LaTeX for Discrete Mathematics

Dr. Isaac Griffith

March 2022

Symbols

Required Packages: mathtools, amssymb, amsmath
Requires: `\DeclareMathOperator{\lcm}{lcm}` in preamble

Basic Math Operators

- $*$ – `*`
- $\cdot$ – `\cdot`
- $+$ – `+`
- $-$ – `-`
- $/$ – `/`
- $<$ – `<`
- $>$ – `>`
- $\geq$ – `\geq`
- $\leq$ – `\leq`
- $\neq$ – `\neq`
- $=$ – `=`
- $\log x$ – `\log x`
- $\max x$ – `\max x`
- $\min x$ – `\min x`
- $\gcd x$ – `\gcd x`
- $\operatorname{lcm} x$
- $\lim_{h \to 0}$ – `\lim_{h \to 0}`
- $\sqrt{x}$ – `\sqrt{x}`
- $\lfloor x \rfloor$ – `\lfloor x \rfloor`
- $\lceil x \rceil$ – `\lceil x \rceil`
- $\ldots$ – `\ldots`
- $\vdots$ – `\vdots`

Sets

- $\mathbb{C}$ – `\mathbb{C}`
- $\mathbb{Z}$ – `\mathbb{Z}`
- $\mathbb{Z}^+$ – `\mathbb{Z}^{+}`
- $\mathbb{N}$ – `\mathbb{N}`
- $\mathbb{R}$ – `\mathbb{R}`
- $\mathbb{Q}$ – `\mathbb{Q}`

Set Theory Operators

- $\cup$ – `\cup`
- $\cap$ – `\cap`
- $\times$ – `\times`
- $\subset$ – `\subset`
- $\subseteq$ – `\subseteq`
- $\nsubseteq$ – `\nsubseteq`
- $\supset$ – `\supset`
- $\supseteq$ – `\supseteq`
- $\not\supseteq$ – `\nsupseteq`
- $\in$ – `\in`
- $\notin$ – `\notin`
- $\setminus$ – `\setminus`
- $\varnothing$ – `\varnothing`

Logic Operators

- $\land$ – `\land`
- $\lor$ – `\lor`
- $\oplus$ – `\oplus`
- $\neg$ – `\lnot`
- $\to$ – `\to`
- $\leftrightarrow$ – `\leftrightarrow`
- $\vdash$ – `\vdash`
- $\models$ – `\models`
- $\top$ – `\top`
- $\bot$ – `\bot`
- $\exists$ – `\exists`
- $\forall$ – `\forall`
- $\therefore$ – `\therefore`
- $\square$ – `\square`

Function Operators

- $\equiv$ – `\equiv`
- $\circ$ – `\circ`
- ∞ – `\infty`

Big Operators

- $\bigvee$ – `\bigvee`
- $\bigwedge$ – `\bigwedge`
- $\bigcap$ – `\bigcap`
- $\bigcup$ – `\bigcap`
- $\sum$ – `\sum`
- $\prod$ – `\prod`

Relations Operators

- $\sqsubset$ – \sqsubset
- $\sqsubseteq$ – \sqsubseteq
- $\sqsupset$ – \sqsubset
- $\sqsupseteq$ – \sqsubseteq
- $\sim$ – \sim
- $\prec$ – \prec
- $\succ$ – \succ
- $\preccurlyeq$ – \preccurlyeq
- $\succcurlyeq$ – \succcurlyeq

Algorithms

- $\coloneqq$ – \coloneqq
- $\leftarrow$ – \gets

Graphs and Trees

- deg – \deg

Matrices

- $\odot$ – \odot

Constructing Truth Tables and Membership Tables

To create a Truth Table, we simply need to create a table. For example the truth table for $((A \to B) \land \neg B) \to \neg A$ is created as follows:

LaTeX Code:

```
\begin{center}
\begin{tabular}{|c|c|cccc|c|}
\hline
$A$ & $B$ & $A \to B$ & $\lnot B$ & $(A \to B) \land \not B$ & $\lnot A$ &
$((A \to B) \land \lnot B) \to \lnot A$ \\ \hline
F & F & T & T & T & T & \textbf{T} \\
F & T & T & F & F & T & \textbf{T} \\
T & F & F & T & F & F & \textbf{T} \\
T & T & T & F & F & F & \textbf{T} \\ \hline
\end{tabular}
\end{center}
```

Results:

A	B	$A \to B$	$\neg B$	$(A \to B) \land \not B$	$\neg A$	$((A \to B) \land \neg B) \to \neg A$
F	F	T	T	T	T	**T**
F	T	T	F	F	T	**T**
T	F	F	T	F	F	**T**
T	T	T	F	F	F	**T**

