The algorithms bundle*

Rogério Brito
rbrito@ime.usp.br

August 24, 2009

Contents

List of Algorithms

1 Introduction

This package provides two environments, `algorithmic` and `algorithm`, which are designed to be used together but may, depending on the necessities of the user, be used separately.

*This document corresponds to algorithms v0.1, dated 2009/08/24.

The `algorithmic` environment provides an environment for describing algorithms and the `algorithm` environment provides a "float" wrapper for algorithms (implemented using `algorithmic` or some other method at the users's option). The reason for two environments being provided is to allow the user maximum flexibility.

Currently, this package consists of the following files:

- `algorithms.ins`: the driver file
- `algorithms.dtx`: the source file
- `COPYING`: the license file
- `README`: remarks about the package
- `THANKS`: mentions of thanks for contributors to the package

Starting with with the 2009-08-24 release, the package is now versioned and this document corresponds to version v0.1.

If you use this package, the author would kindly appreciate if you mentioned it in your documents, so as to let the package be better known and have more contributors, to make it better for the community itself. This is *not* required by the license: it's just a friendly request.

2 Installation

The installation procedure of algorithms follows the usual practice of packages shipped with a pair of `.ins`/`.dtx`—simply type the comand:

```
latex algorithms.ins
```

and the `.sty` files will be generated. Copy them to a place that is referenced by your LaTeX distribution. To generate the documentation, type:

```
latex algorithms.dtx
```

3 The `algorithmic` Environment

Within an `algorithmic` a number of commands for typesetting popular algorithmic constructs are available. In general, the commands provided can be arbitrarily nested to describe quite complex algorithms. An optional argument to the `\begin{algorithmic}` statement can be used to turn on line numbering by giving a positive integer indicating the required frequency of line numbering. For example, `\begin{algorithmic}[5]` would cause every fifth line to be numbered.

3.1 The Simple Statement

The simple statement takes the form

```
\STATE <text>
```

and is used for simple statements. For example,

```
\begin{algorithmic}
  \STATE $S \leftarrow 0$
\end{algorithmic}
```

would produce

$S \leftarrow 0$

With line numbering selected for every line, using,

```
\begin{algorithmic}[1]
  \STATE $S \leftarrow 0$
\end{algorithmic}
```

we would get

1: $S \leftarrow 0$

Warning *For users of earlier versions of* `algorithmic` *this construct is a cause of an incompatibility. In the earlier version, instead of starting simple statements with the* `\STATE` *command, simple statements were entered as free text and terminated with* `\\` *command. Unfortunately, this simpler method failed to survive the modifications necessary for statement numbering. However, the* `\\` *command can still be used to force a line break within a simple statement.*

3.2 The *if-then-else* Statement

The *if-then-else* construct takes the forms:

```
\IF{<condition>} <text> \ENDIF
\IF{<condition>} <text1> \ELSE <text2> \ENDIF
\IF{<condition1>} <text1> \ELSIF{<condition2>} <text2> \ELSE <text3> \ENDIF
```

In the third of these forms there is no limit placed on the number of `\ELSIF{<condition>}` that may be used. For example,

```
\begin{algorithmic}
\IF{some condition is true}
\STATE do some processing
\ELSIF{some other condition is true}
\STATE do some different processing
\ELSIF{some even more bizarre condition is met}
```

```
\STATE do something else
\ELSE
\STATE do the default actions
\ENDIF
\end{algorithmic}
```

would produce

if some condition is true **then**
  do some processing
else if some other condition is true **then**
  do some different processing
else if some even more bizarre condition is met **then**
  do something else
else
  do the default actions
end if

with appropriate indentations.

3.3 The *for* Loop

The *for* loop takes two forms. Namely:

```
\FOR{<condition>} <text> \ENDFOR
\FORALL{<condition>} <text> \ENDFOR
```

For example,

```
\begin{algorithmic}
\FOR{$i=0$ to $10$}
\STATE carry out some processing
\ENDFOR
\end{algorithmic}
```

produces

for $i = 0$ to 10 **do**
  carry out some processing
end for

and

```
\begin{algorithmic}[1]
\FORALL{$i$ such that $0\leq i\leq 10$}
\STATE carry out some processing
\ENDFOR
\end{algorithmic}
```

produces

1: **for all** i such that $0 \leq i \leq 10$ **do**
2: carry out some processing
3: **end for**

