Part 2 of Combinatorial Optimisation and Constraint Programming (1DL441)
Uppsala University – Autumn 2019
Report for Assignment n by Team t

Clara CLÄVER          Whiz KIDD
24th October 2019

This document shows the ingredients of a good assignment report for this part of the course. The \LaTeX{} source code of this document exemplifies almost everything you need to know about \LaTeX{} in order to typeset a professional-looking report (for this course). Use it as a starting point for imitation and delete everything irrelevant. The usage of \LaTeX{} is optional, but highly recommended, for reasons that will soon become clear to those who have never used it before: any learning time is outside the time budget of this course, but will hugely pay off, if not in this course then in the next courses you take and when writing a thesis or other report.

Address each task of each problem, using the numbering and ordering in which the problems and tasks appear in the assignment statement. For each task requiring a model, follow the advice and structure of Section C. Delete all unnecessary text to save trees and suitably replace all other text in this document.

Part 1

The Sudoku Problem

All experiments were run under Linux Ubuntu 18.04 (64 bit) on an Intel Xeon E5520 of 2.27 GHz, with 4 processors of 4 cores each, with a 70 GB RAM and an 8 MB L3 cache (a ThinLinc computer of the IT department). (If different hardware was used for different tasks, then justify this and replicate a paragraph like this one within each relevant part or section.)

A Heading for Task A

(Write your answer for Task A here.)

B Use Appropriate Headings for the Sections

(Remember that this document is a guideline, so you should change its text when appropriate.)

\footnote{1Hint: Under Linux, do \texttt{lscpu} to find this information. Under macOS, you find this information via “About This Mac” in the Apple menu.}
C  Model

(You need not describe a problem specified in an assignment statement. Assume the assignment was to model the Sudoku problem in Topic 1.) Our model, with the prescribed comments, is uploaded as file sudoku.cpp and only its important parts (namely the parameters, decision variables, constraints, objective, and brancher, but neither the solution printing nor any boilerplate code, so as to save trees: see the \LaTeX{} source code of this document in order to learn how to import a sub-model after clustering its important parts) are given in Listing 1. A Gecode model must include comments that explain:

- the parameters that are not part of the possibly provided skeleton code;
- the decision variables, and which problem constraints, if any, they automatically enforce;
- the redundant decision variables; if none, then justify this below.
- the problem constraints not enforced by the choice of decision variables;
- the objective, including possibly the objective function;
- the channelling constraints; if none, then justify this below.
- the implied constraints; if none, then justify this below.
- the symmetry-breaking constraints; if none, then justify this below.
- a reason for the chosen consistency for each constraint; and
- a reason for the chosen search strategy.

The quality of model comments is considered while grading. You must include your model in your report in addition to uploading it as a separate file (so that we can run it).

Listing 1: Core of a Gecode model for the Sudoku problem

```cpp
// Copyright:
// Mikael Lagerkvist, 2005
// Guido Tack, 2005
// Christian Schulte, 2005
// [deleted original lines 12-54]

class Sudoku : public Script {
protected:
  const int n; // the width (and height) of a block of the Sudoku
  IntVarArray x;
public:
  // Constructor:
  Sudoku(const BranchOptions& opt)
    : Script(opt),
      n(example_size(examples[opt.size()])),
      // the Sudoku as a 1D array of n^4 variables in 1..n^2:
      x(*this, n*n*n*n, 1, n*n) {
    const int nn = n*n; // the width (and height) of the Sudoku itself
    // m[r,c] = the value at row r and column c of the Sudoku x:
```
Matrix<IntVarArray> m(x, nn, nn);

// All values in each row and column are different:
for (int i=0; i<nn; i++) {
    distinct(*this, m.row(i), opt.ipl());
    distinct(*this, m.col(i), opt.ipl());
}

// All values in each block are different:
for (int i=0; i<nn; i+=n) {
    for (int j=0; j<nn; j+=n) {
        distinct(*this, m.slice(i, i+n, j, j+n), opt.ipl());
    }
}

// Fix the variables corresponding to the hints:
for (int i=0; i<nn; i++)
    for (int j=0; j<nn; j++)
        if (int v = sudokuField(examples[opt.size()], nn, i, j))
            rel(*this, m(i,j), IRT_EQ, v);

// Search alternatives (see report):
if (opt.branching() == size_min) {
    branch(*this, x, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
} else if (opt.branching() == AFC) {
    branch(*this, x, INT_VAR_AFC_SIZE_MAX(), INT_VAL_MIN());
}

(Provide, here in the report, a more detailed explanation (i) why each redundant decision variable is deemed useful, (ii) why each channelling, implications, and symmetry-breaking constraint is correct, as well as (iii) how each consistency and the search strategy were chosen. Use any combination of in-lined code, mathematical notation, and plain English to explain your choices.)