Creating Proof Trees

Required Packages

- cancel
- bussproofs

Proof Trees

Proof trees are created using the following commands:

- \AxiomC{value} - Create an assumption or axiom for which there is nothing before it.
- \UnaryInfC{value} - Creates an inference for a single (last defined) result.
- \BinaryInfC{value} - Creates an inference for the prior two results.
- \TrinaryInfC{value} - Creates an inference for the prior three results.
- \QuinaryInfC{value} - Creates an inference for the prior four results.
- \RightLabel{value} - Creates a label (on the right) of the line for an inference

Example

LaTeX Code:

```
    \begin{prooftree}
        \AxiomC{$\left(P \land Q) \lor (P \land R)\right)$}
        \AxiomC{$\cancel{P \land Q}$}
          \RightLabel{$\{\land E_L\}$}
          \UnaryInfC{$P$}
        \AxiomC{$\cancel{P \land R}$}
          \RightLabel{$\{\land E_L\}$}
          \UnaryInfC{$P$}
        \RightLabel{$\{\lor E\}$}
        \TrinaryInfC{$P$}
    \end{prooftree}
```

Results:

$$\dfrac{(P \land Q) \lor (P \land R)) \qquad \dfrac{\cancel{P \land Q}}{P}\,\{\land E_L\} \qquad \dfrac{\cancel{P \land R}}{P}\,\{\land E_L\}}{P}\,\{\lor E\}$$

Creating Graphs and Trees

Required Packages

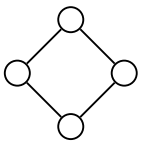
- tikz

Creating a Simple Graph

LaTeX Code:

```
\begin{tikzpicture}
[node distance={10mm},main/.style = {draw, circle, line width=0.25mm}]

\node[main] (1) {};
\node[main] (2) [above right of=1] {};
\node[main] (3) [below right of=1] {};
\node[main] (4) [above right of=3] {};

\draw [line width=0.25mm] (1) -- (2);
\draw [line width=0.25mm] (1) -- (3);
\draw [line width=0.25mm] (2) -- (4);
\draw [line width=0.25mm] (3) -- (4);
\end{tikzpicture}
```

Results:



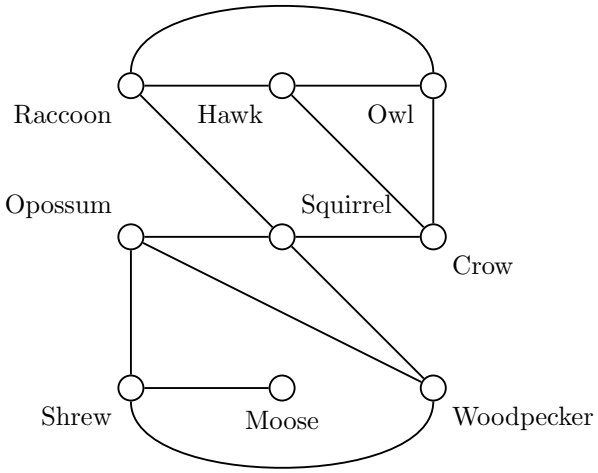
A More Interesting Example

LaTeX Code:

```
\begin{tikzpicture}
[node distance={20mm},main/.style = {draw, circle, line width=0.25mm}]

\node[main] (1) [label=below left:{Raccoon}] {};
\node[main] (2) [label=below left:{Hawk},right of=1] {};
\node[main] (3) [label=below left:{Owl},right of=2] {};

```

```
\node[main] (4) [label=above left:{Opossum},below of=1] {};
\node[main] (5) [label=above right:{Squirrel},right of=4] {};
\node[main] (6) [label=below right:{Crow},right of=5] {};

\node[main] (7) [label=below left:{Shrew},below of=4] {};
\node[main] (8) [label=below:{Moose},right of=7] {};
\node[main] (9) [label=below right:{Woodpecker},right of=8] {};

\draw [line width=0.25mm] (1) -- (2);
\draw [line width=0.25mm] (2) -- (3);

\draw [line width=0.25mm] (1) -- (5);
\draw [line width=0.25mm] (2) -- (6);
\draw [line width=0.25mm] (3) -- (6);

\draw [line width=0.25mm] (4) -- (5);
\draw [line width=0.25mm] (5) -- (6);

\draw [line width=0.25mm] (4) -- (7);
\draw [line width=0.25mm] (4) -- (9);
\draw [line width=0.25mm] (5) -- (9);

\draw [line width=0.25mm] (7) -- (8);

\draw [line width=0.25mm] (1) to [out=90, in=90, looseness=.75] (3);
\draw [line width=0.25mm] (7) to [out=270, in=270, looseness=.75] (9);
\end{tikzpicture}
```

