

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,
      l-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}] l-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.

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 7, 2020

目录

1	简介	6
1.1	TikZ 系统下的层级	6
1.2	与其它图形宏包的比较	7
1.3	工具包	7
1.4	如何阅读本手册	8
1.5	作者与致谢	8
1.6	获取帮助	8
I	教程和准则	9
2	教程：卡尔学生的图片	10
2.1	问题描述	10
2.2	配置环境	10
2.2.1	在 L ^A T _E X 中配置环境	10
2.2.2	在 Plain T _E X 中配置环境	11
2.2.3	在 ConT _E Xt 中配置环境	12
2.3	直线路径的创建	12
2.4	曲线路径的创建	12
2.5	圆路径的创建	13
2.6	矩形路径的创建	14
2.7	网格路径的创建	14
2.8	增加一点风格	15
2.9	绘图选项	15
2.10	弧线路径的创建	16
2.11	剪切路径	17
2.12	抛物线和正弦线路径构造	17
2.13	填充和绘图	18
2.14	阴影	18



*本文档的编辑。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。

2.15	指定坐标	19
2.16	相交路径	20
2.17	添加箭头	21
2.18	作用域	22
2.19	坐标变换	22
2.20	重复工作: For 循环	23
2.21	添加文本	24
2.22	Pics: 再访 Angle	28
3	教程: 哈根的 Petri 网	29
3.1	问题陈述	29
3.2	配置环境	29
3.2.1	在 L ^A T _E X 中配置环境	29
3.2.2	在 Plain T _E X 中配置环境	30
3.2.3	在 ConT _E Xt 中配置环境	30
3.3	节点简介	30
3.4	使用 At 语法放置节点	31
3.5	使用样式	31
3.6	节点尺寸	32
3.7	节点命名	32
3.8	使用相对位置放置节点	33
3.9	在节点旁添加标签	33
3.10	连接节点	35
3.11	在线旁添加标签	37
3.12	添加隐藏线和多行文本	38
3.13	使用图层: 背景矩形	39
3.14	完整的代码	39
4	教程: 欧几里得的琥珀版本的几何原本	42
4.1	书 I, 命题 I	42
4.1.1	配置环境	42
4.1.2	线段 AB	43
4.1.3	以点 A 为圆心的圆	43
4.1.4	圆的交集	45
4.1.5	完整代码	46
4.2	书 I, 命题 II	47
4.2.1	使用分割运算进行点 D 的构造	48
4.2.2	线段与圆相交	49
4.2.3	完整代码	50
5	教程: 简单的图表	52
5.1	为节点设置样式	52
5.2	使用定位选项对齐节点	54
5.3	使用矩阵对齐节点	56
5.4	作为图形的图解	58
5.4.1	连接已经定位的节点	58
5.4.2	使用 Graph 命令创建节点	59

6	教程：约翰内斯的演讲图	63
6.1	问题描述	63
6.2	树状图简介	63
6.3	创建讲义图	66
6.4	添加讲义注释	71
6.5	添加背景	73
6.6	添加日历	74
6.7	完整代码	75
7	绘图准则	80
7.1	规划创建的图形所需要的时间	80
7.2	创建图形的工作流程	80
7.3	将图形与正文紧密结合	81
7.4	图形和文本之间的一致性	81
7.5	图形中的标签	82
7.6	图表	82
7.7	注意力和注意力分散	85

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“TikZist *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。由于每个附加命令使将 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：

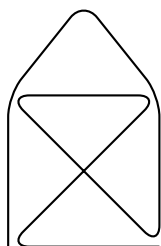
1. 阅读手册，至少是与您的问题有关的部分。
2. 如果那不能解决问题，请尝试查看 GitHub 上 PGF 和 TikZ 的开发页面（请参见本文档的标题）。也许有人已经报告了类似的问题，并且有人找到了解决方案。
3. 在网站上，您会找到许多论坛以获得帮助。在这里，您可以编写帮助讨论，文件错误报告，加入邮件列表等等。
4. 在提交错误报告（尤其是有关安装的错误报告）之前，请确保这确实是一个错误。特别要看一下你用 TeX 编译生成的 .log 文件。这个 .log 文件应显示所有正确的文件均已从正确的目录中加载。通过查看 .log 文件，几乎可以解决所有安装问题。
5. 作为最后的手段您可以尝试给我（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的准则。



```
\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

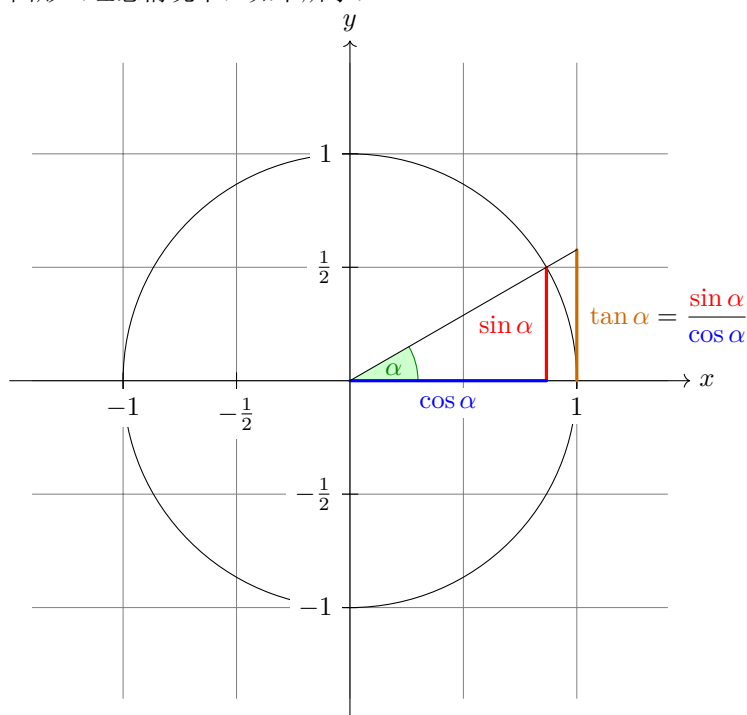
本教程适用于 TikZ 的新用户。它没有详尽介绍 TikZ 的所有功能，只是您可能立即使用的那些功能。

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

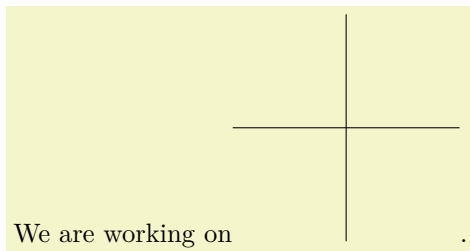
卡尔作为一个 \LaTeX 用户，他的文件如下：

```

\documentclass{article} % say
\usepackage{tikz}
\begin{document}
We are working on
\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：



```

We are working on
\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 TeX 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 `LaTeX` 用户，而使用 `Plain TeX`，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：

```

%% Plain TeX file
\input tikz.tex
\baselineskip=12pt
\hsize=6.3truein
\vsize=8.7truein
We are working on
\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:

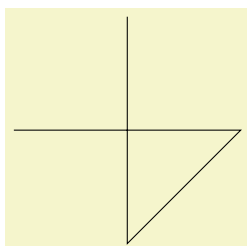
```
%% ConTeXt file
\usemodule[tikz]

\starttext
  We are working on
  \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 直线路径的创建

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:



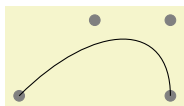
```
\tikz \draw (-1.5,0) -- (1.5,0) -- (0,-1.5) -- (0,1.5);
```

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 曲线路径的创建

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。

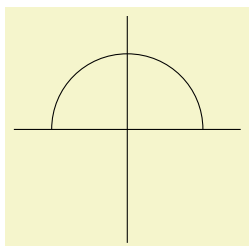
这是一个示例(为清楚起见添加了控制点):



```
\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。

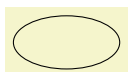
2.5 圆路径的创建

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];
```

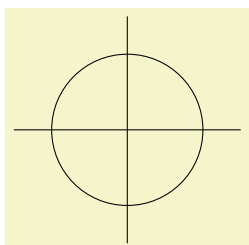
您也可以使用 `ellipse` 操作添加椭圆路径。您可以指定两个，而不是单个半径：



```
\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 `rotate`，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：

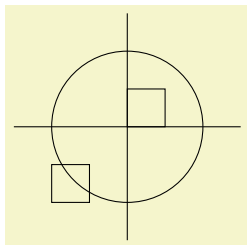


```
\begin{tikzpicture}
\draw (-1.5,0) -- (1.5,0);
\draw (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\end{tikzpicture}
```

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}
```

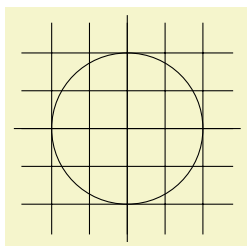
虽然这在其他情况下可能很好，但这并没有真正解决卡尔的问题：首先，我们需要大量的矩形，然后边界没有“封闭”。

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

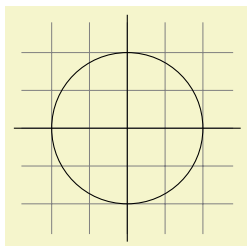
`grid` 路径操作向当前路径添加一个网格。它将添加组成网格的线，填充矩形，矩形的一个角点是当前点，另一个角点是紧跟 `grid` 操作的点。例如，代码 `\tikz \draw[step=2pt] (0,0) grid (10pt,10pt);` 生成 。请注意，可以使用 `\draw` 的可选参数来指定网格宽度（还有 `xstep` 和 `ystep` 来分别定义步长）。卡尔很快就会了解到，使用这些选项可以影响到很多事情。

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:

```
help lines/.style={color=blue!50,very thin}
```

这种“样式设置器”的作用是, 在当前作用域或环境中 `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

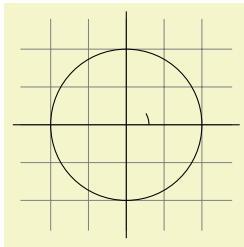
卡尔想知道还有什么其他选项可以影响路径的绘制。他已经看到可以使用 `color=⟨color⟩` 选项来设置线条的颜色。选项 `draw=⟨color⟩` 做的几乎是一样的, 只是它只设置了线条的颜色, 可以使用不同的颜色填充(卡尔填充圆弧的角度会用到这个选项)。

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。

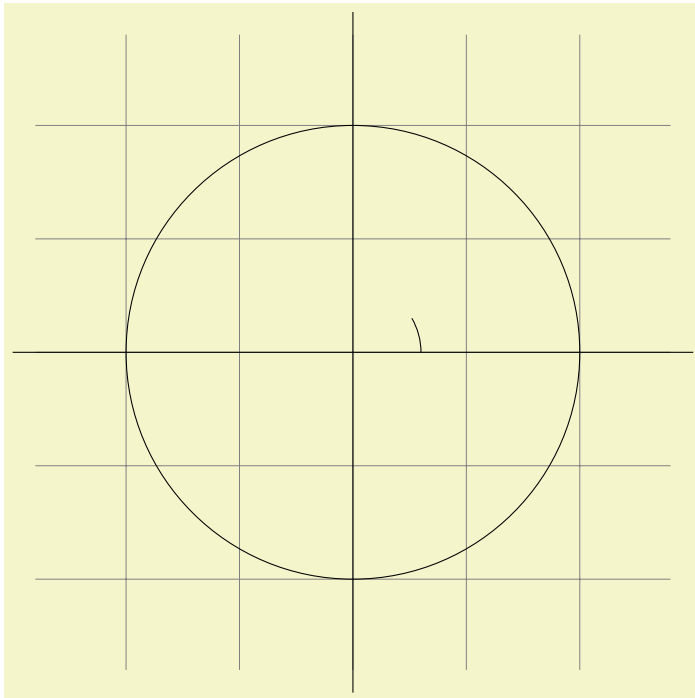
2.10 弧线路径的创建

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”。



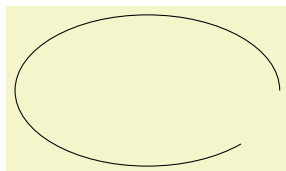
```
\begin{tikzpicture}
  \draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
  \draw (-1.5,0) -- (1.5,0);
  \draw (0,-1.5) -- (0,1.5);
  \draw (0,0) circle [radius=1cm];
  \draw (3mm,0mm) arc [start angle=0, end angle=30, radius=3mm];
\end{tikzpicture}
```

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}
```

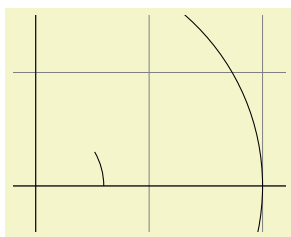
对于圆而言，您可以指定“两个”半径以获得椭圆弧。



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

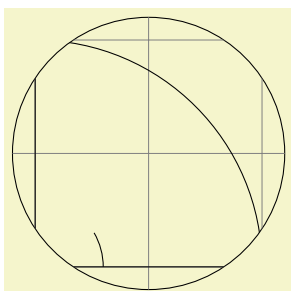
2.11 剪切路径

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) parabola bend (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.