Redundant Decision Variables and Channelling Constraints. We were unable to define any redundant decision variables where experiments (not reported here) revealed that these variables and appropriate channelling constraints accelerate our model.

Implied Constraints. We were unable to design any implied constraints where experiments (not reported here) revealed that they accelerate our model.

Symmetry-Breaking Constraints. We were unable to design any symmetry-breaking constraints where experiments (not reported here) revealed that they accelerate our model.

Consistency. We request domain consistency (DC) for all the distinct(...) constraints because the Sudoku problem is highly combinatorial and Gecode has an efficient DC propagator for that predicate: experiments (not reported here) revealed that this accelerates our model.

Search Strategy. Our search strategy is INT_VAR_AFC_SIZE_MAX with INT_VAL_MIN: experiments (not reported here) revealed that this scales very well in practice.
Part 2
Consistency, Propagation, and Search

First CSP

Consider the following CSP $P_1$ and store $s_1$:

\[
\begin{align*}
  c_1 & : x < y \\
  c_2 & : x + y < 8 \\
  c_3 & : y > z \\
  s_1 : & \{x \mapsto \{2, \ldots, 10\}, y \mapsto \{0, \ldots, 10\}, z \mapsto \{3, \ldots, 10\}\}
\end{align*}
\]

A Pre-Search Propagation

Here are our answers to the three sub-tasks on pre-search propagation.

A.a Propagator Conditions

The following are the sets of conditions that should trigger the scheduling of idempotent propagators achieving \textit{bounds($\mathbb{Z}$)} consistency for the arithmetic constraints and \textit{domain} consistency for the other constraints, as a strictly stronger store might then be obtained:

- $\text{PropConds}(p_1) = \{?\}$
- $\text{PropConds}(p_2) = \{?\}$
- $\text{PropConds}(p_3) = \{?\}$

where $p_i$ is the propagator for constraint $c_i$, with $1 \leq i \leq 3$.

A.b Root Fixpoint

Table 1 gives the initialisation (in the first row) and every pre-search iteration of the call $\text{Propagate}([p_1, p_2, p_3], [p_1, p_2, p_3], s_1)$, assuming that we post the constraints in the textual order in which they appear in the CSP $P_1$ above, that we handle the decision variables in the textual order in which they appear in the store $s_1$ above, and that we use a first-in first-out queue (FIFO) for implementing the set $Q$ of propagators that are not known to be at fixpoint.

A.c Switching to Domain Consistency

When instead achieving domain consistency for \textit{all} the constraints, \textit{all hell breaks loose, because there is nothing new under the sun.}

B Search

The search tree in Figure 1 has pairs of non-subsumed propagator sets and constraint stores as nodes, and decisions (which are propagator sets) as labelled arcs. It is obtained upon largest-minimum variable selection and bottom-up domain splitting.

Table 2 traces the execution of each call $\text{Propagate}(R \cup \{p_{d(x,y,z)}\}, [p_{d(x,y,z)}], s)$ from a node $(R, s)$ upon a decision $\{p_{d(x,y,z)}\}$, where $p_{d(x,y,z)}$ is a propagator for a constraint $d(x, y, z)$.
<table>
<thead>
<tr>
<th>Chosen prop.</th>
<th>Resulting store</th>
<th>Status message</th>
<th>Modification events</th>
<th>Dependent prop.s DepProps</th>
<th>Non-subsumed propagators R</th>
<th>FIFO queue Q of propagators</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>$x \mapsto {2..10}$, $y \mapsto {0..10}$, $z \mapsto {3..10}$</td>
<td>(none)</td>
<td>(none)</td>
<td>(none)</td>
<td>${p_1, p_2, p_3}$</td>
<td>$[p_1, p_2, p_3]$</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$x \mapsto {\ldots}$, $y \mapsto {\ldots}$, $z \mapsto {\ldots}$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1: Computing the root fixpoint of the search tree for CSP $P_1$