Results:



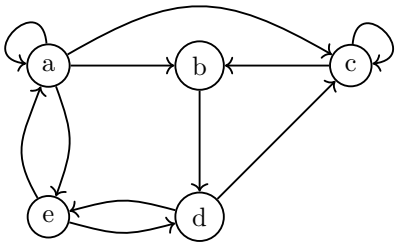
Digraphs

LaTeX Code:

```
\begin{tikzpicture}
[node distance={20mm},main/.style = {draw, circle, line width=0.25mm}]

\node[main] (1) {a};
\node[main] (2) [right of=1] {b};
\node[main] (3) [right of=2] {c};
\node[main] (4) [below of=1] {e};
\node[main] (5) [right of=4] {d};

\draw [->,line width=0.25mm] (1) -- (2);
\draw [->,line width=0.25mm] (3) -- (2);
\draw [->,line width=0.25mm] (2) -- (5);
```

```
\draw [->,line width=0.25mm] (5) -- (3);

\draw [->,line width=0.25mm] (1) to [out=95, in=175, looseness=5] (1);
\draw [->,line width=0.25mm] (1) to [out=30, in=150, looseness=1.25] (3);
\draw [->,line width=0.25mm] (3) to [out=85, in=5, looseness=5] (3);
\draw [->,line width=0.25mm] (4) to [out=120, in=250, looseness=1.25] (1);
\draw [->,line width=0.25mm] (1) to [out=290, in=70, looseness=1.25] (4);
\draw [->,line width=0.25mm] (4) to [out=-15, in=195, looseness=1.25] (5);
\draw [->,line width=0.25mm] (5) to [out=165, in=15, looseness=1.25] (4);

\end{tikzpicture}
```

Results:

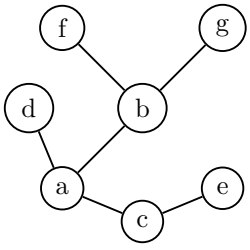


A Simple Tree

LaTeX Code:

```
\begin{tikzpicture}
[node distance={15mm},main/.style = {draw, circle, line width=0.25mm}]

\node[main] (1) {b};
\node[main] (2) [left of=1] {d};
\node[main] (3) [below left of=1] {a};
\node[main] (4) [above left of=1] {f};
\node[main] (5) [above right of=1] {g};
\node[main] (6) [below of=1] {c};
\node[main] (7) [below right of=1] {e};

\draw [line width=0.25mm] (1) -- (4);
\draw [line width=0.25mm] (1) -- (5);
\draw [line width=0.25mm] (1) -- (3);
\draw [line width=0.25mm] (3) -- (2);
\draw [line width=0.25mm] (3) -- (6);
\draw [line width=0.25mm] (6) -- (7);

\end{tikzpicture}
```

Results:



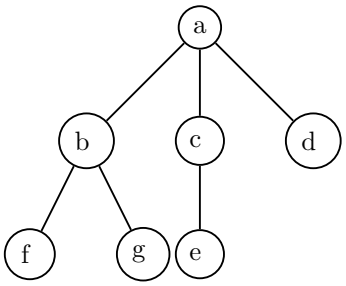
A Basic Rooted Tree

LaTeX Code:

```
\begin{tikzpicture}
```

```
[every node/.style = {shape=circle, draw, line width=0.25mm, align=center}]
\node {a}
    child {node {b}
    child {node {f}}
    child {node {g}} edge from parent [line width=0.25mm]}
    child {node {c}
    child {node {e}} edge from parent [line width=0.25mm]}
    child {node {d} edge from parent [line width=0.25mm]};

\end{tikzpicture}
```

Results:

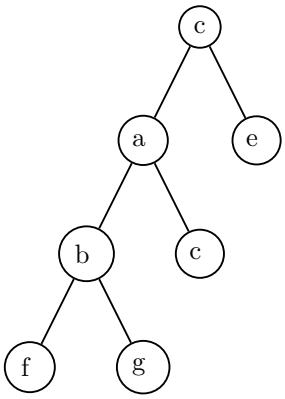


A Simple Binary Tree

LaTeX Code:

```
\begin{tikzpicture}
[every node/.style = {shape=circle, draw, line width=0.25mm, align=center}]
\node {c}
  child {node {a}
    child {node {b}
      child {node {f}}
      child {node {g}} edge from parent [line width=0.25mm]}
    child {node {c}} edge from parent [line width=0.25mm]}
  child {node {e} edge from parent [line width=0.25mm]};

\end{tikzpicture}
```

