)[fill=black!10,rounded corners,fit=(w1)(w2)(e1)(e2)(l1)(l2)]{};
\node(r2)[fill=black!10,rounded corners,fit=(w1')(w2')(e1')(e2')(l1')(l2')]{};
\end{scope}

\draw[shorten >=1mm,->,thick,decorate,
decoration={snake,amplitude=.4mm,segmentlength=2mm,
pre=moveto,prelength=1mm,postlength=2mm}]
(r1)--(r2)node[above=1mm,midway,text width=3cm,align=center]
{replacement of the \textcolor{red}{capacity} by \textcolor{red}{two places}};
\end{tikzpicture}
```

4 教程：欧几里得的琥珀版本的几何原本

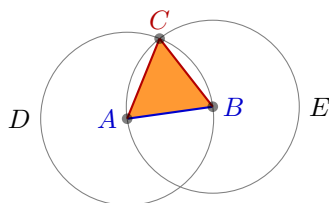
在第三个教程中，我们将了解如何使用 TikZ 绘制几何构造。

欧几里德（Euclid）目前正忙于撰写他的新书系列，其标题是“几何原本”（欧克利德不确定该标题是否会正确地向后人传达该系列的信息，但他打算在它交给出版商之前更改书名）。据他所知，他在纸莎草纸上写下了文字和图形，但是他的出版商突然坚持要求他必须以电子形式提交。欧几里得试图与出版商争辩，电子技术只会在数千年后才被发现，但是出版商告诉他，纸莎草纸的使用不再是尖端技术，欧几里得将不得不跟上现代工具的发展。

欧几里得有些不满，开始将名为“书 I，命题 I”的纸莎草纸改成琥珀版本。

4.1 书 I，命题 I

他的纸莎草纸上的图形看起来像这样：¹



Proposition I

To construct an equilateral triangle on a given finite straight line.

Let AB be the given finite straight line. It is required to construct an equilateral triangle on the straight line AB .

Describe the circle BCD with center A and radius AB . Again describe the circle ACE with center B and radius BA . Join the straight lines CA and CB from the point C at which the circles cut one another to the points A and B .

Now, since the point A is the center of the circle CDB , therefore AC equals AB . Again, since the point B is the center of the circle CAE , therefore BC equals BA . But AC was proved equal to AB , therefore each of the straight lines AC and BC equals AB . And things which equal the same thing also equal one another, therefore AC also equals BC . Therefore the three straight lines AC , AB , and BC equal one another. Therefore the triangle ABC is equilateral, and it has been constructed on the given finite straight line AB .

让我们看看欧几里得如何将其转换为 TikZ 代码。

4.1.1 配置环境

与以前的教程一样，欧几里得需要加载 TikZ 以及一些库。这些库是包括 `calc`、`intersections`、`through` 和 `backgrounds`。根据他使用的格式，欧几里得将在序言中使用以下其中之一：

```
%ForLaTeX:
\usepackage{tikz}
\usetikzlibrary{calc,intersections,through,backgrounds}
```

```
%ForplainTeX:
\inputtikz.tex
\usetikzlibrary{calc,intersections,through,backgrounds}
```

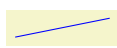
```
%ForConTeXt:
\usemodule[tikz]
\usetikzlibrary[calc,intersections,through,backgrounds]
```

¹该文本摘自 David E. Joyce 的 Euclid's Elements 精彩互动版本，可在他在 Clark University 的网站上找到。

4.1.2 线段 AB

欧几里得希望绘制图片的第一部分是 AB 线段。这很容易，像 `\draw (0,0) --(2,1);` 就可以做到。但是，欧几里得不希望随后将两个点 A 和 B 分别称为 $(0,0)$ 和 $(2,1)$ 。相反，他希望只写 A 和 B ，确实，他的书的要点是 A 和 B 点可以是任意的，其他所有点（如 C ）都是根据其位置构造的。如果欧几里得明确地写下 C 的坐标，它就不会这样做了。??

因此，欧几里得从使用 `\coordinate` 命令定义两个坐标开始：



```
\begin{tikzpicture}
\coordinate(A)at(0,0);
\coordinate(B)at(1.25,0.25);

\draw[blue](A)--(B);
\end{tikzpicture}
```

这已经足够简单。此时缺少的是坐标的标签。欧几里得不希望它们在这些坐标点上，而是在坐标点旁边。他决定使用 `label` 选项：



```
\begin{tikzpicture}
\coordinate[label=left:\textcolor{blue}{$A$}](A)at(0,0);
\coordinate[label=right:\textcolor{blue}{$B$}](B)at(1.25,0.25);

\draw[blue](A)--(B);
\end{tikzpicture}
```

在这一点上，欧几里得认为如果点 A 和 B 在某种意义上是“随机的”，那就更好了。这样，无论是欧几里得还是读者都不会犯错误，认为这些点的位置是“理所当然的”。欧几里得很高兴得知 TikZ 中的 `rand` 函数完全满足他的需要：它产生介于 -1 和 1 之间的数字。由于 TikZ 可以进行一些数学运算，因此欧几里得可以如下更改点的坐标：

```
\coordinate[...](A)at(0+0.1*rand,0+0.1*rand);
\coordinate[...](B)at(1.25+0.1*rand,0.25+0.1*rand);
```

这确实很好。但是，欧几里得并不十分满意，因为他希望“主坐标” $(0,0)$ 和 $(1.25,0.25)$ 与扰动 $0.1(rand,rand)$ “保持分离”。这意味着，他想将坐标 A 指定为“在 $(0,0)$ 上加上向量 $(rand,rand)$ 的十分之一”。

事实证明 `calc` 库允许他精确地进行这种计算。加载此库后，可以使用以 $(\$$ 开始以及以 $\$)$ 结束的特殊坐标，而不只是 $($ 和 $)$ 。在这些特殊坐标内，您可以给出坐标的线性组合。（请注意，美元符号仅用于表示正在进行“计算”；不进行数学排版。）

坐标新的代码如下：

```
\coordinate[...](A)at($ (0,0) + .1*(rand,rand) $);
\coordinate[...](B)at($ (1.25,0.25) + .1*(rand,rand) $);
```

请注意，如果此类计算中的坐标有一个乘数（例如 $.1$ ），则必须直接在坐标的圆括号之前放置 $*$ 。您可以嵌套进行这样的计算。

4.1.3 以点 A 为圆心的圆

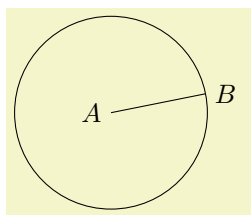
第一个棘手的结构是以点 A 为圆心的圆。稍后我们将看到如何以非常简单的方式执行此操作，但首先让我们以一种更“困难”方式执行此操作。

想法如下：我们在点 A 周围画一个圆，其半径由线 AB 的长度给定。困难在于计算这条线的长度。

有两种“差不多”可以解决这个问题的想法：首先，我们可以说 $(\$ (A) - (B) \$)$ 是对于 A 和 B 之差的向量。我们需要的只是该向量的长度。第二，给定两个数字 x 和 y ，可以在数学表达式中使用 `vecLen(x,y)`。该表达式的值为 $\sqrt{x^2 + y^2}$ ，恰好是所需的长度。

唯一剩下的问题是访问向量 AB 的 x 和 y 坐标。为此，我们需要一个新概念：*let* 操作。*let* 操作可以在路径上的任何位置进行，在这些位置可以进行常规的路径操作（如 *line-to* 或 *move-to*）。*let* 操作的作用是计算一些坐标，并将结果分配给特殊的宏。这些宏可以方便地访问坐标点的 x 和 y 坐标。

欧几里得将编写以下内容：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);

\draw(A)let
\p1=($(B)-(A)$)
in
circle({veclen(\x1,\y1)});
\end{tikzpicture}
```

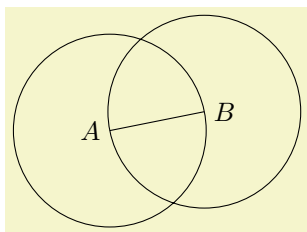
let 操作中的每个赋值都以 $\backslash p$ 开始，通常后跟一个 \langle 数字 \rangle ，然后是等号和坐标。坐标将被计算，结果被存储在内部。之后你可以使用下面的表达方式：

1. $\backslash x\langle$ 数字 \rangle 产生指定点的 x 坐标。
2. $\backslash y\langle$ 数字 \rangle 产生指定点的 y 坐标。
3. $\backslash p\langle$ 数字 \rangle 产生与 $\backslash x\langle$ 数字 \rangle , $\backslash y\langle$ 数字 \rangle 相同的结果。

在 *let* 操作中可以有多个赋值，只需用逗号分隔即可。在以后的赋值中，您已经可以使用以前的赋值的结果。

注意 $\backslash p1$ 不是通常意义上的坐标。相反，它只是扩展为一个字符串，如 $10pt,20pt$ 。例如，你不能写 $(\backslash p1.center)$ 因为它会扩展到 $(10pt,20pt.center)$ ，这没有意义。

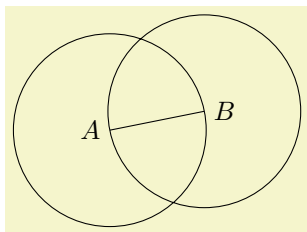
接下来，我们想要同时画两个圆。每个圆半径是 $veclen(\backslash x1, \backslash y1)$ 。只计算一次半径似乎很自然。为此，我们还可以使用 *let* 操作：他使用 $\backslash n2 = \dots$ 代替 $\backslash p1 = \dots$ 。这里，“ n ”代表“数字”（而“ p ”代表“点”）。数字的赋值后面应该跟一个用花括号括起来的数字。



```
\usetikzlibrary {calc}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);

\drawlet\p1=($(B)-(A)$),
\n2={veclen(\x1,\y1)}
in
(A)circle(\n2)
(B)circle(\n2);
\end{tikzpicture}
```

在上面的例子中，您可能想知道 $\backslash n1$ 会产生什么？答案是它将是未定义的， $\backslash p$ ， $\backslash x$ ，和 $\backslash y$ 宏引用相同的逻辑坐标点，而 $\backslash n$ 宏有“它自己的名称空间”。我们甚至可以把例子中的 $\backslash n2$ 替换成 $\backslash n1$ 而且仍然工作正常。实际上，这些宏后面的数字只是普通的 $T_E X$ 参数。我们也可以使用更长的名字，但我们必须使用花括号：

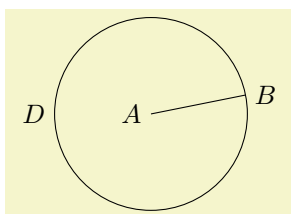


```
\usetikzlibrary {calc}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);

\drawlet\p1=($(B)-(A)$),
\n{radius}={veclen(\x1,\y1)}
in
(A)circle(\n{radius})
(B)circle(\n{radius});
\end{tikzpicture}
```

在本节的开始，我们承诺有一种更简单的方法来创建想要的圆。诀窍是通过使用 `through` 库。顾名思义，它包含用于创建经过给定点的形状的代码。

我们需要的选项是 `circle through`。将此选项提供给节点并具有以下效果：首先，它将使节点的内部和外部的间隔设置为零。然后它将节点的形状设置为 `circle`。最后，它设置节点的半径，使其通过给定的 `circle through`。这个半径的计算方法和上面一样。



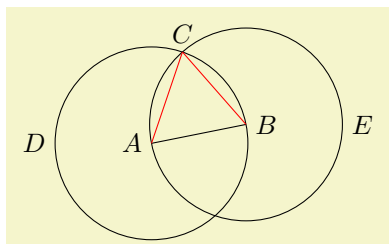
```
\usetikzlibrary {through}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);

\node[draw,circle through=(B),label=left:$D$]at(A){};
\end{tikzpicture}
```

4.1.4 圆的交集

欧几里得现在可以画线段和圆了。最后一个问题是计算两个圆的交点。如果你想手工做的话，这个计算有点复杂。幸运的是，`intersections` 库允许我们计算任意路径的交集。

其思想很简单：首先，使用 `name path` 选项给两条路径“命名”。然后，在以后的某个时候，您可以使用名称为 `name intersections` 选项，它在路径的所有交点上创建名为 `intersec-1`、`intersec-2` 的坐标，以此类推。欧几里德将名称 `D` 和 `E` 分配给两个圆的路径（这两个圆恰好与节点本身的名称相同，但节点及其路径位于不同的“名称空间”中）。



```

\usetikzlibrary {intersections,through}
\begin{tikzpicture}
\coordinate[label=left:$A$] (A) at (0,0);
\coordinate[label=right:$B$] (B) at (1.25,0.25);
\draw(A)--(B);

\node(D)[name path=D,draw,circle through=(B),label=left:$D$] at (A){};
\node(E)[name path=E,draw,circle through=(A),label=right:$E$] at (B){};

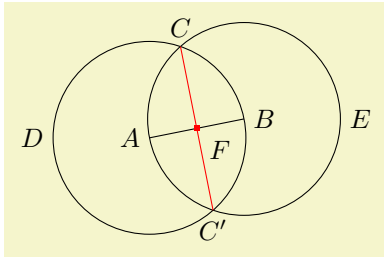
%Name the coordinates, but don't draw anything:
\path[name intersections={of=D and E}];

\coordinate[label=above:$C$] (C) at (intersection-1);

\draw[red] (A)--(C);
\draw[red] (B)--(C);
\end{tikzpicture}

```

事实证明，这个过程可以进一步缩减：`name intersections` 接受一个可选参数 `by`，它允许您为坐标指定名称和选项。这将创建更紧凑的代码。虽然欧几里得在现在的图中不需要它，但它只是计算直线 AB 的等分线的一个小步骤：



```

\usetikzlibrary {intersections,through}
\begin{tikzpicture}
\coordinate[label=left:$A$] (A) at (0,0);
\coordinate[label=right:$B$] (B) at (1.25,0.25);
\draw[name path=A--B] (A)--(B);

\node(D)[name path=D,draw,circle through=(B),label=left:$D$] at (A){};
\node(E)[name path=E,draw,circle through=(A),label=right:$E$] at (B){};

\path[name intersections={of=D and E,by={ [label=above:$C$] C, [label=below:$C'$] C' }}];

\draw[name path=C--C',red] (C)--(C');

\path[name intersections={of=A--B and C--C',by=F}];
\node[fill=red,inner sep=1pt,label=-45:$F$] at (F){};
\end{tikzpicture}

```

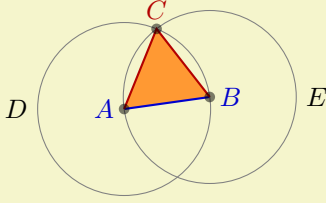
4.1.5 完整代码

回到欧几里得的代码。他引入了一些宏来简化工作，比如用于排版蓝色的 A 的 `\A` 宏。他还使用了 `background` 层来绘制最后所有东西后面的三角形。

Proposition I

To construct an *equilateral triangle* on a given *finite straight line*.

Let AB be the given finite straight line. ...



```
\usetikzlibrary {backgrounds,calc,intersections,through}
\begin{tikzpicture}[thick,help lines/.style={thin,draw=black!50}]
\def\A{\textcolor{input}{A}}\def\B{\textcolor{input}{B}}
\def\C{\textcolor{output}{C}}\def\D{\textcolor{D}{D}}
\def\E{\textcolor{E}{E}}

\colorlet{input}{blue!80!black}\colorlet{output}{red!70!black}
\colorlet{triangle}{orange}

\coordinate[label=left:\A](A)at(0,0)+.1*(rand,rand);
\coordinate[label=right:\B](B)at(1.25,0.25)+.1*(rand,rand);

\draw[input](A)--(B);

\node[name path=D,help lines,draw,label=left:\D](D)at(A)[circle through=(B)]{};
\node[name path=E,help lines,draw,label=right:\E](E)at(B)[circle through=(A)]{};

\path[name intersections={of=DandE,by={label=above:\C}}];

\draw[output](A)--(C)--(B);

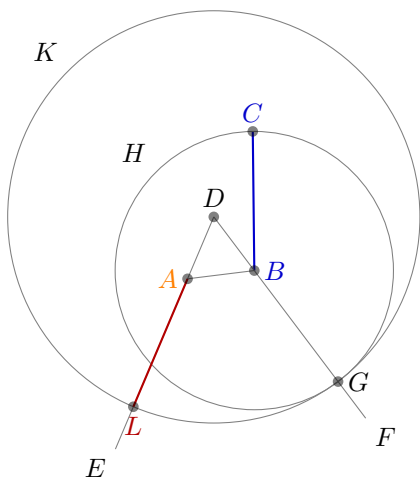
\foreach\pointin{A,B,C}
\fill[black,opacity=.5](\point)circle(2pt);

\begin{pgfonlayer}{background}
\fill[triangle!80](A)--(C)--(B)--cycle;
\end{pgfonlayer}

\node[below right,text width=10cm,align=justify]at(4,3){
\small\textbf{Proposition I}\par
\emph{To construct an \textcolor{triangle}{equilateral triangle}
on a given \textcolor{input}{finite straight line}.}
\par\vskip1em
Let \A\B be the given \textcolor{input}{finite straight line}. \dots
};
\end{tikzpicture}
```

4.2 书 I, 命题 II

几何原本中的第二个命题如下:



Proposition II

To place a *straight line* equal to a given *straight line* with one end at a *given point*.

Let A be the given point, and BC the given *straight line*. It is required to place a *straight line* equal to the given *straight line* BC with one end at the point A .

Join the *straight line* AB from the point A to the point B , and construct the equilateral triangle DAB on it.

Produce the *straight lines* AE and BF in a *straight line* with DA and DB . Describe the circle CGH with center B and radius BC , and again, describe the circle GKL with center D and radius DG .

Since the point B is the center of the circle CGH , therefore BC equals BG . Again, since the point D is the center of the circle GKL , therefore DL equals DG . And in these DA equals DB , therefore the remainder AL equals the remainder BG . But BC was also proved equal to BG , therefore each of the *straight lines* AL and BC equals BG . And things which equal the same thing also equal one another, therefore AL also equals BC .

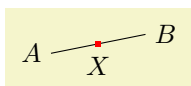
Therefore the *straight line* AL equal to the given *straight line* BC has been placed with one end at the *given point* A .

4.2.1 使用分割运算进行点 D 的构造

欧几里得的构造从“引用”命题 I 开始，用来构造点 D 。现在，虽然我们可以简单地重复这个结构，但要画出所有这些圆并做所有这些复杂的结构似乎有点麻烦。

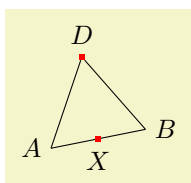
因此，TikZ 支持一些简化。首先，有一种简单的语法可以计算从 p 到 q 的直线上的“中间”点：将这两点放在坐标计算中，请记住，它们以 $(\$$ 开头，以 $\$)$ 结束，然后使用 $!(\text{分割比})!$ 将它们组合起来。 $\langle\text{分割比}\rangle$ 为 0 表示第一个坐标， $\langle\text{分割比}\rangle$ 为 1 表示第二个坐标，其间的值表示从 p 到 q 的线段上的点。因此，语法类似于用于混合颜色的 `xcolor` 语法。

这是 AB 线段中点的计算：



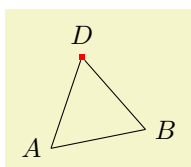
```
\usetikzlibrary {calc}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);
\node[fill=red,inner sep=1pt,label=below:$X$](X)at($(A)!.5!(B)$){};
\end{tikzpicture}
```

欧几里得第二个命题中的点 D 的计算要复杂一些。可以表示为：考虑从 X 到 B 的线段。假设我们将这条线段绕 X 旋转 90° ，然后将其拉伸 $\sin(60^\circ) \cdot 2$ 倍，这将产生所需的点 D 。我们可以使用上面的分割运算编辑器进行拉伸，对于旋转，我们需要一个新的编辑器：旋转编辑器。这个想法是，在进行中间计算时，第二个坐标可以加上一个角度作为前缀。然后正常计算中间点（好像没有给出角度），但是结果点围绕第一个点旋转了这个角度。



```
\usetikzlibrary {calc}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);
\node[fill=red,inner sep=1pt,label=below:$X$](X)at($(A)!.5!(B)$){};
\node[fill=red,inner sep=1pt,label=above:$D$](D)at
($(X)!.5!(B)!.5!(B)!.5!(B)$){};
\draw(A)--(D)--(B);
\end{tikzpicture}
```

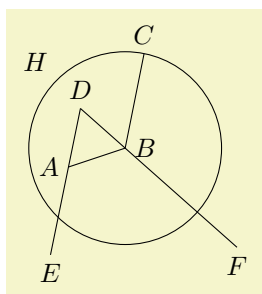
最后，不必显式命名点 X 。相反，就像在 `xcolor` 宏包中一样，可以链接分割编辑器：



```
\usetikzlibrary {calc}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(1.25,0.25);
\draw(A)--(B);
\node[fill=red,inner sep=1pt,label=above:$D$](D)at
($(A)!.5!(B)!.5!(B)!.5!(B)$){};
\draw(A)--(D)--(B);
\end{tikzpicture}
```

4.2.2 线段与圆相交

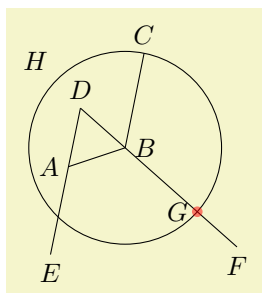
绘图的下一步是在 B 到 C 之间画一个圆，使用 `circle through` 选项很容易做到。可以使用分割运算来延伸 DA 和 DB 线段，只是这里的分割比值超出 $[0,1]$ 的范围：



```
\usetikzlibrary {calc,through}
\begin{tikzpicture}
\coordinate[label=left:$A$](A)at(0,0);
\coordinate[label=right:$B$](B)at(0.75,0.25);
\coordinate[label=above:$C$](C)at(1,1.5);
\draw(A)--(B)--(C);
\coordinate[label=above:$D$](D)at
($(A)!.5!(B)!.5!(B)!.5!(B)$){};
\node(H)[label=135:$H$,draw,circle through=(C)]at(B){};
\draw(D)--(D)!.35!(B)coordinate[label=below:$F$](F);
\draw(D)--(D)!.25!(A)coordinate[label=below:$E$](E);
\end{tikzpicture}
```

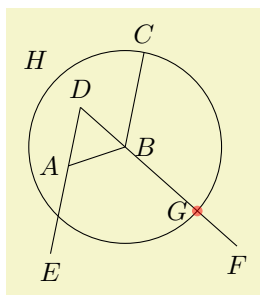
现在，我们面临寻找点 G 的问题，该点是线 BF 和圆 H 的交点。一种方法是使用分割运算的另一种变体：通常，分割运算的形式为 $\langle p \rangle ! \langle \text{因子} \rangle ! \langle q \rangle$ ，得出点 $(1 - \langle \text{因子} \rangle) \langle p \rangle + \langle \text{因子} \rangle \langle q \rangle$ 。另外，除了使用 $\langle \text{因子} \rangle$ 外，您还可以在两点之间使用 $\langle \text{距离} \rangle$ 。在这种情况下，您得到的点是从 $\langle p \rangle$ 到 $\langle q \rangle$ 的直线上距 $\langle p \rangle$ 的距离为 $\langle \text{距离} \rangle$ 的点。

我们知道点 G 在点 B 到点 F 的中间。距离由圆的半径 H 给出。这是计算 H 的代码：



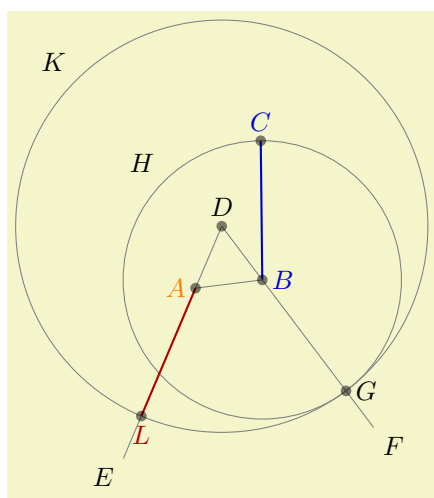
```
\usetikzlibrary {calc,through}
\node(H)[label=135:$H$,draw,circle through=(C)]at(B){};
\pathlet\p1=($(B)-(C)$)in
coordinate[label=left:$G$](G)at($(B)!\text{veclen}(\x1,\y1)!(F)$);
\fill[red,opacity=.5](G)circle(2pt);
```

但是，有一种更简单的方法：我们可以简单地对圆和所讨论的线段的路径进行命名，然后使用 `name intersections` 计算交点。



```
\usetikzlibrary {calc,intersections,through}
\node(H)[name path=H,label=135:$H$,draw,circle through=(C)] at (B){};
\path[name path=B--F](B)--(F);
\path[name intersections={of=HandB--F,by={[label=left:$G$]G}}];
\fill[red,opacity=.5](G)circle(2pt);
```

4.2.3 完整代码



```

\usetikzlibrary {calc,intersections,through}
\begin{tikzpicture}[thick,help lines/.style={thin,draw=black!50}]
\def\A{\textcolor{orange}{\$A\$}}\def\B{\textcolor{input}{\$B\$}}
\def\C{\textcolor{input}{\$C\$}}\def\D{\$D\$}
\def\E{\$E\$}\def\F{\$F\$}
\def\G{\$G\$}\def\H{\$H\$}
\def\K{\$K\$}\def\L{\textcolor{output}{\$L\$}}

\colorlet{input}{blue!80!black}\colorlet{output}{red!70!black}

\coordinate[label=left:\A](A)at(0,0)+.1*(rand,rand);
\coordinate[label=right:\B](B)at(1,0.2)+.1*(rand,rand);
\coordinate[label=above:\C](C)at(1,2)+.1*(rand,rand);

\draw[input](B)--(C);
\draw[help lines](A)--(B);

\coordinate[label=above:\D](D)at(0.5!B)!(sin(60)*2)!(90:(B));

\draw[help lines](D)--(D)!(3.75!(A))coordinate[label=-135:\E](E);
\draw[help lines](D)--(D)!(3.75!(B))coordinate[label=-45:\F](F);

\node(H)at(B)[name path=H,help lines,circle through=(C),draw,label=135:\H];
\path[name path=B--F](B)--(F);
\path[name intersections={of=HandB--F,by={label=right:\G}}];

\node(K)at(D)[name path=K,help lines,circle through=(G),draw,label=135:\K];
\path[name path=A--E](A)--(E);
\path[name intersections={of=KandA--E,by={label=below:\L}}];

\draw[output](A)--(L);

\foreach\pointin{A,B,C,D,G,L}
\fill[black,opacity=.5](\point)circle(2pt);

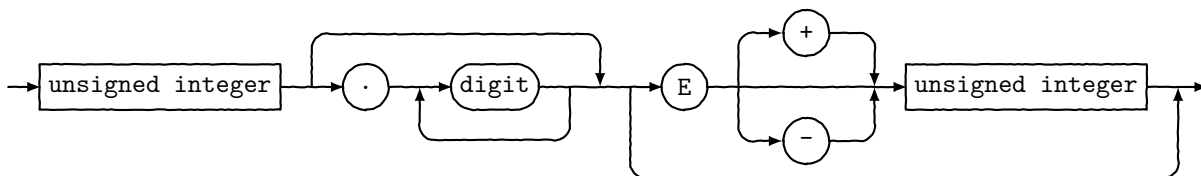
%\node...
\end{tikzpicture}

```

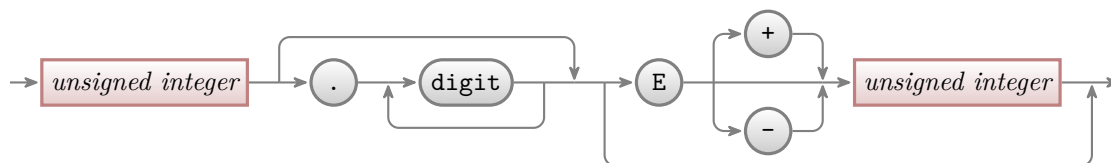
5 教程：简单的图表

在本教程中，我们将了解如何使用图形和矩阵来排版图表。

伊卡尔（Ilka）刚刚获得她那古老而可爱的编程语言的终身教授职位，最近在她的大学图书馆满是灰尘的地窖里挖出了一份题为“编程语言帕斯卡”的技术报告。是在过去的美好时光中使用钢笔和直尺创建的，它看起来像这样²：



在下次演讲中，伊尔卡决定重做这张图，但这一次也许更干净，也可能更“酷”。



阅读了前面的教程后，伊尔卡已经知道如何为她的图表配置环境，即使用 `tikzpicture` 环境。她想知道她将需要哪些库。她决定推迟这个决定，并在创建图片时根据需要添加必要的库。

5.1 为节点设置样式

本教程的大部分内容都是关于如何排列节点并使用链进行连接，让我们开始首先为节点设置样式。

图中有两种节点，即理论家喜欢称之为终端和非终端的节点。对于终端节点，伊尔卡决定使用黑色，这从视觉上表明“无需进行任何操作”。非终端节点则（仍需要进一步“处理”）混入了一些红色。

伊尔卡从更简单的非终端节点开始，因为不涉及圆角。自然地，她创建了一种风格：

unsigned integer

```
\usetikzlibrary {positioning}
\begin{tikzpicture}[
  nonterminal/.style={
    %Theshape:
    rectangle,
    %Thesize:
    minimum size=6mm,
    %Theborder:
    very thick,
    draw=red!50!black!50,%50%redand50%black,
    %andthatmixedwith50%white
    %Thefilling:
    top color=white,%ashadingthatiswhiteatthetop...
    bottom color=red!50!black!20,%andsomethingelseatthebottom
    %Font
    font=\itshape
  }
]
\node[nonterminal]{unsignedinteger};
\end{tikzpicture}
```

伊尔卡对 `minimum size` 选项的使用感到非常自豪。顾名思义，此选项可确保节点至少为 6 毫米 × 6 毫米，但会根据需要扩展其大小以容纳更长的文本。通过为所有节点提供此选项，它们都将具有相同的 6 毫米高度。

²所示的图表不是扫描出来的，而是使用 `TikZ` 进行排版。抖动线是使用 `randomsteps` 创建的。

由于圆角的缘故，对终端节点进行样式设置比较困难。伊尔卡有几种选择可以实现它们。一种方法是使用 `rounded corners` 选项。它以尺寸作为参数，并以指定尺寸作为半径的小圆弧替换所有角。通过将半径设置为 3 毫米，如果节点形状正好为 6 毫米 × 6 毫米时，她将得到一个完全满足她的需要的圆形，否则则节点的侧边为一半圆：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}[node distance=5mm,
terminal/.style={
%Theshape:
rectangle,minimum size=6mm,rounded corners=3mm,
%Therest
very thick,draw=black!50,
top color=white,bottom color=black!20,
font=\ttfamily}]
\node(dot)[terminal]{.};
\node(digit)[terminal,right=of dot]{digit};
\node(E)[terminal,right=of digit]{E};
\end{tikzpicture}
```

另一种可能性是使用专门为排版矩形而在侧面带有弧线的形状（她必须使用 `shapes.misc` 库来创建这样的图形）。这种形状使伊尔卡可以更好地控制外观。例如，她可能只在左侧有一个弧，但她不需要这样做。



```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm,
terminal/.style={
%Theshape:
rounded rectangle,
minimum size=6mm,
%Therest
very thick,draw=black!50,
top color=white,bottom color=black!20,
font=\ttfamily}]
\node(dot)[terminal]{.};
\node(digit)[terminal,right=of dot]{digit};
\node(E)[terminal,right=of digit]{E};
\end{tikzpicture}
```

在这一点上，她注意到一个问题。节点中文本的基线未对齐：



```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm]
\node(dot)[terminal]{.};
\node(digit)[terminal,right=of dot]{digit};
\node(E)[terminal,right=of digit]{E};

\draw[help lines]let\p1=(dot.base),
\p2=(digit.base),
\p3=(E.base)
in(-.5,\y1)--(3.5,\y1)
(-.5,\y2)--(3.5,\y2)
(-.5,\y3)--(3.5,\y3);
\end{tikzpicture}
```

（伊尔卡通过使用 `\tikzset{terminal/.style=...}` 将样式定义移到了序言中，以便她可以在所有图片中使用它。）

对于 `digit` 和 `E` 与基线之间的差异几乎是不可察觉的，但是对于点而言，问题非常严重：它看起来更像是一个乘法点而不是一个句点。

伊尔卡不是很认真地考虑使用 `base right=of...` 选项而不是 `right=of...`，以基线都在同一条线上的方式对齐节点（`base right` 选项将节点放置在某个对象的右侧，以使基线位于另一个对象的基线的右侧）。

但是，并没有达到预期的效果：



```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm]
\node(dot)[terminal]{.};
\node(digit)[terminal,base right=of dot]{digit};
\node(E)[terminal,base right=of digit]{E};
\end{tikzpicture}
```

节点突然在“跳舞”！使用锚点不可能改变节点内文本的位置。相反，伊尔卡必须使用一个技巧：基线不匹配的问题是由 `.`、`digit` 和 `E` 分别具有不同的高度和深度。如果它们的尺寸都是一样，它们将以相同的方式垂直放置。因此，卡尔所需要做的是使用 `text height` 和 `text depth` 选项显式地为节点指定高度和深度。