<table>
<thead>
<tr>
<th>Chosen prop.</th>
<th>Resulting store</th>
<th>Status message</th>
<th>Modification events</th>
<th>Dependent prop.s DepProps</th>
<th>Non-subsumed propagators R</th>
<th>FIFO queue Q of propagators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{d(x,y,z)}$</td>
<td>$x \mapsto {\ldots}$, $y \mapsto {\ldots}$, $z \mapsto {\ldots}$</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 2: Computing the non-root fixpoints of the search tree for CSP $P_1$
Figure 1: Search tree for CSP $P_1$
Second CSP

Consider the following CSP $P_2$ and store $s_2$:

\[
\begin{align*}
  c_4 &: \text{element}([2, 1, 8, 2, 1, 0], x, y) \\
  c_5 &: x + y \leq 7 \\
  c_6 &: z \cdot z \leq 5 \\
  c_7 &: \text{distinct}([x, y, z]) \\
  s_2 &: \{x, y, z \mapsto \{1..9\}\}
\end{align*}
\]

A Pre-Search Propagation

Here are our answers to the three sub-tasks on pre-search propagation.

A.a Propagator Conditions

The following are the sets of conditions that should trigger the scheduling of idempotent propagators achieving $\text{bounds}(\mathbb{Z})$ consistency for the arithmetic constraints and domain consistency for the other constraints, as a strictly stronger store might then be obtained:

- $\text{PropConds}(p_4) = \{?\}$
- $\text{PropConds}(p_5) = \{?\}$
- $\text{PropConds}(p_6) = \{?\}$
- $\text{PropConds}(p_7) = \{?\}$

where $p_i$ is the propagator for constraint $c_i$, with $4 \leq i \leq 7$.

A.b Root Fixpoint

Table 3 gives the initialisation (in the first row) and every pre-search iteration of the call $\text{Propagate}(\{p_4, p_5, p_6, p_7\}, [p_4, p_5, p_6, p_7], s_2)$, assuming that we post the constraints in the textual order in which they appear in the CSP $P_2$ above, that we handle the decision variables in the textual order in which they appear in the store $s_2$ above, and that we use a first-in first-out queue (FIFO) for implementing the set $Q$ of propagators that are not known to be at fixpoint.

A.c Switching to Domain Consistency

When instead achieving domain consistency for all the constraints, all hell breaks loose, because there is nothing new under the sun.

B Search

The search tree in Figure 2 has pairs of non-subsumed propagator sets and constraint stores as nodes, and decisions (which are constraints) as labelled arcs. It is obtained upon largest-minimum variable selection and bottom-up domain splitting.

Table 4 traces the execution of each call $\text{Propagate}(R \cup \{p_d(x, y, z)\}, [p_d(x, y, z)], s)$ from a node $\langle R, s \rangle$ upon a decision $\{p_d(x, y, z)\}$, where $p_d(x, y, z)$ is a propagator for a constraint $d(x, y, z)$. 
### Table 3: Computing the root fixpoint of the search tree for CSP $P_2$

<table>
<thead>
<tr>
<th>Chosen prop.</th>
<th>Resulting store</th>
<th>Status message</th>
<th>Modification events</th>
<th>Dependent prop.s $\text{DepProps}$</th>
<th>Non-subsumed propagators $R$</th>
<th>FIFO queue $Q$ of propagators</th>
</tr>
</thead>
<tbody>
<tr>
<td>(none)</td>
<td>$x \mapsto {1..9}$, $y \mapsto {1..9}$, $z \mapsto {1..9}$</td>
<td>(none)</td>
<td>(none)</td>
<td>(none)</td>
<td>${p_4, p_5, p_6, p_7}$</td>
<td>$[\ldots]$</td>
</tr>
<tr>
<td>$p_i$</td>
<td>$x \mapsto {\ldots}$, $y \mapsto {\ldots}$, $z \mapsto {\ldots}$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$[\ldots]$</td>
</tr>
</tbody>
</table>