Results:



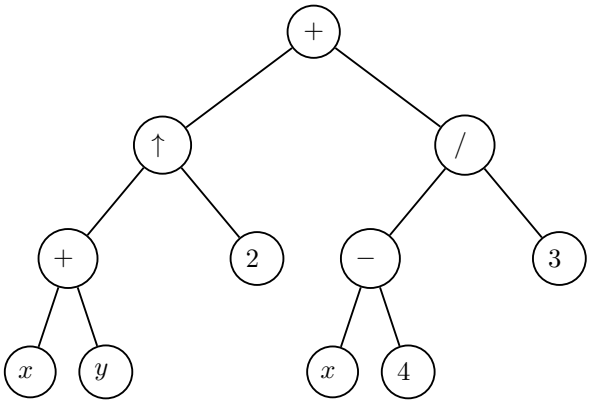
Binary Tree Example

LaTeX Code:

```
\begin{tikzpicture}
[
  level 1/.style = {sibling distance = 4 cm},
  level 2/.style = {sibling distance = 2.5cm},
```

```
  level 3/.style = {sibling distance = 1cm},
  every node/.style = {shape=circle, draw, line width=0.25mm, align=center}
]
\node {$+$}
    child {node {$\uparrow$}
      child {node {$+$}
        child {node {$x$}}
        child {node {$y$}}}
      child {node {$2$}}
      edge from parent [line width=0.25mm]}
    child {node {$/$}
      child {node {$-$}
        child {node {$x$}}
        child {node {$4$}}}
      child {node {$3$}}
    edge from parent [line width=0.25mm]};

\end{tikzpicture}
```

Results:



Writing Algorithms with Pseudocode

Required Packages

- algorithm
- algpseudocode with/without option “noend”

Binary Search

LaTeX Code:

```
\begin{algorithm}[H]
\begin{algorithmic}
\caption{Binary Search Algorithm}
\Procedure{binarySearch}{$x, a_1, a_2, \ldots, a_n, Loc$}
  \State $i \coloneqq 1$
  \State $j \coloneqq n$

  \While{$i < j$}
    \State $m \coloneqq \lfloor(i + j) / 2\rfloor$
    \If{$x > a_m$}
      \State $i \coloneqq m + 1$
    \Else
      \State $j \coloneqq m$
    \EndIf
  \EndWhile

```

```
  \If{$x = a_i$}
    \State $Loc \coloneqq i$
  \Else
    \State $Loc \coloneqq 0$
  \EndIf
\EndProcedure
\end{algorithmic}
\end{algorithm}
```

Results:

Algorithm 1 Binary Search Algorithm

procedure BINARYSEARCH($x, a_1, a_2, \ldots, a_n, Loc$)
  $i \coloneqq 1$
  $j \coloneqq n$
  while $i < j$ **do**
    $m \coloneqq \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ **then**
      $i \coloneqq m + 1$
    else
      $j \coloneqq m$
  if $x = a_i$ **then**
    $Loc \coloneqq i$
  else
    $Loc \coloneqq 0$

Bubble Sort

LaTeX Code:

```
\begin{algorithm}[H]
\caption{Bubble Sort Algorithm}
\begin{algorithmic}
\Procedure{bubbleSort}{$a_1, a_2, \ldots, a_n$}
  \For{$i \gets 1$ \textbf{to} $n - 1$}
    \For{$j \gets 1$ \textbf{to} $n - i$}
      \If{$a_j > a_{j+1}$}
        \State $a_j \leftrightarrow a_{j+1}$
      \EndIf
    \EndFor
  \EndFor
\EndProcedure
\end{algorithmic}
\end{algorithm}
```

Results:

Algorithm 2 Bubble Sort Algorithm

procedure BUBBLESORT($a_1, a_2, \ldots, a_n$)
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    for $j \leftarrow 1$ **to** $n - i$ **do**
      if $a_j > a_{j+1}$ **then**
        $a_j \leftrightarrow a_{j+1}$