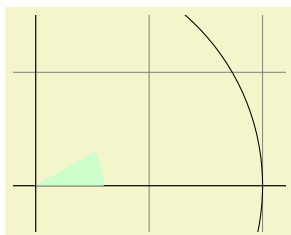
```
A sine \tikz \draw[x=1ex,y=1ex] (0,0) sin (1.57,1); curve.
```



```
\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：

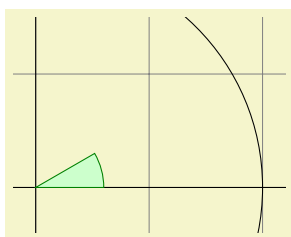
```
\fill[green!20!white] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
```

`--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); % make bounding box higher
\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。



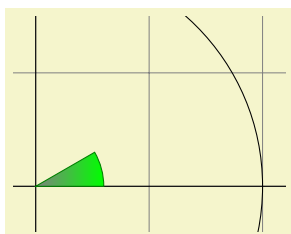
```
\tikz \shade (0,0) rectangle (2,1) (3,0.5) circle (.5cm);
```

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。

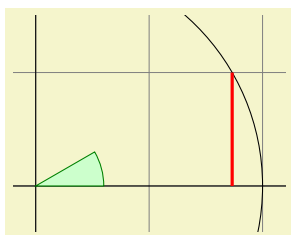
2.15 指定坐标

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 $(\cos 30^\circ, \sin 30^\circ)$ ”。

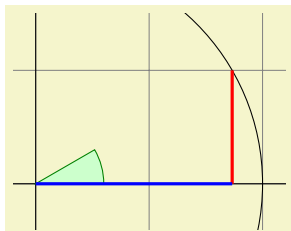
您可以在坐标前添加一个或两个 $+$ 符号，如 $+(0\text{cm}, 1\text{cm})$ 或 $++(2\text{cm}, 0\text{cm})$ 。这些坐标的解释是不同的：第一种形式表示“上一指定位置上方 1cm ”，第二种形式表示“上一指定位置右方 2cm ，使其成为新的指定位置”。例如，我们可以画出正弦线如下：



```
\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”。

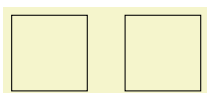
接下来，我们来画余弦线。一种方法是 $(30:1\text{cm} |- 0,0) -- (0,0)$ 。另一种方法是这样的：我们从正弦结束的地方“继续”：



```
\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}
```

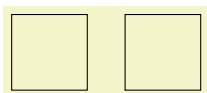
注意，在 $(30:1\text{cm})$ 和 $++(0,-0.5)$ 之间不存在 $--$ 。具体来说，这个路径解释如下：“首先， $(30:1\text{cm})$ 告诉我将笔移动到 $(\cos 30^\circ, 1/2)$ 。接下来，指定了另一个坐标点，所以我移动我的笔，但不绘制任何东西。这个新点比上一个位置下降了半个单位，因此它在 $(\cos 30^\circ, 0)$ 。最后，我把笔移到原点，但这次画一些东西（因为用到了 $--$ ）。”

要理解 $+$ 和 $++$ 的区别，请考虑下面的例子：



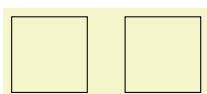
```
\begin{tikzpicture}
\def\rectanglepath{-- ++(1cm,0cm) -- ++(0cm,1cm) -- ++(-1cm,0cm) -- cycle}
\draw (0,0) \rectanglepath;
\draw (1.5,0) \rectanglepath;
\end{tikzpicture}
```

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。

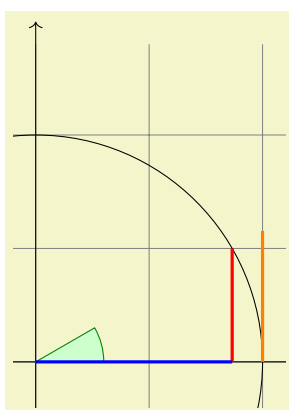
```
\path [name path=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm); % a bit longer, so that there is an intersection

% (add '\usetikzlibrary{intersections}' after loading tikz in the preamble)
\draw [name intersections={of=upward line and sloped line, 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：

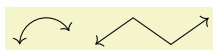


```
\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
\clip (-0.1,-0.2) rectangle (1.1,1.51);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\draw[->] (-1.5,0) -- (1.5,0);
\draw[->] (0,-1.5) -- (0,1.5);
\draw (0,0) circle [radius=1cm];
\filldraw[fill=green!20,draw=green!50!black] (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=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm);
\draw [name intersections={of=upward line and sloped line, 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： \rightarrow 。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 [<<-,>=Stealth] (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。

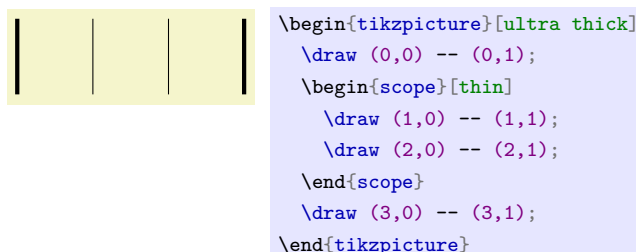
除了 **Stealth** 卡尔还可以选择其他几种预定义的箭头，请参见??节。此外，如果需要新的箭头类型，他可以自己定义箭头类型。

2.18 作用域

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：



作用域还有另一个有趣的作用：裁剪区域的任何更改都是该作用域的局部更改。因此，如果您用 `\clip` 在某个作用域内的某个地方，`\clip` 命令的效果将在作用域的末尾结束。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

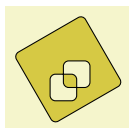
当您指定 (1cm,1cm) 之类的坐标时，该坐标在页面上的什么位置？为了确定位置，TikZ，TeX 和 PDF 或 PostScript 都会使用一种确定的坐标变换，以确定其在页面上的最终位置。

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 \TeX 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

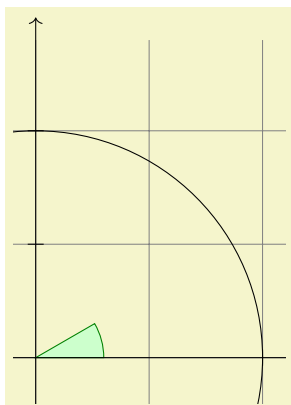
卡尔的下一个目标是在 -1 , $-1/2$, $1/2$ 和 1 的位置的坐标轴上添加小刻度。为此, 最好使用某种 ‘循环’, 特别是因为他希望在这些位置上做同样的事情。可以使用不同的包来完成这项工作。 \LaTeX 对此有自己的内部命令, `psstricks` 带有强大的 `\multido` 命令。所有这些都可以与 `TikZ` 一起使用, 所以如果您熟悉它们, 可以随意使用它们。`TikZ` 引入了另一个命令 `\foreach`, 我引入这个命令是因为我永远记不住其他宏包的语法。`\foreach` 命令定义在宏包 `pgffor` 中, 可以独立于 `TikZ` 使用, 但 `TikZ` 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 \x in {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 \x in {-1cm,-0.5cm,1cm}
    \draw (\x,-1pt) -- (\x,1pt);
  \foreach \y in {-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 \x in {-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 \x in {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 \x in {-1,-0.5,...,1}
  \draw (\x cm,-1pt) -- (\x cm,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

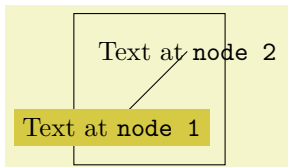
```
\begin{tikzpicture}
  \foreach \x in {1,2,...,5,7,8,...,12}
  \foreach \y in {1,...,5}
  {
    \draw (\x,\y) +(-.5,-.5) rectangle ++(.5,.5);
    \draw (\x,\y) node{\x,\y};
  }
\end{tikzpicture}
```

\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

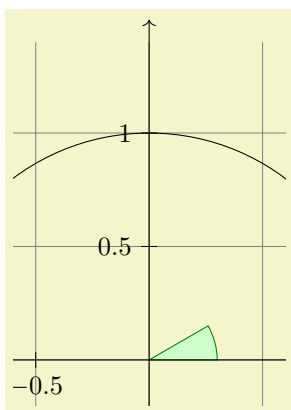
到目前为止，卡尔对图片非常满意。但是，最重要的部分，即文字标签，仍然缺失！

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]
    {Text at \verb!node 1!}
    -- (1.5,1.5) node {Text at \verb!node 2!};
\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：



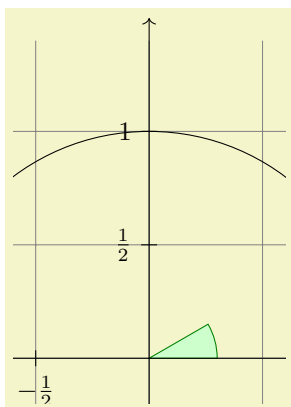
```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[>-] (-1.5,0) -- (1.5,0); \draw[>-] (0,-1.5)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

\foreach \x in {-1,-0.5,1}
  \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {\x};
\foreach \y in {-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：

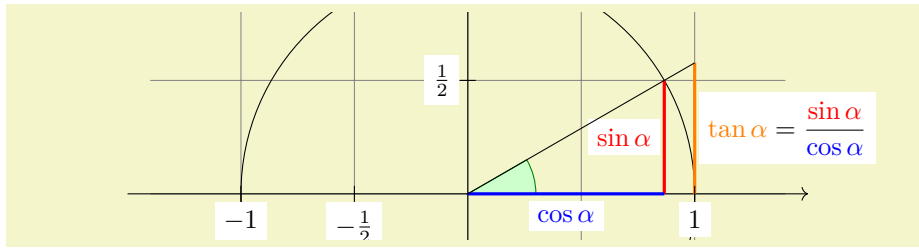


```
\begin{tikzpicture}[scale=3]
\clip (-0.6,-0.2) rectangle (0.6,1.51);
\draw[step=.5cm,help lines] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
  arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[>-] (-1.5,0) -- (1.5,0); \draw[>-] (0,-1.5)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

\foreach \x/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
  \draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north] {\xtext};
\foreach \y/\ytext in {-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：