### Table 4: Computing the non-root fixpoints of the search tree for CSP $P_2$

<table>
<thead>
<tr>
<th>Chosen prop.</th>
<th>Resulting store</th>
<th>Status message</th>
<th>Modification events</th>
<th>Dependent prop.s $\text{DepProps}$</th>
<th>Non-subsumed propagators $R$</th>
<th>FIFO queue $Q$ of propagators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{d(x,y,z)}$</td>
<td>$x \mapsto {\ldots}$, $y \mapsto {\ldots}$, $z \mapsto {\ldots}$</td>
<td>$\ldots$</td>
<td>${\ldots}$</td>
<td>${\ldots}$</td>
<td>${\ldots}$</td>
<td>$[\ldots]$</td>
</tr>
</tbody>
</table>
Figure 2: Search tree for CSP $P_2$
Feedback to the Teachers

(Please write a paragraph, which will not be graded, describing your experience with this assignment: which aspects were too difficult or too easy, and which aspects were interesting or boring? This may help us improve the course for the next year.)

References

Checklist before Submitting

In order to protect yourself against an unnecessary loss of points, use the following checklist before submitting:

- Crosscheck your models against the checklists at the ends of the slides of Topics 2 and 3.
- Crosscheck your report against the assignment instructions.
- Remember that when submitting you implicitly certify (a) that your report and all its uploaded attachments were produced solely by your team, except where explicitly stated otherwise and clearly referenced, (b) that each teammate can individually explain any part starting from the moment of submitting your report, and (c) that your report and attachments are not freely accessible on a public repository.
- Spellcheck all documents, including the comments in the source code.
- Proofread, if not grammar-check, your report at least once per teammate.

More \LaTeX\ and Technical Writing Advice

Unnumbered itemisation (only to be used when the order of the items does not matter):

- Unnumbered displayed formula:

\[ E = m \cdot c^2 \]

- Numbered displayed formula, which is cross-referenced somewhere:

\[ E = m \cdot c^2 \]

- Formula — the same as formula [B] — spanning more than one line:

\[
E = m \cdot c^2
\]

Numbered itemisation (only to be used when the order of the items does matter):

1. First do this.
2. Then do that.
3. If we are not finished, then go back to Step 2 else stop.
Figure 3: A binary search tree (on the left), a binary min-heap (in the middle), and a binomial tree of rank 3 (on the right).

```
1: function f(n)               // optional comment
2: if n < 0 then                // optional comment
3:   n := −2 \cdot n              // n ≥ 0
4: else
5:   n := 3 \cdot n
6: while n > 0 do
7:   n := n − 1
8: return n
```

**Algorithm 1:** Silly algorithm


Use \texttt{\textit{...}} in mathematical mode for each multiple-letter identifier in order to avoid typesetting the identifier like the product of single-letter ones. For example, note the typographic difference between the identifier \texttt{WL}, obtained through $\mathit{WL}$, and the product \texttt{WL}, where there is a small space between the \texttt{W} and the \texttt{L}, obtained through $\texttt{WL}$.

Do \texttt{not} use programming-language-style lower-ASCII notation (such as ! for negation, \texttt{&&} for conjunction, || for disjunction, and the equality sign = for assignment) in algorithms or formulas (but rather use $\neg$ or \texttt{not}, \texttt{\&\&} or \texttt{and}, $\texttt{||}$ or \texttt{or}, and $\leftarrow$ or $\Leftrightarrow$, respectively), as this testifies to a very strong confusion of concepts.

Figures can be imported with \texttt{\includegraphics} or drawn inside the \LaTeX source code using the highly declarative notation of the \texttt{tikz} package: see Figure 3 for sample drawings. It is perfectly acceptable in this course to include scans or photos of drawings that were carefully done by hand.

Algorithms can be typeset as pseudo-code as exemplified in Algorithm 1; study its \LaTeX source code.