```
\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm,
text height=1.5ex,text depth=.25ex]
\node(dot)[terminal]{.};
\node(digit)[terminal,right=of dot]{digit};
\node(E)[terminal,right=of digit]{E};
\end{tikzpicture}
```

5.2 使用定位选项对齐节点

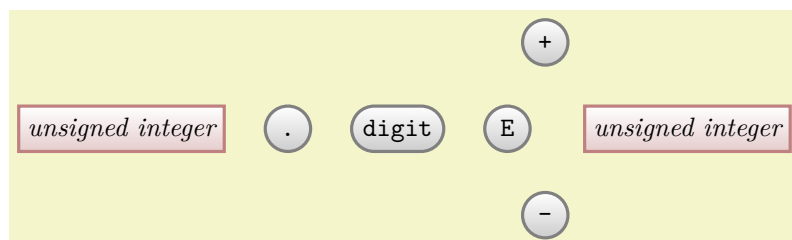
伊尔卡现在已准备好节点的“样式”。下一个问题是将它们放置在正确的位置。有几种方法可以做到这一点。最简单的方法是简单地将节点明确地放置在“手工计算”的特定坐标处。对于非常简单的图形，这是完全可以的，但是它有几个缺点：

1. 对于更复杂的图形，计算可能会变得非常复杂。
2. 更改节点的文本可能需要重新计算坐标。
3. 图形的源代码不是很清楚，因为节点位置之间的关系没有明确。

基于这些原因，伊尔卡决定尝试以其他方式在页面上排列节点。

第一种方法是使用定位选项。要使用它们，您需要加载 `position` 库。这使您可以访问 `above` 或 `left` 之类的选项的高级实现，因为您现在可以使用 `above=of some node` 将一个节点放置在 `some node` 上方，其边界由 `node distance` 分隔开来。

伊尔卡可以使用这个来绘制一个长排节点：



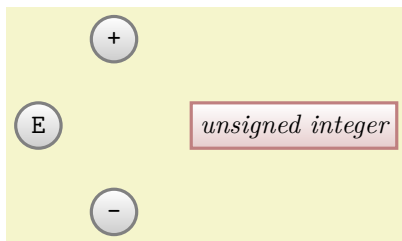

```

\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm]
\node(ui1)[nonterminal]{unsigned integer};
\node(dot)[terminal,right=of ui1]{.};
\node(digit)[terminal,right=of dot]{digit};
\node(E)[terminal,right=of digit]{E};
\node(plus)[terminal,above right=of E]{+};
\node(minus)[terminal,below right=of E]{-};
\node(ui2)[nonterminal,below right=of plus]{unsigned integer};
\end{tikzpicture}

```

对于加号和减号节点，伊尔卡对其放置位置感到有些惊讶。他们不应该更右边吗？以这种方式放置它们的原因如下：E 节点的 `north east` 锚点位于“右圆弧的上起点”，在这种情况下，不幸的是，它恰好位于节点的顶部。同样，+ 节点的 `south west` 锚点实际上在其底部，实际上，E 节点顶部与 + 节点底部之间的水平和垂直距离均为 5 毫米。

有几种解决此问题的方法。最简单的方法是简单地手动添加一点点水平移位：

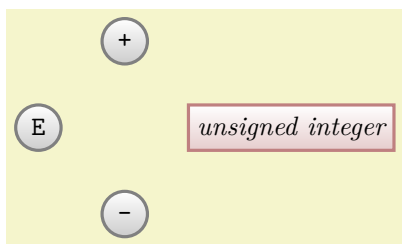


```

\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm]
\node(E)[terminal]{E};
\node(plus)[terminal,above right=of E,xshift=5mm]{+};
\node(minus)[terminal,below right=of E,xshift=5mm]{-};
\node(ui2)[nonterminal,below right=of plus,xshift=5mm]{unsigned integer};
\end{tikzpicture}

```

第二种方法是回到使用标准矩形作为终端节点的想法，但是带有圆角。由于角的倒角不会影响锚点，她得到如下结果：



```

\usetikzlibrary {positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm,terminal/.append style={rectangle,rounded corners=3mm}]
\node(E)[terminal]{E};
\node(plus)[terminal,above right=of E]{+};
\node(minus)[terminal,below right=of E]{-};
\node(ui2)[nonterminal,below right=of plus]{unsigned integer};
\end{tikzpicture}

```

第三种方法是使用矩阵，稍后我们将做。

现在已经放置了节点，伊尔卡需要添加连接。在这里，有些连接比其他连接更困难。例如，考虑 `digit` 周围的“循环”线。描述这条线的一种方式，它从 `digit` 右边开始，然后往下，然后向左，最后在 `digit` 左边一点结束。伊尔卡可以将其放入代码如下：



```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm]
\node(dot) [terminal] {.};
\node(digit) [terminal, right=of dot] {digit};
\node(E) [terminal, right=of digit] {E};

\path(dot) edge[->] (digit) %simple edges
(digit) edge[->] (E);

\draw[->]
%start right of digit.east, that is, at the point that is the
%linear combination of digit.east and the vector (2mm, 0pt). We
%use the ($...$) notation for computing linear combinations
($ (digit.east) + (2mm, 0) $)
%Now go down
--++(0, -.5)
%And back to the left of digit.west
-| ($ (digit.west) - (2mm, 0) $);
\end{tikzpicture}
```

由于 Ilka 需要这样的“向上/向下，然后水平，然后向上/向下到目标”几次，似乎有必要为此定义一个特殊的 *to-path*。每当使用 `edge` 命令时，它只是将 `to path` 的当前值添加到路径中。因此，伊尔卡可以这样使用包含正确路径的样式：



```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm,
skip loop/.style={to path={--++(0, -.5)-|(\tikztotarget)}}]
\node(dot) [terminal] {.};
\node(digit) [terminal, right=of dot] {digit};
\node(E) [terminal, right=of digit] {E};

\path(dot) edge[->] (digit) %simple edges
(digit) edge[->] (E)
($ (digit.east) + (2mm, 0) $)
edge[->, skip loop] ($ (digit.west) - (2mm, 0) $);
\end{tikzpicture}
```

伊尔卡甚至可以更进一步，将她的 `skip loop` 样式参数化。为此，将 `skip loop` 的垂直偏移量作为参数 #1 传递。另外，在下面的代码中，伊尔卡以不同的方式指定了起点和目标，即“在节点中间”的位置。



```
\usetikzlibrary {calc,positioning,shapes.misc}
\begin{tikzpicture}[node distance=5mm and 5mm,
skip loop/.style={to path={--++(0, #1)-|(\tikztotarget)}}]
\node(dot) [terminal] {.};
\node(digit) [terminal, right=of dot] {digit};
\node(E) [terminal, right=of digit] {E};

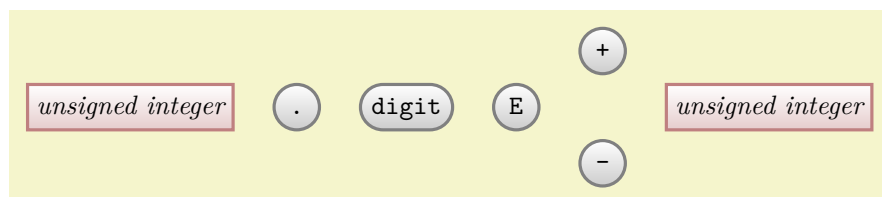
\path(dot) edge[->] (digit) %simple edges
(digit) edge[->] (E)
($ (digit.east)!.5!(E.west) $)
edge[->, skip loop=-5mm] ($ (digit.west)!.5!(dot.east) $);
\end{tikzpicture}
```

5.3 使用矩阵对齐节点

伊尔卡仍然对正负节点的放置感到困扰。不知为什么，必须添加一个显式的 `xshift` 似乎似乎是一种耍小聪明。

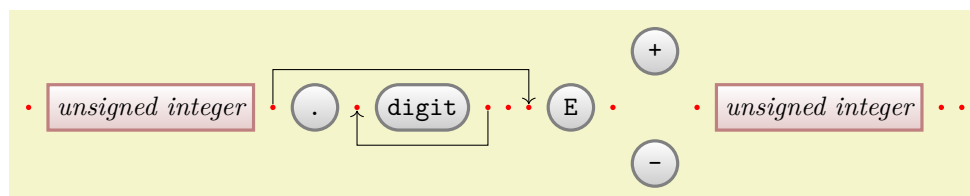
或许更好的定位节点的方法是使用矩阵。在 TikZ 中，矩阵可用于对齐行和列中的任意图形对象。其语法非常类似于 \TeX 中使用数组和表（实际上，在内部使用了 \TeX 表，但还有很多其他内容）。

在伊尔卡的图形中，矩阵将有三行：一行仅包含加号节点，一行包含主节点，另一行仅包含减号节点。



```
\usetikzlibrary {shapes.misc}
\begin{tikzpicture}
\matrix[row sep=1mm,column sep=5mm]{
%Firstrow:
&&&&\node[terminal]{+};&\\
%Secondrow:
\node[nonterminal]{unsignedinteger};&
\node[terminal]{.};&
\node[terminal]{digit};&
\node[terminal]{E};&
&
\node[nonterminal]{unsignedinteger};\\
%Thirdrow:
&&&&\node[terminal]{-};&\\
};
\end{tikzpicture}
```

这好简单！通过摆弄行和列的分隔，伊尔卡可以实现各种令人愉快的节点安排。伊尔卡现在面临着和以前一样的连接问题。这一次，她有了一个想法：她在所有她希望连接开始和结束的地方添加小节点（它们稍后会变成坐标并且不可见）。



```

\usetikzlibrary {shapes.misc}
\begin{tikzpicture}[point/.style={circle,inner sep=0pt,minimum size=2pt,fill=red},
skip loop/.style={to path={--++(0,#1)-/(\tikztotarget)}}]
\matrix[row sep=1mm,column sep=2mm]{
%Firstrow:
&&&&&&&&&&&&\node{plus}[terminal]{+};\\
%Secondrow:
\node{p1}[point]{};&\node{ui1}[nonterminal]{unsignedinteger};&
\node{p2}[point]{};&\node{dot}[terminal]{.};&
\node{p3}[point]{};&\node{digit}[terminal]{digit};&
\node{p4}[point]{};&\node{p5}[point]{};&
\node{p6}[point]{};&\node{e}[terminal]{E};&
\node{p7}[point]{};&&
\node{p8}[point]{};&\node{ui2}[nonterminal]{unsignedinteger};&
\node{p9}[point]{};&\node{p10}[point]{};&\\
%Thirdrow:
&&&&&&&&&&&&\node{minus}[terminal]{-};\\
};

\path{p4}edge[->,skip loop=-5mm]{p3}
{p2}edge[->,skip loop=5mm]{p6};
\end{tikzpicture}

```

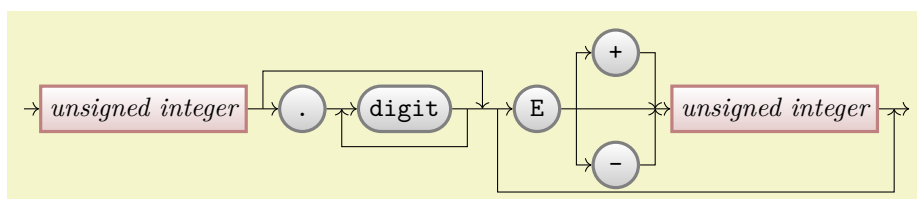
现在，添加所有缺失的线段只是一小步。

5.4 作为图形的图解

矩阵允许伊尔卡很好地对齐节点，但是连接不是很完美。问题在于该代码并没有真正反映该图的基础路径。为此，伊尔卡似乎很自然地使用 `graphs` 库通过边连接节点，这正是图中发生的情况。`graphs` 库既可以用于连接已创建的节点，也可以用于“动态”创建节点，并且这些过程也可以混合使用。

5.4.1 连接已经定位的节点

伊尔卡已经有了一种很好的方法来定位她的节点（使用 `matrix`），所以她所需要的只是一种指定边的简单方法。为此，她使用 `\graph` 命令（实际上只是 `\path graph` 的简写）。该命令允许她以一种简单的方式写下节点之间的线段的代码（宏 `\matrixcontent` 正好包含了前面示例中的矩阵内容；这里不需要重复了）：



```

\usetikzlibrary {graphs,shapes.misc}
\begin{tikzpicture}[skip loop/.style={to path={--++(0,#1)-/(\tikztotarget)}}},
hv path/.style={to path={-/(\tikztotarget)}}},
vh path/.style={to path={/-/(\tikztotarget)}}}]
\matrix[row sep=1mm,column sep=2mm]{\matrixcontent};

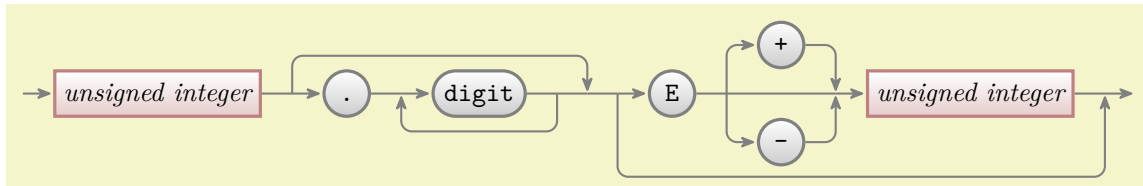
\graph{
(p1)->(ui1)--(p2)->(dot)--(p3)->(digit)--(p4)
--(p5)--(p6)->(e)--(p7)--(p8)->(ui2)--(p9)->(p10);
(p4)->[skip loop=-5mm](p3);
(p2)->[skip loop=5mm](p5);
(p6)->[skip loop=-11mm](p9);
(p7)->[vh path](plus)->[hv path](p8);
(p7)->[vh path](minus)->[hv path](p8);
};
\end{tikzpicture}

```

这已经很接近预期的结果了，只需要几次“修饰”就可以更好地对线段进行样式化。

然而，伊尔卡并不认为示例中的 `graph` 命令有那么的优美。这当然减少了她所需要写的字符的数量，但是总体的图结构并不是那么清晰——它仍然主要是图形的一些路径列表。把它具体化会很好，比如将 (p7) 路径分割为 (+) 和 (-) 然后再合并到 (p8)。而且，所有的这些括号都很难键入。

事实证明，从一个节点到整个节点组的边非常容易指定，如下例所示。另外，通过使用 `use existing nodes` 选项，伊尔卡还可以省略所有括号（此外，一些选项已移到外面，以使示例更短）：



```

\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\begin{tikzpicture}[>={Stealth[round]},thick,black!50,text=black,
every new ->/.style={shorten >=1pt},
graphs/every graph/.style={edges=roundedcorners}]
\matrix[column sep=4mm]{\matrixcontent};

\graph[use existing nodes]{
p1->ui1--p2->dot--p3->digit--p4--p5--p6->e--p7--p8->ui2--p9->p10;
p4->[skip loop=-5mm]p3;
p2->[skip loop=5mm]p5;
p6->[skip loop=-11mm]p9;
p7->[vh path]{plus,minus}->[hv path]p8;
};
\end{tikzpicture}

```

5.4.2 使用 Graph 命令创建节点

伊尔卡听说 `graph` 命令还可以使创建节点变得更容易，不仅仅是连接它们。确实，这是正确的：如果没有使用 `use existing nodes` 选项，并且节点名没有被括号括起来，那么 TikZ 将实际创建一个节点，其名称和文本为节点名：

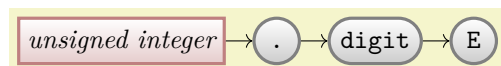
```
unsigned integer → d → digit → E
```

```

\usetikzlibrary {graphs}
\tikz\graph[grow right=2cm]{unsignedinteger->d->digit->E};

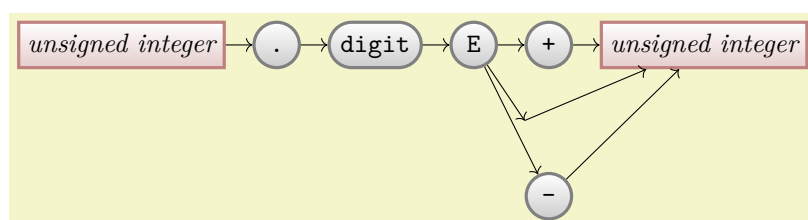
```

这并不完美，但我们正在取得一些进展。首先，通过使用 `grow right sep`，我们可以改变定位算法，，这将导致新节点以一定的固定间隔放置在前一个节点的右侧（默认为 `1em`）。其次，我们添加了一些选项使节点看起来更漂亮。第三，注意上面有趣的 `d` 节点：伊尔卡尝试使用 `.`，但有一些错误消息。原因是在 `TikZ` 中节点不能被称为 `.`，所以她必须选择一个不同的名字——这是不好的，因为她想要一个点显示！诀窍是把点放在引号中，这允许你使用“相当随意的文本”作为节点名：



```
\usetikzlibrary {graphs,shapes.misc}
\tikz\graph[grow right sep]{
unsignedinteger[nonterminal]->"."[terminal]->digit[terminal]->E[terminal]
};
```

现在轮到正号和负号了。在这里，伊尔卡可以使用 `graph` 命令的分组机制来创建节点的分裂：

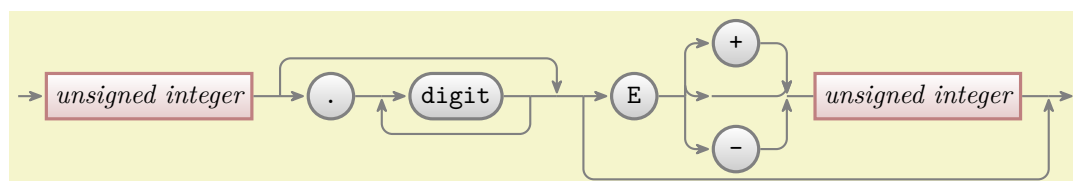


```
\usetikzlibrary {graphs,shapes.misc}
\tikz\graph[grow right sep]{
unsignedinteger[nonterminal]->
"."[terminal]->
digit[terminal]->
E[terminal]->
{
"+"[terminal],
""[coordinate],
"-"[terminal]
}->
ui2/unsignedinteger[nonterminal]
};
```

让我们看看，这里发生了什么。我们想要两个 `unsigned integer` 节点，但是如果我们只是使用这个文本两次，那么 `TikZ` 已经注意到在当前图使用了相同的名称，这看起来很智能的样子（实际上在这种情况下并不太智能），就会创建一条回到已经创建的节点的边。因此，这里需要一个新的名称。但是伊尔卡也不能就写 `unsigned integer2`，毕竟她想要显示原始文本！诀窍是在节点名中使用斜杠：为了“呈现”节点，使用斜杠后面的文本，而不是节点名，节点名是斜杠前面的文本。另外，也可以使用 `as` 选项，它也允许您指定应该如何呈现节点。

事实证明，伊尔卡不需要发明 `ui2` 之类的名称。对于她来说，她不会再次引用这个节点。在这种情况下，她可以省略节点名（在/之前不写任何内容），该名称始终代表“新的，匿名的”节点名称。

接下来，伊尔卡需要在反循环包含的一些节点上增加一些坐标，她需要对节点进行一些移动：



```

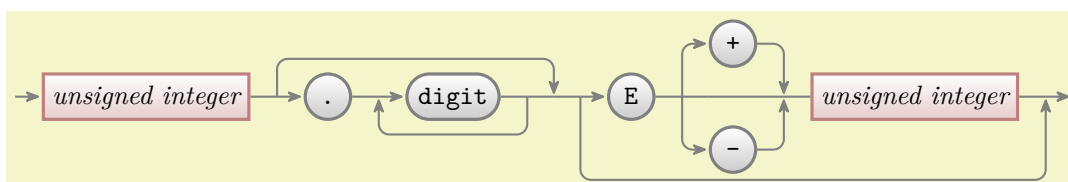
\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\begin{tikzpicture}[>={Stealth[round]},thick,black!50,text=black,
every new ->/.style={shorten >=1pt},
graphs/every graph/.style={edges=roundedcorners}]
\graph[grow right sep,branch down=7mm]{
/[coordinate]->
unsignedinteger[nonterminal]--
p1[coordinate]->
"."[terminal]--
p2[coordinate]->
digit[terminal]--
p3[coordinate]--
p4[coordinate]--
p5[coordinate]->
E[terminal]--
q1[coordinate]->[vh path]
{[nodes={yshift=7mm}]
"+"[terminal],
q2/[coordinate],
"-"[terminal]
}->[hv path]
q3[coordinate]--
/unsignedinteger[nonterminal]--
p6[coordinate]->
/[coordinate];

p1->[skip loop=5mm]p4;
p3->[skip loop=-5mm]p2;
p5->[skip loop=-11mm]p6;
};
\end{tikzpicture}

```

剩下要做的就是以某种方式摆脱 E 和 unsigned integer 之间的奇怪曲线。它们是由于 TikZ 试图创建一个先垂直后水平前进的边引起的，但实际上只是水平前进的边。另外，实际上箭头不应指向边；但是似乎很难摆脱这一点，因为来自 q1 的其他边，即 plus 和 minus，应该被指向。

事实证明，有一个解决此问题的好方法：您可以指定图是 **simple** 的。这意味着任何两个节点之间最多可以有一条边。现在，如果您指定两次边，则第二个的选项为“赢得胜利”。因此，通过多添加两条线来“校正”这些边，我们得到了带有完整代码的最终图：



```

\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\tikz[>={Stealth[round]},black!50,text=black,thick,
every new ->/.style      ={shorten >=1pt},
graphs/every graph/.style ={edges=roundedcorners},
skip loop/.style         ={to path={--+(0,#1)-(\tikztotarget)}}},
hv path/.style           ={to path={-/(\tikztotarget)}}},
vh path/.style           ={to path={/-(\tikztotarget)}}},
nonterminal/.style       ={
rectangle,minimum size=6mm,very thick,draw=red!50!black!50,top color=white,
bottom color=red!50!black!20,font=\itshape,text height=1.5ex,text depth=.25ex},
terminal/.style          ={
rounded rectangle,minimum size=6mm,very thick,draw=black!50,top color=white,
bottom color=black!20,font=\ttfamily,text height=1.5ex,text depth=.25ex},
shape                    =coordinate
]
\graph[grow right sep,branch down=7mm,simple]{
/->unsignedinteger[nonterminal]--p1->" "[terminal]--p2->digit[terminal]--
p3--p4--p5->E[terminal]--q1->[vh path]
{[nodes={yshift=7mm}]
"+"[terminal],q2,"-"[terminal]
}->[hv path]
q3--/unsignedinteger[nonterminal]--p6->/;

p1->[skip loop=5mm]p4;
p3->[skip loop=-5mm]p2;
p5->[skip loop=-11mm]p6;

q1--q2--q3;%maketheseedgesplain
};

```


6 教程：约翰内斯的演讲图

在本教程中，我们探索 TikZ 的树状图和思维导图机制。

约翰内斯（Johannes）非常兴奋：在即将到来的一个学期中，他将第一次独自一人教授一门课程！不幸的是，这门课程不是他最喜欢的科目，当然是理论免疫学，但是基于复杂性理论，但是作为一个年轻的学者，约翰内斯不太可能大声抱怨。为了帮助学生大致了解整个课程中将要发生的事情，他打算画一些包含基本概念的树或图。他从他的老教授那里得到了这个想法，他的老教授似乎成功地使用了这些“讲义图”。约翰内斯认为，撇开这些图的成功不谈，它们看起来相当整洁。

6.1 问题描述

约翰内斯希望创建一个具有以下功能的讲义图：

1. 它应该是包含描述主要概念的树或图。
2. 该地图还应包含一个日历，该日历显示何时举行各个讲座。
3. 出于美学原因，整张图应具有视觉美观且信息丰富的背景。

与往常一样，约翰内斯将必须引入正确的库并配置环境。约翰内斯将使用 `mindmap` 库，并且由于他希望显示日历，因此他还需要 `calendar` 库。为了在背景层上放置一些东西，最好还包含 `background` 库。

6.2 树状图简介

约翰内斯必须做出的第一选择是他将这些概念组织成一棵树，其带有根的概念，分支的概念和叶子的概念，亦或是组织成一般的图形。树状图隐式地组织了概念，而一般的图形则更加灵活。约翰内斯决定做出让步：基本上，这些概念将被组织成一棵树。然而，他会选择性地在相关概念之间添加联系，但这些概念出现在树的不同层次或分支上。

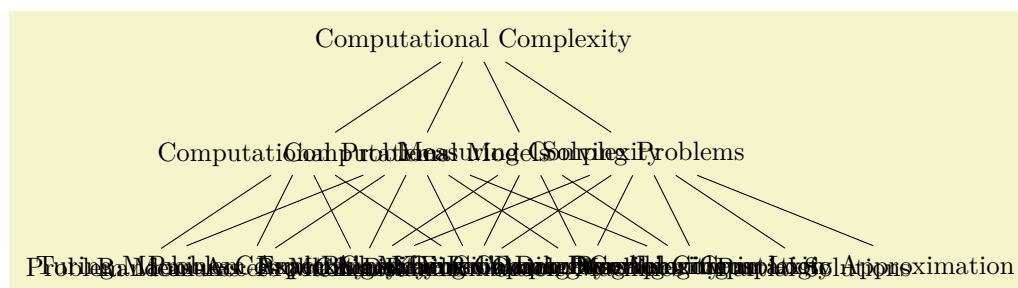
约翰内斯首先在树上列出了一些概念，他认为这些概念对于计算复杂性很重要：

- Computational Problems
 - Problem Measures
 - Problem Aspects
 - Problem Domains
 - Key Problems
- Computational Models
 - Turing Machines
 - Random-Access Machines
 - Circuits
 - Binary Decision Diagrams
 - Oracle Machines
 - Programming in Logic
- Measuring Complexity
 - Complexity Measures
 - Classifying Complexity
 - Comparing Complexity
 - Describing Complexity

- Solving Problems
 - Exact Algorithms
 - Randomization
 - Fixed-Parameter Algorithms
 - Parallel Computation
 - Partial Solutions
 - Approximation

约翰内斯肯定以后需要修改此列表，但作为初步近似看起来不错。他还需要添加一些子主题（例如“分类复杂度”主题下的大量复杂度类别），但是他在创建图形时才会这样做。

原则上，将主题列表转换为 TikZ 树很容易。基本思想是节点可以具有子节点，而后者又可以具有自己的子节点，依此类推。要将子节点添加到节点，约翰内斯只需在一个节点之后添加 `child {⟨节点⟩}`。而 `⟨节点⟩` 应该是用于创建一个节点的代码。要添加另一个节点，约翰内斯可以再一次使用 `child`，依此类推。约翰内斯渴望尝试这种构造并写下以下内容：



```
\tikz
\node{ComputationalComplexity}%root
child{node{ComputationalProblems}}
child{node{ProblemMeasures}}
child{node{ProblemAspects}}
child{node{ProblemDomains}}
child{node{KeyProblems}}
}

child{node{ComputationalModels}}
child{node{TuringMachines}}
child{node{Random-AccessMachines}}
child{node{Circuits}}
child{node{BinaryDecisionDiagrams}}
child{node{OracleMachines}}
child{node{ProgramminginLogic}}
}

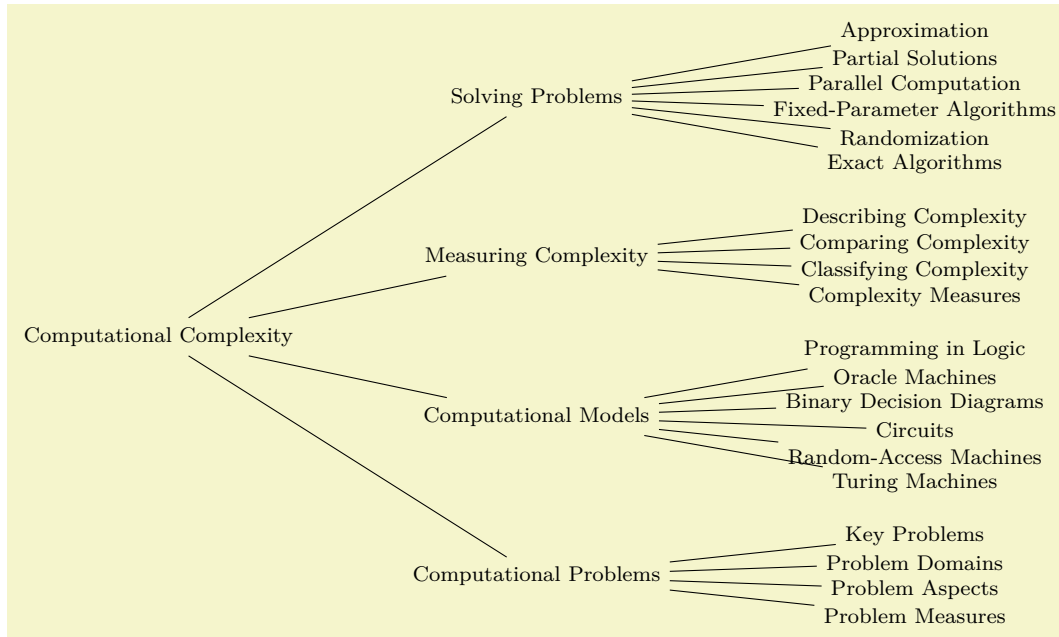
child{node{MeasuringComplexity}}
child{node{ComplexityMeasures}}
child{node{ClassifyingComplexity}}
child{node{ComparingComplexity}}
child{node{DescribingComplexity}}
}

child{node{SolvingProblems}}
child{node{ExactAlgorithms}}
child{node{Randomization}}
child{node{Fixed-ParameterAlgorithms}}
child{node{ParallelComputation}}
child{node{PartialSolutions}}
child{node{Approximation}}
};
```

好吧，这并没有完全按预期进行（尽管，人们究竟期望什么呢？）。有两个问题：

1. 节点的重叠是由于 TikZ 在放置子节点时并不是特别聪明。尽管可以将 TikZ 配置为使用更聪明的放置方法，但是 TikZ 无法将子节点的实际大小考虑在内。这可能看起来很奇怪，但原因是子节点一次被渲染并放置一个，因此在处理第一个节点时最后一个节点的大小是未知的。本质上，您必须“手动”指定适当的级别和同级节点间距。
2. 树的标准计算机科学自上而下的渲染方式非常不适合可视化概念。最好将地图旋转 90 度，甚至最好使用某种圆形排列。

约翰尼斯重新绘制了树，他或多或少地通过反复试验发现并使用了一些更合适的选项集：



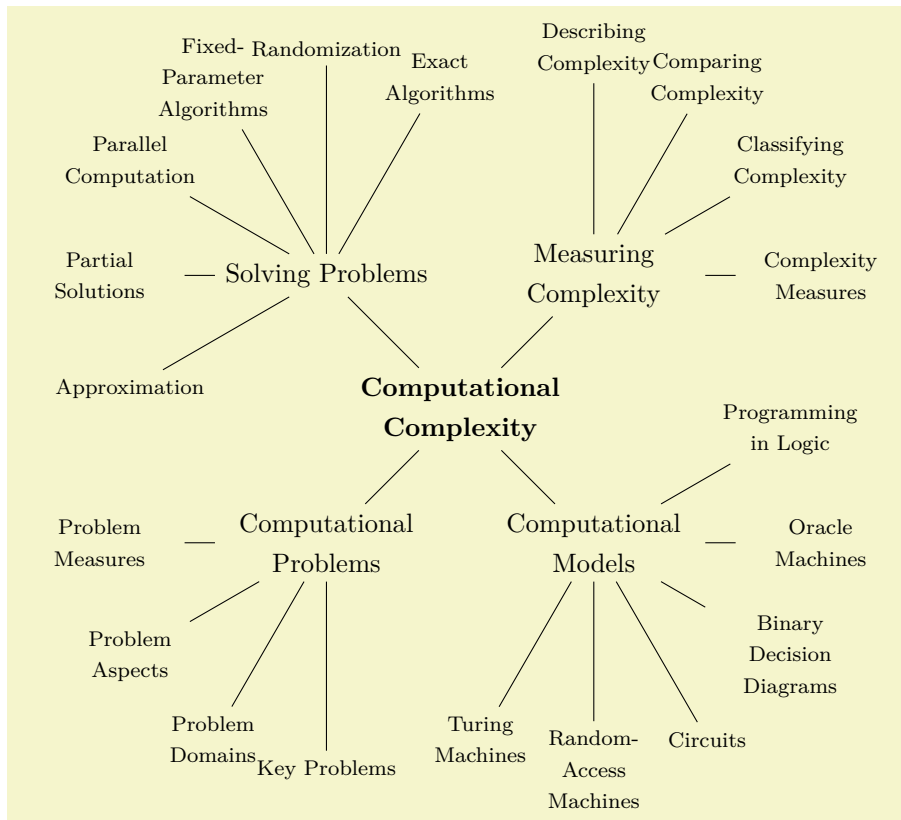
```
\usetikzlibrary {trees}
\tikz[font=\footnotesize,
grow=right,level 1/.style={sibling distance=6em},
level 2/.style={sibling distance=1em},level distance=5cm]
\node{ComputationalComplexity}%root
child{node{ComputationalProblems}}
child{node{ProblemMeasures}}
child{node{ProblemAspects}}
...%asbefore
```

约翰内斯的想法仍然不尽如人意，但他正在进步。

对于配置树而言，有两个参数特别重要：**level distance** 告诉 TikZ 树的相邻级别或上层的节点（中心）之间的距离。**sibling distance**，顾名思义，是树的同级节点（中心）之间的距离。

您可以通过在树开始绘制之前在某个位置来全局地设置树的这些参数，但是您通常希望它们对于树的不同级别是不同的。在这种情况下，您应该设置像 **level 1** 或 **level 2** 这样的样式。对于树的第一级，使用 **level 1** 样式，对于第二级，使用 **level 2** 的样式，以此类推。您还可以通过将这些选项作为选项传递给 **child** 命令，从而仅为某些节点设置同级距离和级别距离。（注意，**node** 命令的选项是该节点的本地选项，对子节点没有影响。还要注意，可以指定对子节点有影响的选项。最后要注意的是，“在正确的地方”为子节点指定选项是一种神秘的艺术，如果你真的感兴趣的话，你应该在一个下雨的星期天下午阅读??节。）

grow 键用于配置树的生长方向。您只需为单个子节点或整个级别更改此键，即可更改“在树的中央”的生长方向。通过包含 **trees** 库，您还可以使用其他增长策略，例如“圆形”增长：