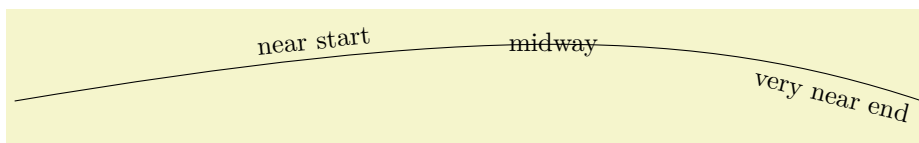
```
\usetikzlibrary {intersections}
\begin{tikzpicture}[scale=3]
\clip (-2,-0.2) rectangle (2,0.8);
\draw[step=.5cm,gray,very thin] (-1.4,-1.4) grid (1.4,1.4);
\filldraw[fill=green!20,draw=green!50!black] (0,0) -- (3mm,0mm)
arc [start angle=0, end angle=30, radius=3mm] -- cycle;
\draw[->] (-1.5,0) --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

\draw[very thick,red]
(30:1cm) -- node[left=1pt,fill=white] {\sin \alpha} (30:1cm |- x axis);
\draw[very thick,blue]
(30:1cm |- x axis) -- node[below=2pt,fill=white] {\cos \alpha} (0,0);
\path [name path=upward line] (1,0) -- (1,1);
\path [name path=sloped line] (0,0) -- (30:1.5cm);
\draw [name intersections={of=upward line and sloped line, 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/\xtext in {-1, -0.5/-\frac{1}{2}, 1}
\draw (\x cm,1pt) -- (\x cm,-1pt) node[anchor=north,fill=white] {\xtext};
\foreach \y/\ytext in {-1, -0.5/-\frac{1}{2}, 0.5/\frac{1}{2}, 1}
\draw (1pt,\y cm) -- (-1pt,\y cm) node[anchor=east,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] {near start}
node {midway}
node[very near end,sloped,below] {very near end} (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}

  % The graphic
  \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(x axis);
    \draw[>-] (0,-1.5) -- (0,1.5) node[above] {$y$} coordinate(y axis);

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

    \foreach \y/\ytext in {-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 |- x axis);

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

  \path [name path=upward line] (1,0) -- (1,1);
  \path [name path=sloped line] (0,0) -- (30:1.5cm);
  \draw [name intersections={of=upward line and sloped line, 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 $30^\circ$ in the
      example ($\pi/6$ in radians). The {\color{sincolor}sine} of
      $\alpha$, which is the height of the red line, is
      \[
      {\color{sincolor} \sin \alpha} = 1/2.
      \]
      By the Theorem of Pythagoras ...
    };
\end{tikzpicture}

```

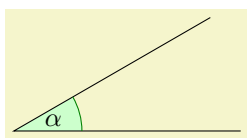
2.22 Pics: 再访 Angle

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。

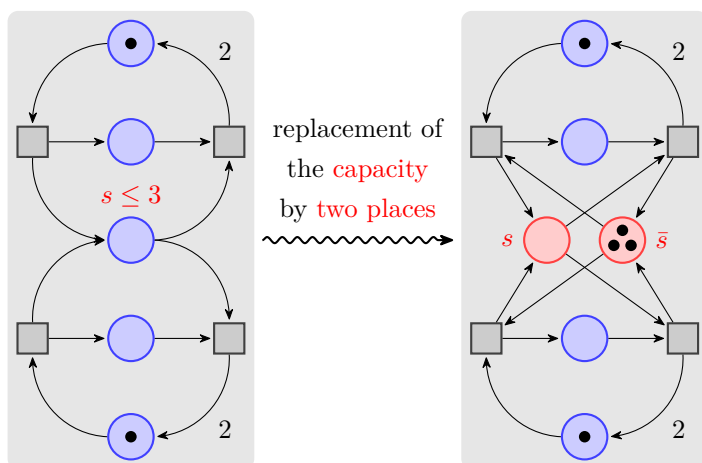
3 教程：哈根的 Petri 网

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：

```
%% Plain TeX file
\input tikz.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：

```
%% ConTeXt file
\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

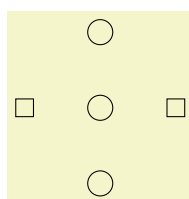
原则上，我们已经知道如何创建哈根想要的图形（也许除了蛇形线之外，我们将介绍到该图形）：我们从大的浅灰色矩形开始，然后添加许多圆形和小矩形以及一些箭头。

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。

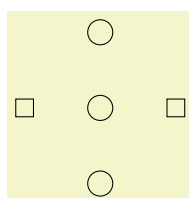
在上一个教程中，我们了解到 `node` 将在最后一个坐标处添加一段文本。因此，五个节点中的每个节点的文本都将添加在不同的位置。在上面的代码中，该文本为空（由于 `{}` 为空）。那么，为什么我们什么都看不到？答案是 `node` 操作的 `draw` 选项：它使“形状周围的文本”被绘制。

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

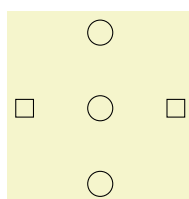
哈根现在了解 `node` 操作将节点添加到路径，但是使用 `\path` 操作创建路径似乎有点愚蠢。包括许多多余的 `move-to` 操作，仅用于放置节点。他很高兴得知有一些方法可以以更明智的方式添加节点。

首先，`node` 操作允许添加 `at (<coordinate>)`，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：



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

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：

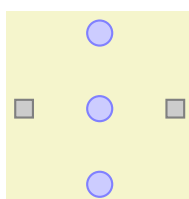


```
\begin{tikzpicture}
  \node at ( 0,2) [circle,draw] {};
  \node at ( 0,1) [circle,draw] {};
  \node at ( 0,0) [circle,draw] {};
  \node at ( 1,1) [rectangle,draw] {};
  \node at (-1,1) [rectangle,draw] {};
\end{tikzpicture}
```

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

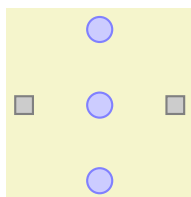
出于冒险的感觉，哈根试图让节点看起来更好。在最终的图片中，圆和矩形需要用不同的颜色填充，从而产生以下代码：



```
\begin{tikzpicture}[thick]
  \node at ( 0,2) [circle,draw=blue!50,fill=blue!20] {};
  \node at ( 0,1) [circle,draw=blue!50,fill=blue!20] {};
  \node at ( 0,0) [circle,draw=blue!50,fill=blue!20] {};
  \node at ( 1,1) [rectangle,draw=black!50,fill=black!20] {};
  \node at (-1,1) [rectangle,draw=black!50,fill=black!20] {};
\end{tikzpicture}
```


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。

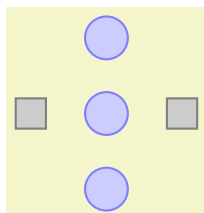
为了解决此问题，哈根使用了样式。他为库所定义了一种样式，为变迁定义了另一种样式：



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

3.6 节点尺寸

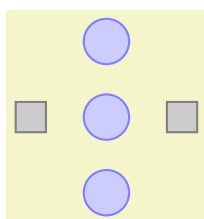
在哈根开始命名和连接节点之前，让我们首先确定节点具有最终外观。它们还有点太小。的确，哈根想知道为什么它们根本没有任何尺寸，毕竟文本是空的。原因是 TikZ 会在文本周围自动添加一些空格。空格的数量是使用选项 `inner sep` 设置的。因此，为了增加节点的大小，哈根可以这样写：



```
\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}]
\node at ( 0,2) [place] {};
\node at ( 0,1) [place] {};
\node at ( 0,0) [place] {};
\node at ( 1,1) [transition] {};
\node at (-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。



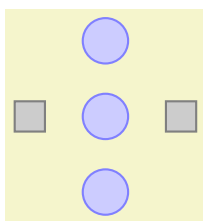
```
\begin{tikzpicture}
[place/.style={circle,draw=blue!50,fill=blue!20,thick,
inner sep=0pt,minimum size=6mm},
transition/.style={rectangle,draw=black!50,fill=black!20,thick,
inner sep=0pt,minimum size=4mm}]
\node at ( 0,2) [place] {};
\node at ( 0,1) [place] {};
\node at ( 0,0) [place] {};
\node at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

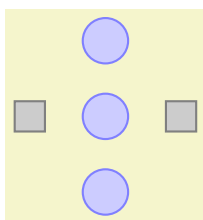
命名节点有两种方法。第一种是使用 `name=` 选项。第二种方法是在 `node` 操作之后的括号中写入节点的名称。哈根认为第二种方法似乎很奇怪，但是他很快就会改变看法。



```
% ... set up styles
\begin{tikzpicture}
  \node (waiting 1)      at ( 0,2) [place] {};
  \node (critical 1)     at ( 0,1) [place] {};
  \node (semaphore)      at ( 0,0) [place] {};
  \node (leave critical) at ( 1,1) [transition] {};
  \node (enter critical) 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。

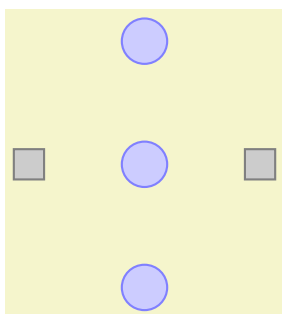
`node` 操作的语法在节点名称、`at` 和必须提供的选项的顺序方面是相当自由的。实际上，你甚至可以在 `node` 和花括号文本之间有多个选项块，它们会累积。你可以随意地重新排列它们，也许下面的方法更可取：



```
\begin{tikzpicture}
  \node[place]      (waiting 1)      at ( 0,2) {};
  \node[place]      (critical 1)     at ( 0,1) {};
  \node[place]      (semaphore)      at ( 0,0) {};
  \node[transition] (leave critical) at ( 1,1) {};
  \node[transition] (enter critical) 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] (leave critical) [right=of critical]         {};
  \node[transition] (enter critical) [left=of critical]          {};
\end{tikzpicture}
```

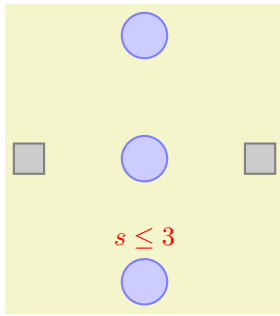
加载 `positioning` 库后，当 `belloe` 选项后面跟着 `of` 时，则将节点的位置移动到指定方向，使其置于与指定节点的距离为 `node distance` 的位置。`node distance` 要么是节点中心之间的距离（当 `on grid` 选项被设置为 `true` 时），要么是边界之间的距离（当 `on grid` 选项被设置为 `false` 时，这是默认值）。

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。

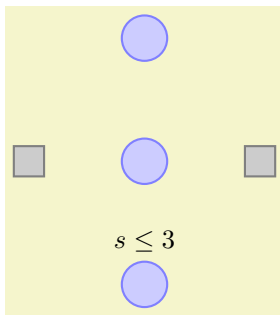


```
\usetikzlibrary {positioning}
\begin{tikzpicture}
  \node[place]      (waiting)                {};
  \node[place]      (critical)      [below=of waiting] {};
  \node[place]      (semaphore)      [below=of critical] {};
  \node[transition] (leave critical) [right=of critical] {};
  \node[transition]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

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

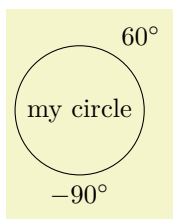
这是一种将“始终有效”的通用方法。

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:$s\le 3$`。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`。



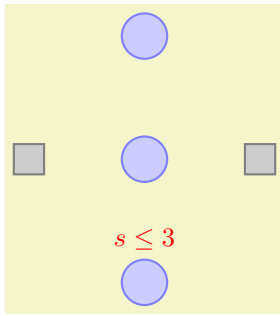
```
\usetikzlibrary {positioning}
\begin{tikzpicture}
  \node[place]      (waiting)                {};
  \node[place]      (critical)      [below=of waiting] {};
  \node[place]      (semaphore)      [below=of critical,
                                         label=above:$s\le 3$] {};
  \node[transition] (leave critical) [right=of critical] {};
  \node[transition] (enter critical) [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:$60^\circ\!\circ$,label=below:$-90^\circ\!\circ$] {my circle};
```