Circuit Diagrams

Required Packages

- tikz

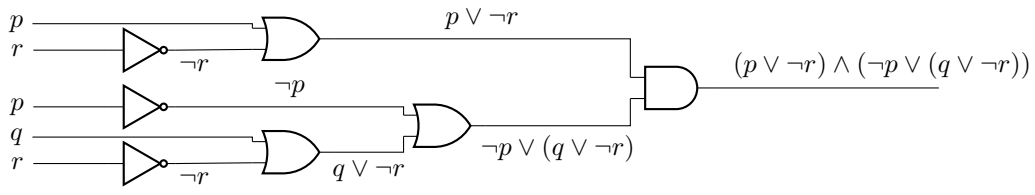
- circuitikz
- tkz-euclide
- Required Tikz Libraries: calc, positioning, intersections, quotes, decorations.markings, matrix, backgrounds

Example

LaTeX Code:

```
\begin{tikzpicture}

\ctikzset{
  logic ports=ieee,
  logic ports/scale=0.5,
}

% Logic ports

\node[not port] (NOTa) at (0,0){};
\node[not port] (NOTb) at (0,-0.75){};
\node[not port] (NOTc) at (0,-1.5){};
\node[or port] (ORa) at (2,0.15){};
\node[or port] (ORb) at (2,-1.35){};
\node[or port] (ORc) at (4,-1){};
\node[and port] (ANDa) at (7,-0.5){};

% Connections

\draw (ANDa.out) -- ++(3.0,0)node[near end,above]{$\left(p \lor \lnot r\right)
\land (\lnot p \lor (q \lor \lnot r))$};
\draw (ORc.out) -| node[near start,below]{$\lnot p \lor (q \lor \lnot r)$}(ANDa.in 2);
\draw (ORa.out) -| node[near start,above]{$p \lor \lnot r$}(ANDa.in 1);
\draw (NOTb.out) -| node[near start,above]{$\lnot p$}(ORc.in 1);
\draw (ORb.out) -| node[near start,below]{$q \lor \lnot r$}(ORc.in 2);
\draw (NOTc.out) -- node[near start,below]{$\lnot r$}(ORb.in 2);
\draw (NOTa.out) -- node[near start,below]{$\lnot r$}(ORa.in 2);
\draw (NOTa.in 1) -- ++(-1.0,0)node[left]{$r$};
\draw (NOTb.in 1) -- ++(-1.0,0)node[left]{$p$};
\draw (NOTc.in 1) -- ++(-1.0,0)node[left]{$r$};
\draw (-1.45,0.35) node[left]{$p$} -| (ORa.in 1);
\draw (-1.45,-1.15) node[left]{$q$} -| (ORb.in 1);

\end{tikzpicture}
```

Results:



Venn Diagrams

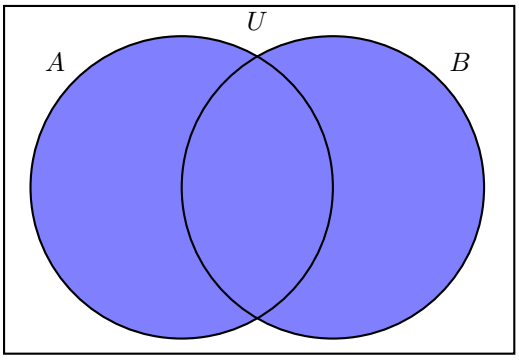
Set Union

LaTeX Code

```
\begin{tikzpicture}

% Set A
```

```
\node [circle,
    fill=blue!50,
    minimum size = 4cm,
    label={135:$A$}] (A) at (0,0){};

% Set B
\node [circle,
    fill=blue!50,
    minimum size = 4cm,
    label={45:$B$}] (B) at (2,0){};

\draw[black,thick] (-2.35,-2.2) rectangle (4.4,2.4);

% Circles outline
\draw[black,thick] (0,0) circle(2cm);
\draw[black,thick] (2,0) circle(2cm);

% Union text label
\node[black] at (1,2.2) {$U$};

\end{tikzpicture}
```

Results



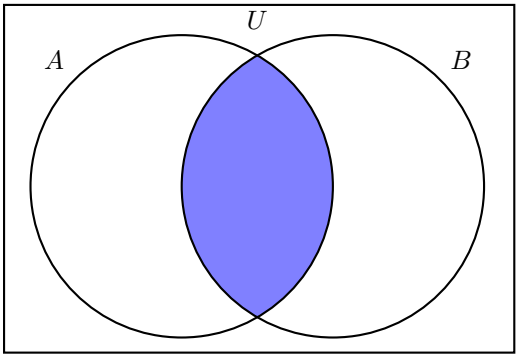
Set Intersection

LaTeX Code

```
\begin{tikzpicture}[thick,
    set/.style = {circle,
        minimum size = 4cm,
        color=black}]

\draw[black,thick] (-2.35,-2.2) rectangle (4.4,2.4);

% Set A
\node[set,label={135:$A$}] (A) at (0,0) {};

% Set B
\node[set,label={45:$B$}] (B) at (2,0) {};

% Intersection
\begin{scope}
    \clip (0,0) circle(2cm);
    \clip (2,0) circle(2cm);
    \fill[blue!50](0,0) circle(2cm);
\end{scope}

% Circles outline
\draw (0,0) circle(2cm);
\draw (2,0) circle(2cm);
```

```
\node[black] at (1,2.2) {$U$};

\end{tikzpicture}
```