```
\usetikzlibrary {trees}
\tikz[text width=2.7cm,align=flushcenter,
grow cyclic,
level 1/.style={level distance=2.5cm,sibling angle=90},
level 2/.style={text width=2cm,font=\footnotesize,level distance=3cm,sibling angle=30}]
\node[font=\bfseries]{ComputationalComplexity}{%root
child{node{ComputationalProblems}
child{node{ProblemMeasures}}
child{node{ProblemAspects}}
...%asbefore
```

约翰内斯很高兴地了解到他可以访问和操作树的节点，就像任何正常的节点一样。特别是，他可以使用 `name=` 选项或 (`<名字>`) 表示法来为它们命名，并且他可以为树节点使用任何可用的形状或样式。稍后，他可以使用普通的 `\draw` (某个节点) -- (另一个节点)；语法连接树。实际上，`child` 命令只是计算节点的适当位置，并在子节点和父节点之间添加一条线。

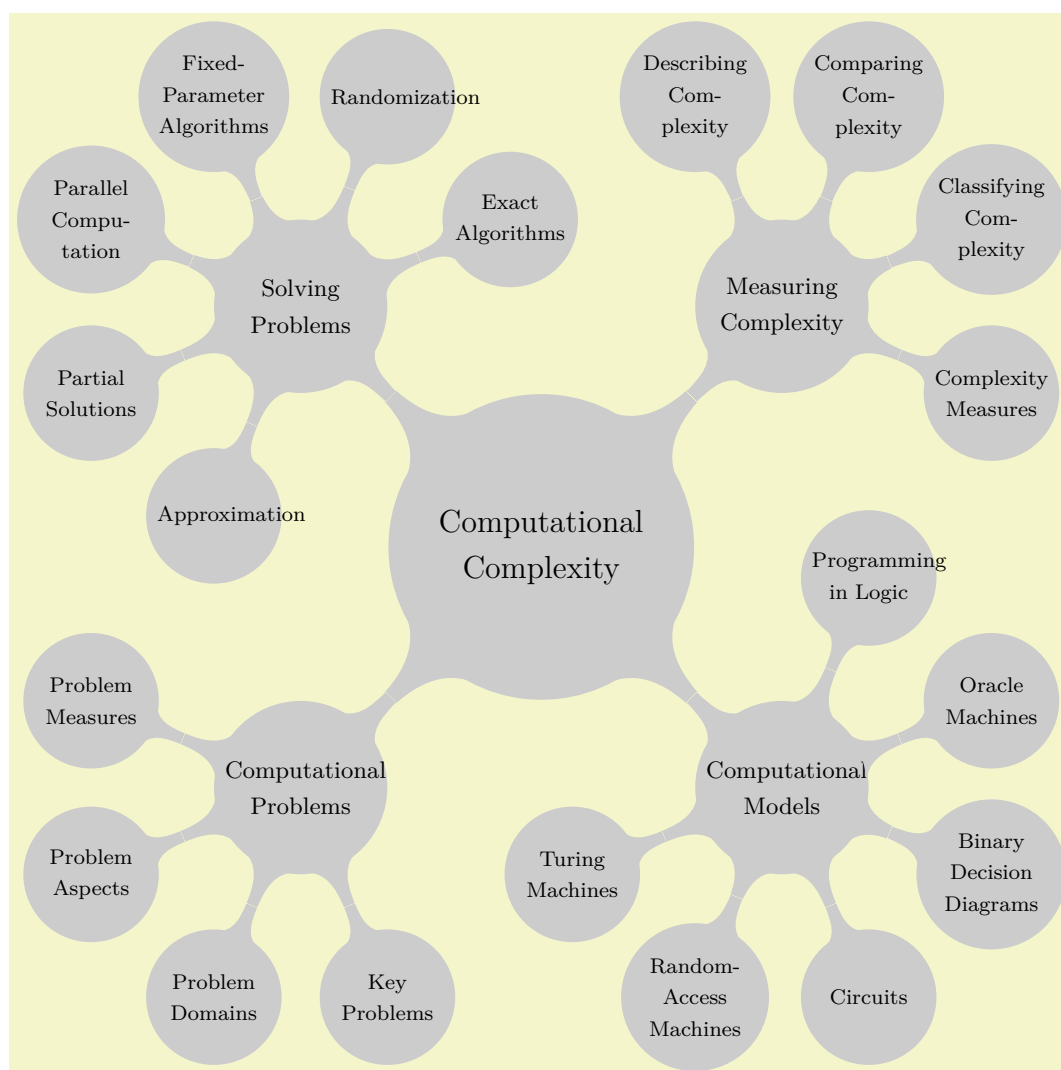
6.3 创建讲义图

约翰内斯现在有他的讲义图的第一个可能的布局。下一步是使其“看起来更好”。为此，`mindmap` 库很有帮助，因为它提供了许多样式，这些样式将使树看起来像一个漂亮的“思维导图”或“概念图”。

第一步是加入 `mindmap` 库，约翰内斯已经这么做了。接下来，他必须将下列选项之一添加到包含讲义图代码范围中：`mindmap` 或 `large mindmap` 或 `huge mindmap`。这些选项都有相同的效果，除了 `large mindmap` 预定义的字体大小和节点大小比标准的 `mindmap` 大一些，`huge mindmap` 甚至更大一些。所以，一个 `large mindmap` 不一定需要有很多概念，但它需要很多纸。

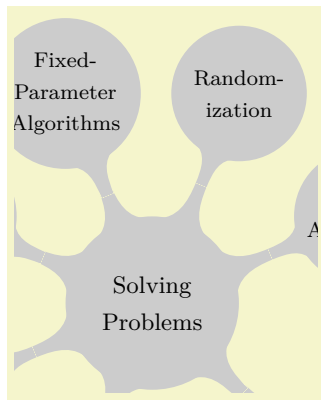
第二步是将 `concept` 选项添加到每一个节点，这些节点实际上是思维导图的一个概念。其思想是树的一些节点将是真实的概念，而其他节点可能只是“简单的子节点”。通常情况下，情况并非如此，因此您可以考虑使用 `every node/.style=concept`。

第三步是设置同级角度（而不是同级距离）以指定同级概念之间的角度。



```
\usetikzlibrary {mindmap}
\tikz[mindmap,every node/.style=concept,concept color=black!20,
grow cyclic,
level 1/.append style={level distance=4.5cm,sibling angle=90},
level 2/.append style={level distance=3cm,sibling angle=45}]
\node[root concept]{ComputationalComplexity}%root
child{node{ComputationalProblems}}
child{node{ProblemMeasures}}
child{node{ProblemAspects}}
...%asbefore
```

当约翰内斯排版上面的地图时， $\text{T}_{\text{E}}\text{X}$ （正当地）开始抱怨盒子太多了，而且确实，像“Randomization”这样的词超出了概念的范围。乍一看，这似乎有点神秘：为什么 $\text{T}_{\text{E}}\text{X}$ 不使用连字符？原因是 $\text{T}_{\text{E}}\text{X}$ 永远不会在段落的第一个字用连字符连接，因为它只在所谓的胶之后才开始寻找“可连字符”的字母。为了把这些单词连字符化，约翰内斯必须耍点小花招：在单词前插入 `\hskip0pt`。并且约翰内斯只能在单词前面插入一个（不可见的）胶从而允许 $\text{T}_{\text{E}}\text{X}$ 将第一个单词用连字符连接。但是，由于约翰内斯不想在每个节点中添加 `\hskip0pt`，他可以使用 `execute at begin node` 选项使 $\text{T}_{\text{E}}\text{X}$ 在每个节点中插入该文本。



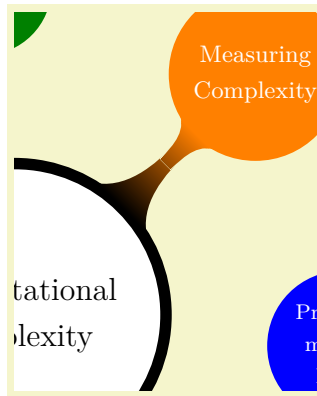
```
\usetikzlibrary {mindmap}
\begin{tikzpicture}
[mindmap,
every node/.style={concept,execute at begin node=\hskip0pt},
concept color=black!20,
grow cyclic,
level 1/.append style={level distance=4.5cm,sibling angle=90},
level 2/.append style={level distance=3cm,sibling angle=45}]
\clip(-1,2)rectangle++(-4,5);
\node[root concept]{ComputationalComplexity}%root
child{node{ComputationalProblems}}
child{node{ProblemMeasures}}
child{node{ProblemAspects}}
...%asbefore
\end{tikzpicture}
```

在上面的示例中，为了节省空间，使用裁剪仅显示了讲义图的一部分。在以下示例中将执行相同的操作，我们将在本教程的末尾回到完整的讲义图。

约翰内斯现在渴望为图形着色。想法是对图形的不同部分使用不同的颜色。然后，他可以在讲座中讨论“绿色”或“红色”主题。这将使他的学生更容易在图形上找到他正在谈论的主题。由于“computational problems”听起来有点像“problematic”，因此约翰内斯为他们选择了红色，而他为“solving problems”选择了绿色。“measuring complexity”和“computational models”主题使用中性色；约翰内斯选择橙色和蓝色。

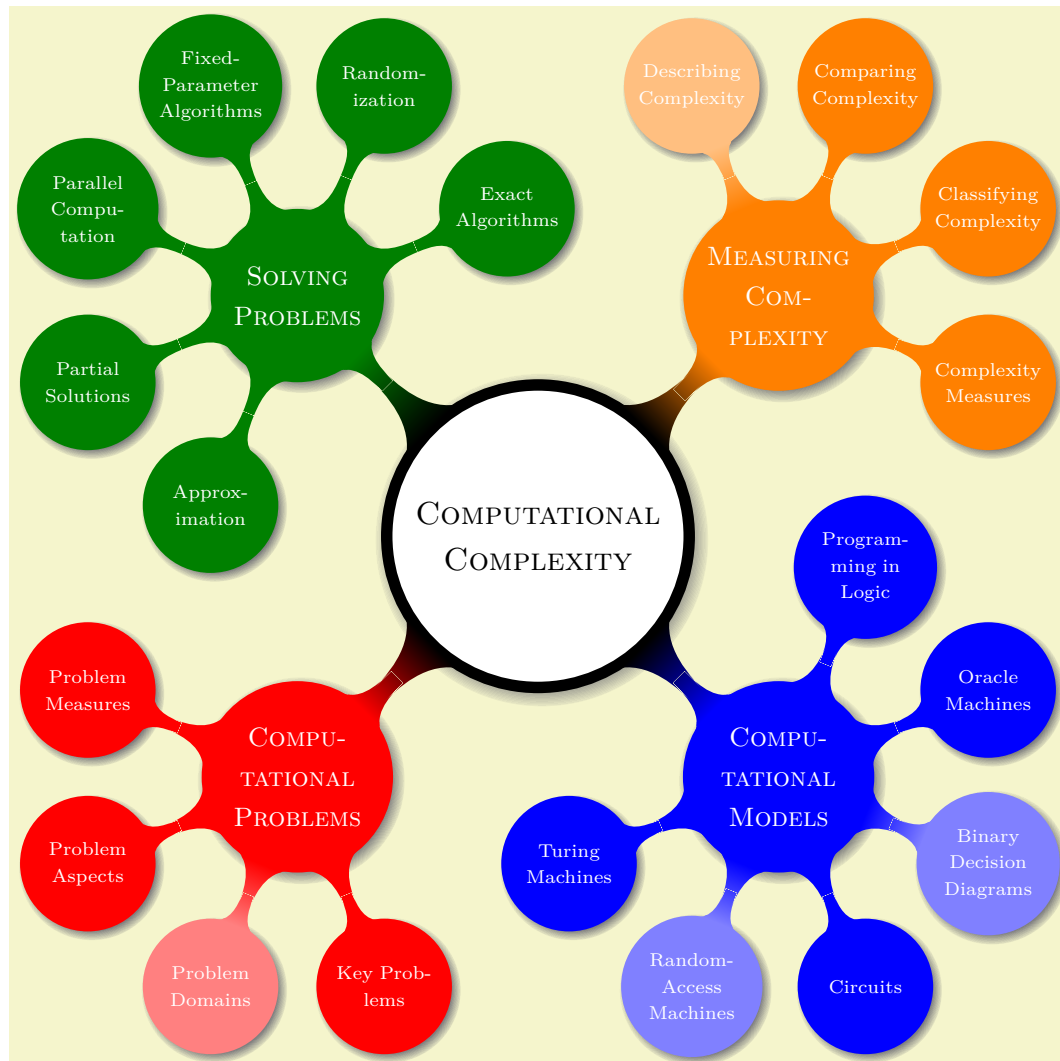
为了设置颜色，约翰内斯必须使用 `concept color` 选项，而不仅仅是 `node [fill=red]`。仅将填充颜色设置为 `red` 确实会使节点为红色，但它将仅使节点为红色，而不是连接概念到父节点和子节点的颜色带。相比之下，特殊的 `concept color` 选项不仅会设置节点及其子节点的颜色，而且还会（神奇地）创建适当的阴影，以便父概念的颜色平稳地变化为子概念的颜色。

对于根概念，约翰内斯决定做一些特殊的事情：他将概念颜色设置为黑色，将线宽设置为较大的值，并将填充颜色设置为白色。这样的效果是，根概念将被一条粗黑线包围，并且子节点通过颜色带与中心概念相连。



```
\usetikzlibrary {mindmap}
\begin{tikzpicture}
[mindmap,
every node/.style={concept,execute at begin node=\hskip0pt},
root concept/.append style={
concept color=black,fill=white,line width=1ex,text=black},
text=white,
grow cyclic,
level 1/.append style={level distance=4.5cm,sibling angle=90},
level 2/.append style={level distance=3cm,sibling angle=45}]
\clip(0,-1)rectangle++(4,5);
\node[root concept]{ComputationalComplexity}%root
child[concept color=red]{node{ComputationalProblems}}
child{node{ProblemMeasures}}
...%asbefore
}
child[concept color=blue]{node{ComputationalModels}}
child{node{TuringMachines}}
...%asbefore
}
child[concept color=orange]{node{MeasuringComplexity}}
child{node{ComplexityMeasures}}
...%asbefore
}
child[concept color=green!50!black]{node{SolvingProblems}}
child{node{ExactAlgorithms}}
...%asbefore
};
\end{tikzpicture}
```

约翰内斯增加了三处收尾工作：首先，他将主要概念的字体更改为小写。其次，他决定应该对某些概念进行“淡化”，即那些原则上很重要并且属于图的概念，但在讲座中将不予讨论。为了实现这一点，约翰内斯定义了四种样式，四个主要分支中的每个分支都使用一种。这些样式（a）为整个分支设置正确的概念颜色，（b）定义了 `faded` 样式以适用于此分支。第三，他添加了 `circular drop shadow`，在 `shadows` 库中定义，从概念上讲，只是为了使图形看起来更漂亮。




```

\usetikzlibrary {mindmap,shadows}
\begin{tikzpicture}[mindmap]
\begin{scope}[
every node/.style={concept,circular drop shadow,execute at begin node=\hskip0pt},
root concept/.append style={
concept color=black,fill=white,line width=1ex,text=black,font=\large\scshape},
text=white,
computational problems/.style={concept color=red,faded/.style={concept color=red!50}},
computational models/.style={concept color=blue,faded/.style={concept color=blue!50}},
measuring complexity/.style={concept color=orange,faded/.style={concept color=orange!50}},
solving problems/.style={concept color=green!50!black,faded/.style={concept color=green!50!black!50}},
grow cyclic,
level 1/.append style={level distance=4.5cm,sibling angle=90,font=\scshape},
level 2/.append style={level distance=3cm,sibling angle=45,font=\scriptsize}]
\node[root concept]{ComputationalComplexity}%root
child[computational problems]{node{ComputationalProblems}}
child{node{ProblemMeasures}}
child{node{ProblemAspects}}
child[faded]{node{ProblemDomains}}
child{node{KeyProblems}}
}
child[computational models]{node{ComputationalModels}}
child{node{TuringMachines}}
child[faded]{node{Random-AccessMachines}}
...
\end{scope}
\end{tikzpicture}

```

6.4 添加讲义注释

约翰内斯将在“computational complexity”课程中进行大约十二次讲座。对于每一次讲座，他都编制了一个（简短的）学习目标清单，陈述了学生在该特定讲座中应获得的知识 and 条件（请注意，学习目标与讲座的内容不同）。对于每次讲座，他打算在图上的每个主题附近的某个位置上在地图上放置一个小矩形，其中包含这些学习目标和讲座名称。mindmap 将这种“小矩形”称为“注释”。图书馆。

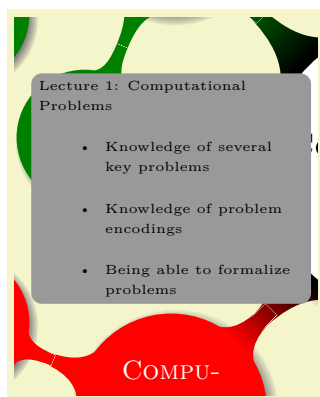
为了将注释放置在概念旁边，约翰内斯必须为概念的节点分配名称。他可以使用 TikZ 对树中节点的自动命名，其中树节点名为 root。节点的子节点被命名为 root-1、root-2、root-3，依此类推。但是，由于约翰内斯不确定树中概念的最终顺序，因此最好以以下方式明确命名树的所有概念：

```

\node[root concept](ComputationalComplexity){ComputationalComplexity}
child[computational problems]{node(ComputationalProblems){ComputationalProblems}}
child{node(ProblemMeasures){ProblemMeasures}}
child{node(ProblemAspects){ProblemAspects}}
child[faded]{node(ProblemDomains){ProblemDomains}}
child{node(KeyProblems){KeyProblems}}
}
...

```

mindmap 库的 annotation 样式主要设置一个大小合适的矩形。约翰内斯通过恰当地定义 every annotation 来设置样式。

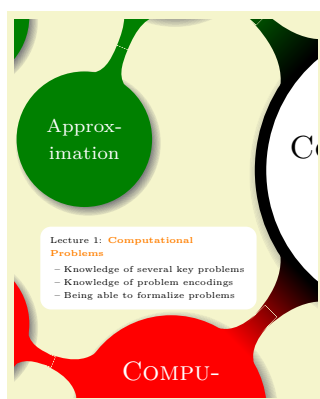


```
\usetikzlibrary {mindmap,shadows}
\begin{tikzpicture}[mindmap]
\clip(-5,-5)rectangle++(4,5);
\begin{scope}[
every node/.style={concept,circular drop shadow,...}]%asbefore
\node[root concept](ComputationalComplexity)...%asbefore
\end{scope}

\begin{scope}[every annotation/.style={fill=black!40}]
\node[annotation,above]at(ComputationalProblems.north){
Lecture1:ComputationalProblems
\begin{itemize}
\itemKnowledgeofseveralkeyproblems
\itemKnowledgeofproblemencodings
\itemBeingabletoformalizeproblems
\end{itemize}
};
\end{scope}
\end{tikzpicture}
```

这看起来还不是很完美。间距或 `{itemize}` 实际上并不合适，而且节点太大。约翰内斯可以“手工”配置这些东西，但是定义一个宏来为他处理这些东西似乎是个好主意。“正确”的方法是定义一个 `\lecture` 宏，它接受键值对列表作为参数，并生成所需的注释。然而，为简单起见，约翰内斯的 `\lecture` 宏只需接受具有以下含义的固定数量的参数：第一个参数是讲座的数量，第二个是讲座的名称，第三个是像 `above` 的定位选项，第四是节点放置的位置，第五个的显示列表项，第六个是举行讲座的日期（尚不需要此参数，但是稍后我们将需要它）。

```
\def\lecture#1#2#3#4#5#6{
\node[annotation,#3,scale=0.65,text width=4cm,inner sep=2mm]at(#4){
Lecture#1:\textcolor{orange}{\textbf{#2}}
\list{--}{\topsep=2pt\itemsep=0pt\parsep=0pt
\parskip=0pt\labelwidth=8pt\leftmargin=8pt
\itemindent=0pt\labelsep=2pt}
#5
\endlist
};
}
```



```
\usetikzlibrary {mindmap,shadows}
\begin{tikzpicture}[mindmap,every annotation/.style={fill=white}]
\clip(-5,-5)rectangle++(4,5);
\begin{scope}[
every node/.style={concept,circular drop shadow,... % as before
\node[root concept] (Computational Complexity) ... % as before
\end{scope}

\lecture{1}{Computational Problems}{above,xshift=-3mm}
{ComputationalProblems.north}{
\itemKnowledgeofseveralkeyproblems
\itemKnowledgeofproblemencodings
\itemBeingabletoformalizeproblems
}{2009-04-08}
\end{tikzpicture}
```

约翰内斯现在可以以相同的方式添加其他讲义注解。显然，约翰内斯在将所有内容都放在一个 A4 尺寸的页面上会遇到一些麻烦，但是通过调整间距和进行一些实验，他可以根据需要快速排列所有注释。

6.5 添加背景

约翰尼斯已经使用颜色将他的讲座图组织成四个区域，每个区域都有不同的颜色。为了更加强调这些区域，他希望为每个区域添加背景色。

事实证明，添加这些背景颜色比约翰尼斯想象的要复杂得多。乍一看，他需要的是某种“色轮”，它在右下方向为蓝色，然后在右上方向平滑地变为橙色，然后在左上方向平滑为绿色，依此类推。不幸的是，没有简单的方法可以创建这样的色轮阴影（尽管原则上可以做到，但是代价是非常高的，作为示例请参见第??页）。

约翰尼斯决定做一些更基础的事情：他创建了四个大矩形，围绕中心概念的四个象限中的每个象限中各有一个，每个象素都以浅色的象限着色。然后，为了“平滑”邻近矩形之间的变化，他在四块区域上加入了底纹。

由于这些背景矩形应位于其他所有内容的“后面”，因此约翰尼斯将其所有背景材料放在 `background` 层上。

在以下代码中，仅显示中心概念以节省一些空间：



6.6 添加日历

约翰内斯打算相当仔细地计划他的讲座。特别是，他已经知道在这门课程中何时举行每一次讲座。自然，这并不意味着约翰内斯会刻苦地遵循该计划，并且某些科目可能比他预期的需要更长的时间，但是尽管如此，他还是有一个详细的计划来计划何时解决该主题。

约翰内斯打算通过在课堂地图上添加日历的方式与他的学生分享这个计划。日历除了可以作为在特定日期举行特定主题的讲座的参考之外，还有助于显示课程的总体时间顺序。

为了将日历添加到 TikZ 图形，`calendar` 库是最有用的。该库提供 `\calendar` 命令，它具有大量选项，并且可以通过多种方式进行配置，以生成几乎可以想象的任何类型的日历。出于约翰内斯的目的，一个简单的 `day list downward` 将是一个不错的选择，因为它会产生一个向下的日期列表。

```
1 \usetikzlibrary {calendar}
2
3 \tiny
4 \begin{tikzpicture}
5
6 \calendar[day list downward,
7
8 name=cal,
9
10 dates=2009-04-01to2009-04-14]
11
12 if(weekend)
13 [black!25];
14
15 \end{tikzpicture}
```

我们可以使用 `name` 选项为日历命名，这将使我们稍后可以引用组成日历的每一天的节点。例如，包含 1 的矩形节点代表 2009 年 4 月 1 日，可以称为 `(cal-2009-04-01)`。`dates` 选项用于指定应该绘制日历的间隔。约翰内斯的日历中将需要几个月的时间，但是上面的示例仅显示了两周的时间来节省一些空间。

注意 `if(weekend)` 构造。`\calendar` 命令后跟选项，然后是 `if` 语句。在日历的每一天都会检查这些 `if` 语句，并且当日期通过此测试时，将执行 `if` 语句之后的选项或代码。在上面的示例中，我们使周末（准确地说是星期六和星期日）比平常更淡。（使用您喜欢的日历检查一下，2009 年 4 月 5 日确实是星期天。）

如上所述，约翰内斯可以引用用于排版日期的节点。回想一下他的 `\lecture` 宏已经接受了一个日期，我们还没有使用过。现在，我们可以使用它将讲座的标题放在讲座举行的日期旁边：

```
\def\lecture#1#2#3#4#5#6{
%Asbefore:
\node[annotation,#3,scale=0.65,text width=4cm,inner sep=2mm]at(#4){
Lecture#1:\textcolor{orange}{\textbf{#2}}
\list{--}{\topsep=2pt\itemsep=0pt\parsep=0pt
\parskip=0pt\labelwidth=8pt\leftmargin=8pt
\itemindent=0pt\labelsep=2pt}
#5
\endlist
};
%New:
\node[anchor=basewest]at(cal-#6.baseeast){\textcolor{orange}{\textbf{#2}}};
}
```

约翰内斯现在可以使用如下这个新的 `\lecture` 命令（在示例中，仅新的定义部分）：

1	<code>\usetikzlibrary {calendar}</code>
2	<code>\tiny</code>
3	<code>\begin{tikzpicture}</code>
4	<code>\calendar[day list downward,</code>
5	<code>name=cal,</code>
6	<code>dates=2009-04-01to2009-04-14]</code>
7	<code>if(weekend)</code>
8	<code>[black!25];</code>
9	<code>%Asbefore:</code>
10	<code>\lecture{1}{ComputationalProblems}{above,xshift=-3mm}</code>
11	<code>{ComputationalProblems.north}{</code>
12	<code>\itemKnowledgeofseveralkeyproblems</code>
13	<code>\itemKnowledgeofproblemencodings</code>
14	<code>\itemBeingabletoformalizeproblems</code>
	<code>}{2009-04-08}</code>
	<code>\end{tikzpicture}</code>

最后，约翰内斯需要向 `\calendar` 命令添加更多选项：他使用 `month text`。选项配置月份文本的呈现方式（有关详细信息，请参见??节），然后在每个月初的特殊位置对月份文本进行排版。

April 2009	<code>\usetikzlibrary {calendar}</code>
1	<code>\tiny</code>
2	<code>\begin{tikzpicture}</code>
3	<code>\calendar[day list downward,</code>
4	<code>month text=\%mt\\%y0,</code>
5	<code>month yshift=3.5em,</code>
6	<code>name=cal,</code>
7	<code>dates=2009-04-01to2009-05-01]</code>
8	<code>if(weekend)</code>
9	<code>[black!25]</code>
10	<code>if(dayofmonth=1){</code>
11	<code>\nodeat(0pt,1.5em)[anchor=basewest]{\small\tikzmonthtext};</code>
12	<code>};</code>
13	<code>\lecture{1}{ComputationalProblems}{above,xshift=-3mm}</code>
14	<code>{ComputationalProblems.north}{</code>
15	<code>\itemKnowledgeofseveralkeyproblems</code>
16	<code>\itemKnowledgeofproblemencodings</code>
17	<code>\itemBeingabletoformalizeproblems</code>
18	<code>}{2009-04-08}</code>
19	<code>\lecture{2}{ComputationalModels}{above,xshift=-3mm}</code>
20	<code>{ComputationalModels.north}{</code>
21	<code>\itemKnowledgeofTuringmachines</code>
22	<code>\itemBeingabletocomparethecomputationalpowerofdifferent</code>
23	<code>models</code>
24	<code>}{2009-04-15}</code>
25	<code>\end{tikzpicture}</code>
26	
27	
28	
29	
30	
May 2009	
1	

6.7 完整代码

放在一起，约翰内斯得到以下代码：

首先是 `\lecture` 命令的定义：

```

\def\lecture#1#2#3#4#5#6{
%Asbefore:
\node[annotation,#3,scale=0.65,text width=4cm,inner sep=2mm,fill=white] at (#4){
Lecture#1:\textcolor{orange}{\textbf{#2}}
\list{--}{\topsep=2pt\itemsep=0pt\parsep=0pt
\parskip=0pt\labelwidth=8pt\leftmargin=8pt
\itemindent=0pt\labelsep=2pt}
#5
\endlist
};
%New:
\node[anchor=basewest] at (cal-#6.baseeast){\textcolor{orange}{\textbf{#2}}};
}

```

接下来是主要的思维导图设置...