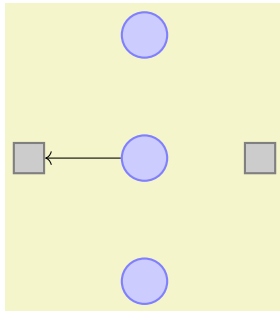
哈根对 `label` 选项并不完全满意, 因为标签不是红色的。为了实现这一点, 他有两个选择: 首先, 他可以重新定义 `every label` 的风格。其次, 他可以向标签的节点中添加选项。这些选项是按照 `label=` 给出的, 所以他在上面写 `label=[red]above:$s\le 3$`。然而, 这并不能完全工作, 因为 `TEX` 认为 `]` 结束了 `semaphore` 节点的整个选项列表。因此, 哈根必须添加括号这样使用 `label={ [red]above:$s\le 3$ }`。由于这看起来有点丑, 哈根决定重新定义 `every label` 风格。



```
\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] (leave critical) [right=of critical] {};
\node[transition] (enter critical) [left=of critical]  {};
\end{tikzpicture}
```

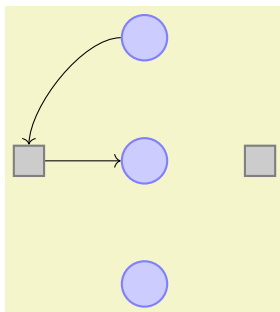
3.10 连接节点

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] (leave critical) [right=of critical] {};
\node[transition] (enter critical) [left=of critical]  {};
\draw [->] (critical.west) -- (enter critical.east);
\end{tikzpicture}
```

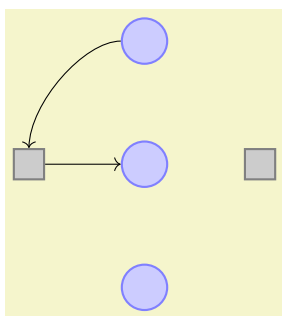
接下来，让我们处理从 `waiting` 到 `enter critical` 的曲线。可以使用曲线和控件来指定：



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

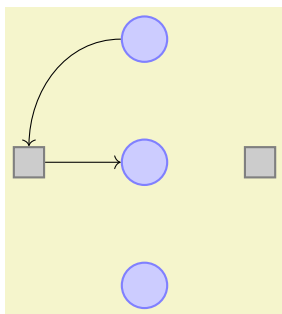
因此，让我们开始改进边的代码。首先，哈根可以省略锚点：



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

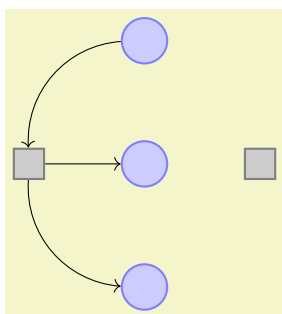
哈根有点惊讶，这是可行的。毕竟，TikZ 怎么知道从 `enter critical` 到 `critical` 的直线实际上应该从边界开始呢？每当 TikZ 遇到作为“坐标”的整个节点名时，它就会尝试“聪明地”为该节点选择锚点。根据接下来发生的情况，TikZ 将选择位于节点边界上的锚点，该锚点位于到下一个坐标或控制点的直线上。确切的规则有点复杂，但选择的点通常是正确的，如果不是这样，哈根仍然可以手工指定所需的锚点。

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：



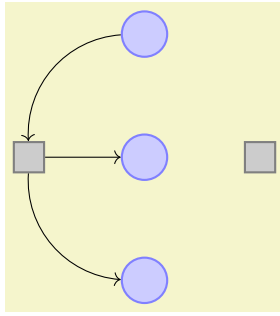
```
\usetikzlibrary {positioning}
\begin{tikzpicture}
  \node[place]      (waiting)                {};
  \node[place]      (critical)      [below=of waiting] {};
  \node[place]      (semaphore)      [below=of critical] {};
  \node[transition] (leave critical) [right=of critical] {};
  \node[transition] (enter critical) [left=of critical]  {};
  \draw [->] (enter critical) to (critical);
  \draw [->] (waiting) to [out=180,in=90] (enter critical);
\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] (leave critical) [right=of critical] {};
  \node[transition] (enter critical) [left=of critical]  {};
  \draw [->] (enter critical) to (critical);
  \draw [->] (waiting) to [bend right=45] (enter critical);
  \draw [->] (enter critical) to [bend right=45] (semaphore);
\end{tikzpicture}
```

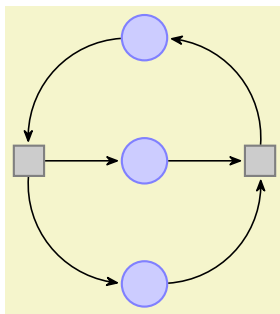
现在是时候让哈根学习指定边的另一种方法了：使用 `edge` 路径操作。此操作与 `to` 非常相似。操作，但是有一个重要的区别：与节点一样，`edge` 操作生成的边不是主路径的一部分，而是仅在以后添加。这看似不太重要，但是会带来一些不错的效果。例如，每条边可以有自己的箭头提示和颜色，依此类推，而且所有边都可以在同一路径上给出。这使哈根可以编写以下内容：



```
\usetikzlibrary {positioning}
\begin{tikzpicture}
  \node[place]      (waiting)                {};
  \node[place]      (critical)      [below=of waiting] {};
  \node[place]      (semaphore)     [below=of critical] {};
  \node[transition] (leave critical) [right=of critical] {};
  \node[transition] (enter critical) [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：



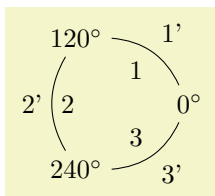
```
\usetikzlibrary {arrows.meta,positioning}
% Styles place and transition as before
\begin{tikzpicture}
  [bend angle=45,
  pre/.style={<-,shorten <=1pt,>={Stealth[round]},semithick},
  post/.style={->,shorten >=1pt,>={Stealth[round]},semithick}]

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

  \node[transition] (leave critical) [right=of critical] {}
  edge [pre]          (critical)
  edge [post,bend right] (waiting)
  edge [pre, bend left] (semaphore);
  \node[transition] (enter critical) [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

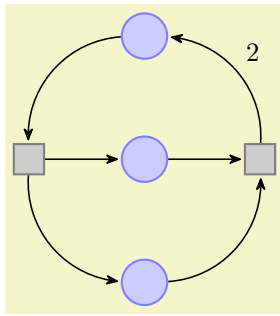
哈根接下来需要添加的是弧线上的“2”。为此，哈根可以使用 TikZ 的自动节点放置：通过添加选项 `auto`，TikZ 可以将节点定位在曲线和线上，使得它们不在曲线上而是位于曲线旁边。添加 `swap` 将对该线的标签进行镜像。这是一个一般示例：



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

  \draw (a) to node {1} node [swap] {1'} (b)
        (b) to node {2} node [swap] {2'} (c)
        (c) to node {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。



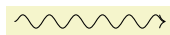
```
\usetikzlibrary {arrows.meta,positioning}
% Styles as before
\begin{tikzpicture}[bend angle=45]
  \node[place]      (waiting)                {};
  \node[place]      (critical)      [below=of waiting] {};
  \node[place]      (semaphore)     [below=of critical] {};

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

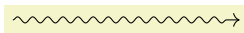
利用节点机制，哈根现在可以轻松创建两个 Petri 网络。他不确定的是如何在网之间创建蛇形线。

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。



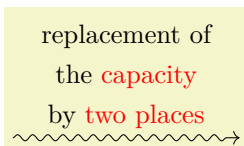
```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
  \draw [->,decorate,decoration=snake] (0,0) -- (2,0);
\end{tikzpicture}
```

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,segment length=2mm,post length=1mm}]
    (0,0) -- (3,0);
\end{tikzpicture}
```

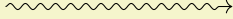
现在，哈根需要在蛇形上方添加文本。该文本是多行文本，因此具有一定的挑战性。哈根对此有两种选择：首先，他可以指定 `align=center`。然后使用 `\\` 命令以在所需位置强制换行。



```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
  \draw [->,decorate,
    decoration={snake,amplitude=.4mm,segment length=2mm,post length=1mm}]
    (0,0) -- (3,0)
    node [above,align=center,midway]
    {
      replacement of\\
      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**



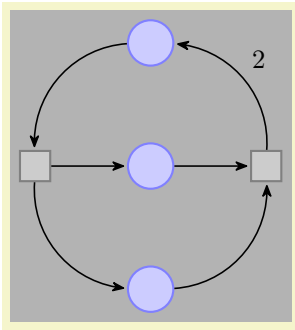
```
\usetikzlibrary {decorations.pathmorphing}
\begin{tikzpicture}
\draw [->,decorate,
decoration={snake,amplitude=.4mm,segment length=2mm,post length=1mm}]
(0,0) -- (3,0)
node [above,text width=3cm,align=center,midway]
{
replacement of the \textcolor{red}{capacity} by
\textcolor{red}{two places}
};
\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}
% Styles as before
\begin{tikzpicture}[bend angle=45]
\node[place] (waiting) {};
\node[place] (critical) [below=of waiting] {};
\node[place] (semaphore) [below=of critical] {};

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

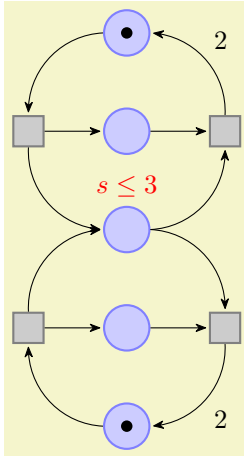
首先，哈根需要较少的样式定义，但他仍然需要指定库所和变迁的颜色。


```

\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：



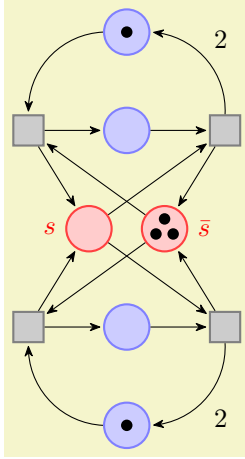
```

\usetikzlibrary {arrows.meta,petri,positioning}

\node [place,tokens=1] (w1) {}
\node [place] (c1) [below=of w1] {}
\node [place] (s) [below=of c1,label=above:$s\le 3$] {}
\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=2] (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,segment length=2mm,
pre=moveto,pre length=1mm,post length=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

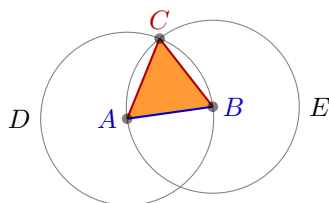
在第三个教程中，我们将了解如何使用 TikZ 绘制几何构造。

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：

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

```
% For plain TeX:
\input tikz.tex
\usetikzlibrary{calc,intersections,through,backgrounds}
```

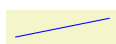
```
% For ConTeXt:
\usemodule[tikz]
\usetikzlibrary[calc,intersections,through,backgrounds]
```

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。??

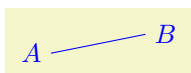
因此，欧几里得从使用 `\coordinate` 命令定义两个坐标开始：



```
\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

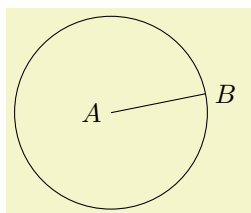
第一个棘手的结构是以点 A 为圆心的圆。稍后我们将看到如何以非常简单的方式执行此操作，但首先让我们以一种更“困难”方式执行此操作。