If you are not sure whether you will stick to your current choice of notation or terminology, then introduce a new (possibly parametric) command. For example, upon

```
\newcommand{\Cardinality}[1]{\left\lvert#1\right\rvert}
```

\footnote{Use footnotes very sparingly, and note that footnote pointers are never preceded by a space and always glued immediately behind the punctuation, if there is any.}
the formula $\text{Cardinality}(S)$ typesets the cardinality of set $S$ as $|S|$ with autosized vertical bars and proper spacing, but upon changing the definition of that parametric command to

\newcommand{\Cardinality}[1]{\# #1}

and recompiling, the formula $\text{Cardinality}(S)$ typesets the cardinality of set $S$ as $\#S$. Similarly, upon

\newcommand{\MiniZinc}{\textit{Mini-Zinc}}

the text \MiniZinc\ typesets into MiniZinc, hyphenation being only possible in the middle, but upon changing the definition of that non-parametric command to

\newcommand{\MiniZinc}{\textsc{Mini-Zinc}}

and recompiling, the text \MiniZinc\ typesets into MINIZINC. You can thus obtain an arbitrary number of changes in the document with a constant-time change in its source code, rather than having to perform a linear-time find-and-replace operation within the source code, which is painstaking and error-prone. The imported file macros.tex has a lot of useful predefined commands about mathematics, CP, Gecode, modelling, MiniZinc, and algorithms.

Use commands on positioning (such as \hspace, \vspace, and \noindent) and appearance (such as \small for reducing the font size, and \textit for italics) very sparingly, and ideally only in (parametric) commands, as the very idea of mark-up languages such as \LaTeX is to let the class designer (usually a trained professional typesetter) decide on where things appear and how they look. For example, \textbf{emph} (for emphasis) compiles (outside italicised environments, such as theorem) into italics under the article class used for this document, but it may compile into boldface under some other class.

If you do not (need to) worry about how things look, then you can fully focus on what you are trying to express!

Note that no absolute numbers are used in the \LaTeX source code for any of the references inside this document. For ease of maintenance, \label is used for giving a label to something that is automatically numbered (such as an algorithm, equation, figure, footnote, item, line, part, section, subsection, or table), and \ref is used for referring to a label. An item in the bibliography file is referred to by \cite instead. Upon changing the text, it suffices to recompile, once or twice, and possibly to run BibTeX again, in order to update all references consistently.

Always write Table~\ref{tab:maths} instead of Table \ref{tab:maths}, by using the non-breaking space (which is typeset as the tilde ~) instead of the normal space, because this avoids that a cross-reference is spread across a line break, as for example in “Table 5”, which is considered poor typesetting.

The rules of English for how many spaces to use before and after various symbols are given in Table 6. Beware that they may be very different from the rules in your native language.