```

\noindent
\begin{tikzpicture}
\begin{scope}[
mindmap,
every node/.style={concept,circular drop shadow,execute at begin node=\hskip0pt},
root concept/.append style={
concept color=black,
fill=white,line width=1ex,
text=black,font=\large\scshape},
text=white,
computational problems/.style={concept color=red,faded/.style={concept color=red!50}},
computational models/.style={concept color=blue,faded/.style={concept color=blue!50}},
measuring complexity/.style={concept color=orange,faded/.style={concept color=orange!50}},
solving problems/.style={concept color=green!50!black,faded/.style={concept color=green!50!black!50}},
grow cyclic,
level 1/.append style={level distance=4.5cm,sibling angle=90,font=\scshape},
level 2/.append style={level distance=3cm,sibling angle=45,font=\scriptsize}]

```

...以及思维导图内容:

```

\documentclass[12pt]{article}
\usepackage{tikz}
\begin{tikzpicture}
\node[root concept] (ComputationalComplexity) {ComputationalComplexity} %root
child[computational problems] {node[yshift=-1cm] (ComputationalProblems) {ComputationalProblems}}
child{node(ProblemMeasures) {ProblemMeasures}}
child{node(ProblemAspects) {ProblemAspects}}
child[faded] {node(problemDomains) {ProblemDomains}}
child{node(KeyProblems) {KeyProblems}}
}
child[computational models] {node[yshift=-1cm] (ComputationalModels) {ComputationalModels}}
child{node(TuringMachines) {TuringMachines}}
child[faded] {node(Random-AccessMachines) {Random-AccessMachines}}
child{node(Circuits) {Circuits}}
child[faded] {node(BinaryDecisionDiagrams) {BinaryDecisionDiagrams}}
child{node(OracleMachines) {OracleMachines}}
child{node(ProgramminginLogic) {ProgramminginLogic}}
}
child[measuring complexity] {node[yshift=1cm] (MeasuringComplexity) {MeasuringComplexity}}
child{node(ComplexityMeasures) {ComplexityMeasures}}
child{node(ClassifyingComplexity) {ClassifyingComplexity}}
child{node(ComparingComplexity) {ComparingComplexity}}
child[faded] {node(DescribingComplexity) {DescribingComplexity}}
}
child[solving problems] {node[yshift=1cm] (SolvingProblems) {SolvingProblems}}
child{node(ExactAlgorithms) {ExactAlgorithms}}
child{node(Randomization) {Randomization}}
child{node(Fixed-ParameterAlgorithms) {Fixed-ParameterAlgorithms}}
child{node(ParallelComputation) {ParallelComputation}}
child{node(PartialSolutions) {PartialSolutions}}
child{node(Approximation) {Approximation}}
};
\end{scope}

```

现在是日历代码:

```

\documentclass[12pt]{article}
\usepackage{tikz}
\begin{tikzpicture}
\tiny
\calendar[day list downward,
month text=\%mt\ \%y0,
monthlyshift=3.5em,
name=cal,
at={(-.5\textwidth-5mm,.5\textheight-1cm)},
dates=2009-04-01to2009-06-last]
if(weekend)
[black!25]
if(dayofmonth=1){
\nodeat(0pt,1.5em)[anchor=base west]{\small\tikzmonthtext};
};
\end{tikzpicture}

```

讲义注释:

```

\documentclass[12pt]{article}
\usepackage{tikz}
\begin{tikzpicture}
\lecture{1}{ComputationalProblems}{above,xshift=-5mm,yshift=5mm}{ComputationalProblems.north}{
\item Knowledge of several key problems
\item Knowledge of problem encodings
\item Being able to formalize problems
}{2009-04-08}

\lecture{2}{ComputationalModels}{above left}{
ComputationalModels.west}{
\item Knowledge of Turing machines
\item Being able to compare the computational power of different
models
}{2009-04-15}
\end{tikzpicture}

```

最后，背景的代码：

```
\begin{pgfonlayer}{background}
\clip[xshift=-1cm](-.5\textwidth,-.5\textheight)rectangle++(\textwidth,\textheight);

\colorlet{upperleft}{green!50!black!25}
\colorlet{upperright}{orange!25}
\colorlet{lowerleft}{red!25}
\colorlet{lowerright}{blue!25}

%Thelargerectangles:
\fill[upperleft](ComputationalComplexity)rectangle++(-20,20);
\fill[upperright](ComputationalComplexity)rectangle++(20,20);
\fill[lowerleft](ComputationalComplexity)rectangle++(-20,-20);
\fill[lowerright](ComputationalComplexity)rectangle++(20,-20);

%Theshadings:
\shade[left color=upperleft,right color=upperright]
([xshift=-1cm]ComputationalComplexity)rectangle++(2,20);
\shade[left color=lowerleft,right color=lowerright]
([xshift=-1cm]ComputationalComplexity)rectangle++(2,-20);
\shade[top color=upperleft,bottom color=lowerleft]
([yshift=-1cm]ComputationalComplexity)rectangle++(-20,2);
\shade[top color=upperright,bottom color=lowerright]
([yshift=-1cm]ComputationalComplexity)rectangle++(20,2);
\end{pgfonlayer}
\end{tikzpicture}
```

下一页显示了最终得到的讲义图（如果有更多讲义注解，那就更好了，但您应该了解其中的思想）。

- 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

May 2009

- 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

- 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



7 绘图准则

本节与 PGF 或 TikZ 无关，而是关于为科学报告、论文和书籍创建图形的一般指导准则或原则。

本节中的指导准则来自不同的地方。其中很多都是我想说的“常识”，一些反映了我的个人经验（尽管，希望不只是我个人的喜好），一些来自于平面设计和排版方面的书籍（抱歉，书目还没找到）。最有影响力的资料来源是爱德华·塔夫特的杰出著作。虽然我并不同意这些书中所写的一切，但塔夫特的许多论点是如此令人信服，因此我决定在下面的指导准则中重复它们。

当有人提出一系列指导准则时，您应该问自己的第一件事是：我是否真的应该遵循这些指导准则？这是一个重要的问题，因为有充分的理由不遵循这些准则。制定准则的人可能有除您之外的其他目标。例如，一条准则可能会说“用红色表示强调”。虽然此准则对于使用投影仪进行演示非常有意义，但是当使用黑白打印机进行打印时，红色的“颜色”会具有“强调”的相反效果¹。准则几乎总是针对特定情况而制定的。如果你不是在这种情况下，那么遵循这些准则可能会弊大于利。

您应该意识到的第二件事是排版的基本规则：“只要您意识到您正在违反规则，那么每个规则都可能被破坏。”该规则也适用于绘图。基本规则用不同的措词表述：“版式中的唯一错误是未知的东西。”当你意识到一个规则，当你决定打破规则会有更理想的效果，请打破规则。

7.1 规划创建的图形所需要的时间

当你创建一篇有大量图形的文章时，创建这些图形所需的时间就成为一个重要的因素。您应该规划多少时间来创建图形？

通常，假设图形创建所需的时间与相同长度的文本所需的时间相同。例如，当我写论文时，初稿每页大约需要一个小时。稍后，我需要每页两到四个小时进行修订。因此，我预计大约需要半小时才能创建半页图形的初稿。稍后，我希望再等一到两个小时才能完成最终图形。

在许多出版物中，甚至在优质期刊中，作者和编辑者显然都花了大量时间在文本上，但似乎只花了大约五分钟来创建所有图形。图形似乎经常被认为是“事后思考”的东西，或者看起来就像作者的统计软件显示的屏幕快照一样。正如稍后将要讨论的那样，默认情况下，像 GNUPLOT 这样的程序所产生的图形质量很差。

创建有助于读者并与主要内容相适应的信息丰富的图形是一个艰难而漫长的过程。

- 将图形视为你的文章的一等公民。它们理应需要和文本一样多的时间和精力。确实，图形的创建可能比主文的写作需要甚至更多的时间，因为将更多地关注图形，并且读者首先会关注它们。
- 与为相同大小的文本的计划一样，计划尽可能多的时间来创建和修改图形。
- 具有高信息密度的复杂图形可能需要更多时间。
- 非常简单的图形将需要较少的时间，但是无论如何您很可能不想在纸上创建“非常简单的图形”；就像您不想拥有相同大小的“非常简单的文字”一样。

7.2 创建图形的工作流程

撰写（科学）论文时，您很可能会遵循以下模式：您有一些要研究的结果或想法。论文的创建通常从编写一个粗略的轮廓开始。然后，在不同部分填充文本以完成第一稿。然后对该草案进行反复修订，直到经常进行实质性修订后才得出最终文件。在一份好的期刊论文中，通常不会有一个句子从初稿起就没有改变。

创建图形遵循相同的模式：

- 决定图表应该传达什么。让这成为一个有意识的决定，也就是说，决定“这个图形要告诉读者什么？”

¹译者注：打印出来颜色更浅。

- 创建一个“轮廓”，即图形的粗略整体“形状”，其中包含最关键的元素。通常，使用铅笔和纸进行此操作很有用。
- 填写图形的详细信息以创建第一稿。
- 重复修改图形以及本文的其余部分。

7.3 将图形与正文紧密结合

图形可以放置在文本的不同位置。可以内联它们，这意味着它们位于“文本中间”的某个位置，或者可以放置在独立的“图形”环境中。由于打印机（人们）喜欢“填满”他们的页面，因此传统上（出于美学和经济方面的考虑），独立的图形可能会放置在文档中远离引用它们的主要文本的页面上。由于技术原因， $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 和 $\text{T}_{\text{E}}\text{X}$ 倾向于鼓励图形的这种“漂移”。

对于内联图形，从某种程度上来说，图形的标签将由周围的文本隐含地解释，它会或多或少地自动与主要文本链接。同样，正文通常会清楚说明图形的含义和显示的内容。

完全不同的是，一个独立的图形通常会在这个图形所属的主要文本还没有被读过或者已经读过一段时间的时候被看到。因此，在创建独立图形时，应该遵循以下指导原则：

- 独立的图形应该有一个标题，而不是让读者“自己可以理解”。
例如，假设一张图显示了快速排序算法不同阶段的示例。然后该图的标题至少应告知读者“该图显示了 xyz 页上介绍的快速排序算法的不同阶段”，不仅仅是“快速排序算法”。
- 一个好的标题会添加尽可能多的上下文信息。例如，您可以说：“该图显示了 xyz 页上介绍的快速排序算法的不同阶段。在第一行中，选择了主元素 5。这会导致 ...虽然这些信息也可以在正文中给出，但将其放在标题中将确保上下文被保留。不要害怕 5 行的标题。（你的编辑可能会因此而讨厌你，考虑反过来讨厌他们。）
- 请在正文中引用图形，如“有关快速排序的实际示例，请参阅第 xyz 页上的图 2.1”。
- 大多数有关样式和排版的书籍建议您不要使用“Fig. 2.1”的缩写版本，而应写“Figure 2.1”。

反对缩写的主要论点是：“句点很有价值，不能浪费在缩写上”。这个想法是，句点将使读者认为句子在“Fig”之后结束，并且需要进行“有意识的回溯”才能意识到句子毕竟没有结束。

支持缩写的观点是它们可以节省空间。

就个人而言，我并没有真正相信任何一种论点。一方面，我还没有看到任何确凿的证据表明缩写会使读者放慢速度。另一方面，在所有文档中均使用“Figure”的缩写版本“Fig”可能性很小。我尽量避免使用缩写。

7.4 图形和文本之间的一致性

也许人们在创建图形时最常见的“错误”（记住设计中的“错误”始终只是“无知”）是图形外观和文本之间看起来不匹配。

作者使用几个不同的程序来创建一篇论文的图形是很常见的。作者可能使用 `gnuplot` 生成一些图，使用 `xfig` 生成一个图，还有一个由共同作者使用一些未知的程序生成 `.eps` 图形。所有这些图形很可能使用不同的线宽、不同的字体和不同的大小。此外，作者经常使用像 `[height=5cm]` 这样的选项，当使用这些图形时，将它们缩放到“合适的大小”。

如果采用同样的方法来书写正文，那么每个部分都将以不同的字体和不同的大小书写。在某些部分，所有的定理都会加下划线，在另一些部分，它们会全部用大写字母打印，而在另一些部分，它们会用红色。此外，每一页的页边距也不一样。读者和编辑不会容忍以这种方式书写的文本，但他们通常却不得不容忍带有图形的文本。

若要使图形和文本之间保持一致，请遵循以下准则：

- 不要缩放图形。

这意味着在使用外部程序生成图形时，请“以适当的尺寸”创建图形。

- 在图形和正文中使用相同的字体。
- 在文本和图形中使用相同的线宽。

普通文本的“线宽”是像 T 这样的字母的主干的宽度。对于 \TeX ，通常是 0.4 pt。但是，有些期刊不接受正常线宽低于 0.5 pt 的图形。

- 使用颜色时，请在文本和图形中使用一致的颜色编码。例如，如果红色应该提醒读者注意正文中的某些内容，则在图形中也将红色用于图形的重要部分。如果将蓝色用于标题和节标题之类的结构元素，请将蓝色也用于图形的结构元素。

但是，图形也可以使用逻辑上的固有颜色编码。例如，无论您通常使用什么颜色，读者通常都会假设绿色是“积极，正常，没问题”，红色是“警告，警告，行动”。

使用不同的图形程序时创建一致性的图形几乎是不可能的。因此，您应该考虑坚持使用单个图形程序。

7.5 图形中的标签

几乎所有图形都将包含标签，即用于解释图形各部分的文本。放置标签时，请遵循以下准则：

- 放置标签时，请遵循一致性规则。您应该通过两种方式进行操作：首先，与主要文本保持一致，即对标签也使用与主文本相同的字体。其次，标签之间应保持一致，即，如果以某种特定方式格式化某些标签，则应以这种方式格式化所有标签。
- 除了在文本和图形中使用相同的字体外，还应该使用相同的符号。例如，如果您在主文本中写入 $1/2$ ，则还应使用“ $1/2$ ”作为图形中的标签，而不是“0.5”。 π 是“ π ”而不是“3.141”。最后， $e^{-i\pi}$ 是“ $e^{-i\pi}$ ”，而不是“-1”，更不用说“-1”了。
- 标签应清晰易读。它们不仅应具有较大的尺寸，而且也不应被线条或其他文本遮盖。这也适用于行标签和标签后面的文本。
- 标签应放在“适当位置”。只要有足够的空间，标签就应该放在它们所标记的对象旁边。只要在必要的时候，才在标签和它们标记的对象用一条（浅色的）线连接起来。尽量避免只参考外部说明的标签。读者不得不在解释和被描述的对象之间来回跳跃。
- 考虑使用例如灰色来排版“无关紧要”的标签。这将使读者的焦点集中在实际的图形上。

7.6 图表

图表是最常见的一种图形，尤其是在科学论文中。它们种类繁多，包括简单的折线图，参数图，三维图，饼图等。

不幸的是，众所周知，图表很难弄对。部分原因是因为 `gnuplot` 或 Excel 之类的程序的默认设置，因为这些程序使创建劣质图表非常方便。

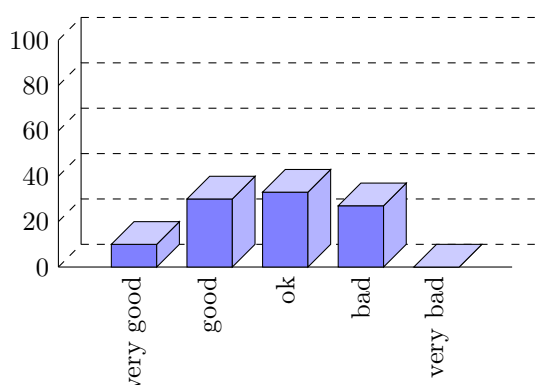
创建图时您应该问自己的第一个问题是：是否有足够的数据点可用于绘制图？如果答案是“否定的”，请使用表格。

不需要绘图的典型情况是人们在条形图中显示一些数字。这是一个真实的示例：在研讨会结束时，一位讲师要求参与者提供反馈。在 50 位参与者中，有 30 位提交了反馈表。根据反馈，三位参与者认为研讨会“非常好”，九位认为是“好”，十位“还好”，八位“一般”，没有人认为研讨会“非常差”。

下表是汇总这些信息的一种简单方法：

给出的评分	给予此评分的参与者 (共 50 个)	百分比
“非常好”	3	6%
“好”	9	18%
“一般”	10	20%
“差”	8	16%
“非常差”	0	0%
未评分	20	40%

讲师所做的就是使用 3D 条形图将数据可视化。它看起来像这样（实际上，数字是使用某些分辨率非常低的位图字体进行排版的，几乎无法读取）：



表和“图”的大小大致相同。如果您首先想到的是“图形看起来比表格更好”，请尝试根据表格或图形中的信息回答以下问题：

1. 总共有多少参与者？
2. 有多少参与者退回了反馈表？
3. 百分之几的参与者返回了反馈表？
4. 有多少参与者的评价“非常好”？
5. 在所有参与者中，有多少百分比表示“非常好”？
6. 是否有超过四分之一的参与者检查“差”或“非常差”？
7. 提交了反馈表的参与者中有百分之多少表示“非常好”？

可悲的是，该图形不允许我们回答这些问题中的任何一个。该表直接回答所有问题，除了最后一个。本质上，图形的信息密度非常接近零。该表具有更高的信息密度；尽管它使用大量空白来表示一些数字。以下是 3D 条形图出现问题的列表：

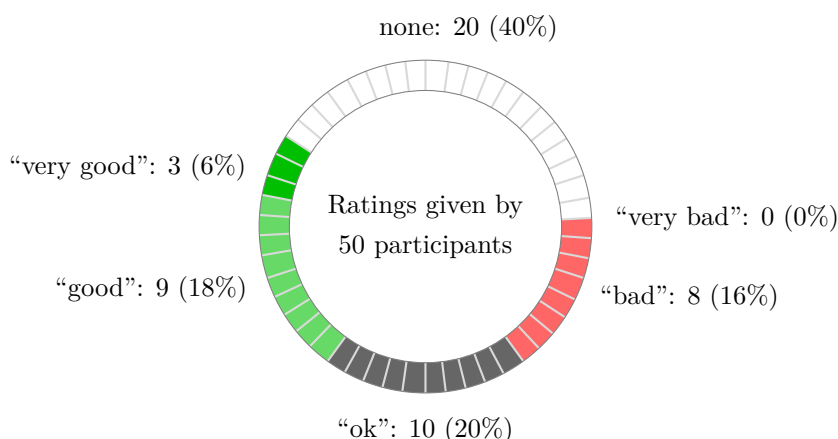
- 整个图形以令人讨厌的背景线为主。
- 目前尚不清楚左边的数字是什么意思。大概是百分比，但也可能是参与者的绝对人数。
- 底部的标签被旋转，使其难以阅读。

（在我看到的真实演示中，文本的分辨率非常低，每个字母大约 10×6 像素，而且字距错误，使得旋转后的文本几乎无法读取。）

- 第三维在不添加信息的情况下增加了图形的复杂性。

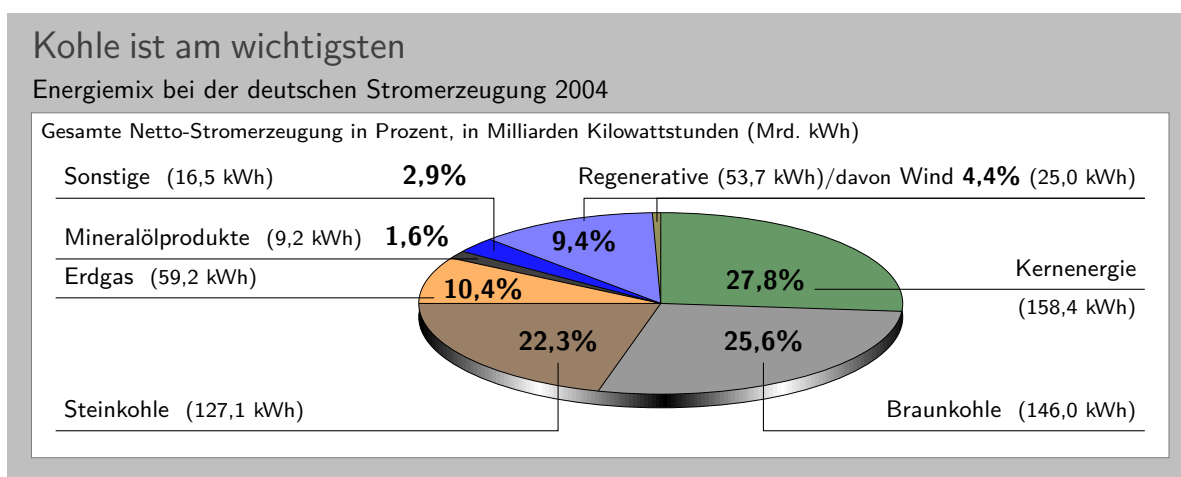
- 三维设置使正确度量柱的高度变得十分困难。考虑一下“差”这一柱。这一条代表的数字是否是大于或等于 20？柱状图的前部在 20 以下，而柱状图的后部确在其上方。
- 我们无法分辨这些柱状图代表的是哪些数字。因此，这些柱不必要地隐藏了这些柱所包含的所有信息。
- 杆高加起来是多少？是 100% 还是 60%？
- “非常差”柱代表 0 还是 1？
- 为什么柱是蓝色的？

你可能会说，在这个例子中，确切的数字对图表并不重要。重要的是“信息”，即“非常好”和“好”的评级要多于“差”和“非常差”的评级。但是，要传达这一信息，可以用句子来表达，也可以用图表来更清楚地表达：



上面的图形与表格具有大约相同的信息密度（显示的大小和数字大致相同）。此外，人们可以直接“看到”好的评级比不良评级更高。人们也可以“看到”，根本没有给出评分的人数是微不足道的，这对于反馈表来说是很普遍的。

图表并非总是一个好主意。让我们看一个我从 2005 年 6 月 4 日时代周刊中的饼图的一个重新绘制的示例：



该图形已用 TikZ 重绘，但原始外观几乎完全相同。

乍一看，该图形看起来“不错且内容丰富”，但是有很多地方出了问题：

- 图表是三维的。但是，阴影没有增加“信息方面”的内容，充其量只能分散注意力。
- 在 3D 饼图中，相对大小会非常严重地变形。例如，灰色的“Braunkohle”占据的面积大于绿色的“Kernenergie”占据的面积，尽管 Braunkohle 的百分比小于 Kernenergie 的百分比。

- 3D 失真在小区域变得更糟。“Regenerative”面积比“Erdgas”面积大。“Wind”的面积略小于“Mineralölprodukte”的面积，（尽管“Wind”的百分比几乎是矿产“Mineralölprodukte”的百分比的三倍。

在最后一种情况下，不同的大小只是部分上归咎于失真。原图的设计者还把“Wind”的切片做得太小，甚至把失真也考虑进去了。（只要比较一下“Wind”和“Regenerative”的一般大小就可以了。）

- 根据其标题，此图表应该告诉我们，煤炭是 2004 年德国最重要的能源。忽略了多余和误导性的 3D 设置所造成的强烈变形，此信息需要花很长时间才能传达出来。

煤炭作为能源分为两部分：一个是“Steinkohle”，另一个是“Braunkohle”（两种不同的煤炭）。将它们加起来后，您会看到饼图的整个下半部分都被煤炭占据了。

不同种类的煤炭的两个区域根本没有视觉上的联系。而是使用两种不同的颜色，标签位于图形的不同侧。相比之下，“Regenerative”和“Wind”连接更为紧密。

- 图形的颜色编码没有遵循任何逻辑模式。为什么核能是绿色的？再生能源是浅蓝色的，“其他能源”是蓝色的。“Braunkohle”（字面意思是“棕色的煤”）的区域是石灰色的，而“Steinkohle”（字面意思是“石煤”）的区域是棕色的，这看起来更像是一个笑话。
- 颜色最亮的区域用于“Erdgas”。该区域最突出的原因是颜色较亮。但是，对于这个图表，“Erdgas”一点都不重要。

爱德华·塔夫特（Edward Tufte）将上述图形称为“垃圾图表”。（不过，我很高兴地宣布时代周刊已经停止使用 3D 饼图，并且其信息图也有所改善。）

以下是一些建议，可以帮助您避免创建垃圾图表：

- 不要使用 3D 饼图。它是邪恶的。
- 考虑使用表格而不是饼图。
- 请勿随机应用颜色；而应用它们来引导读者的注意力并对事物进行分组。
- 请勿使用背景图案（例如交叉线或对角线）代替背景颜色。他们会使读者分心。信息图中的背景图案是邪恶的。

7.7 注意力和注意力分散

挑选您最喜欢的小说，并浏览典型的页面。您会注意到页面非常统一。在阅读时，没有什么可以分散读者的注意力。没有大的标题，没有粗体的文字，没有大的白色区域。确实，即使作者确实希望强调某些内容，也可以使用斜体字母。这样的字母与主要文本很好地融合在一起，在一段距离内您将无法分辨页面是否包含斜体字母，但是您会立即注意到一个粗体字。小说被排版的原因是以下范例：避免分神。

好的排版（如好的组织）是您要做到不引起注意。排版的工作是使文本阅读尽可能轻松，即“吸收”其内容。对于小说来说，读者通过逐行阅读文本来吸收内容，就好像他们在听某人讲故事一样。在这种情况下，页面上的任何东西都会分散眼睛的注意力，使他们无法快速、均匀地一行一行地阅读，这会使文本难以阅读。

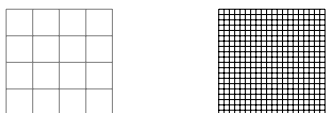
现在，选择您喜欢的每周杂志或报纸，然后浏览典型的页面。您会注意到页面上有很多“正在进行中”。字体以不同的大小和排列方式使用，文本以窄列组织，通常与图片交错。杂志以这种方式排版的原因是另一个范例：引导注意力。

读者不会像小说一样阅读杂志。与其逐行阅读杂志，不如使用标题和简短摘要来检查我们是否要阅读某篇文章。排版工作首先是要引起我们对这些摘要和标题的关注。但是，一旦决定要阅读一篇文章，我们就不再容忍分心，这就是为什么文章的正文排版与小说完全一样的原因。

“避免分心”和“引导注意力”这两个原则也适用于图形。设计图形时，应消除一切“分散眼睛注意力”的东西。同时，您应该尝试通过使用字体/颜色/线宽突出显示不同部分来积极帮助读者“看透图形”。

这是一份可能会分散读者注意力的详尽清单：

- 强烈的对比总是首先被眼睛抓住。例如，考虑以下两个网格：



即使左边的栅格在英语阅读顺序中排在第一位，右边的栅格也更有可能首先被读者看到：白对黑对比度高于灰对白对比度。此外，更多的“网格”也增加了右侧网格中的整体对比度。

诸如网格之类的东西，通常是辅助线，通常不应引起读者的注意，因此，其排版应与背景有较低的对比度。而且，与非常紧密间隔的网格相比，间隔较疏的网格更不会分散注意力。

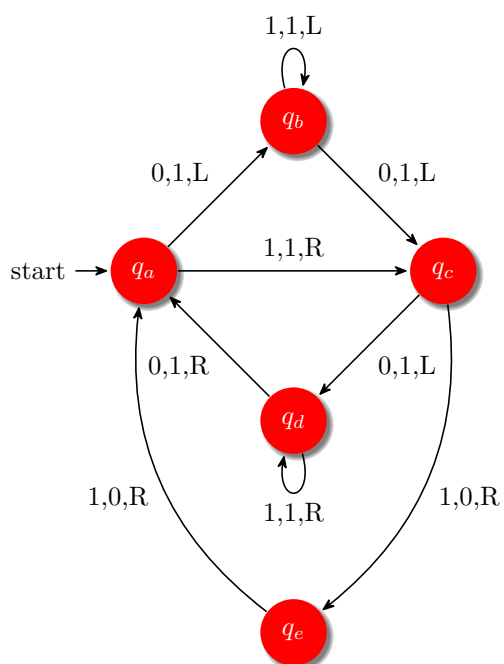
- 虚线创建了许多点，这些点之间存在黑白对比。虚线可能非常分散注意力，因此通常应避免使用。
不要使用不同的虚线图案来区分图表中的曲线。您会以这种方式丢失数据点，并且眼睛并不是特别擅长“按照虚线模式对事物进行分组”。眼睛在根据颜色进行分组方面更胜一筹。
- 用斜线、水平线和垂直线或点来填充区域的背景图案几乎总是会分散注意力，而且通常没有真正的作用。
- 背景图片和阴影分散了注意力，并且很少给图形添加任何重要的东西。
- 可爱的小剪贴画可以很容易地把注意力从数据上转移开来。

Part II

安装与配置

by Till Tantau

这部分解释了系统是如何安装的。通常，已经有人为您的系统做了这样的操作，因此这部分可以略过；但如果情况不是这样，那你是一个可怜的家伙，必须安装，阅读本部分。



The current candidate for the busy beaver for five states. It is presumed that this Turing machine writes a maximum number of 1's before halting among all Turing machines with five states and the tape alphabet $\{0, 1\}$. Proving this conjecture is an open research problem.

```

\usetikzlibrary {arrows.meta,automata,positioning,shadows}
\begin{tikzpicture}[->,>={Stealth[round]},shorten >=1pt,auto,node distance=2.8cm,on grid,semithick,
every state/.style={fill=red,draw=none,circular drop shadow,text=white}]

\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(A)edgenode{0,1,L}(B)
edgenode{1,1,R}(C)
(B)edge[loop above]node{1,1,L}(B)
edgenode{0,1,L}(C)
(C)edgenode{0,1,L}(D)
edge[bend left]node{1,0,R}(E)
(D)edge[loop below]node{1,1,R}(D)
edgenode{0,1,R}(A)
(E)edge[bend left]node{1,0,R}(A);