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}
```

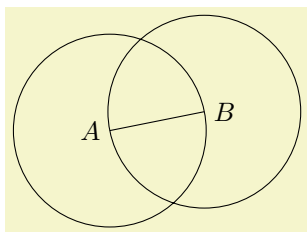
let 操作中的每个赋值都以 $\backslash p$ 开始，通常后跟一个 \langle 数字 \rangle ，然后是等号和坐标。坐标将被计算，结果被存储在内部。之后你可以使用下面的表达方式：

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。

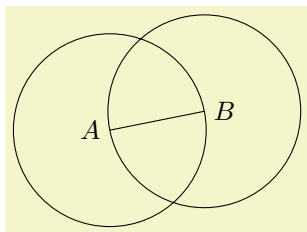
接下来，我们想要同时画两个圆。每个圆半径是 $veclen(\backslash x1, \backslash y1)$ 。只计算一次半径似乎很自然。为此，我们还可以使用 *let* 操作：他使用 $\backslash n2 = \dots$ 代替 $\backslash p1 = \dots$ 。这里，“n”代表“数字”（而“p”代表“点”）。数字的赋值后面应该跟一个用花括号括起来的数字。



```
\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);

  \draw let \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：

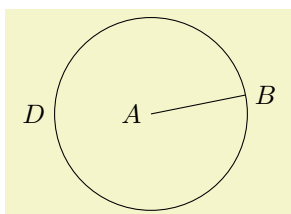


```
\usetikzlibrary {calc}
\begin{tikzpicture}
  \coordinate [label=left:$A$] (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \draw let \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。



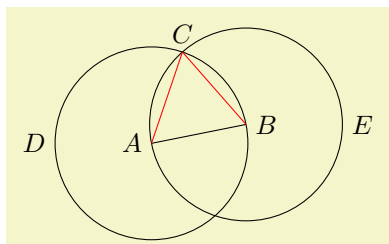
```
\usetikzlibrary {through}
\begin{tikzpicture}
  \coordinate [label=left:$A$] (A) at (0,0);
  \coordinate [label=right:$B$] (B) at (1.25,0.25);
  \draw (A) -- (B);

  \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) {};

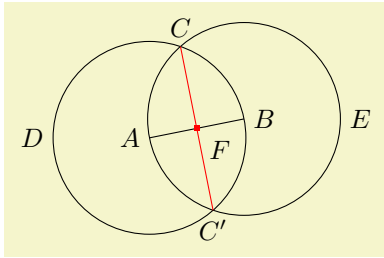
  % Name the coordinates, but do not draw anything:
  \path [name intersections={of=D and E}];

  \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}

```

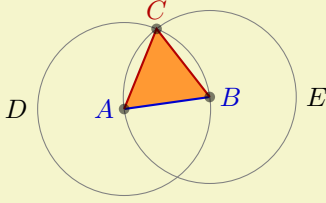
4.1.5 完整代码

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$}} \def\D{$D$}
\def\E{$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=D and E,by={ [label=above:\C]C}}];

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

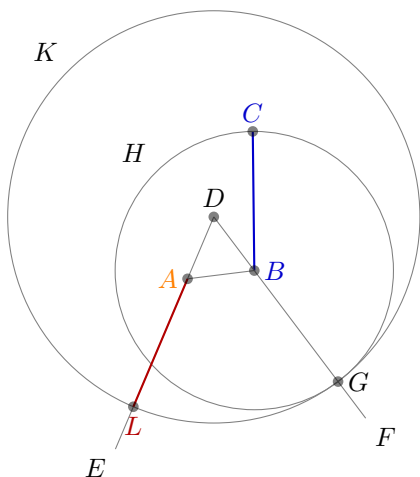
\foreach \point in {A,B,C}
\fill [black,opacity=.5] (\point) circle (2pt);

\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*.

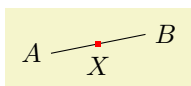
Therefore the *straight line* *AL* equal to the given *straight line* *BC* has been placed with one end at the *given point* *A*.

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 !(<分割比>! 将它们组合起来。(<分割比> 为 0 表示第一个坐标，(<分割比> 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。

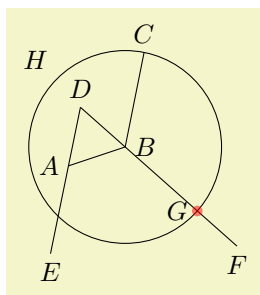
这是 *AB* 线段中点的计算：



```
\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}
```

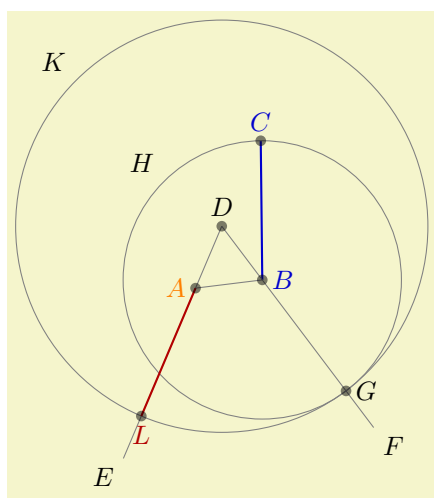
欧几里得第二个命题中的点 *D* 的计算要复杂一些。可以表示为：考虑从 *X* 到 *B* 的线段。假设我们将这条线段绕 *X* 旋转 90° ，然后将其拉伸 $\sin(60^\circ) \cdot 2$ 倍，这将产生所需的点 *D*。我们可以使用上面的分割运算编辑器进行拉伸，对于旋转，我们需要一个新的编辑器：旋转编辑器。这个想法是，在进行中间计算时，第二个坐标可以加上一个角度作为前缀。然后正常计算中间点（好像没有给出角度），但是结果点围绕第一个点旋转了这个角度。

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=H and B--F,by={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 ($ (A)!.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=H and B--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=K and A--E,by={label=below:\L}}];

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

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

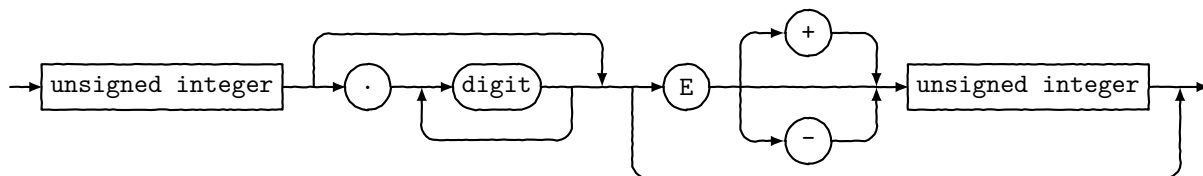
  % \node ...
\end{tikzpicture}

```

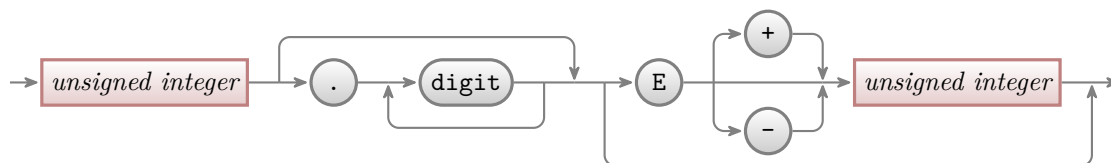
5 教程：简单的图表

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：



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={
        % The shape:
        rectangle,minimum size=6mm,rounded corners=3mm,
        % The rest
        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={
        % The shape:
        rounded rectangle,
        minimum size=6mm,
        % The rest
        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};

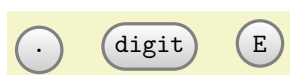
\draw [help lines] let \p1 = (dot.base),
    \p2 = (digit.base),
    \p3 = (E.base)
    in (-.5,\y1) -- (3.5,\y1)
    (-.5,\y2) -- (3.5,\y2)
    (-.5,\y3) -- (3.5,\y3);
\end{tikzpicture}
```

（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

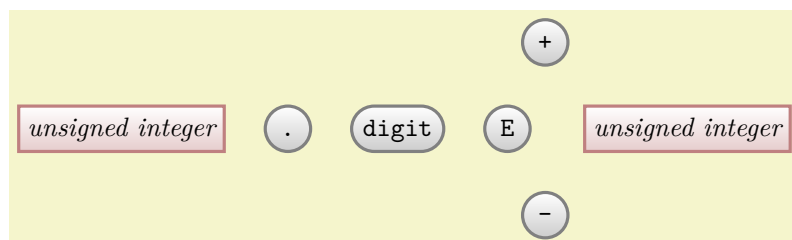
伊尔卡现在已准备好节点的“样式”。下一个问题是将它们放置在正确的位置。有几种方法可以做到这一点。最简单的方法是简单地将节点明确地放置在“手工计算”的特定坐标处。对于非常简单的图形，这是完全可以的，但是它有几个缺点：

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：



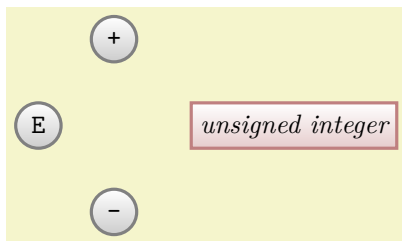
```

\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：

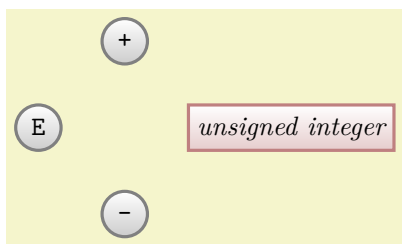


```

\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};

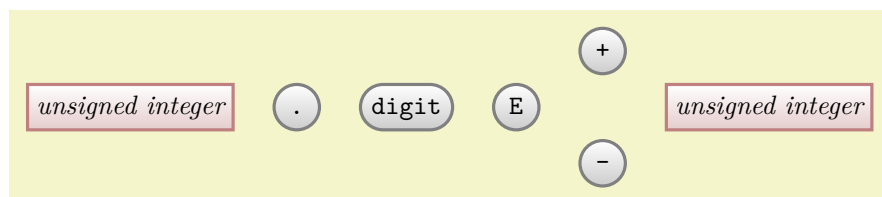
  \path (dot) edge[->] (digit) % simple edges
        (digit) edge[->] (E)
        ($ (digit.east)!.5!(E.west) $)
        edge[->,skip loop=-5mm] ($ (digit.west)!.5!(dot.east) $);
\end{tikzpicture}
```

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。

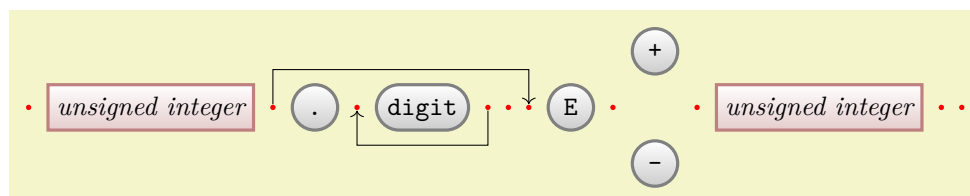
或许更好的定位节点的方法是使用矩阵。在 TikZ 中，矩阵可用于对齐行和列中的任意图形对象。其语法非常类似于 \TeX 中使用数组和表（实际上，在内部使用了 \TeX 表，但还有很多其他内容）。

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] {
    % First row:
    & & & \node [terminal] {+}; & \\
    % Second row:
    \node [nonterminal] {unsigned integer}; & & & & \\
    \node [terminal] {\.}; & & & & \\
    \node [terminal] {digit}; & & & & \\
    \node [terminal] {E}; & & & & \\
    & & & & \\
    \node [nonterminal] {unsigned integer}; & & & & \\
    % Third row:
    & & & \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] {
% First row:
& & & & & & & & \node (plus) [terminal] {+};\\
% Second row:
\node (p1) [point] {}; & & \node (ui1) [nonterminal] {unsigned integer}; & & \\
\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] {unsigned integer}; & & \\
\node (p9) [point] {}; & & \node (p10) [point] {}; & & \\
% Third row:
& & & & & & & & \node (minus) [terminal] {-};\\
};