Results

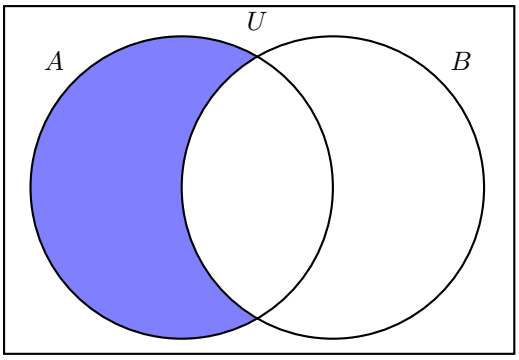


Set Difference

LaTeX Code

```
\begin{tikzpicture}[thick,
    set/.style = { circle, minimum size = 4cm}]

\draw[black,thick] (-2.35,-2.2) rectangle (4.4,2.4);

% Set A
\node[set,fill=blue!50,label={135:$A$}] (A) at (0,0) {};

% Set B
\node[set,fill=white,label={45:$B$}] (B) at (0:2) {};

% Circles outline
\draw (0,0) circle(2cm);
\draw (2,0) circle(2cm);

\node[black] at (1,2.2) {$U$};

\end{tikzpicture}
```

Results

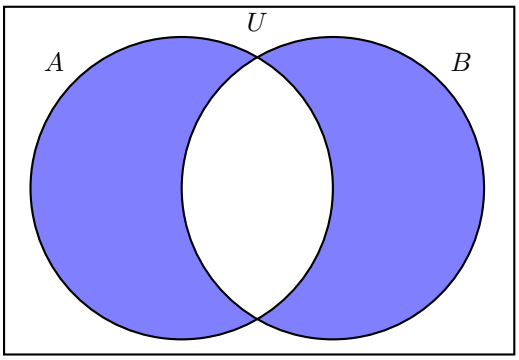


Symmetric Difference

LaTeX Code

```
\begin{tikzpicture}[thick,
    set/.style = {circle,
        minimum size = 4cm}]

\draw[black,thick] (-2.35,-2.2) rectangle (4.4,2.4);

% Set A
\node[set,fill=blue!50,label={135:$A$}] (A) at (0,0) {};

% Set B
\node[set,fill=blue!50,label={45:$B$}] (B) at (2,0) {};

% Intersection
\begin{scope}
    \clip (0,0) circle(2cm);
    \clip (2,0) circle(2cm);
    \fill[white](0,0) circle(2cm);
\end{scope}

% Circles outline
\draw (0,0) circle(2cm);
\draw (2,0) circle(2cm);

\node[black] at (1,2.2) {$U$};


\end{tikzpicture}
```

Results



Plotting Functions

You will need to use the tikz library: shapes.geometric

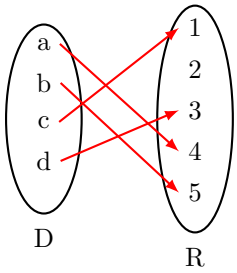
Domain and Range

LaTeX Code

```
\begin{tikzpicture}
  \node[ellipse,
    draw = black,
    thick,
    text = white,
    minimum width = 1cm,
    minimum height = 2.5cm] (A) at (0,0) {};
```

```
  \node[ellipse,
    draw = black,
    thick,
    minimum width = 1cm,
    minimum height = 3cm] (B) at (2,0) {};

  \node[anchor=north] at ([yshift=-2pt]A.south) {D};
  \node[anchor=north] at ([yshift=-2pt]B.south) {R};

  \node[anchor=north] at ([yshift=-2pt]A.north) (a) {a};
  \node[anchor=north] at ([yshift=-2pt]a.south) (b) {b};
  \node[anchor=north] at ([yshift=-2pt]b.south) (c) {c};
  \node[anchor=north] at ([yshift=-2pt]c.south) (d) {d};
  \node[anchor=north] at ([yshift=-2pt]B.north) (r1) {1};
  \node[anchor=north] at ([yshift=-2pt]r1.south) (r2) {2};
  \node[anchor=north] at ([yshift=-2pt]r2.south) (r3) {3};
  \node[anchor=north] at ([yshift=-2pt]r3.south) (r4) {4};

  \node[anchor=north]
    at ([yshift=-2pt]r4.south) (r5) {5};

  \draw[-latex, thick, red] (a.east) -- (r4.west) {};
  \draw[-latex, thick, red] (b.east) -- (r5.west) {};
  \draw[-latex, thick, red] (c.east) -- (r1.west) {};
  \draw[-latex, thick, red] (d.east) -- (r3.west) {};
\end{tikzpicture}
```