\node[right=1cm,text width=8cm]at(C)
{
The current candidate for the busy beaver for five states. It is
presumed that this Turing machine writes a maximum number of
 $1$ 's before halting among all Turing machines with five states
and the tape alphabet  $\{0,1\}$ . Proving this conjecture is an
open research problem.
};
\end{tikzpicture}

```

8 安装

有不同的安装 PGF 的方法，这取决于您的系统和需要，您可能还需要安装其他软件包，请参阅下面。在安装之前，您可能希望查看软件包发布所依据的许可证，请参阅9节。

通常，该软件包将已经安装在您的系统上。自然，在这种情况下，您完全不必担心安装过程，可以跳过本节的其余部分。

8.1 宏包和驱动程序版本

该文档是 PGF 宏包的 3.1.5b 版本的一部分。为了运行 PGF，您需要安装一个相当新的 T_EX。使用 L^AT_EX 时，您需要安装以下软件包（更新的版本也应正常工作）：

- `xcolor` 2.00 版本.

使用 Plain T_EX 时，不需要 `xcolor`，但您显然无法获得其（完整）功能。

目前，PGF 支持以下后端驱动程序：

- `luatex` v0.76 或更高版本。大多数早期版本可能也可以使用。
- `pdftex` v0.14 或更高版本。早期版本不起作用。
- `dvips` v5.94a 或更高版本。较早的版本可能也可以使用。

对于图片间连接，需要使用在 DVI 模式下运行的 `pdftex` 版本 1.40 或更高版本处理图片。

- `dvipdfm` v0.13.2c 或更高版本。较早的版本可能也可以使用。

对于图片间连接，需要使用在 DVI 模式下运行的 `pdftex` v1.40 或更高版本处理图片。

- `dvipdfmx` v0.13.2c 或更高版本。较早的版本可能也可以使用。
- `dvisvgm` v1.2.2 或更高版本。较早的版本可能也可以使用。
- `tex4ht` v2003-05-05 或更高版本。较早的版本可能也可以使用。
- `vtex` v8.46a 或更高版本。较早的版本可能也可以使用。
- `textures` v2.1 或更高版本。较早的版本可能也可以使用。
- `xetex` v0.996 或更高版本。较早的版本也可以使用。

目前，PGF 支持以下格式：

- 具有完整功能的 `latex`。
- 具有完整功能的 `plain`，除了图形的包含，它只适用于 pdfT_EX。
- 具有完整功能的 `context`，除了图形包含，它只适用于 pdfT_EX。

有关更多详细信息，请参见10节。

8.2 安装预捆绑的软件包

我不创建或管理 PGF 的预捆绑宏包，但是，幸运的是，其他人也可以。由于我不管理这些宏包，因此无法提供有关如何安装这些宏包的详细说明，但是我可以告诉您要找到它们。如果您在安装时遇到问题，则不妨先看一下 Debian 页面或 MiK_T_EX 的页面。

8.2.1 Debian

命令“`aptitude install pgf`”应该可以解决问题。高枕无忧。

8.2.2 MiKTeX

对于 MiKTeX，请使用更新向导安装名为 `pgf` 和 `xcolor` 的宏包（最新版本）。

8.3 在 texmf 目录中安装

对于永久安装，可以将 PGF 包的文件放在适当的 `texmf` 树中。

当您要求 T_EX 使用某个类或程序包时，它通常会在所谓的 `texmf` 目录中查找必要的文件。这些树只是包含这些文件的巨大目录。默认情况下，T_EX 在三个不同的 `texmf` 目录中查找文件：

- 根 `texmf` 目录，通常位于 `/usr/share/texmf/` 或 `c:\texmf\` 或类似的地方。
- 本地 `texmf` 目录，通常位于 `/usr/local/share/texmf/` 或 `c:\localtexmf\` 或类似的地方。
- 您的个人 `texmf` 目录，通常位于您的主目录中 `~/texmf/` 或 `~/Library/texmf/` 目录。

您应该将软件包安装在本地目录还是个人目录中，具体取决于您是否具有对本地树的写访问权。在根目录树中进行安装可能会导致问题，因为整个 T_EX 安装程序的更新将替换整个目录。

8.3.1 可以将所有内容整合在一起的安装

找到正确的 `texmf` 目录后，必须确定是否要以“将其所有文件都保存在一个位置”的方式来安装 PGF，还是要成为“适用于 TDS”，其中 TDS 表示“T_EX 目录结构”。

如果您想将“一切都保留在一个地方”，请在所选的 `texmf` 目录内创建一个名为 `texmf/tex/generic/pgf` 或 `texmf/tex/generic/pgf-3.1.5b` 的子目录，如果你想这么做的话。然后将 `pgf` 包的所有文件放在此目录中。最后，重建 T_EX 的文件数据库。这是通过运行 `texhash` 或 `mktxlsr` 命令（它们相同）来完成的。在 MiKTeX 中，有一个菜单选项可以执行此操作。

8.3.2 适用于 TDS 的安装

虽然上述安装过程是最“自然”的过程，但我还是建议您这样做，因为它使 `pgf` 宏包的更新和管理变得容易，但它不适用于 TDS。如果要适用于 TDS 的要求，请按照下列步骤操作：（如果您不了解 TDS 的含义，则可能不希望适用于 TDS 的要求。）

`pgf` 宏包的 `.tar` 文件在其根目录中包含以下文件和目录：`README`，`doc`，`generic`，`plain` 和 `latex`。您应将“四个”目录中的每一个“合并”到以下目录：`texmf/doc`，`texmf/tex/generic`，`texmf/tex/plain` 和 `texmf/tex/latex`。例如，在 `.tar` 文件中，`doc` 目录仅包含目录 `pgf`，并且该目录必须移至 `texmf/doc/pgf`。根 `README` 文件可以忽略，因为它是在 `doc/pgf/README` 中复制的。

您也可以考虑将所有内容都放在一个位置，并使用符号链接将适用于 TDS 的目录指向主安装目录。

有关宏包标准安装过程的更详细说明，您可能希望参考 <http://www.ctan.org/installationadvice/>。但是，请注意 `pgf` 宏包没有 `.ins` 文件（只需跳过该部分）。

8.4 更新安装

要从以前的版本更新安装，您需要做的就是用新版本的文件（或如果安装了 `pgf` 的所有目录）替换 `texmf/tex/generic/pgf` 目录中的所有内容，或适用于 TDS 的安装。最简单的方法是先删除旧版本，然后按如上所述进行操作。有时，某些命令的语法会因版本而异。如果事情不像过去那样工作，则不妨查看发行说明和更改日志。

9 许可证和版权

9.1 必需遵守哪些许可证？

分发 PGF 宏包的不同部分遵守不同的许可证：

1. 宏包的代码是双重许可的。这意味着当使用 PGF 宏包时您可以选择其中一个许可证。这两个许可证是：

- (a) 您可以使用 GNU 公共许可证，v2。
- (b) 您可以使用 L^AT_EX 项目公共许可证，v1.3c。

2. 宏包的文档也是双重许可的，同样，您可以选择其中一个许可证。这两个许可证是：

- (a) 您可以使用 GNU 免费文档许可证，v1.2。
- (b) 您可以使用 L^AT_EX 项目公共许可证，v1.3c。

“宏包的文档”是指 pgf 宏包子目录 doc 中的所有文件。详细列表可在文件 doc/generic/pgf/licenses/manifest-
中找到。其他目录中的所有文件都是“程序包代码”的一部分。详细列表可在文件 doc/generic/pgf/licenses/manifest-co
中找到。

在本节的其余部分，将介绍许可证。以下文本受版权保护，请在目录 doc/generic/pgf/licenses 中查看这些许可证有关的详细信息的纯文本版本。

本手册中使用的示例图片 Brave GNU World 徽标取自 Brave GNU World 主页，其版权如下：“Copyright (C) 1999, 2000, 2001, 2002, 2003, 2004 Georg C. F. Greve。只要出现版权和本许可声明，就可以授予制作和分发该笔录的逐字记录副本的权限。”

9.2 GNU 通用公共许可证，V2

9.2.1 前言

大多数软件许可证的用意在于剥夺您共享和修改软件的自由。相反，GNU 通用公共许可证力图保证您共享和修改自由软件的自由——保证自由软件对所有使用者都是自由的。GNU GPL 适用于大多数自由软件基金会的软件，以及任何经作者授权使用的其他软件。（有些自由软件基金会软件受 GNU 函数库通用许可证 GNU LGPL 的保护）。您也可以将它用到您的程序中。

当我们谈到自由软件时，我们谈的是自由而不是价格。我们把 GNU 通用公共许可证设计成您的保障，确保您拥有发布自由软件的自由（您可以自由决定是否要对此项服务收费）；确保您能收到程序源码或者在您需要时能得到它；确保您能修改软件或将它的一部分用于新的自由软件；而且还确保您知道您拥有这些权利。

为了保护您的权利，我们需要作出规定：禁止任何人剥夺您的权利，或者要求您放弃这些权利。如果您修改了自由软件或者发布了软件的副本，这些规定就转化为您的责任。

例如，如果您发布这样一个程序的副本，不管是免费的还是收费的，您必须将您具有的一切权利给予您的接受者；您必须确认他们能收到或得到源码；并且必须向他们展示这些条款的内容，使他们知道他们有这样的权利。

我们采取两项措施来保护您的权利：（1）用版权来保护软件；以及（2）提供您许可证，赋予您复制、发布和/或修改这些软件的法律许可。

同样，为了保护每个作者和我们自己，我们需要清楚地让每个人明白，自由软件没有担保。如果由于某人修改了软件，并继续加以传播，我们需要它的接受者明白：他们所得到的并不是原来的自由软件。由其他人引入的任何问题，不应损害原作者的声誉。

最后，由于任何自由软件不断受到软件专利的威胁，故我们希望避免这样的风险，即如果自由软件的再发布者以个人名义获得专利许可证，也就等同将软件变为私有。为防止这一点，我们必须明确声明：任何专利必须以允许每个人自由使用为前提，否则就不应授予专利。

下面是有关复制、发布和修改的确切的条款和条件。

9.2.2 复制、分发与修改的条款与条件

0. 凡是版权所有者在其程序和作品中声明其程序和作品可以在 GNU GPL 条款的约束下发布，这样的程序或作品都受到本许可证约束。下面提到的“程序”指的是任何这样的程序或作品。而“程序的衍生作品”指的是这样的程序或者版权法认定下的衍生作品，也就是说包含此程序或程序的一部分的套件，可以是原封不动的，或经过修改的，和/或翻译成其他语言的（程序）。（在下文中，“修改”一词的涵义一律包含翻译作品。）每个许可证接受人用“您”来称呼。

本许可证条款不适用于复制、发布和修改以外的行为。这些行为超出这些条款的范围。执行本程序的行为不受条款的限制。而程序的输出只有在其内容构成本程序的衍生作品（并非只是因为该输出由本程序所产生）时，这一条款才适用。至于程序的输出内容是否构成本程序的衍生作品，则取决于程序具体的用途。

1. 只要您在每一程序副本上明显和恰当地宣告版权声明和无担保的声明，并原封不动保持此许可证的声明和无担保的声明，并将此许可证连同程序一起给其他每位程序接受者，您就可以用任何媒体复制和发布您收到的程序的源码。

您可以据转让副本的实际行动收取一定费用。您也可以自由决定是否以提供担保来换取一定的费用。

2. 您可以修改程序的一个或几个副本或程序的任何部分，以此形成基于这些程序的衍生作品。只要您同时满足下面的所有条件，您就可以按前面第一款的要求复制和发布这一经过修改的程序或作品：

- (a) 您必须在修改过的文件上附加明显的说明：叙明您修改过这些文件，以及修改的日期。
- (b) 您必须让您发布或出版的作品，包括本程序的全部或一部分，或内含本程序的全部或部分所衍生的作品，允许第三方在此许可证条款下使用，并且不得因为此项授权行为而收费。
- (c) 如果修改的程序在执行时以互动方式读取命令，您必须使它在开始进入最常使用的方式时列印或显示这样的声明：适当的版权声明和无担保的声明（或者您提供担保的声明）；使用者可以按此许可证条款重新发布程序的声明，并告诉使用者如何看到这一许可证的副本。（例外的情况：如果原始程序以互动方式工作，但它通常并不列印这样的声明，那么您基于此程序的作品也就不用列印声明）。

这些要求适用于整个修改过的作品。如果能够确定作品的一部分并非是本程序的衍生产品，且可以合理地单独考虑并将它与原作品分开的话，则当您将它作为独立的作品发布时，它不受此许可证和其条款的约束。但是当您将这部分与基于本程序的作品一同发布时，则整个套件将受到本许可证条款约束，因为本许可证对于其他许可证持有人的授权扩大到整个产品，也就是套件的每个部分，不管它是谁写的。因此，本条款的意图不在于剥夺您拥有对完全由您自身完成作品的权利，而在于履行权利来控制基于本程序的集体作品或衍生作品的发布。

此外，将与本程序无关的作品和本程序（或本程序的衍生作品）一起放在贮存媒体或发布媒体的同一卷上，并不使其他作品置于此许可证的约束范围之内。

3. 只要您遵守前面的第 1、2 款，并同时满足下列三条中的任一条，您就可以以目标码或可执行形式复制或发布程序（或符合第 2 款，本程序的衍生作品）：

- (a) 在通常用作软件交换的媒体上，和目标码一起附上机器可读的完整的本程序源码。这些源码的发布应符合上面第 1、2 款的要求。或者，
- (b) 在通常用作软件交换的媒体上，和目标码一起，附上书面报价，提供替第三方复制源码的服务。该书面报价有效期不得少于 3 年，费用不得超过完成原程序发布的实际成本，源码的发布应符合上面的第 1、2 款的要求。或者，
- (c) 和目标码一起，附上您收到的发布源码的报价信息。（这一条款只适用于非商业性发布，而且您只收到程序的目标码或可执行码，和按 b 款要求提供的报价。）

作品的源码指的是对作品进行修改最优先择取的形式。对可执行的作品而言，完整的源码套件包括：所有模组的所有原始程序，加上有关的界面的定义，加上控制可执行作品的安装和编译的脚本。至于那些通常伴随着执行本程序所需的作业系统元件（如编译器、核心等）而发布的软件（不论是源码或可执行码），则不在本许可证要求以程序源码形式伴随发布之列，除非它是本程序的一部分。

如果可执行码或目标码是以指定复制地点的方式来发布，那么在同一地点提供等价的源码复制服务也可以算作源码的发布，然而第三方并不需因此而负有必与目标码一起复制源码的义务。

4. 除了本许可证明白声明的方式之外，您不能复制、修改、转发许可证和发布程序。任何试图用其他方式复制、修改、转发许可证和发布程序是无效的，而且将自动结束许可证赋予您的权利。然而，对那些从您那里按许可证条款得到副本和权利的人们，只要他们继续全面履行条款，许可证赋予他们的权利仍然有效。
5. 您没有在本许可证上签字，因而您没有必要一定接受此许可证。然而，没有任何其他东西赋予您修改和发布程序及其衍生作品的权利。如果您不接受许可证，这些行为是法律禁止的。因此，如果您修改或发布程序（或本程序的衍生作品），您就表明您接受这一许可证以及它的所有有关复制、发布和修改程序或基于程序的作品条款和条件。
6. 每当您重新发布程序（或任何程序的衍生作品）时，接受者自动从原始许可证颁发者那里接到受这些条款和条件支配的复制、发布或修改本程序的许可。您不可以增加任何条款来进一步限制本许可证赋予他们的权利。您也没有强求第三方履行许可证条款的义务。
7. 如果因法院判决或因违反专利的指控或任何其他原因（不限于专利问题），使得强加于您的条件（不管是法院判决、协议或其他）和许可证的条件有冲突时，他们也不能令您背离许可证的条款。在您不能同时满足本许可证规定的义务及其他相关的义务来发布程序时，结果是您只能根本不发布程序。例如，如果某一专利许可证不允许所有直接或间接从您那里接受副本的人们，在不付专利费的情况下重新发布程序，唯一能同时满足两方面要求的办法是停止发布程序。

如果本条款的任一部分在特定的环境下无效或无法实施，本条其余部分仍应适用，并将这部分条款作为整体用于其他环境。

本条款的目的不在于引诱您侵犯专利或其他财产权的主张，或争论这种主张的有效性。本条款的主要目的在于保护自由软件发布系统的完整性。它是通过公共许可证的应用来实现的。许多人已依赖同是出自此系统的应用程序，经由此系统发布大量自由软件而做出慷慨的奉献。作者/捐献者有权决定他/她是否通过任何其他系统发布软件，许可证接受者不能强迫作者/捐献者做某种特定的选择。

我们相信许可证其他部分已涵盖本节所述状况，本节目的只在更明确说明许可证其余部分可能产生的结果。

8. 如果由于专利或者由于有版权的介面问题使程序在某些国家的发布和使用受到限制，则以本许可证发布程序的原始作者可以增加发布地区的限制条款，将这些国家明确排除在外，并在这些国家以外的地区发布程序。在这种情况下，这些限制条款如同写入本许可证一样，成为许可证的条款。
9. 自由软件基金会可能随时出版通用公共许可证的修改版和/或新版。新版和当前的版本在精神上保持一致，但在细节上可能有所不同，以便处理新的问题与状况。

每一版本都有不同的版本号。如果程序指定可适用的许可证版本号以及“任何更新的版本”，您有权选择遵循指定的版本或自由软件基金会以后出版的新版本。如果程序未指定许可证版本，您可选择自由软件基金会已经出版的任何版本。
10. 如果您愿意将程序的一部分结合到其他自由程序中，而它们的发布条件不同，请写信给作者，要求准予使用。如果是自由软件基金会加以版权保护的软件，请写信给自由软件基金会，我们有时会作为例外的情况处理。我们的决定受两个主要目标的指导，这两个主要目标是：我们的自由软件的衍生作品继续保持自由状态，以及从整体上促进软件的共享和重复利用。

9.2.3 无担保声明

10. 由于准予免费使用本程序，因此在法律许可范围内，对本程序并不负担担保责任。除非另有书面说明，版权所有者和/或其他提供程序的人们“一样”不提供任何类型的担保，不论是明确的，还是隐含的，包括但不限于可销售和适合特定用途的隐含保证。全部的风险，如程序的质量和性能问题都由您来承担。如果程序出现缺陷，您应当承担所有必要的服务、修复和改正的费用。
11. 非经法律要求或书面同意，在任何情况下，任何版权所有者或任何按许可证条款修改和/或发布程序的人们都不对您的损失负有任何责任。包括由于使用或不能使用程序引起的任何一般的、特殊的、偶然发生的或重大的损失（包括但不限于数据的损失，或者数据变得不精确，或者您或第三方的持续的损失，或者程序不能和其他程序相兼容）。即使版权所有者和其他人已被告知这种损失的可能性也不例外。

9.3 L^AT_EX 项目公共许可证 (LPPL), V1.3c 2006-05-20

9.3.1 导言

L^AT_EX 项目公共许可证 (LPPL) 是分发 L^AT_EX 内核和基本的 L^AT_EX 宏包时必须遵守的许可证。

您可以将该许可证用于您的维护版权和分发的的工作。该许可证很适合您的 T_EX 相关的工作（如 L^AT_EX 宏包），但它是写于这样一种情况之下，即您的工作与 T_EX 毫不相干，您也可以使用该许可证。

本节的内容是“是否以及如何在这个许可证下做分发工作”，下面给出一些说明、例子和建议以方便那些正在考虑在这个许可证下分发他们作品的作者。

这个许可证在某个作品可能被分发和修订，以及在那个作品的修订版的情况下可能被分发。

L^AT_EX3 项目下述的条件给您制作和发布您作品修订版本的自由，同时与您期望的技术规范一致，保持您作品的可用性、完整性、可靠性。如果您不知道如何实现您的目标，同时满足这些条件，请在分发的 L^AT_EX 中阅读文档 ‘`cfgguide.tex`’ 和 ‘`modguide.tex`’ 来获得建议。

9.3.2 定义

本许可证文档使用了如下术语：

作品 指在这个许可证许可下进行的任何分发工作。

衍生作品 指衍生于作品的任何合法的其它作品。

修正 指产生一个合法衍生作品的任何过程，例如，一个文件的产生过程，包括与这个文件相关的原始文件或者像这样文件的一个重要部分无论是直接引用或修改和/或翻译成另一种语言。

修改 指实施产生合法衍生作品的任何过程。

分发 指从一个人复制作品的全部或部分给另一个人。分发包括但不限于制作作品的电子元件，该电子元件适合文件传协议如 FTP 或 HTTP，或适合共享文件系统如升阳公司的网络文件系统（NFS）。

编译作品 指将作品处理成可以直接在计算机系统里可用的格式的过程。这种处理可能包括作品提供的安装工具的使用、作品的转变、作品部分的复制，或其他处理。请注意，任何对由作品提供的安装工具的修改构成了对作品的修改。

当前维护者 指在作品里被提名的一个或者几个人。如果没有这样明确的提名则是合法的“版权持有人”。

基本解释程序 指一个软件或程序，它对运行或解释作品的一部分或全部通常是必需的。

一个基本解释程序可能依赖外部组件，但这些外部组件不认为是基本解释程序的一部分，规定无论何时在交互使用外部组件时都必须明确标识每一个外部组件。否则，将许可证应用于作品时，必须明确指明唯一可用的基本解释程序是“L^AT_EX-格式”或程序执行“T_EX”语言的文件属于“L^AT_EX-格式”。

9.3.3 分发和修改的条件

1. 除分发和/或修改作品的行为之外，其它行为没有涵盖在许可证之中。特别地，作品运转的行为不受限制，并且没有任何关于提供支持这项作品的要求。
2. 您可以分发一个完整的、未经修改的您收到的作品的副本。只有部分作品的分发是需要修改的，但没有权利分发这样的符合该条款中术语的衍生作品。
3. 在符合上面第 2 条款时，您可以分发一件从一个完整的、未经修改的作品副本产生的编译作品。只要该编译作品是直接来自作品中产生的，并按照这样的分发方式，收件人便可以在他们的系统中完全正确地安装编译作品。
4. 如果您是作品当前的维护者，则可以没有限制地修改作品，从而创建一个衍生作品。您也可以没有限制地分发衍生作品，包括衍生作品中产生的编译作品。当前维护者用这种方式分发的衍生作品被认为是作品的新版本。
5. 如果您不是作品当前维护者，则可以修改您的作品副本，从而在作品的基础上创建一个衍生作品，并编译此衍生作品从而在这个衍生作品的基础上创建一个编译作品。
6. 除了版权说明中明确声明的部分，只要对作品的每个部分符合下列条件时，即使您不是作品的当前维护者，您也可以分发衍生作品。只有当前维护者可以对作品的一个组成部分增加这样的声明。
 - (a) 如果衍生作品的一个部分可以直接替换作品中一个基本解释程序使用的部分，然后，当交互式地使用基本解释程序时，作品这个部分被用户识别的任何地方，衍生作品的替代部分清楚地被用户识别为这个部分的一个修改版本。
 - (b) 衍生作品的每个部分都包含针对该部分详述性质变化的突出声明，或对另一个文件的突出参考条目，这个文件是作为衍生作品的一部分分发的，并且包含一个完整准确的更改日志。
 - (c) 衍生作品的信息中没有针对该衍生作品收件人的任何暗示，这些人包括（但不限于）作品的初始版本作者、提供任何支持包括（但不限于）错误报告和处理的人，除非这些人已经明确表示为衍生作品提供了这种支持。
 - (d) 您可以至少分发以下衍生作品中的一种：
 - i. 一个完整的、未经修改的作品副本；如果修改部分的分发是通过提供权限从一个指定的地点复制修改的部分来实现，那么，会提供相同的权限用于从相同或相似的符合该条件的地点复制作品，即使第三方不会随着修改部分被迫复制作品。
 - ii. 足以获得一个完整的、未经修改的作品副本的信息。
7. 如果您不是作品的当前维护者，只要衍生作品是分发给编译作品的全部收件人，并且满足上述第 6 条的关于衍生作品的条件，您就可以分发来自衍生作品的编译作品。
8. 上述条件的目的不是禁止，并且因此不适用以任何方式修改任何部分，以便使其成为作品的那个部分相同的更新版本，正如上述第 4 条当前维护者分发的那样。
9. 以可选的方式分发作品或衍生作品，不是放宽或者废弃该许可证的任何部分，因为它涉及应用过程的结果，在可选的方式中，作品或衍生作品（全部或部分）由适应于该方式的应用过程产生。
10.
 - (a) 衍生作品可能在不同的许可证许可下进行分发，关于作品，如果那个许可证本身尊重上述第 6 条中的条件，虽然它并没有必须尊重本许可证的其他条件。
 - (b) 就作品的变化而言，如果派生作品在不同的许可证许可下进行分发，衍生作品必须提供足够的文件作为它的一部分来允许每一个收件人，兑现上述第 6 条中的限制。
11. 本许可证对与该作品无关的作品没有限制，也对以任何方式聚集这些作品的工作没有任何限制。
12. 本许可证没有任何企图或用于破坏适应的法律。

9.3.4 无担保声明

作品没有保修。除非另有书面说明，版权持有人提供作品“按照原样”，没有任何形式的担保，明示或暗示，包括但不限于，针对特定用途的适销性和适用性的暗示保证。作为作品的质量和性能的全部风险与您同在。作品应证明有缺陷，否则您将承担所有必要的维修、修理、或改正的费用。

除非适用法律要求，或版权持有人书面同意，或该作品任何署名作者，分发和/或修改作品的其他人员，同意上述或任何其他方，负责对您损害赔偿，否则将不赔偿。损害包括任何一般，特殊，附带或间接损害所产生的任何无法使用作品的情况，（包括但不限于数据丢失，所呈现的数据不准确，或持续受损失人作为工作的任何失败与任何其他程序）的结果，即使版权持有人或者作者，或者对方已被告知此类损害的可能性。

9.3.5 作品的维护

工作中有“作者维持”这么一个状态，如果版权所有人明确地、突出地说要注意作品的主要版权，那么作品只能通过版权所有人或者是“作者维持”这么一个简单的状态去维持。

如果当前维护者表明他们愿意接受来自作品中的错误（例如，通过提供一个有效地电子邮箱），则作品处于“维持”状态。但并不要求当前维护者对这些错误有深刻理解或者采取措施。

如果没有当前维护者，或者是有人声明要做作品的当前维护者，然后在 6 个月的时间内，他并没有通过指定的手段使得他能胜任，且没有其他显著的有效维护的迹象，则作品从“维护”状态进入“未维护”状态。

通过与任何一位当前维护者达成一致，您就可以取代他的角色成为一名维护者。

如果作品处于未维护状态，您也可以成为作品的当前维护者，只需通过以下几个步骤：

1. 通过互联网或类似的搜索手段做出一个合理的打算去跟随当前维护者（和版权持有人，如果两者不是同一人）。
2. 如果这个搜索是成功的，然后询问作品是否仍处于维护状态。
 - (a) 如果它正处于维护状态，然后向当前维护者要求更新他们一个月内的通讯数据。
 - (b) 如果搜索不成功，或当前维护者没采取行动以恢复有效维护，那么相关的社区宣布您接管维护。（如果作品是一个 L^AT_EX 作品，可以这样做，例如，张贴到 `comp.text.tex`。）
3.
 - (a) 如果当前维护者做到了，并同意作品的维护交给您，那么公布后将立即生效。
 - (b) 如果当前维护者做不到，并且版权持有人同意把作品的维护交给您，那么公布后将立即生效。
4. 如果您做一个像在上面 2b 所述的那样的“意向性声明”，三个月后您的意向不被当前维护者反驳，也不被版权持有人或其它人反驳，那么，您可以改编作品，从而您成了（新的）当前维护者。
5. 如果在 3b 或 4 的条件下 3 个月内完成了某个更改，以前的未达当前维护者再一次成为可达当前维护者，那么，那个当前维护者必须成为或保持为上述的当前维护者，要求提供他们一个月内更新的通讯数据。

当前维护者的变化，本身并不改变在 LPPL 许可证下这项工作的分发的现实。

如果您成为作品的当前维护者，您应该立即提供，在作品中，突出自己的地位和毫不含糊的声明，您也应该向有关社区宣布新的状态，像上面在 2b 所述。

9.3.6 该许可证允许下能否和如何分发

本节为那些正在考虑在这个许可证下分发他们作品的作者提供了重要指示、例子和建议。这些作者在本节中以“您”的形式出现。

9.3.7 选择这个或其它许可证

如查您想或需要在您作品的任何一部分使用与该许可证显著不同的分发条件，那么，请勿在您作品的任何地方提及该许可证，而应在不同的许可证下分发您的作品。您可以将该许可证的文本作为您的许可证的模板，但您的许可证不应该提及 LPPL，否则会给人您的作品在 LPPL 下分发的印象。

基本 L^AT_EX 分发中的 ‘modguide.tex’ 文档解释了这个许可证条件后面的动机。它解释说，例如，为什么 gnu 通用公共许可证（GPL）下分发 L^AT_EX 被认为是不恰当的。即使您的作品与 L^AT_EX 无关，而在 ‘modguide.tex’ 里的讨论仍可能是相关的，鼓励那些想在某些许可证下分发自己作品的作者去读一下。

9.3.8 不进行分发时的修改的建议

不去修改作品的任何部分是明智的，即使是您自己用的，而且分发修改部分没有满足上述条件时也不要修改。尽管您可能希望永远不会分发这样的修改，但这还是常常发生-您也许忘记曾经修改过，或者其他进入您修改过的版本，您这样分发并且违反了本许可证的条件，从而可能引起法律纠纷，更糟糕的是，引发社会问题。因此，通常在最有利的情况下保持您的作品副本与公开发布的一样。许多作品在没有改变任何得到许可的部分的情况下提供一些方式来控制那种行为。

9.3.9 如何使用该许可证

要使用此许可证，可以在您作品的每一个组成部分里放置一个明确的版权声明，包括您的姓名和作品的撰写和/或最后大幅修改的年份。还包括一个受该许可证约束的分发和/或修改该组件的声明。

下面是一个这样的注意和声明的例子：

```
%%pig.dtx
%%Copyright2005M.Y.Name
%
%Thisworkmaybedistributedand/ormodifiedunderthe
%conditionsoftheLaTeXProjectPublicLicense,eitherversion1.3
%ofthislicenseor(atyouroption)anylaterversion.
%Thelatestversionofthislicenseisin
%http://www.latex-project.org/lppl.txt
%andversion1.3orlaterispartofalldistributionsofLaTeX
%version2005/12/01orlater.
%
%ThisworkhastheLPPLmaintenancestatus`maintained'.
%
%TheCurrentMaintainerofthisworkisM.Y.Name.
%
%Thisworkconsistsofthefiles pig.dtx and pig.ins
%andthederivedfile pig.sty.
```

在一个文件中给出这样的一个注意和声明，将适应该许可证的条件，“Work” 适应三个文件即 “pig.dtx”、“pig.dtx”、“pig.sty”（最后这个文件产生于用 “pig.ins” 产生的 “pig.dtx”），“基本解释程序” 适应任何 “L^AT_EX-格式”，“版权持有人” 和 “当前维护者” 适应 “M.Y.Name”。

如果您不希望 LPPL 的维护这一节适应于您的作品，请将上述的“维护” 更改为“作者维护”。然而，我们建议您使用“维护”，因为添加维护这一节就是为了确保您的作品对社区有用，即使您自己都不再维护并支持您的作品。

9.3.10 不可替代的衍生作品

LPPL 所指定的若干条款为用户社区提供可靠性和稳定性。因此，他们关心这种情形，即衍生作品是用作初始作品的（合适的或不合适的）替代品。如果是这种情形并非如此（例如，如果一个完全不同的任务中重

复使用几行代码)，第 6b 和 6d 的条款不适应。

9.3.11 重要的建议

定义是什么构成了作品 LPPL 要求作品的分发需包含作品的全部文件。因此，您提供给许可用以证判断哪些文件构成作品的方式是非常重要的，例如，通过明确列出每个文件的版权声明或在那里使用诸如这样的一行字：