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

```

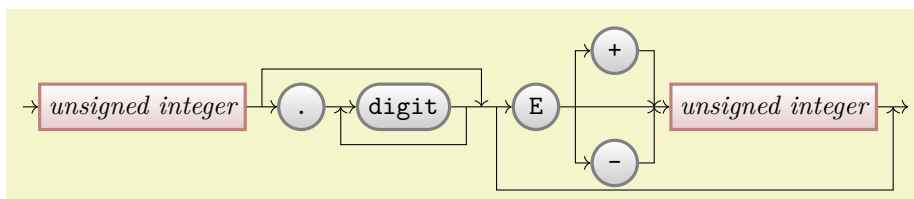
现在，添加所有缺失的线段只是一小步。

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 };

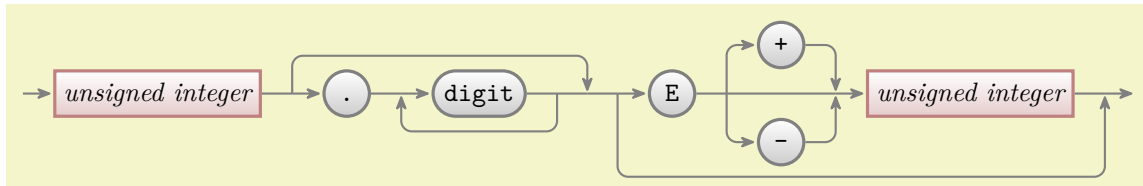
\graph {
    (p1) -> (ui1) -- (p2) -> (dot) -- (p3) -> (digit) -- (p4)
        -- (p5) -- (p6) -> (e) -- (p7) -- (p8) -> (ui2) -- (p9) -> (p10);
    (p4) ->[skip loop=-5mm] (p3);
    (p2) ->[skip loop=5mm] (p5);
    (p6) ->[skip loop=-11mm] (p9);
    (p7) ->[vh path] (plus) -> [hv path] (p8);
    (p7)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

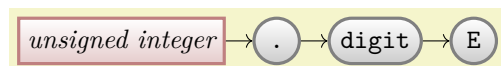
```
unsigned integer -> d -> digit -> E
```

```

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

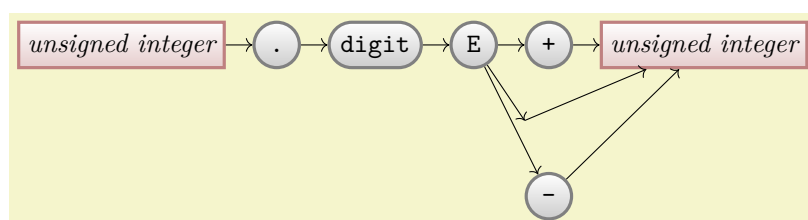
```

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] {
  unsigned integer[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：

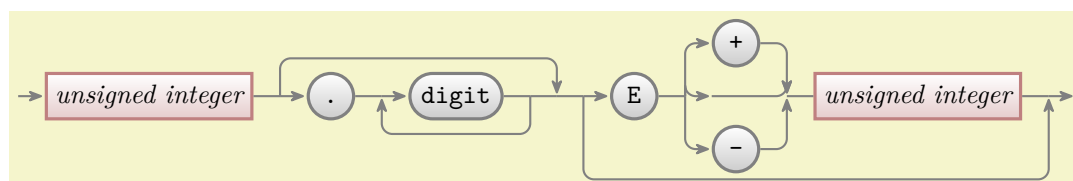


```
\usetikzlibrary {graphs,shapes.misc}
\tikz \graph [grow right sep] {
  unsigned integer [nonterminal] ->
  "." [terminal] ->
  digit [terminal] ->
  E [terminal] ->
  {
    "+" [terminal],
    "" [coordinate],
    "-" [terminal]
  } ->
  ui2/unsigned integer [nonterminal]
};
```

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：



```

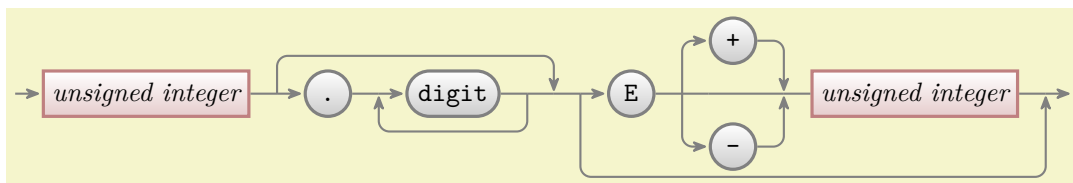
\usetikzlibrary {arrows.meta,graphs,shapes.misc}
\begin{tikzpicture}[>={Stealth[round]}, thick, black!50, text=black,
                    every new ->/.style={shorten >=1pt},
                    graphs/every graph/.style={edges=rounded corners}]
\graph [grow right sep, branch down=7mm] {
  / [coordinate]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

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

q1 -- q2 -- q3; % make these edges plain
};

```

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。

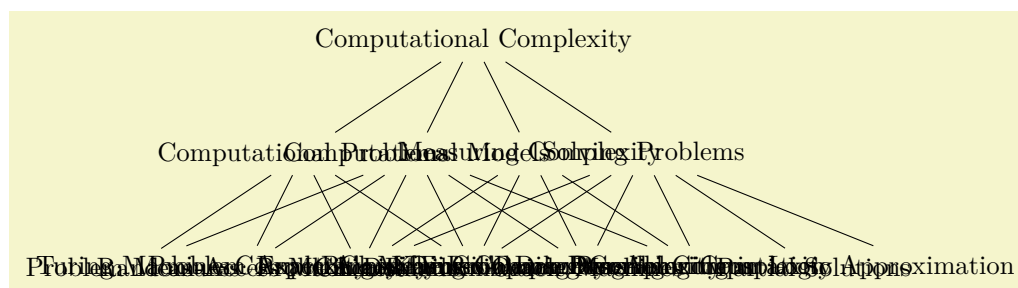
约翰内斯首先在树上列出了一些概念，他认为这些概念对于计算复杂性很重要：

- 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：

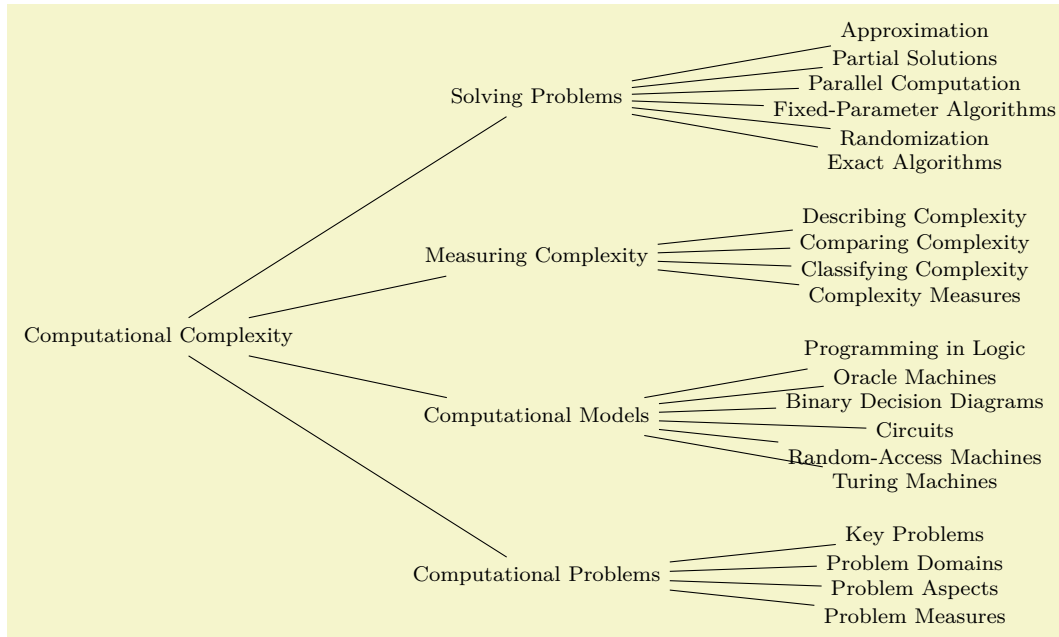


```
\tikz
\node {Computational Complexity} % root
  child { node {Computational Problems}
    child { node {Problem Measures} }
    child { node {Problem Aspects} }
    child { node {Problem Domains} }
    child { node {Key Problems} }
  }
  child { node {Computational Models}
    child { node {Turing Machines} }
    child { node {Random-Access Machines} }
    child { node {Circuits} }
    child { node {Binary Decision Diagrams} }
    child { node {Oracle Machines} }
    child { node {Programming in Logic} }
  }
  child { node {Measuring Complexity}
    child { node {Complexity Measures} }
    child { node {Classifying Complexity} }
    child { node {Comparing Complexity} }
    child { node {Describing Complexity} }
  }
  child { node {Solving Problems}
    child { node {Exact Algorithms} }
    child { node {Randomization} }
    child { node {Fixed-Parameter Algorithms} }
    child { node {Parallel Computation} }
    child { node {Partial Solutions} }
    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：



```

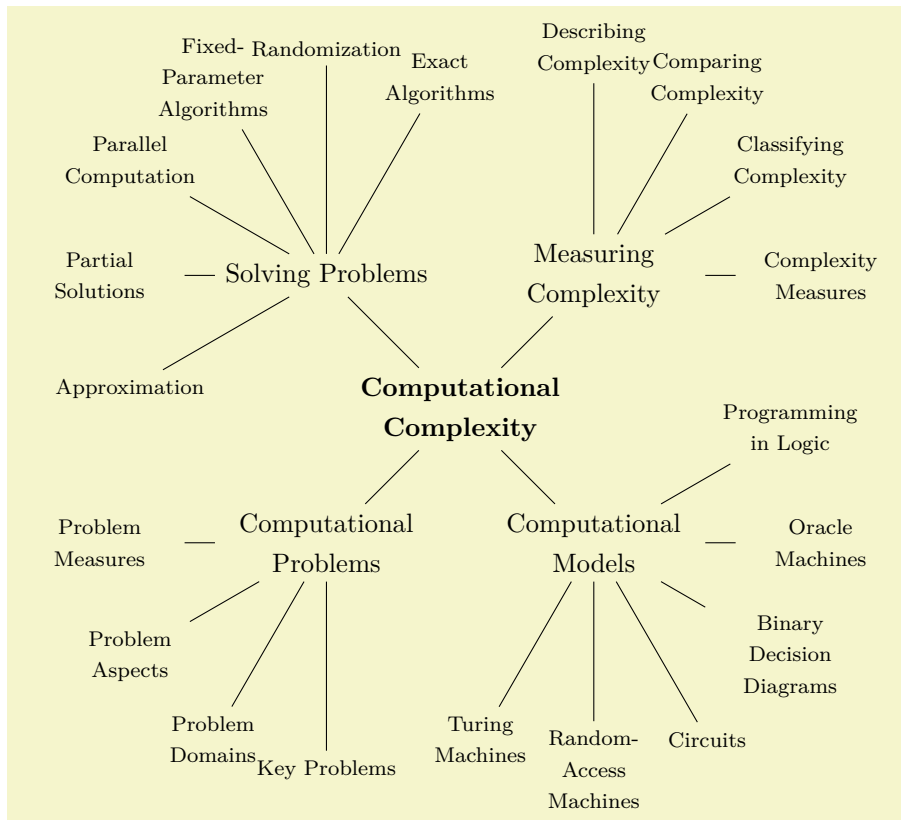
\usetikzlibrary {trees}
\tikz [font=\footnotesize,
       grow=right, level 1/.style={sibling distance=6em},
       level 2/.style={sibling distance=1em}, level distance=5cm]
\node {Computational Complexity} % root
  child { node {Computational Problems}
    child { node {Problem Measures} }
    child { node {Problem Aspects} }
    ... % as before
  