Results

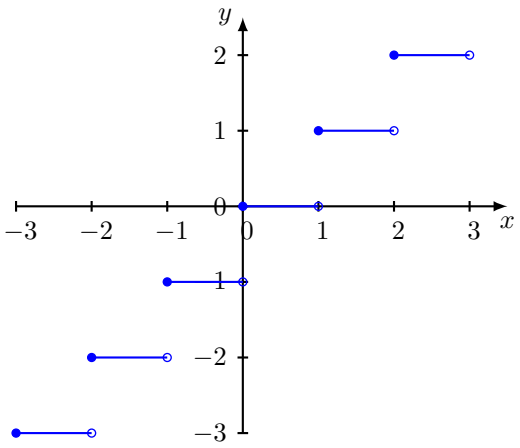


Plotting the Floor Function

LaTeX Code

```
\begin{tikzpicture}
    \tkzInit[xmin = -3, xmax = 3, ymin = -3, ymax = 2]
    \tkzDrawXY[color=black,thick]
    \tkzLabelXY[text=black,fill=none]
    \foreach \a in {-3,...,2}{
        \draw[blue,thick] (\a, \a) -- (\a + 1, \a);
        \node [circle, draw, thick, fill, line width = .5pt, color = blue, inner sep = 0pt, minimum size = 3pt] (ca) at (\a, \a) {};
        \node [circle, draw, thick, fill=none, line width = .5pt, color = blue, inner sep = 0pt, minimum size = 3pt] (ca) at (\a + 1, \a) {};
    }
\end{tikzpicture}
```

Results



Matrices

Required Packages

- amsmath

```
$$
\mathbf{A} = \begin{bmatrix}
             1 & 0 \\
             0 & 1 \\
             1 & 0
             \end{bmatrix},
\text{\hspace{1em}}
\mathbf{B} = \begin{bmatrix}
             1 & 1 & 0 \\
             0 & 1 & 1
             \end{bmatrix}
$$
```

Result:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Code Listings

Required Packages

- xcolor – if you want color listings
- lstlistings

Customizing

Codestyle and Colors

Colors and styles can be set in the preamble (after `\documentclass{...}` but before `\begin{document}`

```

% You can setup colors if you included the xcolor package

\definecolor{name}{rgb}{r,g,b}
```

Customization Options

- backgroundcolor - colour for the background. External color or xcolor package needed.
- commentstyle - style of comments in source language.
- basicstyle - font size/family/etc. for source (e.g. `basicstyle=\ttfamily\small`)
- keywordstyle - style of keywords in source language (e.g. `keywordstyle=\color{red}`)
- numberstyle - style used for line-numbers
- numbersep - distance of line-numbers from the code
- stringstyle - style of strings in source language
- showspaces - emphasize spaces in code (true/false)
- showstringspaces - emphasize spaces in strings (true/false)
- showtabs - emphasize tabulators in code (true/false)
- numbers - position of line numbers (left/right/none, i.e. no line numbers)
- prebreak - displaying mark on the end of breaking line (e.g. `prebreak=\raisebox{0ex}[0ex][0ex]{\ensuremath{\hookleftarrow}}`)
- captionpos - position of caption (t/b)
- frame - showing frame outside code (none/leftline/topline/bottomline/lines/single/shadowbox)
- breakwhitespace - sets if automatic breaks should only happen at whitespaces
- breaklines - automatic line-breaking
- keepspaces - keep spaces in the code, useful for indetation
- tabsize - default tabsize
- escapeinside - specify characters to escape from source code to LATEX (e.g. `escapeinside={\%*}{*)}`)
- rulecolor - Specify the colour of the frame-box