```
%Thisworkconsistsofallfileslistedinmanifest.txt.
```

在没有一个明确的清单的情况下，许可证不可能判定哪些文件组成了作品，在这种情况下，许可证将有权作出合理的猜测哪些文件组成了作品。

9.4 GNU 自由文档许可证, V1.2, 2002 年 11 月

9.4.1 引言

本授权的目的在于作为一种手册、教科书或其它的具有功能性的有用文件获得自由：确保每一个人都具有复制和重新发布它的有效自由，而不论是否作出修改，也不论其是否具有商业行为。其次，本授权保存了作者以及出版者由于他们的工作而得到名誉的方式，同时也不被认为应该对其他人所作出的修改而担负责任。

本授权是一种“公共版权”，这表示文件的衍生作品本身必须具有相同的自由涵义。它补足了 GNU 公共通用授权——一种为了自由软件而设计的“公共版权”授权。

我们设计了本授权是为了将它使用到自由软件的使用手册上，因为自由软件需要自由的文档：一种自由的程序应该提供与此软件具有相同的自由的使用手册。但是本授权并不被限制在软件使用手册的应用上；它可以被用于任何以文字作基础的作品，而不论其主题内容，或者它是否是一个被出版的印刷书籍。我们建议本授权主要应用在以使用说明或提供参考作为目的的作品上。

9.4.2 效力与定义

本授权的效力在于任何媒体中的任何的使用手册或其它作品，只要其中包含由版权所有人所指定的声明，说明它可以在本授权的条款下被发布。这样的一份声明提供了全球范围内的、免版税的和没有期限的许可，在此所陈述条件下使用那个作品。以下所称的“文件”，指的是任何像这样的使用手册或作品。公众中的任何成员都是被许可人，并且称作为“你”。如果你以一种需要在版权法下取得允许的方式进行复制、修改或发布作品，你就接受了这项许可。

“修改版本”指的是任何包含文件或是它的其中一部份，不论是逐字的复制或是经过修正，或翻译成其它语言的任何作品。

“次要章节”是一个具名的附录，或是文件的本文之前内容的章节，专门用来处理文件的出版者或作者，与文件整体主题（或其它相关内容）的关系，并且不包含任何可以直接落入那个整体主题的内容。（因此，如果文件的部分内容是作为数学教科书，那么其次要章节就可以不用来解释任何数学。）它的关系可以是与主题相关的历史连接，或是与其相关的法律、商业、哲学、伦理道德或政治立场。

“不变章节”是标题已被指定的某些次要章节，在一个声明了是以本授权加以发行的文件中，依此作为不变章节。如果一个章节并不符合上述有关于次要的定义时，则它并不允许被指定为不变。文件可以不包含不变章节。如果文件并没有指出任何不变章节，那么就是没有。

“封面文字”是某些被加以列出的简短文字段落，在一个声明了是以本授权加以发行的文件中，依此作为前封面文字或后封面文字。前封面文字最多可以包含 5 个单词，后封面文字最多可以包含 25 个单词。

文件的“透明”拷贝指的是一份机器可读的拷贝，它以一种一般公众可以取得其规格说明的格式来表现，适合于直接用一般文字编辑器、一般点阵图像程序用于由图元像素构成的影像或一些可以广泛取得的绘图程序用于由向量绘制的图形直接地进行修订；并且适合于输入到文字格式化程式，或是可以自动地转换到适合于输入到文字格式化程序的各种格式。一份以透明以外的档案格式所构成的拷贝，其标记或缺少标记，若是

被安排成用来挫折或是打消读者进行其后续的修改，则此拷贝并非透明。一种影像格式，如果仅仅是用来充斥文本的资料量时，就不是透明的。一个不是透明的拷贝被称为混浊。

透明拷贝适合格式的例子包括有：没有标记的纯 ASCII、Texinfo 输入格式、LaTeX 输入格式、使用可以公开取得其 DTD 的 SGML 或 XML、合乎标准的简单 HTML、PostScript 或 PDF。透明影像格式的例子有 PNG、XCF 和 JPG。混浊格式包括只能以私人文书处理器阅读以及编辑的私人格式、DTD 以及或处理工具不能够一般地加以取得的 SGML 或 XML、以及由某些文书处理器只是为了输出的目的而做出的，由机器制作的 HTML、PostScript 或 PDF。

“标题页”对一本印刷书籍来说，指的是标题页本身，以及所需要用来容纳本授权必须出现在标题页的易读内容的，如此的接续数页。对于并没有任何如此页面的作品的某些格式，“标题页”指的是本文主体开始之前作品标题最显着位置的文字。

一个标题为“XYZ”的章节指的是文件的一个具名的次要单元，其标题精确地为“XYZ”或是将“XYZ”包含在跟着翻译为其它语言的“XYZ”文字后面的括号内 – 这里“XYZ”代表的是名称于下提及的特定章节，像是感谢、贡献、背书或历史。当你修改文件时，给像这样子的章节保存其标题指的是，它保持为一个根据这个定义的标题为“XYZ”

文件可以在用来陈述本授权效力及于文件的声明后，包括担保放弃。这些担保放弃被考虑为以提及的方式，包括在本授权中，但是只被看作为放弃担保之用：任何这些担保放弃可能会有的其它暗示都是无效的，并且也对本授权的含义没有影响。

以下是有关复制、发布及修改的明确条款及条件。

9.4.3 逐字的复制

你可以复制或发布文件于任何媒体，而不论其是否具有商业买卖行为，其条件为具有本授权、版权声明和许可声明，说明本授权效力于文件的所有重制拷贝，并且你没有增加任何其它条件到本授权的条件中。你不可以使用技术手段，来妨碍或控制你所制作或发布的拷贝阅读或进一步的发布。然而，你可以接受补偿以作为拷贝的交换。如果你发布了数量足够大的拷贝，你也必须遵循第三条的条件。

你也可以在上述的相同条件下借出拷贝，并且你可以公开地陈列拷贝。

9.4.4 大量地复制

如果你出版文件的印刷拷贝或者通常具有印刷封面的媒体的拷贝，数量上超过一百个单位，而且文件许可声明要求有封面文字，那么你必须将这些拷贝附上清楚且易读的文字：前封面文字于前封面上、后封面文字于后封面上。这两种封面必须清楚易读地辨认出，你是这些拷贝的出版者。前封面文字必须展示完整的标题，而标题的文字应当同等地显著可见。你可以增加额外的内容于封面上。仅在封面作出改变的复制，只要它们保存了文件的标题，并且满足了这些条件，可以在其它方面被看作为逐字的复制。

如果对于任意一个封面所需要的文字，数量过于庞大以至于不能符合易读的原则，你应该在实际封面的最前面列出所能符合易读原则的内容，然后将剩下的接续在相邻的页面。

如果你出版或发布数量超过一百个单位文件的混浊拷贝，你必须与此混浊拷贝一起包含一份机器可读的透明拷贝，或是与一份混浊拷贝一起或其陈述一个电脑网络位址，使一般的网络使用公众具有存取权，可以使用公开标准的网络协定，下载一份文件的完全透明拷贝，此拷贝中并且没有增加额外的内容。如果你使用后面的选项，当你开始大量地发布混浊拷贝时，你必须采取合理的审慎步骤，以保证这个透明拷贝将会在发布的一开始就保持可供存取，直到你最后一次发布那个发行版的一份混浊拷贝给公众后，至少一年为止。以保证这个透明拷贝，将会在所陈述的位址保持如此的可存取性，直到你最后一次发布那个发行版直接或经由你的代理商或零售商的一份混浊拷贝给公众后，至少一年为止。

你被要求，但不是必须，在重新发布任何大数量的拷贝之前与文件的作者联络，给予他们提供你一份文件的更新版本的机会。

9.4.5 修改

你可以在上述第二条和第三条的条件下，复制和发布文件的修改版本，其条件为你要精确地在本授权下发布修改版本，且修改版本补足了文件的角色，从而允许修改版本的发布和修改权利给任何拥有它拷贝的人。另外，你必须在修改版本中做这些事：

- A. 在标题页或在封面上使用，如果有与先前版本不同的文件，应该被列在文件的历史章节不同的标题。如果版本的原始出版者允许，你可以使用与某一个先前版本相同的标题。
- B. 在修改版本的标题页上列出担负作者权的一个或多个人或实体作为作者，并且列出至少五位文件的主要作者。如果少于五位，则列出全部的主要作者，除非他们免除了你这个要求。
- C. 在标题页陈述修改版本的出版者的名称作为出版者。
- D. 保存文件的所有版权声明。
- E. 为你的修改增加一个与其它版权声明相邻的适当的版权声明。
- F. 在版权声明后面，以授权附录所显示的形式，包括一个给予公众在本授权条款下使用修改版本的许可声明。
- G. 在那个许可声明中保存恒常章节和文件许可声明中必要封面文字的全部列表。
- H. 包括一个未被改变的本授权的副本。
- I. 保存标题为“历史”的章节和其标题，并且增加一项至少陈述如同在标题页中所给的修改版本的标题、年份、新作者和出版者。如果在文件中没有标题为“历史”的章节，则制作出一个陈述如同在它的标题页中所给的文件的标题、年份、新作者和出版者，然后增加一项描述修改版本如前面句子所陈述的情形。
- J. 如果有的话，保存在文件中为了给公众存取文件的透明拷贝，而给予的网络位址，以及同样地在文件中为了它所根据的先前版本，而给予的网络位址。这些可以被放置在历史章节。你可以省略一个在文件本身之前，已经至少出版了四年的作品的网络位址，或是如果它所参照的那个版本的原始出版者给予允许的情形下也可以省略它。
- K. 在任何标题为感谢或贡献的章节，保存章节的标题，并且在那章节保存到那时候为止，每一个贡献者的感谢以及或贡献的所有声色。
- L. 保存文件的所有恒常章节，于其文字以及标题皆不得变更。章节号码或其同等物并不被认为是章节标题的一部份。
- M. 删除任何标题为背书的章节。这样子的章节不可以被包括在修改版本中。
- N. 不要重新命名任何现存的章节，而使其标题为背书，或造成与任何恒常章节相冲突的标题。
- O. 保存任何的担保放弃。

如果修改版本包括新的本文之前内容的章节，或合乎作为次要章节的附录，并且没有包含复制自文件的内容，则你具有选择可以指定一些或全部这些章节为恒常的。要这样做，将它们的标题增加到在修改版本许可声明中的恒常章节列表中。这些标题必须可以和任何其它章节标题加以区别。

你可以增加一个标题为“背书”的章节，其条件为它仅只包含由许多团体所提供的你的修改版本的背书——举例来说，同侪评审的说明，或本文已经被一个机构认可为一个标准的权威定义。

你可以增加一个作为前封面文字的最多五个字的段落，以及一个作为后封面文字的最多二十五个字的段落，到修改版本的封面文字列表的后面。前封面文字和后封面文字都只能有一个段落，可以经由任何一个实体，或经由任何一个实体所作出的安排而被加入。如果文件已经在同样的封面包括了封面文字——先前由你或

由你所代表的相同实体所作出的安排而加入，则你不可以增加另外一个；但是你可以在先前出版者的明确允许下替换掉旧的。

文件的作者和出版者并不由此授权，而给予允许使用他们的名字以为了或经由声称或暗示任何修改版本背书为自己所应得的方式而获得名声的权利。

9.4.6 组合文件

你可以在上述第四条的条款中对于修改版本的定义之下，将文件与其它在本授权下发行的文件组合起来，其条件是你要在组合品中，包括所有原始文件的所有恒常章节，不做修改，同时在组合作品的许可声明中将它们全部列为恒常章节，并且你要保存它们所有的担保放弃。

组合作品只需包含本授权的一份副本，并且重复的恒常章节可以仅以单一个拷贝来取代。如果名称重复但内容不同的恒常章节，则将任此章节的标题，以在它的后面增加的方式加以独特化，如果已知的话，于括号中指出那个章节的原始作者或出版者的名称，或是指定一个独特的号码。在此组合作品许可声明中恒常章节的列表中，对其章节标题也作出相同的调整。

在组合品中，你必须组合在不同原始文件中，标题为历史的任何章节，形成一个标题为历史的章节；同样组合任意标题为感谢或贡献的章节。你必须删除标题为背书的所有章节。

9.4.7 文件的收集

你可以制作含有文件以及其它以本授权发行文件的收集品，并且将本授权对不同文件中的个别副本，以单一个包括在收集品的副本取代，其条件是你遵循在其它方面，给予一个文件逐字复制的允许本授权的规则。

你可以从这样的—个收集品中抽取出一份单一的文件，并且在本授权下将它单独地发布，其条件是你要在抽取出的文件中插入本授权的一份副本，并且在关于那份文件的逐字的复制的所有其它方面，遵循本授权。

9.4.8 独立作品的聚集

一个文件的编辑物，其中或附加于储存物或发布媒体的一册的，具有其它分别且独立的文件或作品的衍生品，如果经由编辑而产生的版权，并没有用来限制此编辑物使用者的法律权力，而超过了个别作品所允许的，则被称为一个聚集品。当文件中包括一个聚集品，本授权的效力并不仅在于此聚集品中的，于其本身并非文件的衍生作品的其它作品。

如果第三条的封面文字要求效力于这些文件的拷贝，并且文件的篇幅少于整个聚集品的一半，则文件的封面文字可以被放在只围绕着文件，并于聚集品内部的封面或是电子的封面同等物上，如果文件是以电子的形式出现的话。否则它们必须出现在绕着整个聚集品的印刷封面上。

9.4.9 翻译

翻译被认为一种修改，因此你可以在第四条的条款下发布文件的翻译。用翻译更换恒常章节需要取得版权所有者的特别允许，但是你可以包括部份或所有恒常章节的翻译，使其附加到这些恒常章节的原始版本之中。你可以包括本授权、文件中的所有许可声明和任何的担保放弃的翻译，其条件为你也必须包括本授权的原始英文版本，以及这些声明与放弃的原始版本。如果发生翻译与本授权、声明或放弃的原始版本有任何的不同意时，将以原始版本为准。

如果在文件中的章节被标题为“感谢”、“贡献”或“历史”，则保存标题第一条的必要条件第四条，典型上将会需要去更动实际的标题。

9.4.10 终止

你不可以复制、修改、在本授权下再设定额外条件的次授权或发布文件，除非明白地表示是在本授权所规范的条件下进行。任何其它的复制、修改、在本授权下再设定额外条件的次授权、或发布文件的意图都是

无效的，并且将会自动地终止你在本授权下所被保障的权利。然而，你在本授权下收到拷贝及权利的团体，只要团体完全遵守本授权的条件，则他们所获得的许可将不会被终止。

9.4.11 本授权的未来改版

自由软件基金会可能偶尔会出版自由文件授权的新修订过的版本。这种新版本在精神上将会类似于现在的版本，但在细节上可能会有不同，以对应新的问题或相关的事。请见 <http://www.gnu.org/copyleft/>。

本授权的任何版本都被指定一个可供区别的版本号码。如果文件指定一个效力于它的特定号码版本的本授权或任何以后的版本，你就具有选择遵循指定的版本，或任何已经由自由软件基金会出版的后来版本并且不是草稿的条款和条件。如果文件并没有指定一个本授权的版本号码，你就可以选择任何一个曾由自由软件基金会所出版不是草稿的版本。

9.4.12 授权附录：如何使用本授权用于你的文件

为使用本授权成为你撰写成的一份文件，必须在文件中包括本授权的一份复本，以及标题页的后面包括许可声明：

Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

如果你有恒常章节、前封面文字和后封面文字，请将 with... Texts 这一行以这些文本取代：

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

如果你有不具封面文字的恒常章节，或一些其它这三者的组合，将可选择的二项合并以符合实际情形。

如果你的文件中包含有并非微不足道的程序码范例，我们建议这些范例平行地在你自由软件授权选择下，比如以 GNU General Public License 的自由软件授权来发布，从而允许它们作为自由软件而使用。

10 支持的格式

T_EX 被设计为一个灵活的系统。无论是对于 T_EX 的输入还是输出，都是如此。本节说明了哪些输入格式以及 PGF 如何支持它们。还说明了可以产生哪些不同的输出格式。

10.1 支持的输入格式：L^AT_EX，Plain T_EX，ConT_EXt

T_EX 并未明确规定应如何格式化输入。例如，通常在 T_EX 中使用开括号来启动绘图作用于，但这绝不是必需的。同样，环境通常以 `\begin` 开头，但是 T_EX 并不太在乎确切的命令名称。

即使可以重新配置 T_EX，用户也不能这样做。因此，某些输入格式指定了一组命令和约定，应如何格式化 T_EX 的输入。目前有三种“主要”格式：Donald Knuth 的原始普通 T_EX 格式，Leslie Lamport 受欢迎的 L^AT_EX 格式和 Hans Hagen 的 ConT_EXt 格式。

10.1.1 使用 L^AT_EX 格式

使用具有 L^AT_EX 格式的 PGF 和 TikZ 很容易：您可以使用 `\usepackage{pgf}` 或 `\usepackage{tikz}`。通常，这就是您所需要做的，所有配置将自动完成（至少我们希望如此）。

用于 L^AT_EX 格式的样式文件位于 PGF 系统的子目录 `latex/pgf/` 中。这些文件的主要作用是将文件包含在 `generic/pgf` 目录中。例如，以下是 `latex/pgf/frontends/tikz.sty` 文件的内容：

```
%Copyright2019byTillTantau
%
%Thisfilemaybedistributedand/ormodified
%
%1.undertheLaTeXProjectPublicLicenseand/or
%2.undertheGNUPublicLicense.
%
%Seethefiledoc/generic/pgf/licenses/LICENSEformoredetails.

\RequirePackage{pgf,pgffor}

\input{tikz.code.tex}

\endinput
```

`generic/pgf` 目录中的文件承担实际的绘图工作。

10.1.2 使用 Plain T_EX 格式

当我们使用 plain T_EX 时，您可以使用 `\input{pgf.tex}` 或 `\input{tikz.tex}`。接下来，使用 `\pgfpicture` 和 `\endpgfpicture` 来代替 `\begin{pgfpicture}` 和 `\end{pgfpicture}`。

与 L^AT_EX 格式不同的是，PGF 不能很好地识别 Plain T_EX 格式的适当配置。特别是如果使用 `pdftex` 或 `tex` 加 `dvips` 时，它只能自动确定正确的输出格式。对于所有其他输出格式，您需要将宏 `\pgfsysdriver` 设置为正确的值。请参阅稍后使用输出格式的描述。

像 L^AT_EX 样式文件一样，像 `tikz.tex` 的 plain T_EX 文件也只是正常包含 `tikz.code.tex` 文件。

10.1.3 使用 ConT_EXt 格式

使用 ConT_EXt 格式时，请使用 `\usemodule[pgf]` 或 `\usemodule[tikz]`。对于 plain T_EX 格式，还必须按如下方式替换环境开始和结束标签：使用 `\startpgfpicture` 和 `\stoppgfpicture` 代替 `\begin{pgfpicture}` 和 `\end{pgfpicture}`；同样地，现在必须使用 `\starttikzpicture` 和 `\starttikzpicture` 代替 `\begin{tikzpicture}` 和 `\end{tikzpicture}`；对于其他环境亦是如此。

ConT_EXt 支持与普通 T_EX 支持非常相似，因此存在相同的限制：您可能必须直接设置输出格式，并且图形包含可能是一个问题。

除了 `pgf` 和 `tikz` 之外，还存在诸如 `pgfcore` 或 `pgfmodulematrix` 之类的模块。要使用它们，您可能需要首先包括模块 `pgfmod`（模块 `pgf` 和 `tikz` 都为您包括 `pgfmod`，因此通常可以跳过此步骤）。这个特殊的模块是必需的，因为 2005 年之前的旧版 ConT_EXt MkII 讽刺地将模块名称的长度限制为 8 个字符，而 `pgf` 的长名称则由模块 `pgfmod` 映射为神秘的 6 个字母的名称。在 ConT_EXt MkIV 中从来没有这个限制，并且现在可以安全地忽略 `pgfmod` 模块。

10.2 支持的输出格式

输出格式是 T_EX 输出其已排版的文本的格式。产生输出是一个（从概念上）分两个阶段的过程：

1. T_EX 排版文本和图形。这种排版的结果主要是一长串字母坐标对，再加上（可能的）一些“特殊”的命令。如此长的字母坐标对被写入 `.dvi` 文件（非正式地称为“与设备无关的文件”）中。
2. 其他一些程序读取此 `.dvi` 文件，并将字母坐标对转换为 PostScript 命令，以便将给定字母放置在给定坐标处。

这个过程经典例子是 `latex` 和 `dvips` 的结合。`latex` 程序（它是使用预安装的 L^AT_EX 宏调用的 `tex` 程序）生成 `.dvi` 文件作为其输出。`dvips` 程序接受此输出并生成一个 `.ps` 文件（一个 PostScript 文件）。这个文件可以使用 `ps2pdf` 进一步转换，它的名字应该是“PostScript to PDF”的意思。使用这个过程的一个例子是 `tex` 和 `dvipdfm` 的结合。`dvipdfm` 程序接受 `.dvi` 文件作为输入，并将其中的字母坐标对转换为 PDF-命令，直接生成 `.pdf` 文件。最后，`tex4ht` 也是一个接受 `.dvi` 文件，并产生一个输出，这次它是 `.html` 文件。`pdftex` 和 `pdflatex` 很特别：它们直接生成没有中间 `.dvi` 阶段的 `pdf` 文件。然而，从程序员的角度来看，它们的行为就好像有一个中间阶段一样。

通常，T_EX 只产生字母坐标对作为它的“输出”。很明显，这使得画曲线变得很困难。为此，可以使用“特殊”命令。不幸的是，对于处理 `.dvi` 文件的不同程序，这些特殊命令并不相同。实际上，每个接受 `.dvi` 文件作为输入的程序对特殊命令有完全不同的语法。

PGF 的主要工作之一是“抽象”不同程序语法中的差异。然而，这意味着对每个项目的支持都必须被“编程”，这是一个耗时而复杂的过程。

10.2.1 选择后端驱动

当 T_EX 排版您的文档时，它不知道您将使用哪个程序来转换 `.dvi` 文件。如果你的 `.dvi` 文件不包含任何特殊命令，这样也就可以完成工作；但现在 `.dvi` 文件几乎都包含许多特殊命令。因此，有必要告诉 T_EX 以后将使用哪个程序。

不幸的是，没有“标准”方法来告诉 T_EX。对于 L^AT_EX 格式，在 `graphics` 包和 PGF 插件中存在一种复杂的机制。对于其他格式，当此插件不能按预期工作时，有必要直接告诉 PGF 您将使用哪个程序。这是通过在加载 `pgf` 之前将宏 `\pgfsysdriver` 重新定义为适当的值来实现的。如果您要使用 `dvips` 程序，您将该宏设置为值 `pgfsys-dvip.def`；如果你使用 `pdftex` 或 `pdflatex`，你可以将其设置为 `pgfsys-pdfTeX.def`；等等。下面将详细讨论不同程序的支持。

10.2.2 输出 PDF 格式的文件

PGF 支持三个程序产生 PDF 输出（PDF 意思是“便携式文档格式”，由 Adobe 公司发明）：`dvipdfm`，`pdftex` 和 `vtex`。`pdflatex` 程序与 `pdftex` 程序相同：它使用不同的输入格式，但输出完全相同。

文件 `pgfsys-pdfTeX.def`

这是用于 pdfT_EX 的驱动程序文件，即使用 `pdftex` 或 `pdflatex` 的命令。它包含 `pgfsys-common-pdf.def`。这个驱动程序有很多功能。（几乎）所有 PGF “能做的事情”都在这个驱动程序中实现了。

文件 `pgfsys-dvipdfm.def`

这是一个驱动程序文件，用于 `(la)tex` 和 `dvipdfm`。该文件包含 `pgfsys-common-pdf.def`。

这个驱动程序支持大多数 PGF 的功能，但有一些限制：

1. 在 `LATEX` 模式下，它使用 `graphicx` 插图机制，不支持屏蔽??。
2. 在 Plain `TEX` 模式下，它不支持插图机制。

文件 `pgfsys-xetex.def`

这是一个驱动程序文件，用于 `xe(la)tex` 和 `xdvipdfmx`。这个驱动程序支持与 `dvipdfm` 驱动程序基本相同的操作。

文件 `pgfsys-vtex.def`

这是用于商业 `VTEX` 程序的驱动程序文件。即使它产生 PDF 输出，它也包含 `pgfsys-common-postscript.def`。请注意，根据命令行参数，`VTEX` 程序可以同时产生 *Postscript* 和 PDF 输出。然而，无论你产生 *Postscript* 或 PDF 输出不会改变任何与驱动有关的东西。

这个驱动程序支持大多数 PGF 的特性，除了以下限制：

1. 在 `LATEX` 模式下，它使用 `graphicx` 作为图形机制，不支持屏蔽??。
2. 在 Plain `TEX` 模式下，它不支持插图机制。
3. 阴影被完全实现，但是产生与 `dvips` 的实现相同的质量。
4. 不支持透明度。
5. 不能记住图形在页面上的位置。

也可以通过首先生成 *PostScript* 文件（参见下文），然后使用诸如 `ps2pdf` 或 *Acrobat Distille* 之类的 *PostScript* to PDF 转换程序来生成 `.pdf` 文件。

10.2.3 输出 *PostScript* 格式的文件

文件 `pgfsys-dvips.def`

这是一个驱动程序文件，用于 `(la)tex` 和 `dvips`。它包含 `pgfsys-common-postscript.def`。

该驱动程序还支持 PGF 的大多数功能，但有以下限制：

1. 在 `LATEX` 模式下，它使用 `graphicx` 用于图形包含。如果使用 `ps2pdf` 进一步处理 *PostScript* 输出产生 PDF，则支持图像屏蔽。
2. 在 Plain `TEX` 模式下，它不支持插图机制。
3. 阴影功能是用 `Type-0` 函数（采样函数）来近似的，因为 `Type-4` 函数在最新的（V3）*PostScript* 语言定义中不可用。由于其固定的分辨率，与来自 PDF 生成驱动程序的阴影功能相比，`Type-0` 型阴影功能在较高的缩放级别上的质量较低。完全支持轴向和径向阴影。与产生 PDF 输出的驱动程序一样，可以获得相同的输出质量（平滑的阴影）。

尽管完全支持不透明度和淡入淡出是 PDF 的功能，这些功能仅在用 `ps2pdf` 进一步处理 *PostScript* 输出后才可见。请注意，更新的 *Ghostscript* 版本对于在 PDF 输出中产生不透明度是必需的。另外，从 *Ghostscript* 的 9.52 版开始，命令行选项 `-dALLOWPSTRANSOPARENCY` 必须添加：

```
ps2pdf-dALLOWPSTRANSOPARENCYexample.ps
```

4. 为了记住图片（图片间的连接），您需要以 `DVI` 模式运行 `pdftex` 的最新版本。

文件 `pgfsys-textures.def`

这是用于 TEXTURES 程序的驱动程序文件。它包含 `pgfsys-common-postscript.def`。

该驱动程序共享 `vtex` 驱动程序的限制，但增加了有限的不透明度支持（尽管没有透明度组，渐变和混合模式）。

You can also use the `vtex` program together with `pgfsys-vtex.def` to produce Postscript output.

10.2.4 输出 SVG 格式的文件

文件 `pgfsys-dvisvgm.def`

此驱动程序将 DVI 文件转换为 SVG 文件，包括文本和字体。选择此驱动程序时，`pgfname` 将为其生成的图片输出所需的原始 SVG 代码。

由于 `graphics` 宏包尚（未？）直接支持该驱动程序， \LaTeX 中对此驱动程序有一条特殊规则：如果选项 `dvisvgm` 赋予 `tikz` 宏包中，将选择该驱动程序（通常使用 `graphics` 选择的驱动程序）。对于像 `beamer` 本身加载 PGF 的宏包，这意味着选项 `dvisvgm` 应该给文档类。

```
%example.tex
\documentclass[dvisvgm]{minimal}

\usepackage{tikz}

\begin{document}
Hello\tikz[baseline]\fill[fill=blue!80!black](0,.75ex)circle[radius=.75ex];
\end{document}
```

然后运行

```
latexexample
dvisvgmexample
```

或

```
lualatex--output-format=dviexample
dvisvgmexample
```

（这是“更好的”，因为它使您可以访问 \TeX 文件中的 Lua \TeX 的全部功能。尤其是在这种情况下，`TikZ` 能够运行图形绘制算法。）

与下面的 `tex4ht` 驱动程序不同，该驱动程序完全支持文本节点。

文件 `pgfsys-tex4ht.def`

这是与 `tex4ht` 程序一起使用的驱动程序文件。它是在加载 `tex4ht` 样式或命令时自动包含的。它包含 `pgfsys-common-svg.def`。

`tex4ht` 程序转换 `.dvi` 为 `.html` 文件。虽然 HTML 格式不能用于绘制图形，但 SVG 格式可以。此驱动程序将要求 PGF 为文本中的每个 PGF 图形生成一个 SVG 图形

使用此驱动程序时，应注意以下限制：

1. 在 \LaTeX 下，它使用 `graphics` 的插图机制。
2. 在 plain \TeX 下，不支持插图机制。
3. 不能记住图形在页面上的位置。
4. 对 `pgfpicture` 环境内的文字的支持不是很好。原因是 SVG 规范当前不能很好地支持文本，尽管可以“转回”HTML，但是 HTML 不得不猜测浏览器渲染的文本的大小。

5. 与其他输出格式不同，图片的边框“确实裁剪”了图片。
6. 不支持矩阵。
7. 不支持颜色渐变。

驱动程序的基本工作原理如下：启动 `{pgfpicture}` 时，将使用适当的 `\special` 命令将 `tex4ht` 的输出定向到名为 `\jobname-xxx.svg` 的新文件，其中 `xxx` 是每个图形增加的数字。然后，直到图片结束，每个（系统层）图形命令都会创建一个特殊的命令，该命令将适当的 `svg` 文字文本插入到输出文件。由于 PostScript/PDF 和 SVG 的成像模型和处理模型并不完全相同，因此确切的细节有些复杂。但对于 PGF 而言，它们“足够接近”。

由于 SVG 标准不很好地支持文本，因此您可能希望使用以下选项来修改文本的处理方式：

`/pgf/tex4ht node/escape=<布尔值>` (default `false`)

使用 `tex4ht` 驱动程序选择文本节点的渲染方法。

当此键设置为 `false` 时，文本将转换为 SVG 文本，该文本在某种程度上受到限制：简单字符（字母，数字，标点， \sum ， \int ，...），下标和上标（但不包括下下标）将显示，但其他所有内容将被过滤掉，忽略或产生的无效 HTML 代码（在最坏的情况下）。这意味着两种文本表现得相当不错：

1. 首先，没有数学模式，特殊字符或其他特殊字符的纯文本。
2. 其次，非常简单的数学文本，包含下标或上标。即使那样，变量也不能正确设置为斜体，并且通常来说，简单文本看起来不是很好。

如果您使用包含特殊内容的文本，即使是 `\alpha` 之类的简单内容，也可能会损坏图形。

```
\tikz\node[draw,/pgf/tex4ht node/escape=false]{Example:$(a+b)^2=a^2+2ab+b^2$};
```

当您编写 `node[/pgf/tex4ht node/escape=true] {<文本>}` 时，PGF 会转回到 HTML 以呈现 `<文本>`。在大多数情况下，此方法会产生有效的 HTML 代码，并且对复杂文本节点的支持要好得多，因为可以在 `{pgfpicture}` 外部很好地渲染的代码也应该在文本节点内部也可以很好地渲染。另一个优点是，在具有固定宽度的文本节点内，HTML 会为长行产生换行符。另一方面，您需要具有良好 SVG 支持的浏览器来显示图片。此外，文本将显示不同，具体取决于您的浏览器，系统上的字体和设置。最后，PGF 必须猜测浏览器呈现的文本的大小才能缩放文本并防止其从节点中粘贴。当它失败时，文本不是太大被裁剪就是太小。

```
\tikz\node[draw,/pgf/tex4ht node/escape=true]
{Example:$\int_0^\infty\frac{1}{1+t^2}dt=\frac{\pi}{2}$};
```

`/pgf/tex4ht node/css=<文件名>` (default `\jobname`)

此选项使您可以告诉浏览器应使用哪个 CSS 文件来设置节点显示的样式（仅当 `tex4ht node/escape=true`）。

`/pgf/tex4ht node/class=<类名>` (default `foreignobject`)

此选项允许您为节点指定一个类名，从而允许它通过 CSS 文件设置样式（仅当 `tex4ht node/escape=true`）。

`/pgf/tex4ht node/id=<ID 名>` (default `\jobname picture number-nodenum`)

此选项允许您为节点提供唯一的 ID，从而可以通过 CSS 文件设置其样式（仅当 `tex4ht node/escape=true`）。

10.2.5 输出完美便携式的 DVI 文件

文件 `pgfsys-dvi.def`

这是一个可以与任何输出驱动程序一起使用的驱动程序文件，`tex4ht` 除外。

该驱动程序将产生完美可移植的 `.dvi` 。通过将所有图片完全由黑色矩形组成文件， \TeX 核心支持该基本和唯一图形形状。在此模型中，即使是直线但倾斜的线也很难正确设置（它们需要由许多小方块组成）。

自然，使用此驱动程序几乎不可能。事实上，可能性是如此之小，所以列出什么是可能会更容易：

- 可以按常规方式放置文本框。
- 可以绘制（描画）直线和曲线。如果它们不是水平或垂直的，则它们由数百个小矩形组成。
- 支持不同宽度的线。
- 支持坐标转换。

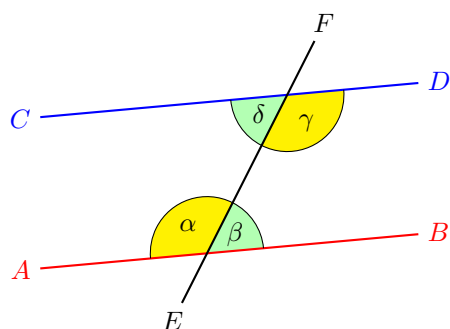
请注意，它甚至不支持填充！（更不用说颜色或其他奇特的东西了。）

该驱动程序只有一个实际的应用程序：当您只需要图片中的水平或垂直线时，它可能会很有用。然后，结果是令人满意的。

Part III

TikZ 不是绘图程序

by Till Tantau



When we assume that AB and CD are parallel, i. e., $AB \parallel CD$, then $\alpha = \gamma$ and $\beta = \delta$.


```

\usetikzlibrary {angles,calc,quotes}
\begin{tikzpicture}[angle radius=.75cm]

\node(A)at(-2,0)[red,left]{$A$};
\node(B)at(3,.5)[red,right]{$B$};
\node(C)at(-2,2)[blue,left]{$C$};
\node(D)at(3,2.5)[blue,right]{$D$};
\node(E)at(60:-5mm)[below]{$E$};
\node(F)at(60:3.5cm)[above]{$F$};

\coordinate(X)at(intersection cs:firstline={(A)--(B)},secondline={(E)--(F)});
\coordinate(Y)at(intersection cs:firstline={(C)--(D)},secondline={(E)--(F)});

\path
(A)edge[red,thick](B)
(C)edge[blue,thick](D)
(E)edge[thick](F)
pic["$\\alpha$",draw,fill=yellow]{angle=F--X--A}
pic["$\\beta$",draw,fill=green!30]{angle=B--X--F}
pic["$\\gamma$",draw,fill=yellow]{angle=E--Y--D}
pic["$\\delta$",draw,fill=green!30]{angle=C--Y--E};

\nodeat($(D)!.5!(B)$)[right=1cm,text width=6cm,rounded corners,fill=red!20,inner sep=1ex]
{
Whenweassumethat$\color{red}AB$and$\color{blue}CD$are
parallel,i.\,e.,$\{\color{red}AB\}\mathbin{\|}\{\color{blue}CD\}$,
then$\alpha=\gamma$and$\beta=\delta$.
};
\end{tikzpicture}

```

11 设计原则

本节描述 TikZ 前端背后的设计原则，其中 TikZ 表示“TikZ ist kein Zeichenprogramm”（TikZ 不是绘图程序）。要使用 TikZ，作为 L^AT_EX 用户使用 `\usepackage{tikz}`，作为 Plain T_EX 用户使用 `\input tikz.tex`。TikZ 的工作是通过提供易于学习和易于使用的图形描述语法来简化您的工作。

TikZ 的命令和语法受到多种语法的影响。基本命令名和路径操作的概念来自 METAFONT，选项机制来自 PSTricks，样式的概念让人联想到 SVG，图形语法来自 GRAPHVIZ。为了让这一切一起工作，一些妥协是必要的。我还添加了一些自己的想法，比如坐标变换。

以下是基于 TikZ 的基本设计原则：

1. 用于指定点的特殊语法。
2. 用于指定路径的句法。
3. 路径上的动作。
4. 图形参数的键值语法。
5. 用于指定 node 的特殊语法。
6. 用于指定 tree 的特殊语法。
7. 用于指定 graph 的特殊语法。
8. 图形参数的分组。
9. 坐标转换系统。

11.1 用于指定点的特殊语法

TikZ 提供了用于指定点和坐标的特殊语法。在最简单的情况下，提供两个 TeX 尺寸，以逗号分隔，用圆括号括起来，如 `(1cm,2pt)`。

您还可以通过使用冒号而不是使用逗号指定点，像 `(30:1cm)` 这样来指定极坐标中的一个点，这意味着“30 度方向上距离原点 1cm 的点”。

如果您不提供单位，如在 `(2,1)` 中，您指定 PGF 的 xy 坐标系中的一个点。默认情况下，单位 x 向量为向右 1cm，单位 y 向量为向上 1cm。

通过指定 `(1,1,1)` 中的三个数字，您还可以在 PGF 的 xyz 坐标系统中指定一个点。

`(first node.south)`。

也可以使用之前定义的形状的锚点，如 `(first node.south)`。

可以在坐标前添加两个加号，如 `++(1cm,0pt)`。意思是“最后一个点往右 1 厘米”。这允许您轻松地指定相对移动。例如 `(1,0) ++(1,0) ++(0,1)` 指定了三个坐标：`(1,0)`，`(2,0)` 和 `(2,1)`。

最后，除了两个加号，您还可以使用一个加号。这也以相对的方式指定了一个点，但是它不会“改变”后续相对命令中使用的当前点。例如 `(1,0) +(1,0) +(0,1)` 指定了三个坐标：`(1,0)`，`(2,0)` 和 `(1,1)`。

11.2 用于指定路径的句法

当使用 TikZ 创建图片时，您的主要任务是指定路径。路径是一系列不需要连接的直线或曲线。TikZ 可以很容易地指定路径，部分使用了 METAPOST 的语法。例如，为指定三角形路径你需要使用

```
(5pt,0pt)--(0pt,0pt)--(0pt,5pt)--cycle
```

当您绘制此路径时，你会得到 \triangle 。

11.3 路径上的动作

路径就是一系列的直线和曲线，还没有制定随着路径命令的执行会发生什么。可以是绘制路径，填充路径，给它添加阴影，剪切路径，或做这些的任何组合。绘制（也称为描边）可以被认为是拿一支一定宽度的钢笔，并沿着路径移动，从而在画布上绘画。填充是指路径的内部用统一的颜色填充。显然，填充只对闭合路径有意义，如有必要，路径在填充之前会自动关闭。

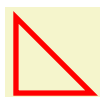
给定 `\path (0,0) rectangle (2ex,1ex);` 中的路径，您可以通过添加 `draw` 选项来绘制它，如 `\path[draw] (0,0) rectangle (2ex,1ex);`，它会产生 \square ；`\draw` 命令是 `\path[draw]` 的缩写。要填充一条路径，使用 `fill` 选项或 `\fill` 命令，它是 `\path[fill]` 的缩写。`\filldraw` 命令是 `\path[fill,draw]` 的缩写。阴影是由 `shade` 选项（`\shade` 和 `\shadedraw` 缩写）以及剪切是由 `clip` 选项完成的。还有一个 `\clip` 命令，它的作用与 `\path[clip]` 相同，但不是像 `\drawclip` 这样的命令，一般来说，使用 `\draw[clip]` 或 `\path[draw,clip]` 代替它。

所有这些命令只能在 `{tikzpicture}` 环境中使用。

TikZ 允许您使用不同的颜色进行填充和描边。

11.4 图形参数的键值语法

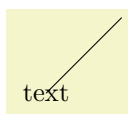
每当 TikZ 绘制或填充路径时，大量图形参数都会影响呈现效果。包括所使用的颜色、点划模式、剪切区域、线宽以及其他许多内容。在 TikZ 中，所有这些选项都指定为所谓的键值对列表，如 `color=red` 中所示，它们作为可选参数传递给路径绘制和填充命令。这种用法类似于 PSTricks。例如，下面会画一个红色的粗三角形；



```
\tikz\draw[line width=2pt,color=red](1,0)--(0,0)--(0,1)--cycle;
```


11.5 用于指定 node 的特殊语法

TikZ 引入了一种特殊语法，用于向图形添加文本，或者更一般地，添加节点。在指定路径时，添加节点，如下例所示：