```

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=flush center,
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] {Computational Complexity} % root
child { node {Computational Problems}
child { node {Problem Measures} }
child { node {Problem Aspects} }
... % as before
```

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。

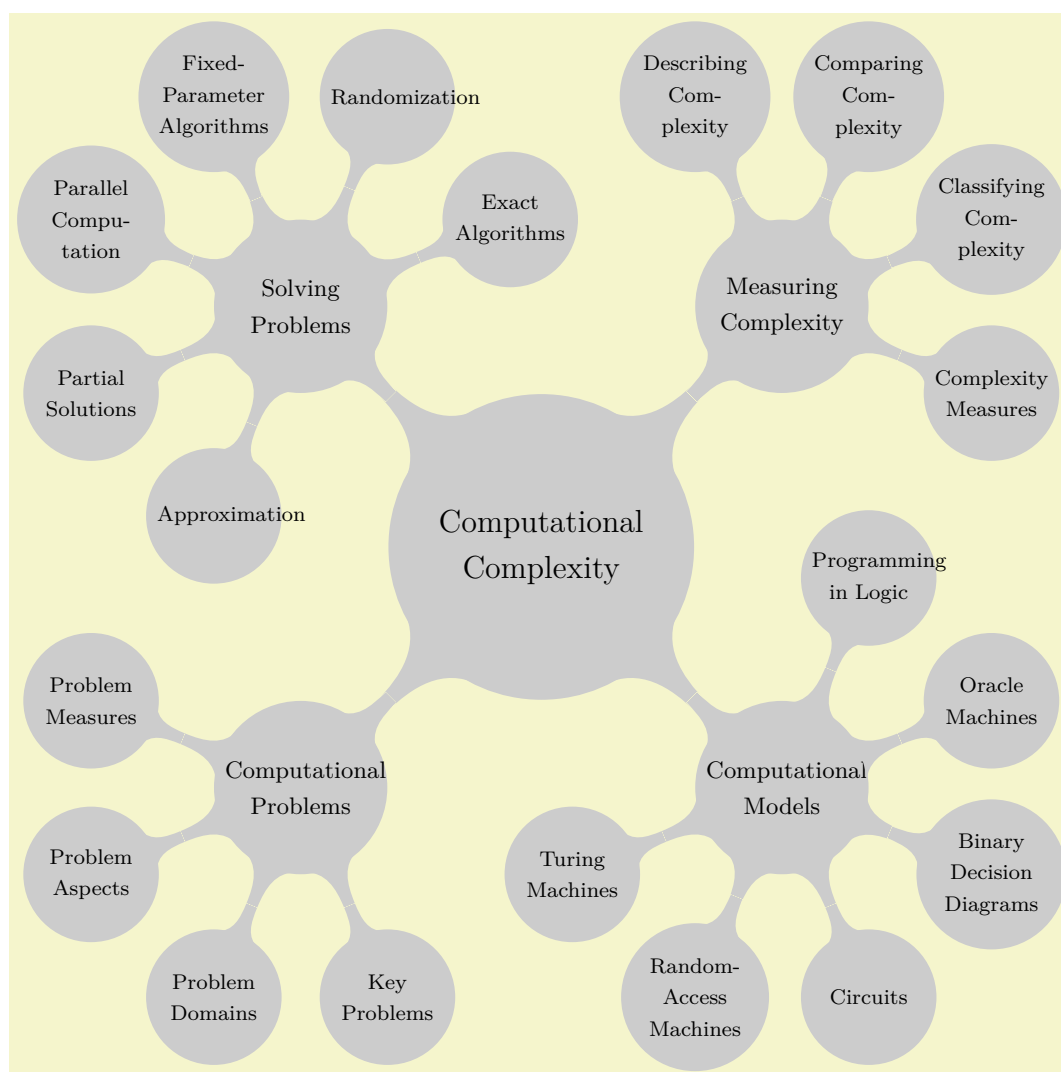
6.3 创建讲义图

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。

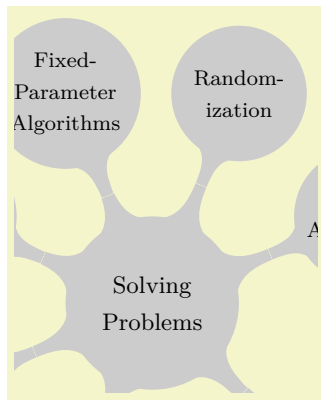
第二步是将 `concept` 选项添加到每一个节点，这些节点实际上是思维导图的一个概念。其思想是树的一些节点将是真实的概念，而其他节点可能只是“简单的子节点”。通常情况下，情况并非如此，因此您可以考虑使用 `every node/.style=concept`。

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] {Computational Complexity} % root
child { node {Computational Problems}
child { node {Problem Measures} }
child { node {Problem Aspects} }
... % as before
```

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 TikZ 在每个节点中插入该文本。



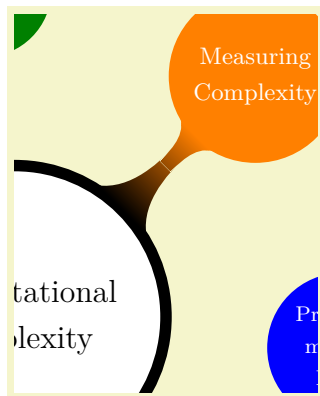
```
\usetikzlibrary {mindmap}
\begin{tikzpicture}
[mindmap,
every node/.style={concept, execute at begin node=\hskip0pt},
concept color=black!20,
grow cyclic,
level 1/.append style={level distance=4.5cm,sibling angle=90},
level 2/.append style={level distance=3cm,sibling angle=45}]
\clip (-1,2) rectangle ++ (-4,5);
\node [root concept] {Computational Complexity} % root
child { node {Computational Problems}
child { node {Problem Measures} }
child { node {Problem Aspects} }
... % as before
\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。

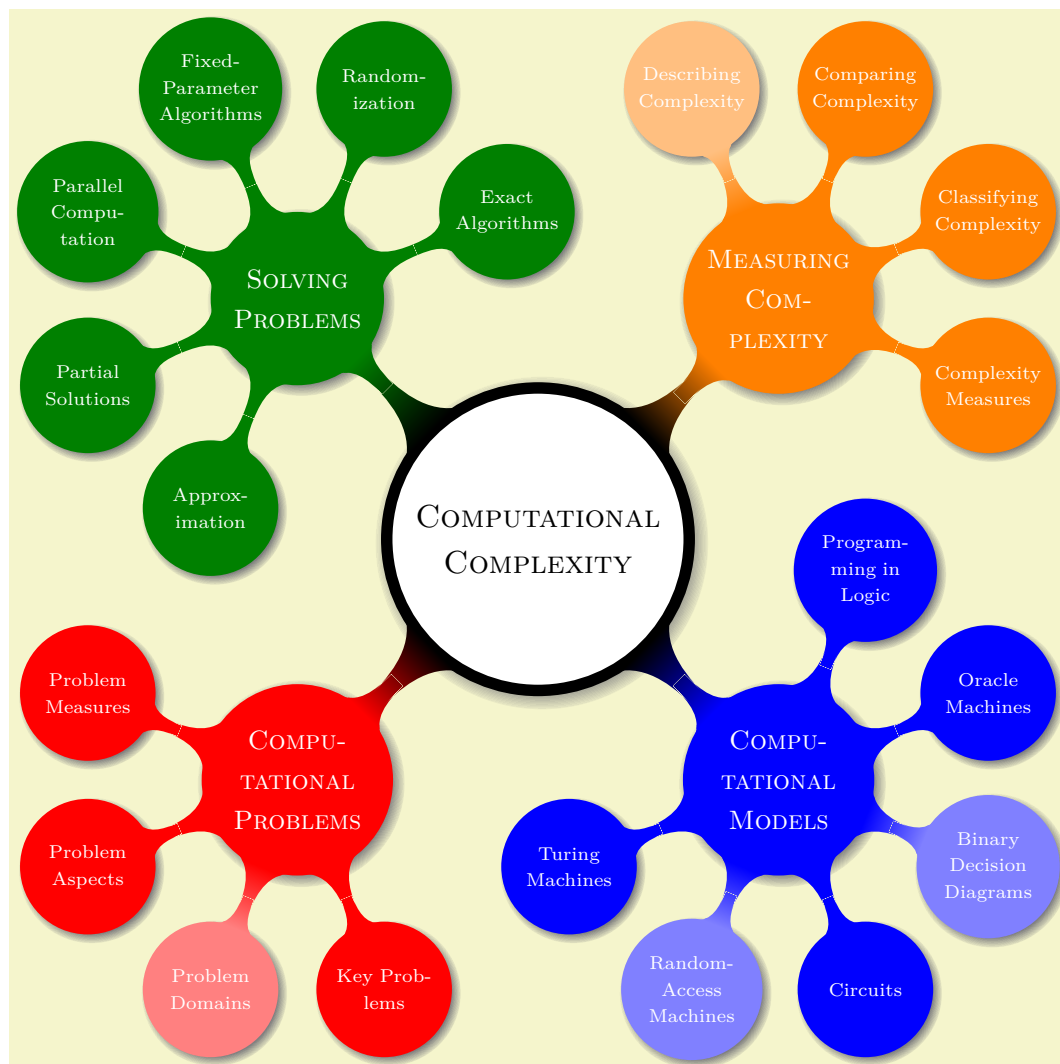
为了设置颜色，约翰内斯必须使用 `concept color` 选项，而不仅仅是 `node [fill=red]`。仅将填充颜色设置为 `red` 确实会使节点为红色，但它将仅使节点为红色，而不是连接概念到父节点和子节点的颜色带。相比之下，特殊的 `concept color` 选项不仅会设置节点及其子节点的颜色，而且还会（神奇地）创建适当的阴影，以便父概念的颜色平稳地变化为子概念的颜色。

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] {Computational Complexity} % root
child [concept color=red] { node {Computational Problems}
child { node {Problem Measures} }
... % as before
}
child [concept color=blue] { node {Computational Models}
child { node {Turing Machines} }
... % as before
}
child [concept color=orange] { node {Measuring Complexity}
child { node {Complexity Measures} }
... % as before
}
child [concept color=green!50!black] { node {Solving Problems}
child { node {Exact Algorithms} }
... % as before
};
\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] {Computational Complexity} % root
      child [computational problems] { node {Computational Problems}
        child { node {Problem Measures} }
        child { node {Problem Aspects} }
        child [faded] { node {Problem Domains} }
        child { node {Key Problems} }
      }
      child [computational models] { node {Computational Models}
        child { node {Turing Machines} }
        child [faded] { node {Random-Access Machines} }
      }
      ...
  \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。

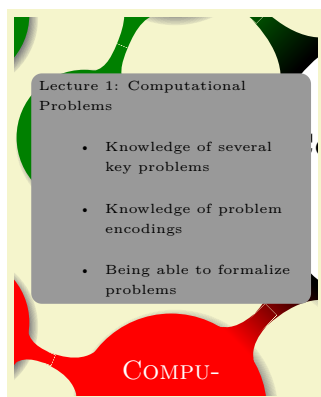
为了将注释放置在概念旁边，约翰内斯必须为概念的节点分配名称。他可以使用 TikZ 对树中节点的自动命名，其中树节点名为 `root`。节点的子节点被命名为 `root-1`、`root-2`、`root-3`，依此类推。但是，由于约翰内斯不确定树中概念的最终顺序，因此最好以以下方式明确命名树的所有概念：

```

\node [root concept] (Computational Complexity) {Computational Complexity}
  child [computational problems] { node (Computational Problems) {Computational Problems}
    child { node (Problem Measures) {Problem Measures} }
    child { node (Problem Aspects) {Problem Aspects} }
    child [faded] { node (Problem Domains) {Problem Domains} }
    child { node (Key Problems) {Key Problems} }
  }
  ...

```

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。

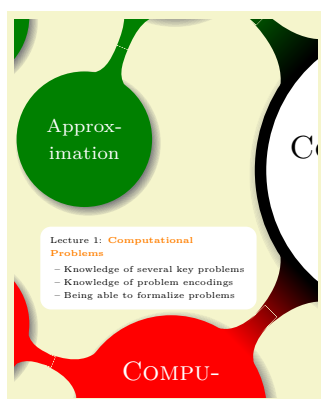