\footnote{Feel free to report to the head teacher any other features that you would have liked to see discussed and exemplified in this template document.}
<table>
<thead>
<tr>
<th>Topic</th>
<th>[\text{LaTeX code}]</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greek letter</td>
<td>$\Theta, \Omega, \epsilon$</td>
<td>$\Theta$, $\Omega$, $\epsilon$</td>
</tr>
<tr>
<td>multiplication</td>
<td>$m \cdot n$</td>
<td>$m \cdot n$</td>
</tr>
<tr>
<td>division</td>
<td>$\frac{m}{n}, m \div n$</td>
<td>$\frac{m}{n}, m \div n$</td>
</tr>
<tr>
<td>rounding down</td>
<td>$\lfloor n \rfloor$</td>
<td>$\lfloor n \rfloor$</td>
</tr>
<tr>
<td>rounding up</td>
<td>$\lceil n \rceil$</td>
<td>$\lceil n \rceil$</td>
</tr>
<tr>
<td>binary modulus</td>
<td>$m \mod n$</td>
<td>$m \mod n$</td>
</tr>
<tr>
<td>unary modulus</td>
<td>$m \equiv n \mod \ell$</td>
<td>$m \equiv n \mod \ell$</td>
</tr>
<tr>
<td>root</td>
<td>$\sqrt[n]{n}, \sqrt[n]{3}$</td>
<td>$\sqrt[n]{n}, \sqrt[n]{3}$</td>
</tr>
<tr>
<td>exponentiation, superscript</td>
<td>$n^i$</td>
<td>$n^i$</td>
</tr>
<tr>
<td>subscript</td>
<td>$n_{i}$</td>
<td>$n_{i}$</td>
</tr>
<tr>
<td>overline</td>
<td>$\overline{n}$</td>
<td>$\overline{n}$</td>
</tr>
<tr>
<td>base 2 logarithm</td>
<td>$\log_2 n$</td>
<td>$\log_2 n$</td>
</tr>
<tr>
<td>base $b$ logarithm</td>
<td>$\log_b n$</td>
<td>$\log_b n$</td>
</tr>
<tr>
<td>binomial</td>
<td>$\binom{n}{k}$</td>
<td>$\binom{n}{k}$</td>
</tr>
<tr>
<td>sum</td>
<td>$\sum_{i=1}^n i$</td>
<td>$\sum_{i=1}^n i$</td>
</tr>
<tr>
<td>numeric comparison</td>
<td>$\leq, &lt;, =, \neq, &gt;, \geq$</td>
<td>$\leq, &lt;, =, \neq, &gt;, \geq$</td>
</tr>
<tr>
<td>non-numeric comparison</td>
<td>$\prec, \nprec, \preceq, \succeq$</td>
<td>$\prec, \nprec, \preceq, \succeq$</td>
</tr>
<tr>
<td>extremum</td>
<td>$\min, \max, +\infty, \bot, \top$</td>
<td>$\min, \max, +\infty, \bot, \top$</td>
</tr>
<tr>
<td>function, tuple</td>
<td>$f: A \to B, o \to$</td>
<td>$f: A \to B, o \to$</td>
</tr>
<tr>
<td>sequence, tuple</td>
<td>$\langle a, b, c \rangle$</td>
<td>$\langle a, b, c \rangle$</td>
</tr>
<tr>
<td>set</td>
<td>${a, b, c}, \emptyset, \mathbb{N}$</td>
<td>${a, b, c}, \emptyset, \mathbb{N}$</td>
</tr>
<tr>
<td>set membership</td>
<td>$\in, \notin$</td>
<td>$\in, \notin$</td>
</tr>
<tr>
<td>set comprehension</td>
<td>${i \mid 1 \leq i \leq n}$</td>
<td>${i \mid 1 \leq i \leq n}$</td>
</tr>
<tr>
<td>set operation</td>
<td>$\cup, \cap, \setminus, \times$</td>
<td>$\cup, \cap, \setminus, \times$</td>
</tr>
<tr>
<td>set comparison</td>
<td>$\cupset, \subseteq, \nsubseteq$</td>
<td>$\cupset, \subseteq, \nsubseteq$</td>
</tr>
<tr>
<td>logic quantifier</td>
<td>$\forall, \exists, \nexists$</td>
<td>$\forall, \exists, \nexists$</td>
</tr>
<tr>
<td>logic connective</td>
<td>$\land, \lor, \neg, \Rightarrow$</td>
<td>$\land, \lor, \neg, \Rightarrow$</td>
</tr>
<tr>
<td>logic</td>
<td>$\models, \equiv, \vdash$</td>
<td>$\models, \equiv, \vdash$</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>$&amp;$, $#$, $\approx$, $\sim$, $\ell$</td>
<td>$&amp;$, $#$, $\approx$, $\sim$, $\ell$</td>
</tr>
<tr>
<td>dots (context-sensitive)</td>
<td>$\ldots, \cdots, \vdots, \ddots$</td>
<td>$\ldots, \cdots, \vdots, \ddots$</td>
</tr>
<tr>
<td>parentheses (autosizing)</td>
<td>$\left(m^n\right), \left(m^n\right)$</td>
<td>$\left(m^n\right), \left(m^n\right)$</td>
</tr>
<tr>
<td>identifier of &gt; 1 character</td>
<td>$\mathit{identifier}$</td>
<td>$\mathit{identifier}$</td>
</tr>
<tr>
<td>hyphen, $n$-dash, $m$-dash, minus</td>
<td>$-, --, ---, ___$</td>
<td>$-, --, ---, ___$</td>
</tr>
</tbody>
</table>

Table 5: The typesetting of elementary mathematics. Note very carefully when italics are used by \textit{LaTeX} and when not, as well as all the horizontal and vertical spacing performed by \textit{LaTeX}.

<table>
<thead>
<tr>
<th>number of spaces after</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of spaces before</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>/ -</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( (</td>
</tr>
</tbody>
</table>

Table 6: Spacing rules of English