```
\tikz\draw(1,1)node{text}--(2,2);
```

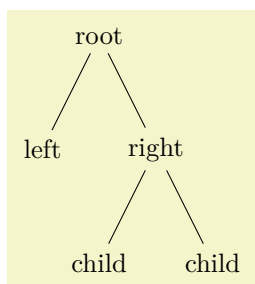
节点插入到路径的当前位置，但是要么在完整路径绘制之后（默认）呈现，要么在完整路径绘制之前呈现。当给出特殊选项时，如 `\draw (1,1) node[circle,draw] {text};`，文本不只是放在当前位置。相反，它被一个圆圈包围着，这个圆圈被“绘制”出来。

您可以通过使用选项 `name=(节点名)` 或在文本外的括号中声明节点名称，如 `node[circle](节点名){text}` 中所示，向节点添加名称以供以后引用。

预定义的形状包括 `rectangle`、`circle` 和 `ellipse`，但也可以定义新形状（尽管这有些挑战性）。

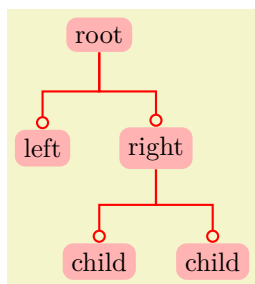
11.6 用于指定 tree 的特殊语法

“节点语法”也可以用来绘制树状结构：一个 `node` 后面可以跟着任意数量的子节点，每个子节点都由关键字 `child` 引入。子节点本身是节点，每个节点可以依次有子节点。

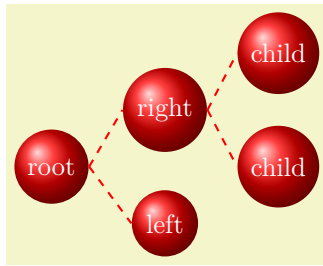


```
\begin{tikzpicture}
\node{root}
child{node{left}}
child{node{right}}
child{node{child}}
child{node{child}}
};
\end{tikzpicture}
```

由于树是由节点组成的，所以可以使用选项来修改树的绘制方式。下面是上述树的两个例子，用不同的选项重新绘制：



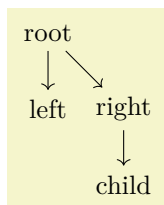
```
\usetikzlibrary {arrows.meta,trees}
\begin{tikzpicture}
[edge from parent fork down,sibling distance=15mm,level distance=15mm,
every node/.style={fill=red!30,rounded corners},
edge from parent/.style={red,-{Circle[open]},thick,draw}]
\node{root}
child{node{left}}
child{node{right}}
child{node{child}}
child{node{child}}
};
\end{tikzpicture}
```



```
\begin{tikzpicture}
[parent anchor=east,child anchor=west,grow=east,
sibling distance=15mm,level distance=15mm,
every node/.style={ball color=red,circle,text=white},
edge from parent/.style={draw,dashed,thick,red}]
\node{root}
child{node{left}}
child{node{right}}
child{node{child}}
child{node{child}}
};
\end{tikzpicture}
```

11.7 用于指定 graph 的特殊语法

`\node` 命令可以很好地控制节点应该放置在何处、节点应该使用什么文本以及节点的外观。但是，在绘制图形时，通常需要创建许多非常类似的节点，这些节点的不同之处在于它们所显示的名称。在这些情况下，可以使用 `graph` 语法，这是在节点语法之上构建的另一个语法层。



```
\usetikzlibrary {graphs}
\tikz\graph[grow down,branch right]{
root->{left,right->{child,child}}
};
```

`\graph` 命令的语法扩展了流行的 GRAPHVIZ 程序中所使用的所谓的 DOT 表示法。

根据您使用的 \TeX 的版本（它必须允许您调用 Lua 代码，这是 \LaTeX 的情况），您还可以要求 `TikZ` 使用几种集成的图形绘制算法之一来自动计算图形节点的良好位置。

11.8 图形参数的分组

图形参数应该经常应用于几个路径绘制或填充命令。例如，我们可能希望绘制多条线，所有线宽都为 1pt。为此，我们将这些命令放在 `{scope}` 环境中，该环境将所需的图形选项作为可选参数。当然，指定的图形参数仅应用于环境中的绘图和填充命令。此外，嵌套 `{scope}` 环境或单个绘制命令可以覆盖外部 `{scope}` 环境的图形参数。在下面的例子中，画了三条红线，两条绿线，一条蓝线：



```
\begin{tikzpicture}
\begin{scope}[color=red]
\draw(0mm,10mm)--(10mm,10mm);
\draw(0mm,8mm)--(10mm,8mm);
\draw(0mm,6mm)--(10mm,6mm);
\end{scope}
\begin{scope}[color=green]
\draw(0mm,4mm)--(10mm,4mm);
\draw(0mm,2mm)--(10mm,2mm);
\draw[color=blue](0mm,0mm)--(10mm,0mm);
\end{scope}
\end{tikzpicture}
```

`{tikzpicture}` 环境本身的行为也类似于 `{scope}` 环境，也就是说，您可以使用可选参数指定图形参数。这些可选命令适用于图片中的所有命令。

11.9 坐标转换系统

TikZ 同时支持 PGF 的坐标转换系统来执行转换，以及画布转换，这是一种更低级的转换系统。（关于坐标转换和画布转换之间区别的详细信息，请参阅第??节。）

语法的设置使得使用画布转换比使用坐标转换更困难。这样做有两个原因：首先，画布转换的使用必须非常小心，经常会导致“糟糕”的图形，改变了线宽和文本的大小。其次，当使用画布转换时，PGF 失去了对节点和形状位置的跟踪。因此，在几乎所有的情况下，您应该使用坐标转换而不是画布转换。

12 层次结构： 宏包，环境，分组和样式

本节解释了当您使用 TikZ 时应该如何构造文件。在顶层，您需要包含 `tikz` 包。在正文中，每个图形都需要放在 `{tikzpicture}` 环境中。在这些环境中，您可以使用 `{scope}` 环境来创建内部组。在分组内，您使用 `\path` 命令来实际绘制一些东西。在所有级别（包级别除外）上，都可以提供适用于环境中的所有内容的图形选项。

12.1 加载宏包和库

```
\usepackage{tikz} %  $\TeX$ 
\input tikz.tex   % plain  $\TeX$ 
\usemodule{tikz}  % Con $\TeX$ t
```

该宏包没有任何选项。

这将自动加载 PGF 和 `pgffor` 包。

PGF 需要知道您打算使用什么 \TeX 驱动程序。在大多数情况下 PGF 足够聪明，可以为您确定正确的驱动程序；如果您使用 \LaTeX ，尤其如此。当您使用普通 \TeX 或 Con \TeX t 与 `dvipdfm` 一起使用时，PGF 不能“自己”知道驱动程序。在这种情况下，您必须在输入 `tikz.tex` 之前使用 `\def\pgfsysdriver{pgfsys-dvipdfs.def}` 告诉 PGF 您要使用什么驱动程序。

```
\usetikzlibrary{<库列表>}
```

加载 TikZ 之后，可以使用此命令加载其他库。库列表应该是包含用逗号分隔的库的名称。除了花括号，您还可以使用方括号，这会是 Con \TeX t 用户想要的。如果尝试第二次加载一个库，不会发生任何错误。

示例：`\usetikzlibrary{arrows.meta}`

上面的命令将加载一大堆额外的箭头提示的定义。

该命令的作用是为“库列表”中的每个“库”加载文件 `tikzlibrary<库>.code.tex`。如果此文件不存在，则改为加载文件 `pgflibrary<库>.code.tex`。如果该文件也不存在，则会显示一条错误消息。因此，要编写自己的库文件，您要做的就是将一个具有适当名称的文件放在 \TeX 可以找到它的地方。然后， \LaTeX 、Plain \TeX 和 Con \TeX t 就用户可以使用您的库。

12.2 创建图片

12.2.1 使用环境创建图片

TikZ 的“最外层”分组是 `{tikzpicture}` 环境。您可以只在这个环境中提供绘图命令，而在外部提供绘图命令（在许多其他包中都是可能的）将导致混乱。

在 TikZ 中，图形的呈现方式受到图形选项的强烈影响。例如，有一个选项用于设置用于绘图的颜色，另一个选项用于设置用于填充的颜色，还有一些更模糊的选项，如设置在使用外部程序绘制函数时写入的临时文件的文件名中使用的前缀。图形选项在键列表中指定，请参阅下面的第 12.4 节了解详细信息。所有图形选项都是适用于 `{tikzpicture}` 的本地选项。

```
\begin{tikzpicture}<动画规则>[<选项>]
  <环境内容>
\end{tikzpicture}
```

除了 `\tikzset` 命令之外，所有 TikZ 命令都应该在这个环境中提供。您不能在此环境之外使用像低级命令 `\pgfpathmoveto` 这样的图形命令，这样做会导致混乱。对于 TikZ，像 `\path` 这样的命令只在这个环境中定义，所以在这里很少有出错的可能。

遇到此环境时，解析(选项)请参见12.4节。这里给出的所有选项将适用于整个情况。如果 `animations` 库已经加载，在指定选项之前，请参阅??节以了解详细信息。

接下来，处理环境的内容，并将其中的图形命令放入一个框中。非图形文本也可能被抑制，但 `{tikzpicture}` 环境中的非 PGF 命令不应该产生任何“输出”，因为这可能会完全扰乱后端驱动程序的定位系统。顺便说一下，抑制普通文本是通过临时将字体切换为 `\nullfont` 来实现的。但是，您可以“跳回”到正常的 `TEX` 排版。例如，当您指定一个节点时，就会发生这种情况。

在环境结束时，PGF 尝试猜测图形边框的大小，然后调整图片框的大小，使其具有此大小。为了“猜测”，每次 PGF 遇到一个新坐标时，它都会更新边界框的大小，以便包含所有这些坐标。这通常会给出一个很好的边界框近似值，但并不总是准确的。首先，斜线的线宽没有被正确考虑。第二，曲线的控制点经常在曲线的“外”，使边界框太大。在这种情况下，您应该使用 `[use as bounding box]` 选项。

以下关键因素会影响生成图像的基线：

`/tikz/baseline=<尺寸或坐标或 默认值>` (default 0pt)

通常，图片的下端放在周围文字的基线上。例如，当您给出代码 `\tikz\draw(0,0)circle(.5ex);`，PGF 会发现图片的下端是 $-.5ex - 0.2pt$ ($0.2pt$ 是线宽 $0.4pt$ 的一半)，上端为 $.5ex + .5pt$ 。然后，下端将被放在基线上，导致如下结果：○。

使用此选项，您可以指定应该升高或降低图片，使其<尺寸>高度位于基线上。例如，我们可以使用 `\tikz[baseline=0pt]\draw(0,0)circle(.5ex);`；因为现在，基线在 x 轴的高度上。

此选项通常对于“内联”图形很有用，如

`A \rightarrow B` `$A\mathbin{\tikz[baseline]\draw[->](0pt,.5ex)--(3ex,.5ex);}B$`

除了<尺寸>，您还可以在括号中提供坐标。这样做的效果是将基线放在在图片末尾给定的“坐标”的 y 坐标上。这意味着，在图片的最后，将计算<坐标>，然后将基线设置为结果点的 y 坐标。这使得引用节点基线的 y 坐标变得很容易。

`Hello world.` `\usetikzlibrary {shapes.misc}`
Hello
`\tikz[baseline=(X.base)]`
`\node[cross out,draw](X){world.};`

Top align: `\tikz[baseline=(currentboundingbox.north)]`
`\draw(0,0)rectangle(1cm,1ex);`

使用 `baseline=default` 将 `baseline` 选项重置为初始设置。

`/tikz/execute at begin picture=<代码>` (no default)

此选项将导致在图片的开头执行<代码>。这个选项必须在 `{tikzpicture}` 环境本身的参数中给出，否则这个选项将不起作用。毕竟，这幅图后来已经“开始”了。多重设置此选项的效果会累积。

此选项主要用于像 `every picture` 这样的样式中，以在图片开始时执行某些代码。

`/tikz/execute at end picture=<代码>` (no default)

此选项安装将在图片末尾执行的<代码>。多次使用此选项将导致代码累积。这个选项也必须在 `{tikzpicture}` 环境的可选参数中提供。



```
\usetikzlibrary {backgrounds}
\begin{tikzpicture}[execute at end picture=%
{
\begin{pgfonlayer}{background}
\path[fill=yellow,rounded corners]
(currentboundingbox.southwest)rectangle
(currentboundingbox.northeast);
\end{pgfonlayer}
}]
\nodeat(0,0){X};
\nodeat(2,1){Y};
\end{tikzpicture}
```

“globally” change the following style:

所有选项均在图片末尾“结束”。要“全局”设置选项，请更改以下样式：

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

此样式安装在每张图片的开头。

```
\tikzset{every picture/.style=semithick}
```

注意，不应该使用 `\tikzset` 直接设置选项。例如，如果您想在默认情况下使用 `1pt` 的行宽，请不要尝试在文档的开头使用 `\tikzset{line width=1pt}`。由于在许多地方更改了线宽，因此无法使用。相反，使用

```
\tikzset{every picture/.style={line width=1pt}}
```

这将达到预期的效果。

在其他 $\text{T}_\text{E}\text{X}$ 格式中，应改为使用以下命令：

`\tikzpicture[⟨选项⟩]`

⟨环境内容⟩

`\endtikzpicture`

这是环境的 Plain $\text{T}_\text{E}\text{X}$ 版本。

`\starttikzpicture[⟨选项⟩]`

⟨环境内容⟩

`\stoptikzpicture`

这是环境的 Con $\text{T}_\text{E}\text{X}$ t 版本。


12.2.2 使用命令创建图片

下面的命令是 `{tikzpicture}` 的另一种选择，它对于由单个或几个命令组成的图形特别有用。

`\tikz⟨动画规则⟩[⟨选项⟩]{⟨路径命令⟩}`

该命令将 ⟨路径命令⟩ 放在 `{tikzpicture}` 环境中 ⟨路径命令⟩ 可能包含段落和脆弱命令（如抄录文本）。如果只有一个路径命令，则不需要用花括号将其括起来，如果有多个，则需要添加它们（这类似于 `\foreach` 语句，也类似于 Java 或 C 等编程语言中有关花括号位置的规则）。

示例: `\tikz{\draw (0,0) rectangle (2ex,1ex);}` 生成 

示例: `\tikz \draw (0,0) rectangle (2ex,1ex);` 生成 

12.2.3 处理类别代号和 Babel 宏包

在 TikZ 图片中,大多数符号需要有类别代码 12 (普通文本),以确保解析器正常工作。当使用像 `babel` 这样的宏包时,通常不会出现这种情况,因为这些包会积极地更改类别代号。

为了解决这个问题,TikZ 提供了一个小型库,也称为 `babel` (但是,它也可以与任何其他全局更改类别代号的包一起使用)。它所做的是重置每个 `{tikzpicture}` 开始处的类别代号,并在每个节点开始处恢复它们。在几乎所有情况下,这正是您所期望和需要的,因此我建议始终通过使用 `\usetikzlibrary{babel}` 来加载此库。要了解类别代号到底发生了什么,请参阅第??节。

12.2.4 添加背景

默认情况下,图片没有任何背景,也就是说,它们在你不绘制任何东西的所有部分都是“透明”的。相反,你可能希望在你的图片后面有一个彩色的背景,或者在它周围有一个黑色的框架,或者在它上面和下面有线条,或者其他一些装饰。

由于背景通常是不需要的,添加背景的样式的定义已经放在 `background` 库中。这个库记录在??节中。


12.3 使用分组构造图片

在 `{tikzpicture}` 环境中,您可以使用 `{scope}` 环境创建分组。此环境仅在 `{tikzpicture}` 环境中可用,再次强调,所以再也没有机会做错任何事情。

12.3.1 Scope 环境

```
\begin{scope}<动画规则>[<选项>]  
  <环境内容>  
\end{scope}
```

所有的 `<选项>` 对于 `<环境内容>` 都是本地的。此外,剪切路径对于环境也是本地的,也就是说,在环境中完成的任何剪切都会在它的末尾结束。



```
\begin{tikzpicture}[ultra thick]  
  \begin{scope}[red]  
    \draw(0mm,10mm)--(10mm,10mm);  
    \draw(0mm,8mm)--(10mm,8mm);  
  \end{scope}  
  \draw(0mm,6mm)--(10mm,6mm);  
  \begin{scope}[green]  
    \draw(0mm,4mm)--(10mm,4mm);  
    \draw(0mm,2mm)--(10mm,2mm);  
  \end{scope}  
  \draw[blue](0mm,0mm)--(10mm,0mm);  
\end{scope}  
\end{tikzpicture}
```

`/tikz/name=<分组名>` (no default)

在动画中为分组分配一个名称。这个名称是一个驱动程序看不到的“高级”名称,所以你可以在一个名称中使用空格、数字、字母,但是你应该不使用任何标点符号,如点、逗号或冒号。

以下风格会影响分组:

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

这种样式安装在每个分组的开始处。

以下选项对分组很有用:

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

这个选项安装了一些将在分组开始时执行的代码。这个选项必须在 `{scope}` 环境的参数中提供。
此效果仅适用于当前分组，而不适用于子分组。

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

此选项安装将在当前范围的末尾执行的一些代码。多次使用此选项将导致代码累积。这个选项还必须在 `{scope}` 环境的可选参数中提供。
同样，该效果仅适用于当前分组，而不适用于子分组。

`\scope<动画规则>[<选项>]`

<环境内容>

`\endscope`

环境的 Plain TeX 版本。

`\startscope<动画规则>[<选项>]`

<环境内容>

`\stopscope`

环境的 ConTeXt 版本。

12.3.2 Scope 环境的简写形式

有一个小库可以使使用分组更容易:

TikZ Library scopes

```
\usetikzlibrary{scopes} % TeX and plain TeX
\usetikzlibrary[scopes] % ConTeXt
```

这个库定义了启动和结束 `{scope}` 环境的简写形式。

当加载此库时，会发生以下情况：在 TikZ 图片的某些地方，允许仅使用单大括号开始分组，前提是单大括号后面跟着方括号中的选项：



```
\usetikzlibrary {scopes}
\begin{tikzpicture}
{[ultra thick]
{[red]
\draw(0mm,10mm)--(10mm,10mm);
\draw(0mm,8mm)--(10mm,8mm);
}
\draw(0mm,6mm)--(10mm,6mm);
}
{[green]
\draw(0mm,4mm)--(10mm,4mm);
\draw(0mm,2mm)--(10mm,2mm);
\draw[blue](0mm,0mm)--(10mm,0mm);
}
\end{tikzpicture}
```

在上面的例子中，`{[ultra thick]}` 实际上会插入一个 `\begin{scope}[ultra thick]`，对应的 `}` 会插入一个 `\end{scope}`。

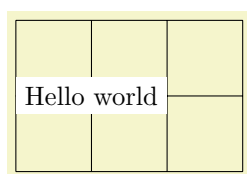
以下是具有特殊含义的开括号的“某些位置”：首先，紧跟在结束路径的分号之后。第二，正好在分组结束之后。第三，在分组的开始，包括一幅图的开始。还要注意，必须在后面加上一些方括号，否则大括号将被视为普通的 TeX 分组。

12.3.3 单命令分组

在某些情况下，为单个命令创建分组是有用的。例如，当您希望使用算法绘制图形来布局树时，树的路径需要被一个分组包围，该分组的唯一目的是使用一个键来选择分组的布局。类似地，为了在背景层上放置一些东西，需要创建一个分组。在这种情况下，仅为单个命令创建 `\begin{scope}` 和 `\end{scope}` 会很麻烦，`\scoped` 命令可能很有用：

`\scoped`(动画规则)[(选项)](路径命令)

这个命令类似于 `\tikz`，只是您可以在 `{tikzpicture}` 中使用它。它将采用下面的 (路径命令)，并将其放在带有 (选项) 集的 `{scope}` 中。(路径命令)可以是一个以分号结束的单个命令，也可以包含多个命令，但是它们必须用大括号括起来。



```
\usetikzlibrary {backgrounds}
\begin{tikzpicture}
\node[fill=white] at (1,1){Helloworld};
\scoped[on background layer]
\draw(0,0)grid(3,2);
\end{tikzpicture}
```

12.3.4 在路径内部使用分组

`\path` 命令（将在后面的部分中更详细地描述）也接受图形选项。这些选项是路径本地的。此外，可以像下面这样简单地使用花括号在路径内创建局部分组



```
\tikz\draw(0,0)--(1,1)
{[rounded corners]--(2,0)--(3,1)}
--(3,0)--(2,1);
```

注意，许多选项仅应用于整个路径，不能以这种方式确定分组。例如，不可能限定路径的 `color` 的分组。有关路径的详细信息，请参阅有关路径部分的说明。

最后，在 `\path` 命令的参数中指定的某些元素也接受本地选项。例如，指定节点接受选项。在这种情况下，选项仅应用于节点，而不应用于周围的路径。

12.4 使用图形选项

12.4.1 图形选项是如何处理的

许多 TikZ 的命令和环境都接受选项。这些选项称为键列表。要处理这些选项，使用以下命令，您也可以自己调用。请注意，通常最好不要直接调用此命令，因为这将确保选项的作用仅限于定义良好的分组。

`\tikzset`{(选项)}

此命令将使用 `\pgfkeys` 命令处理 (选项)，该命令的默认路径设置为 `/tikz`，详细记录在第??节中。在正常情况下，(选项)将是 (键)=(值) 形式的逗号分隔对列表，但当您使用强大的 `pgfkeys` 机制时，可能会发生更奇妙的事情，请再次参阅第??节。

处理一对 (键)=(值) 时，会发生以下情况：

1. 如果 (键) 是一个完整的键（以斜杠开始），它将被直接处理，如第??节所述。
2. 否则（通常情况下），检查 `/tikz/(键)` 是否是一个键，如果是，则执行它。
3. 否则，检查 `/pgf/(键)` 是否是一个键，如果是，则执行它。
4. 否则，检查 (键) 是否是一种颜色，如果是，则执行 `color=(键)`。

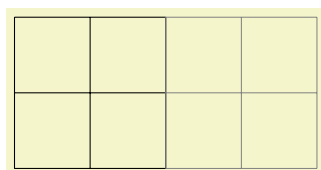
5. 否则，检查 `<键>` 是否包含连字符，如果是，则执行 `arrows=<键>`。
6. 否则，检查 `<键>` 是否是形状的名称，如果是，则执行 `shape=<键>`。
7. 否则，将显示错误消息。

注意，根据上面的描述，所有以 `/tikz` 开头的键以及所有以 `/pgf` 开头的键都可以在一个 `<选项>` 列表中作为 `<键>` 使用。

12.4.2 使用样式来管理图片的外观

有一种组织图形选项集的方法与正常的分组机制“正交”。例如，您可能希望所有的“帮助线”都以某种特定的方式绘制，比如灰色和细的（做“竖线”，那会分散注意力）。为此，您可以使用样式。

样式是一个键，使用时会导致一组图形选项被处理。定义样式后，就可以像使用其他任何键一样使用它。例如，预定义的 `help lines` 样式，您应该将其用于背景中的线条，例如网格线或辅助线。



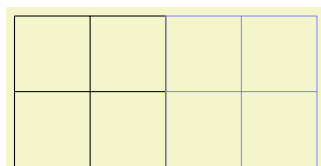
```
\begin{tikzpicture}
\draw(0,0)grid+(2,2);
\draw[help lines](2,0)grid+(2,2);
\end{tikzpicture}
```

还可以使用选项来定义样式。假设我们希望定义一个名为 `my style` 的样式，当使用这个样式时，我们希望绘制颜色设置为 `red`，填充颜色设置为 `red!20`。为此，我们使用以下选项：

```
mystyle/.style={draw=red,fill=red!20}
```

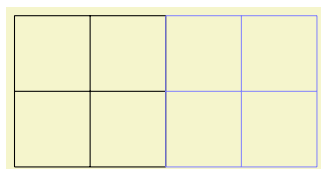
`/.style` 的含义如下：“关键 `my style` 这里不是使用，而是定义。因此，请使用 `my style` 键进行设置。在下面，将具有与我们使用 `draw=red,fill=red!20` 相同的效果。”

回到帮助线示例，假设我们更喜欢蓝色的帮助线。这方面可以实现如下：



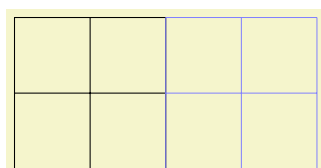
```
\begin{tikzpicture}[help lines/.style={blue!50,very thin}]
\draw(0,0)grid+(2,2);
\draw[help lines](2,0)grid+(2,2);
\end{tikzpicture}
```

当然，样式背后的主要思想之一是它们可以被用于不同的图片。在本例中，我们必须在开头使用 `\tikzset` 命令。



```
\tikzset{help lines/.style={blue!50,very thin}}
%...
\begin{tikzpicture}
\draw(0,0)grid+(2,2);
\draw[help lines](2,0)grid+(2,2);
\end{tikzpicture}
```

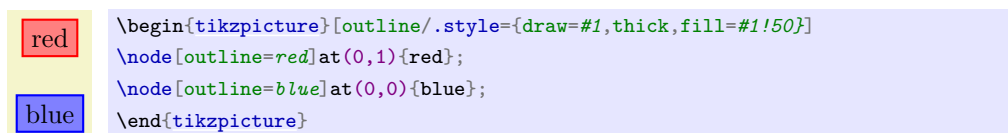
由于样式只是 `pgfkeys` 通用样式工具的特殊情况，实际上您可以做的更多。让我们从向一个已经存在的样式添加选项开始。这是使用 `/.append style` 而不是 `/.style` 做到的：



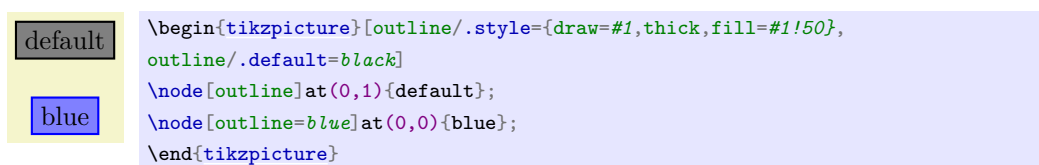
```
\begin{tikzpicture}[help lines/.append style=blue!50]
\draw(0,0)grid+(2,2);
\draw[help lines](2,0)grid+(2,2);
\end{tikzpicture}
```

在上面的示例中,选项 `blue!50` 被附加到样式 `help lines`,它现在具有与 `black!50,very thin,blue!50` 相同的效果。注意,如果设置了两中颜色,最后一个会“赢得胜利”。还存在一个名为 `/.prefix style` 的样式,它在样式的开头添加了一些东西。

与普通键一样,样式也可以参数化。这意味着您需要在`使用样式时使用 $\langle \text{样式} \rangle = \langle \text{值} \rangle$,而不只是 $\langle \text{样式} \rangle$ 。在本例中, $\langle \text{样式} \rangle$ 中出现的所有 #1 都被 $\langle \text{值} \rangle$ 替换。下面是一个示例,展示了如何使用它。`



对于参数化样式,您也可以使用 `/.default` 处理程序设置默认值:



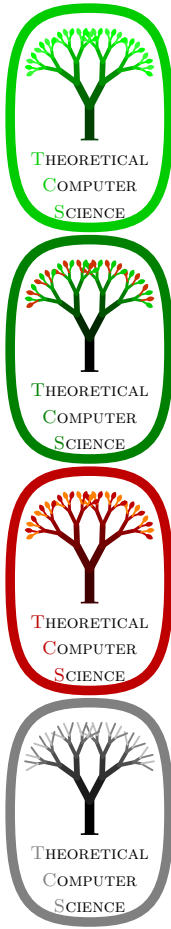
有关使用和设置样式的更多细节,请参见第??节。

Part IV

库

by Till Tantau

在这一部分中，介绍了库。它们提供了其他预定义的图形对象，例如新的箭头或新的地标，有些地方还提供了基本 PGF 或 TikZ 系统的扩展。默认情况下不加载库，因为许多用户不会需要到它们。



```

\usetikzlibrary {arrows,trees}
\tikzset{
ld/.style={level distance=#1},lw/.style={line width=#1},
level 1/.style={ld=4.5mm,trunk!lw=1ex,sibling angle=60},
level 2/.style={ld=3.5mm,trunk!80!leaf a,lw=.8ex,sibling angle=56},
level 3/.style={ld=2.75mm,trunk!60!leaf a,lw=.6ex,sibling angle=52},
level 4/.style={ld=2mm,trunk!40!leaf a,lw=.4ex,sibling angle=48},
level 5/.style={ld=1mm,trunk!20!leaf a,lw=.3ex,sibling angle=44},
level 6/.style={ld=1.75mm,leaf a,lw=.2ex,sibling angle=40},
}

\pgfarrowsdeclare{leaf}{leaf}
{\pgfarrowsleftextend{-2pt}\pgfarrowsrightextend{1pt}}
{
\pgfpathmoveto{\pgfpoint{-2pt}{0pt}}
\pgfpatharc{150}{30}{1.8pt}
\pgfpatharc{-30}{-150}{1.8pt}
\pgfusepathqfill
}

\newcommand{\logo}[5]
{
\colorlet{border}{#1}
\colorlet{trunk}{#2}
\colorlet{leafa}{#3}
\colorlet{leafb}{#4}
\begin{tikzpicture}
\scriptsize\scshape
\draw[border,line width=1ex,yshift=.3cm,
yscale=1.45,xscale=1.05,looseness=1.42]
(1,0)to[out=90,in=0](0,1)to[out=180,in=90](-1,0)
to[out=-90,in=-180](0,-1)to[out=0,in=-90](1,0)--cycle;

\coordinate(root)[grow cyclic,rotate=90]
child{
child[line cap=round]foreach\ain{0,1}{
childforeach\bin{0,1}{
childforeach\cin{0,1}{
childforeach\din{0,1}{
childforeach\leafcolorin{leafa,leafb}
{edgefromparent[color=\leafcolor,-#5]}
}}}}
edgefromparent[shorten >=-1pt,serif cm-,line cap=butt]
};

\node[align=center,below]at(0pt,-.5ex)
{\textcolor{border}{T}heoretical\\
\textcolor{border}{C}omputer\\
\textcolor{border}{S}cience};
\end{tikzpicture}
}

\begin{minipage}{3cm}
\logo{green!80!black}{green!25!black}{green}{green!80}{leaf}\\
\logo{green!50!black}{black}{green!80!black}{red!80!green}{leaf}\\
\logo{red!75!black}{red!25!black}{red!75!black}{orange}{leaf}\\
\logo{black!50}{black}{black!50}{black!25}{}
\end{minipage}

```

13 思维导图图形库

TikZ Library `mindmap`

```
\usetikzlibrary{mindmap} % LATEX and plain TEX
\usetikzlibrary{mindmap} % ConTEXt
```

该包提供了用于绘制思维导图的样式。

13.1 概述

该库旨在简化思维导图或概念图的创建。思维导图是一个用于表示概念和相关概念以及注释的图形。思维导图本质上是树，可能添加了一些额外的边，但通常以特殊的方式绘制：根概念位于页面的中间，并绘制为巨大的圆形，椭圆形或云形。然后，相关的概念通过类似树枝的卷须“离开”这个根概念。

由 TikZ 的 `mindmap` 库绘制的思维导图看起来与标准思维导图有些不同：虽然大根概念仍然是一个圆，相关概念也被描述为（较小）圆。相关概念通过外观相似的连接链接到根概念。总体效果在视觉上令人愉悦，但是当读者看到使用此库创建的图片时，可能不会立即想到思维导图。

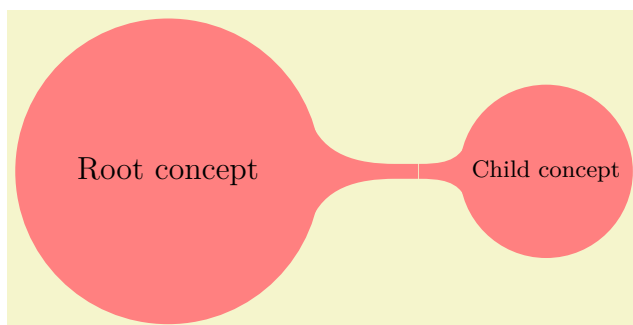
尽管不是绝对必要的，但是通常您会使用 TikZ 的树机制来创建思维导图，并且在树内使用时，包的某些样式和宏效果最佳。但是，仍然可能并且有时有必要将思维导图的各个部分视为具有任意边的图，这也是可能的。

13.2 思维导图样式

每个思维导图都应放在使用 `mindmap` 样式的作用域或图片环境中。此样式将加载一些内部设置。

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

对所有图片或至少包含思维导图的图片使用这种风格。它安装了一大堆对绘制思维导图有用的设置。

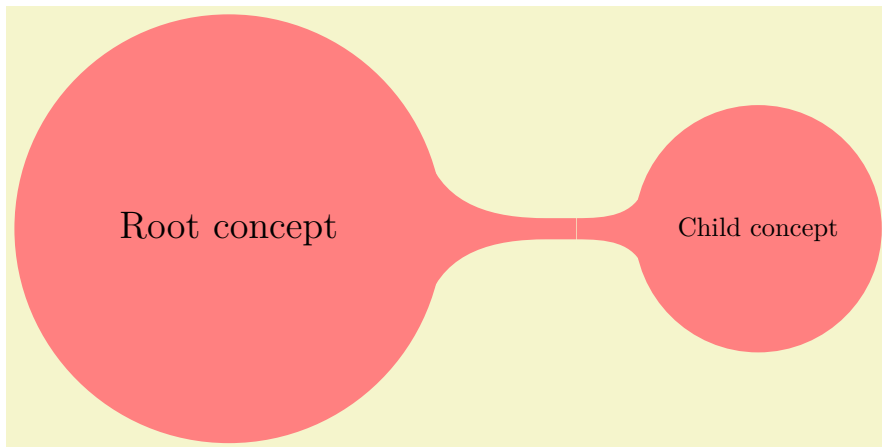