```
\usetikzlibrary {mindmap,shadows}
\begin{tikzpicture}[mindmap]
\clip (-5,-5) rectangle ++ (4,5);
\begin{scope}[
every node/.style={concept, circular drop shadow, ...}] % as before
\node [root concept] (Computational Complexity) ... % as before
\end{scope}

\begin{scope}[every annotation/.style={fill=black!40}]
\node [annotation, above] at (Computational Problems.north) {
Lecture 1: Computational Problems
\begin{itemize}
\item Knowledge of several key problems
\item Knowledge of problem encodings
\item Being able to formalize problems
\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}
{Computational Problems.north}{
\item Knowledge of several key problems
\item Knowledge of problem encodings
\item Being able to formalize problems
}{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：



```
\usetikzlibrary {backgrounds,mindmap,shadows}
\begin{tikzpicture}[
  mindmap,
  concept color=black,
  root concept/.append style={
    concept,
    circular drop shadow,
    fill=white, line width=1ex,
    text=black, font=\large\scshape}
]

\clip (-1.5,-5) rectangle ++(4,10);

\node [root concept] (Computational Complexity) {Computational Complexity};

\begin{pgfonlayer}{background}
  \clip (-1.5,-5) rectangle ++(4,10);

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

  % The large rectangles:
  \fill [upperleft] (Computational Complexity) rectangle ++(-20,20);
  \fill [upperright] (Computational Complexity) rectangle ++(20,20);
  \fill [lowerleft] (Computational Complexity) rectangle ++(-20,-20);
  \fill [lowerright] (Computational Complexity) rectangle ++(20,-20);

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

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             name=cal,
8             dates=2009-04-01 to 2009-04-14]
9
10   if (weekend)
11     [black!25];
12
13 \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{
  % As before:
  \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=base west] at (cal-#6.base east) {\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	
2	
3	
4	
5	
6	
7	
8	Computational Problems
9	
10	
11	
12	
13	
14	

```

\usetikzlibrary {calendar}
\tiny
\begin{tikzpicture}
  \calendar [day list downward,
    name=cal,
    dates=2009-04-01 to 2009-04-14]
    if (weekend)
      [black!25];

  % As before:
  \lecture{1}{Computational Problems}{above,xshift=-3mm}
  {Computational Problems.north}{
    \item Knowledge of several key problems
    \item Knowledge of problem encodings
    \item Being able to formalize problems
  }{2009-04-08}
\end{tikzpicture}

```

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	
1	
2	
3	
4	
5	
6	
7	
8	Computational Problems
9	
10	
11	
12	
13	
14	
15	Computational Models
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
May 2009	
1	

```

\usetikzlibrary {calendar}
\tiny
\begin{tikzpicture}
  \calendar [day list downward,
    month text=\%mt\ \%y0,
    month yshift=3.5em,
    name=cal,
    dates=2009-04-01 to 2009-05-01]
    if (weekend)
      [black!25]
    if (day of month=1) {
      \node at (Opt,1.5em) [anchor=base west] {\small\tikzmonthtext};
    };

  \lecture{1}{Computational Problems}{above,xshift=-3mm}
  {Computational Problems.north}{
    \item Knowledge of several key problems
    \item Knowledge of problem encodings
    \item Being able to formalize problems
  }{2009-04-08}

  \lecture{2}{Computational Models}{above,xshift=-3mm}
  {Computational Models.north}{
    \item Knowledge of Turing machines
    \item Being able to compare the computational power of different
      models
  }{2009-04-15}
\end{tikzpicture}

```

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{
  % As before:
  \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=base west] at (cal-#6.base east) {\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] (Computational Complexity) {Computational Complexity} % root
  child [computational problems] { node [yshift=-1cm] (Computational Problems) {Computational Problems}
    child { node (Problem Measures) {Problem Measures} }
    child { node (Problem Aspects) {Problem Aspects} }
    child [faded] { node (problem Domains) {Problem Domains} }
    child { node (Key Problems) {Key Problems} }
  }
  child [computational models] { node [yshift=-1cm] (Computational Models) {Computational Models}
    child { node (Turing Machines) {Turing Machines} }
    child [faded] { node (Random-Access Machines) {Random-Access Machines} }
    child { node (Circuits) {Circuits} }
    child [faded] { node (Binary Decision Diagrams) {Binary Decision Diagrams} }
    child { node (Oracle Machines) {Oracle Machines} }
    child { node (Programming in Logic) {Programming in Logic} }
  }
  child [measuring complexity] { node [yshift=1cm] (Measuring Complexity) {Measuring Complexity}
    child { node (Complexity Measures) {Complexity Measures} }
    child { node (Classifying Complexity) {Classifying Complexity} }
    child { node (Comparing Complexity) {Comparing Complexity} }
    child [faded] { node (Describing Complexity) {Describing Complexity} }
  }
  child [solving problems] { node [yshift=1cm] (Solving Problems) {Solving Problems}
    child { node (Exact Algorithms) {Exact Algorithms} }
    child { node (Randomization) {Randomization} }
    child { node (Fixed-Parameter Algorithms) {Fixed-Parameter Algorithms} }
    child { node (Parallel Computation) {Parallel Computation} }
    child { node (Partial Solutions) {Partial Solutions} }
    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,
    month yshift=3.5em,
    name=cal,
    at={(-.5\textwidth-5mm,.5\textheight-1cm)},
    dates=2009-04-01 to 2009-06-last]
  if (weekend)
    [black!25]
  if (day of month=1) {
    \node at (Opt,1.5em) [anchor=base west] {\small\tikzmonthtext};
  };
\end{tikzpicture}

```

讲义注释:

```

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

  \lecture{2}{Computational Models}{above left}{
    {Computational Models.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}

% The large rectangles:
\fill [upperleft] (Computational Complexity) rectangle ++(-20,20);
\fill [upperright] (Computational Complexity) rectangle ++(20,20);
\fill [lowerleft] (Computational Complexity) rectangle ++(-20,-20);
\fill [lowerright] (Computational Complexity) rectangle ++(20,-20);

% The shadings:
\shade [left color=upperleft,right color=upperright]
([xshift=-1cm]Computational Complexity) rectangle ++(2,20);
\shade [left color=lowerleft,right color=lowerright]
([xshift=-1cm]Computational Complexity) rectangle ++(2,-20);
\shade [top color=upperleft,bottom color=lowerleft]
([yshift=-1cm]Computational Complexity) rectangle ++(-20,2);
\shade [top color=upperright,bottom color=lowerright]
([yshift=-1cm]Computational Complexity) 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

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。虽然我并不完全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。

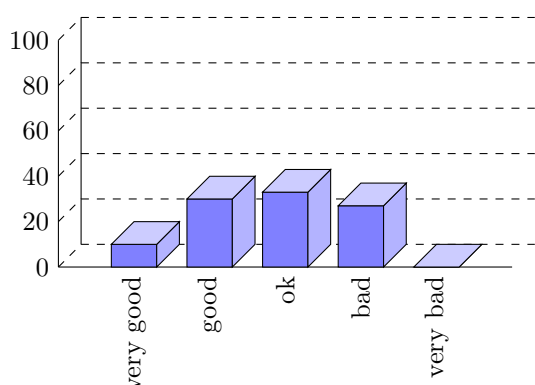
创建图时您应该问自己的第一个问题是：是否有足够的数据点可用于绘制图？如果答案是“否定的”，请使用表格。

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：

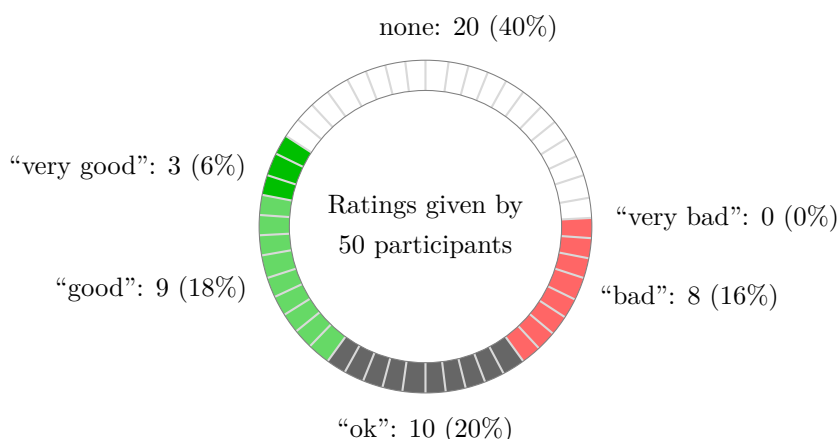
1. 总共有多少参与者？
2. 有多少参与者退回了反馈表？
3. 百分之几的参与者返回了反馈表？
4. 有多少参与者的评价“非常好”？
5. 在所有参与者中，有多少百分比表示“非常好”？
6. 是否有超过四分之一的参与者检查“差”或“非常差”？
7. 提交了反馈表的参与者中有百分之多少表示“非常好”？

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。

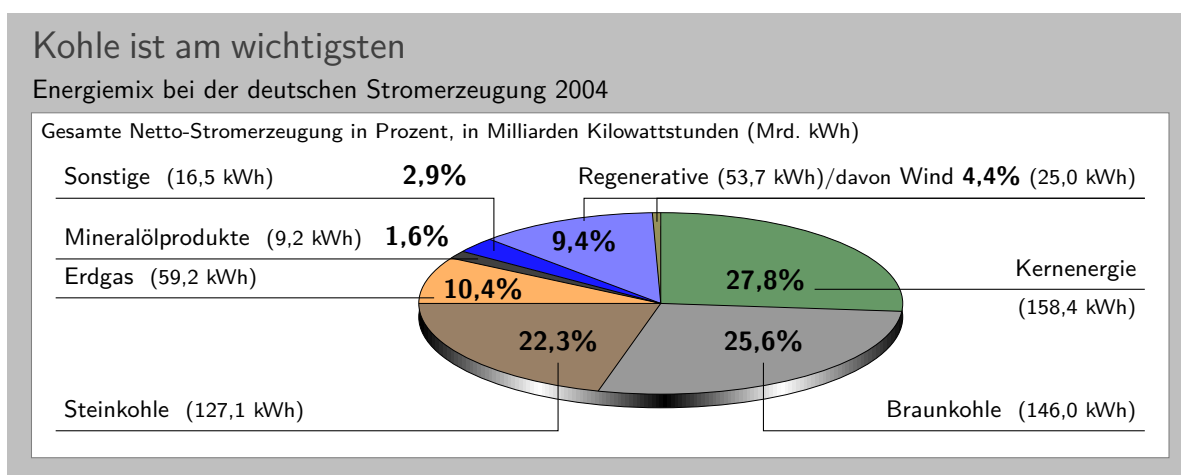
- 三维设置使正确度量柱的高度变得十分困难。考虑一下“差”这一柱。这一条代表的数字是否是大于或等于 20？柱状图的前部在 20 以下，而柱状图的后部确在其上方。
- 我们无法分辨这些柱状图代表的是哪些数字。因此，这些柱不必要地隐藏了这些柱所包含的所有信息。
- 杆高加起来是多少？是 100% 还是 60%？
- “非常差”柱代表 0 还是 1？
- 为什么柱是蓝色的？

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。

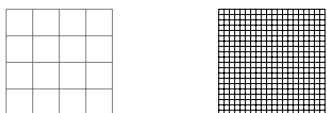
现在，选择您喜欢的每周杂志或报纸，然后浏览典型的页面。您会注意到页面上有很多“正在进行中”。字体以不同的大小和排列方式使用，文本以窄列组织，通常与图片交错。杂志以这种方式排版的原因是另一个范例：引导注意力。

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会首先被读者看到：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。