Supported Languages

- ABAP
- ACSL
- Ada
- Algol
- Ant
- Assembler
- Awk
- bash
- Basic
- C
- C++
- Caml
- CIL
- Clean
- Cobol
- Comal 80
- command.com
- Comsol
- csh
- Delphi
- Eiffel
- Elan
- erlang
- Euphoria
- Fortran
- GCL
- Gnuplot
- Haskell
- HTML
- IDL
- inform
- Java
- JVMIS
- ksh
- Lingo
- Lisp

- Logo
- make
- Mathematica
- Matlab
- Mercury
- MetaPost
- Miranda
- Mizar
- ML
- Modula-2
- MuPAD
- NASTRAN
- Oberon-2
- OCL
- Octave
- Oz
- Pascal
- Perl
- PHP
- PL/I
- Plasm
- PostScript
- POV
- Prolog
- Promela
- PSTricks
- Python
- R
- Reduce
- Rexx
- RSL
- Ruby
- S
- SAS
- Scilab
- sh
- SHELXL
- Simula
- SPARQL
- SQL
- tcl
- TeX
- VBScript
- Verilog
- VHDL
- VRML
- XML
- XSLT

Haskell Examples:

Linear Search

LaTeX Code:

```
\begin{lstlisting}[language=Haskell]
linSearch :: Eq a => [a] -> Int -> a -> Int
linSearch [] _ _     = 0
linSearch (y:ys) i x =
    if x == y then i
    else linSearch ys (i + 1) x
\end{lstlisting}
```

Results:

```haskell
linSearch :: Eq a => [a] -> Int -> a -> Int
linSearch [] _ _     = 0
linSearch (y:ys) i x =
    if x == y then i
    else linSearch ys (i + 1) x
```

Binary Search

LaTeX Code:

```
\begin{lstlisting}[language=Haskell]
binSearch :: (Ord a) => [a] -> a -> Int -> Int -> Int
binSearch arr x lo hi
  | hi < lo = -1
  | pivot > x = binSearch arr x lo (mid - 1)
  | pivot < x = binSearch arr x (mid + 1) hi
  | otherwise =  mid
  where
    mid = lo + (hi - lo) `div` 2
    pivot = arr!!mid
\end{lstlisting}
```

Results:

```haskell
binSearch :: (Ord a) => [a] -> a -> Int -> Int -> Int
binSearch arr x lo hi
  | hi < lo = -1
  | pivot > x = binSearch arr x lo (mid - 1)
  | pivot < x = binSearch arr x (mid + 1) hi
  | otherwise =  mid
  where
    mid = lo + (hi - lo) `div` 2
    pivot = arr!!mid
```

Merge Sort

LaTeX Code:

```
\begin{lstlisting}[language=Haskell]
merge :: (Ord a) => [a] -> [a] -> [a]
merge [] [] = []
merge [] ys = ys
merge xs [] = xs
merge allX@(x:xs) allY@(y:ys)
  | x > y     = y : merge allX ys
  | otherwise = x : merge xs allY

sort :: (Ord a) => [a] -> [a]
sort [] = []
sort [a] = [a]
sort [a,b]
  | a > b     = [b, a]
  | otherwise = [a, b]
sort list =
  let split = splitAt(length list `div` 2) list
      firstHalf  = sort (fst split)
      secondHalf = sort (snd split)
  in merge firstHalf secondHalf
\end{lstlisting}
```

Results:

```haskell
merge :: (Ord a) => [a] -> [a] -> [a]
merge [] [] = []
merge [] ys = ys
merge xs [] = xs
merge allX@(x:xs) allY@(y:ys)
  | x > y     = y : merge allX ys
  | otherwise = x : merge xs allY

sort :: (Ord a) => [a] -> [a]
sort [] = []
sort [a] = [a]
sort [a,b]
  | a > b     = [b, a]
  | otherwise = [a, b]
sort list =
  let split = splitAt(length list `div` 2) list
      firstHalf  = sort (fst split)
      secondHalf = sort (snd split)
  in merge firstHalf secondHalf
```

Quick Sort

LaTeX Code:

```
\begin{lstlisting}[language=Haskell]
quickSort :: Ord a => [a] -> [a]
quickSort [] = []
quickSort (pivot:xs) =
  quickSort [y | y <- xs, y < pivot]
  ++ [pivot]
  ++ quicksort [y | y <- xs, y >= pivot]
\end{lstlisting}
```

Results:

```haskell
quickSort :: Ord a => [a] -> [a]
quickSort [] = []
quickSort (pivot:xs) =
  quickSort [y | y <- xs, y < pivot]
  ++ [pivot]
  ++ quicksort [y | y <- xs, y >= pivot]
```