```
\usetikzlibrary {mindmap}
\tikz[mindmap,concept color=red!50]
\node[concept]{Rootconcept}
child[grow=right]{node[concept]{Childconcept}};
```

预定义概念的大小，以使中型思维导图适合 A4 页面（或多或少）。

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

这种样式包含在 `mindmap` 样式中。改变这种样式，为你的思维导图添加特殊设置。



```
\usetikzlibrary {mindmap}
\tikz[large mindmap,concept color=red!50]
\node[concept]{Rootconcept}
child[grow=right]{node[concept]{Childconcept}};
```

备注 注意 `mindmap` 重新定义 `font` 大小和 `sibling angle`, 取决于当前概念级别 (即 `level 1 concept`, `level 2 concept` 等中的概念)。因此, 如果您需要重新定义这些变量, 请使用

```
level 1 concept/.append style={font=\small}
```

或

```
level 2 concept/.append style={sibling distance=90}
```

应在 `mindmap` 样式之后使用。

/tikz/small mindmap (style, no value)

default size of concepts, fonts and distances so that a medium-sized mindmap will fit on an A5 page (A5 pages are half as large as A4 pages). Mindmaps with `small mindmap` will also fit onto a standard frame of the `beamer` package.

此样式包含 `mindmap` 样式, 但另外会更改概念, 字体和距离的默认大小, 以使中型思维导图适合 A5 页面 (A5 页面的大小是 A4 页面的一半)。具有 `small mindmap` 的思维导图适合 `beamer` 宏包的标准框架。

/tikz/large mindmap (style, no value)

此样式包含 `mindmap` 样式, 但另外会更改概念, 字体和距离的默认大小, 以使中型思维导图适合 A3 页面 (A3 页面的大小是 A4 页面的两倍)。

/tikz/huge mindmap (style, no value)

这种样式会使概念更大, 最好与 A2 纸及更高版本一起使用。

13.3 概念节点

思维导图的基本实体在 TikZ 中被称为概念。概念是样式为 `concept` 的节点, 它必须是圆形的, 这样一些连接宏才能正常工作。

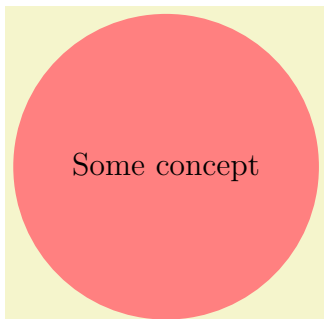
13.3.1 孤立的节点

以下样式影响孤立概念的呈现方式:

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

尽管某些样式（如 `extra concept`）自动加载此样式，该样式应与所有作为概念的节点一起使用。

基本上，这种样式使概念节点成为圆形，并加载被称为 `concept color` 的统一颜色，请参见下文。此外，样式 `every concept` 被自动加载。



```
\usetikzlibrary {mindmap}
\tikz[mindmap,concept color=red!50]\node[concept]{Someconcept};
```

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

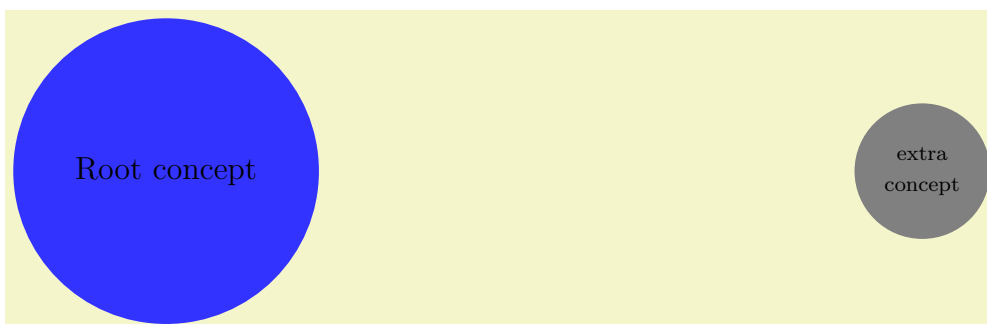
为了更改概念节点的外观，您应该更改此样式。但是请注意，对于某些连接带来说，概念的颜色应该统一，因此您不应使用此选项更改概念的颜色或绘制/填充状态。它对于更改文本颜色和字体很有用。

`/tikz/concept color=⟨颜色⟩` (no default)

这个选项告诉 TikZ 哪种颜色应该用于填充和描边概念。此选项与直接将 `every concept` 设置为所需颜色的区别在于，此选项允许 TikZ 跟踪用于概念的颜色。当你在两个相连的概念之间改变颜色时，这一点很重要。在这种情况下，TikZ 可以自动创建阴影，提供新旧概念颜色之间的平滑过渡；我们将在下一节中回到这个问题。

`/tikz/extra concept` (style, no value)

此样式适用于不属于“思维导图”但位于其旁边的概念。通常，它们将具有更淡的颜色或更小的尺寸。为了绘制这些概念并在代码中指出这些概念是附加的，可以使用此样式。



```
\usetikzlibrary {mindmap}
\begin{tikzpicture}[mindmap,concept color=blue!80]
\node[concept]{Rootconcept};
\node[extra concept]at(10,0){extraconcept};
\end{tikzpicture}
```

`/tikz/every extra concept` (style, no value)

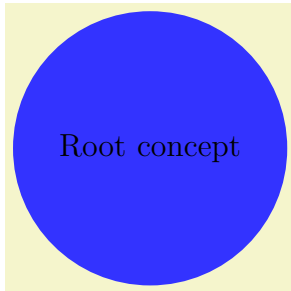
更改此样式可更改其他概念的外观。

13.3.2 树中的概念

正如前面指出的，TikZ 假设您的思维导图是使用 TikZ 的 `child` 工具构建的。有许多选项影响如何在树的不同级别上呈现概念。

`/tikz/root concept` (style, no value)

此样式用于思维导图树的根节点。通过添加一些东西，您可以更改思维导图的根节点的渲染方式。

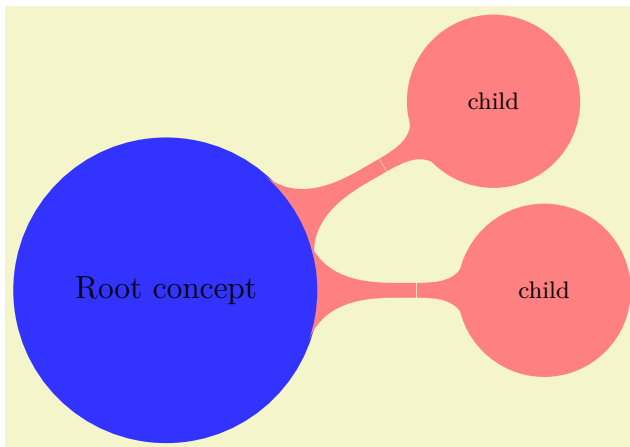


```
\usetikzlibrary {mindmap}
\tikz
[root concept/.append style={concept color=blue!80,minimum size=3.5cm},
mindmap]
\node[concept]{Rootconcept};
```

请注意 `large mindmap` 重新定义这些样式，因此您应该仅在图片内向此样式添加一些内容。

`/tikz/level 1 concept` (style, no value)

`mindmap` 样式将此样式添加到 `level 1` 样式，这意味着思维导图树的第一级子代将使用此样式。



```
\usetikzlibrary {mindmap}
\tikz
[root concept/.append style={concept color=blue!80},
level 1 concept/.append style={concept color=red!50},
mindmap]
\node[concept]{Rootconcept}
child[grow=30]{node[concept]{child}}
child[grow=0]{node[concept]{child}};
```

`/tikz/level 2 concept` (style, no value)

类似于 `level 1 concept`，仅适用于第二级子代。

`/tikz/level 3 concept` (style, no value)

类似于 `level 1 concept`。

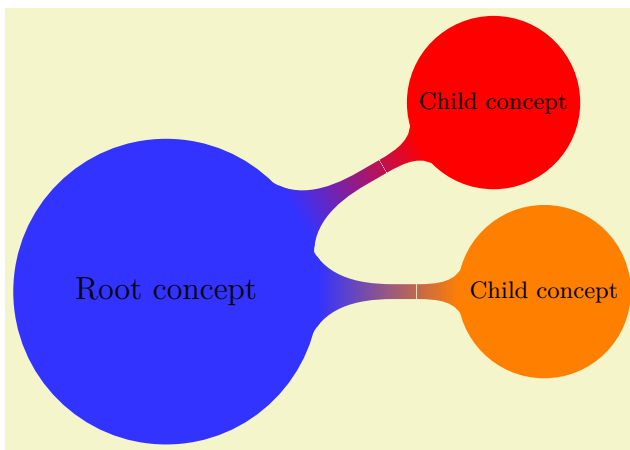
`/tikz/level 4 concept` (style, no value)

类似于 `level 1 concept`。注意，没有第五种和更高级别的样式，在这种情况下您需要直接修改 `level 5`。

`/tikz/concept color=<颜色>` (no default)

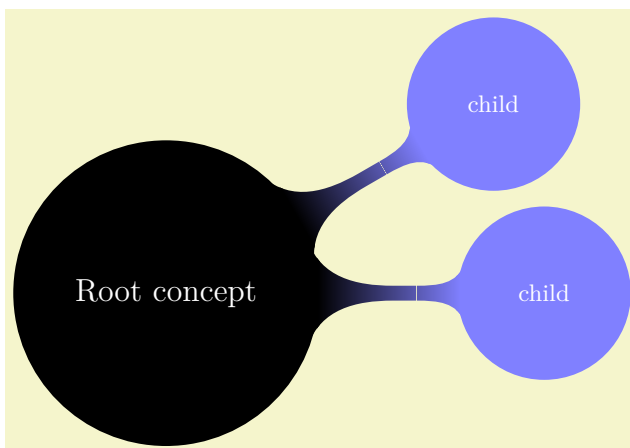
我们已经看到该选项用于更改概念的颜色。现在来看看它在概念的子节点上使用时的效果。通常，此选项仅更改子项的颜色。但是，当将该选项作为 `child` 操作的选项（而不是对 `node` 操作的选项，也不是通过 `level 1` 样式给所有子级提供的选项），TikZ 将平滑地将概念颜色从父级概念颜色过渡为子级概念的颜色。

这是一个例子：



```
\usetikzlibrary {mindmap}
\tikz[mindmap,concept color=blue!80]
\node[concept]{Rootconcept}
child[concept color=red,grow=30]{node[concept]{Childconcept}}
child[concept color=orange,grow=0]{node[concept]{Childconcept}};
```

为了使概念颜色随层次结构级别而变化，需要一点魔法：



```

\usetikzlibrary {mindmap}
\tikz[mindmap,text=white,
root concept/.style={concept color=blue},
level 1 concept/.append style=
{every child/.style={concept color=blue!50}}]
\node[concept]{Rootconcept}
child[grow=30]{node[concept]{child}}
child[grow=0]{node[concept]{child}};

```

13.4 连接概念

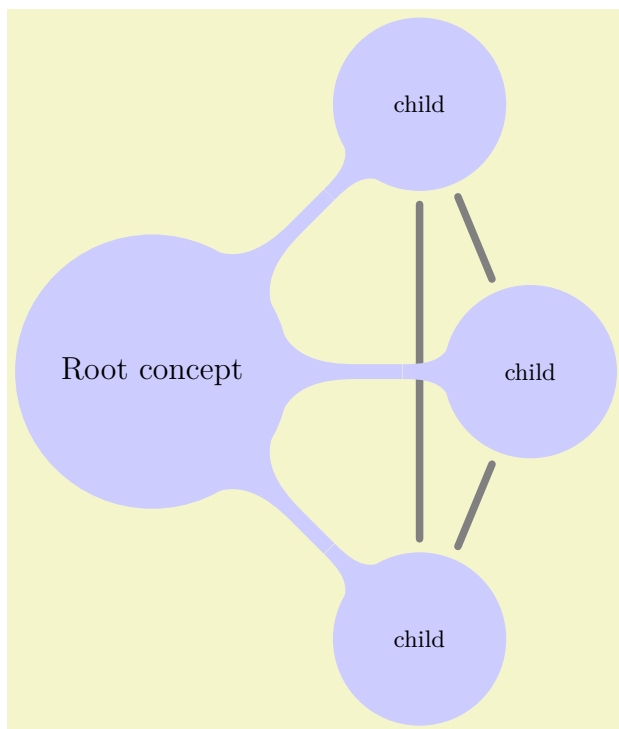
13.4.1 简单连接

连接两个概念的最简单方法是在它们之间画一条线。为了使此类线条具有一致的外观，建议在绘制此类线条时使用以下样式：

`/tikz/concept connection` (style, no value)

此样式可用于两个概念之间的线条。可随意重新定义此样式。

在绘制主思维导图之后需要连接概念时，会出现问题。在这种情况下，您将希望连接线位于主思维导图的后面。但是，仅在确定概念的坐标之后才能绘制线。在这种情况下，应将连接线放在背景层上，如以下示例所示：



```

\usetikzlibrary {backgrounds,mindmap}
\begin{tikzpicture}
[root concept/.append style={concept color=blue!20,minimum size=2cm},
level 1 concept/.append style={sibling angle=45},
mindmap]
\node[concept]{Rootconcept}
[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}

```

13.4.2 圆形带连接

除了两个概念之间创建简单的线条外，您还可以在两个节点之间添加一条稍微带有一点有机末端的带状图。默认情况下，这些带也用作思维导图树中父级概念的边。

对于带的绘制，使用了特殊的装饰，该装饰在 `mindmap` 库中定义：

装饰 `circle connection bar`

to-be-decorated path should lie on the border of the first circle, the end should lie on the border of the second circle. The following two decoration keys should be initialized with the sizes of the circles:

此装饰可用于连接两个圆。待装饰路径的起点应在第一个圆的边界上，终点应在第二个圆的边界上。以下两个装饰键应使用圆的大小初始化：

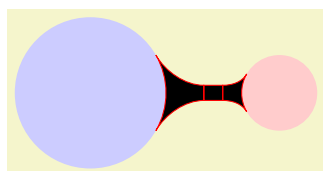
- `start radius`
- `end radius`

此外，以下两个装饰键会影响装饰：

- `amplitude`
- `angle`

该装饰将直线转变为路径，该路径从第一个圆的边界开始，相对于连接圆的圆心的线的指定角度。然后，路径变成一个矩形，其厚度由幅度给定。最后，路径以第二个圆上相同的角度结束。

这是一个应该更清楚地说明示例：



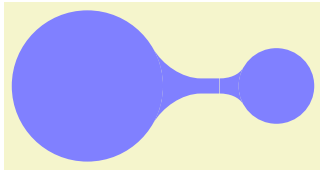
```

\usetikzlibrary {mindmap}
\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=circleconnectionbar](1,0)--(2,0);
\end{tikzpicture}

```

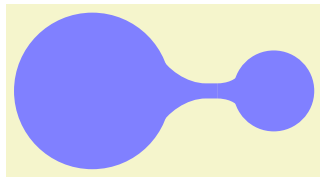
可以看出，装饰的路径由三部分组成，对于绘图并没有真正的用处。但是，如果仅填充装饰的路径，并且使用与圆相同的颜色，则效果会更好。



```
\usetikzlibrary {mindmap}
\begin{tikzpicture}
[blue!50,decoration={startradius=1cm,
endradius=.5cm,amplitude=2mm,angle=30}]
\fill(0,0)circle(1cm);
\fill(2.5,0)circle(.5cm);

\fill[decorate,decoration=circleconnectionbar](1,0)--(2,0);
\end{tikzpicture}
```

在上面的示例中，您可能会注意到圆和装饰路径之间的白色小线。这是由于舍入误差造成的。不幸的是，对于更大的距离，错误会累积得非常强烈，特别是因为 TikZ 和 TeX 不太善于计算平方根。因此，最好使圆稍大一些以掩盖此类问题。当使用形状为 `circ` 的节点时，只需添加 `draw` 的 `line width` 的选项增加 1pt 或 2pt（对于非常长的距离，您可能需要最大 4pt 的线宽）。



```
\usetikzlibrary {mindmap}
\begin{tikzpicture}
[blue!50,decoration={startradius=1cm,
endradius=.5cm,amplitude=2mm,angle=30}]
\fill(0,0)circle(1cm+1pt);
\fill(2.4,0)circle(.5cm+1pt);

\fill[decorate,decoration=circleconnectionbar](1,0)--(1.9,0);
\end{tikzpicture}
```

13.4.3 圆形连接带的 To-Path

`circle connection bar` 装饰使用起来有点复杂。特别是指定半径非常麻烦（可以一劳永逸地设置幅度和角度）。因此，`mindmap` 库定义了一个特殊的 to-path，可以为您执行必要的计算。

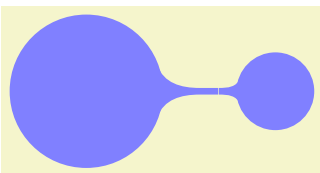
`/tikz/circle connection bar` (style, no value)

这种样式会加载相当复杂的 to-path。与正常的 to-path 不同，此路径要求 to-path 的起点和目标为被其形状命名为 `circle` 的节点。如果不是这种情况，该路径将产生错误。

假设起点和目标是圆，则 to-path 将首先计算这些圆的半径（通过测量从 `center` 锚点到边界上某个锚点的距离）并设置 `start circle` 键。接下来，及时当 `draw=none` 选项被设置时 `fill` 选项也被设置为 `concept color`。装饰设置为 `circle connection bar`。最后，该样式包含以下样式：

`/tikz/every circle connection bar` (style, no value)

重新定义此样式将改变圆形连接带路径的 to-path。



```
\usetikzlibrary {mindmap}
\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]{};

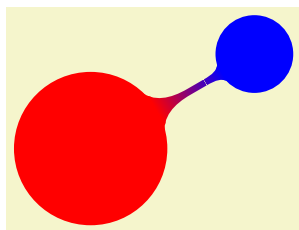
\path(n1)to[circle connection bar](n2);
\end{tikzpicture}
```

注意，在一个 `\path` 中同时使用多个 `to` 操作和 `circle connection bar` 选项不是一个好主意。相反，使用 `edge` 操作创建多个连接，该操作为每条边创建一个新的范围。

在思维导图图中，有时我们希望概念的颜色从一种颜色变为另一种。然后，理想情况下，连接带应由这两种颜色之间的平滑过渡组成。如果您“手工”尝试使用阴影完成这项工作，会有些棘手，因此 `mindmap` 库提供了一个特殊的选项来简化此过程。

`/tikz/circle connection bar switch color=from(<first color>)to(<second color>)` (no default)

此样式的作用类似于 `circle connection bar`。唯一的区别是，它不是使用单色填充路径，而是使用阴影。



```
\usetikzlibrary {mindmap}
\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]{};

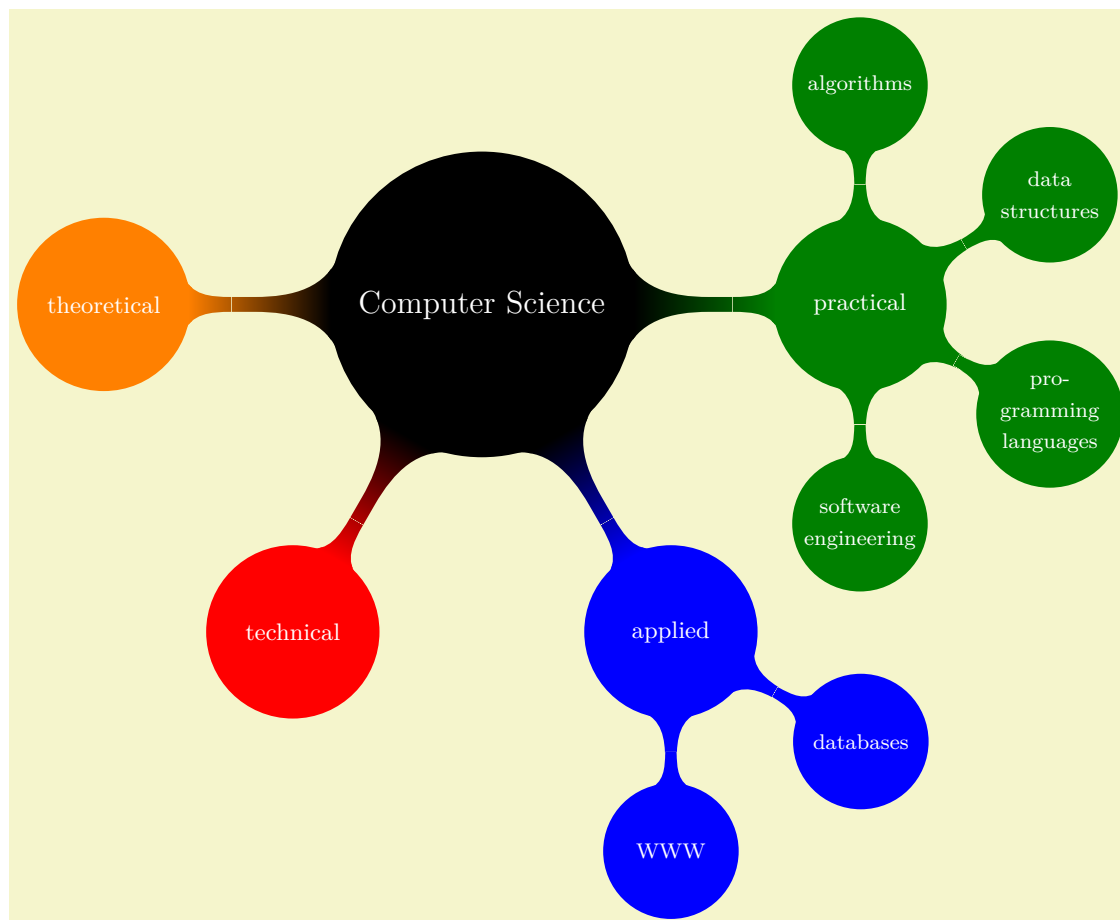
\path(n1)to[circle connection bar switch color=from(red)to(blue)](n2);
\end{tikzpicture}
```

13.4.4 树的边

在大多数情况下，将思维导图构建为树时，思维导图中的概念会自动连接。原因是 `mindmap` 加载了一个 `circle connection bar` 路径作为 `edge from parent path`。另外，`mindmap` 选项还可以处理诸如设置正确的 `draw` 和 `outer sep` 设置之类的事情。

详细地说，`mindmap` 选项将 `edge from parent path` 设置为使用 `to-path circle connection bar` 连接父节点和子节点的路径。`concept color` 选项（本地）通过使用 `circle connection bar switch color` 来改变这一点，而不是使用从颜色设置为旧的（父节点的）概念颜色和到颜色设置为新的（子节点的）概念颜色。这意味着，当您向一个 `child` 命令提供 `concept color` 选项时，颜色将从父节点概念颜色更改为指定的颜色。

现在让我们用这种方法来构建一棵树。请注意，我们将 `concept color` 传递给相应的 `child`，而不是它下面的 `node`。



```

\usetikzlibrary {mindmap}
\begin{tikzpicture}
\path[mindmap,concept color=black,text=white]
node[concept]{ComputerScience}
[clockwise from=0]
%notethat`siblingangle`canonlybedefinedin
%`level1concept/.appendstyle={}`
child[concept color=green!50!black]{
node[concept]{practical}
[clockwise from=90]
child{node[concept]{algorithms}}
child{node[concept]{datastructures}}
child{node[concept]{pro\-\gramminglanguages}}
child{node[concept]{softwareengineer\-\ing}}
}
%notethatthe`conceptcolor`ispassedtothe`child`(!)
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}

```

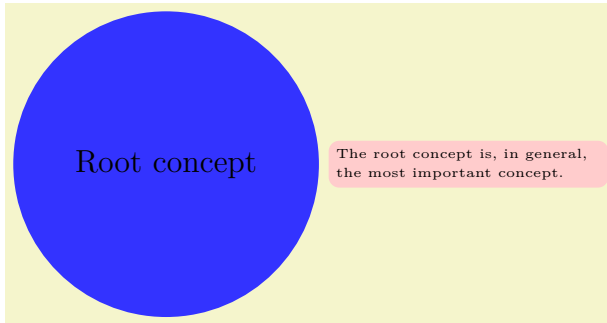
13.5 添加注释

注释是思维导图之外的一些文本，与其他概念不同，它只是解释了思维导图中的某些内容。下面的样式主要用于帮助代码的读者查看注释节点中的节点。

`/tikz/annotation`

(style, no value)

此样式表示节点是注释节点。它包含 `every annotation` 样式，这使您可以方便地更改此样式。



```
\usetikzlibrary {mindmap}
\begin{tikzpicture}
[mindmap, concept color=blue!80,
every annotation/.style={fill=red!20}]
\node[concept](root){Rootconcept};

\node[annotation, right] at (root.east)
{Therootconceptis, ingeneral, themostimportantconcept.};
\end{tikzpicture}
```

`/tikz/every annotation`

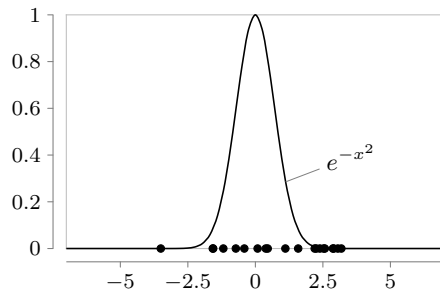
(style, no value)

此样式被 `annotation` 包含。

Part V

数据可视化

by Till Tantau



• $\sum_{i=1}^{10} x_i$, where $x_i \sim U(-1, 1)$

```
\usetikzlibrary {datavisualization.formats.functions}
\tikz\datavisualization[scientific axes=clean]
[
  visualize as smooth line=Gaussian,
  Gaussian={pinindata={text={\mathit{e}^{-x^2}}},when=axis1}}
]
data[format=function]{
  varx:interval[-7:7]samples51;
  funcy=exp(-\valuex*\valuex);
}
[
  visualize as scatter,
  legend={southeastoutside},
  scatter={
    style={mark=*,marksize=1.4pt},
    labelinlegend={text={
      \sum_{i=1}^{10} x_i, where x_i \sim U(-1, 1)}}
  }
]
data[format=function]{
  vari:interval[0:1]samples20;
  funcy=0;
  funcx=(rand+rand+rand+rand+rand+
  rand+rand+rand+rand+rand);
};
```

14 数据可视化简介

数据可视化是将通常由多个数值组成的数据点转换为用图形表示的过程。包括众所周知的函数图形，饼图、条形图、箱形图和矢量图也可看成是数据可视化的示例。

PGF 的数据可视化子系统采用一种通用的开放式方法来进行数据可视化。像 PGF 中的所有其他内容一样，数据可视化系统中有一个功能强大但不太容易使用的基本层，以及灵活性较差但使用起来却更简单的前端层。本节概述了数据可视化系统背后的基本思想。

14.1 概念：数据点

数据可视化的最重要的输入始终是原始数据。该数据通常以不同的格式存在，并且数据可视化子系统提供了读取此类格式以及定义新输入格式的方法。但是，与输入格式无关，我们可能会问数据可视化子系统应该能够处理哪种数据。对于二维图，我们需要成对的实数列表。对于条形图，我们通常需要一个实数列表，可能还需要一些颜色和标签。对于表面图，我们需要一个由实数组成的三维矩阵。对于矢量场，我们需要更复杂的数据。

数据可视化子系统不假设正在处理哪种类型的数据。取而代之的是，整个“渲染管道”都围绕一个称为数据点的概念。从概念上讲，数据点是任意复杂的记录，代表应可视化的一条数据。数据点不仅是平面中的坐标或需要可视化的数值。相反，它们代表需要可视化的数据的基本单位。

请考虑以下示例：在一个实验中，我们沿着道路驾驶汽车并安装了不同的测量仪器。我们测量汽车的位置，时间，速度，汽车前进的方向，加速度以及可能的其他值。数据点将由一个记录组成，该记录包括一个时间戳记以及汽车的当前位置（大概两个或三个数字），速度矢量（另一个两个或三个数字），加速度（另一个两个或三个数字）以及也许是当前实验的标签文字。

数据点应该是“信息丰富”的。它们甚至可能包含比实际的可视化图形更多的信息。呈现管道的工作是挑选与一种特定数据可视化图形相关的信息——同一数据的另一种可视化图形可能会选择数据点的不同方面，从而有望允许对数据进行一种新的洞察。

从技术上讲，数据点没有特殊的数据结构。而是当调用名为 `\pgfdatapoint` 的特殊宏时，当前作用域中所有的设置键带有 `/data point/` 前缀的“全体”构成数据点。这既是一种非常通用的方法，又非常快，因为不需要创建额外的数据结构。

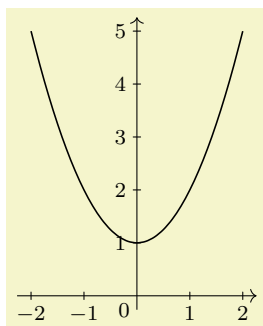
14.2 概念：可视化管道

可视化管道是对要可视化的数据执行的一系列操作。数据以一长串复杂数据点的形式呈现给可视化管道。可视化管道对该数据点流进行了多次传递。在被称为调查阶段的在第一阶段期间，将收集有关数据点的信息，例如最小值和最大值，这对于将数据自动拟合到给定区域很有用。在数据的主要传递过程中，称为可视化阶段，实际上以线或点的形式可视化数据点。

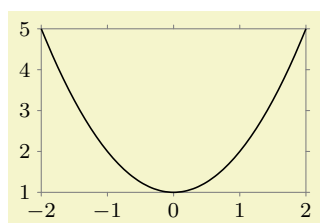
像数据点一样，可视化管道不对所需的可视化做出任何假设。确实，甚至可以使用它来生成纯文本表。通过大量使用对象和信号来实现这种灵活性：数据可视化开始时，会初始化许多信号（有关信号的介绍，请参见??节）。然后，创建了许多监听这些信号的“可视化对象”。这些对象都涉及数据点的处理。例如，`interval mapper` 对象的工作是将数据点的一个属性（例如汽车的速度）映射到另一个属性，例如图表的 y 轴。对于每个数据点，按一定顺序发出不同的信号，并且现在不同的可视化对象都有机会为实际的可视化准备数据点。继续上面的示例，可能还有第二个 `interval mapper`，它采用计算出的 y 位置并求其对数，因为请求了对数图。然后是另一个映射器，这次是 `polar mapper`，可用于将所有数据点映射到极坐标。此后，`plot mark visualizer` 可在实际上计算出的位置绘制一些东西。

呈现管道背后的整个想法是，理想地，只需在可视化管道中添加一个或两个新对象，就可以实现新型的数据可视化。此外，不同类型的图形可以通过这种方式以新颖的方式结合起来，这通常是很难做到的。例如，可视化管道可以轻松创建极坐标半对数箱形图。乍一看，这种新的图可能看起来很无聊，但数据可视化就是要从尽可能多的不同角度来洞察数据。

自然，为呈现管道创建新的类和对象并不是一件简单的事情，因此大多数用户将只使用现有的类，这样应该尽可能灵活。但是，即使只打算使用现有的类，正确地设置管道仍然需要技巧，因为顺序显然很重要，而且轴和刻度之类的东西需要配置和处理。因此，前端库提供了预先配置的渲染管道，这样我们就可以简单地说，一个数据可视化应该看起来像一个带有 `school book axes` 或 `scientific axes` 的 `line plot`，它选择一个特定的可视化管道，适合这类图：



```
\usetikzlibrary {datavisualization.formats.functions}
\begin{tikzpicture}[scale=.7]
\datavisualization[school book axes,visualize as smooth line]
data[format=function]{
varx: interval[-2:2];
funcy=\valuex*\valuex+1;
};
\end{tikzpicture}
```



```
\usetikzlibrary {datavisualization.formats.functions}
\begin{tikzpicture}[scale=.7]
\datavisualization[scientific axes,visualize as smooth line]
data[format=function]{
varx: interval[-2:2];
funcy=\valuex*\valuex+1;
};
\end{tikzpicture}
```

我们仍然必须配置这样的绘图（选择样式和主题，并指定应使用数据点的哪些属性），但总体而言，绘图的指定相当简单。