3.3.1 The *to* Connective

As may be clear from the usage of loops above, we usually want to specify ranges over which a variable will assume values. To help make this typographically distinct, the `algorithmic` package now supports the **to** connective, which can be used like:

```
\begin{algorithmic}
\FOR{$i=0$ \TO $10$}
\STATE carry out some processing
\ENDFOR
\end{algorithmic}
```

to produce the output

for $i = 0$ **to** 10 **do**
carry out some processing
end for

3.4 The *while* Loop

The *while* loop takes the form

```
\WHILE{<condition>} <text> \ENDWHILE
```

For example,

```
\begin{algorithmic}
\WHILE{some condition holds}
\STATE carry out some processing
\ENDWHILE
\end{algorithmic}
```

produces

while some condition holds **do**
carry out some processing
end while

3.5 The *repeat-until* Loop

The *repeat-until* loop takes the form.

```
\REPEAT <text> \UNTIL{<condition>}
```

For example,

```
\begin{algorithmic}
\REPEAT
\STATE carry out some processing
\UNTIL{some condition is met}
\end{algorithmic}
```

produces

repeat
carry out some processing
until some condition is met

3.6 The Infinite Loop

The infinite loop takes the form.

```
\LOOP <text> \ENDLOOP
```

For example,

```
\begin{algorithmic}
\LOOP
\STATE this processing will be repeated forever
\ENDLOOP
\end{algorithmic}
```

produces

loop
this processing will be repeated forever
end loop

3.7 The Logical Connectives

The connectives **and**, **or**, **xor** and **not** can be used in boolean expressions in the familiar, expected way:

```
<expression> \AND <expression>
<expression> \OR <expression>
<expression> \XOR <expression>
\NOT <expression>
```

according to their arity.[1] For example,

```
\begin{algorithmic}
```

[1] But there is nothing that prevents the user from violating the arity, from a syntatic point of view.

```
\IF{\NOT ($year \bmod 400$ \XOR $year \bmod 100$ \XOR $year \bmod 4$)}
\STATE $year$ does not represent a leap year.
\ENDIF
\end{algorithmic}
```

produces

if not (*year* mod 400 **xor** *year* mod 100 **xor** *year* mod 4) **then**
  year does not represent a leap year.
end if

3.8 The Precondition

The precondition (that must be met if an algorithm is to correctly execute) takes the form:

```
\REQUIRE <text>
```

For example,

```
\begin{algorithmic}
\REQUIRE $x \neq 0$ and $n \geq 0$
\end{algorithmic}
```

produces

Require: $x \neq 0$ and $n \geq 0$

3.9 The Postcondition

The postcondition (that must be met after an algorithm has correctly executed) takes the form:

```
\ENSURE <text>
```

For example,

```
\begin{algorithmic}
\ENSURE $x \neq 0$ and $n \geq 0$
\end{algorithmic}
```

produces

Ensure: $x \neq 0$ and $n \geq 0$

3.10 Returning Values

The `algorithmic` environment offers a special statement for explicitly returning values in algorithms. It has the syntax:

```
\RETURN <text>
```

For example,

```
\begin{algorithmic}
\RETURN $(x+y)/2$
\end{algorithmic}
```

produces

return $(x + y)/2$

3.10.1 The "true" and "false" Values

Since many algorithms have the necessity of returning *true* or *false* values, `algorithms`, starting with version 2006-06-02, includes the keywords `\TRUE` and `\FALSE`, which are intented to print the values in a standard fashion, like the following snippet of an algorithm to decide if an integer n is even or odd:

```
\begin{algorithmic}
  \IF{$n$ is odd}
     \RETURN \TRUE
  \ELSE
     \RETURN \FALSE
  \ENDIF
\end{algorithmic}
```

The code above produces the following output:

if n is odd **then**
  return **true**
else
  return **false**
end if

3.11 Printing Messages

Another feature of the `algorithmic` environment is that it currently provides a standard way of printing values (which is an operation used enough to merit its own keyword). It has the syntax:

```
\PRINT <text>
```

For example,

```
\begin{algorithmic}
\PRINT \texttt{``Hello, World!''}
\end{algorithmic}
```

produces

print ``Hello, World!''

3.12 Comments

Comments may be inserted at most points in an algorithm using the form:

```
\COMMENT{<text>}
```

For example,

```
\begin{algorithmic}
\STATE do something \COMMENT{this is a comment}
\end{algorithmic}
```

produces

do something {this is a comment}

Because the mechanisms used to build the various algorithmic structures make it difficult to use the above mechanism for placing comments at the end of the first line of a construct, the commands `\IF`, `\ELSIF`, `\ELSE`, `\WHILE`, `\FOR`, `\FORALL`, `\REPEAT` and `\LOOP` all take an optional argument which will be treated as a comment to be placed at the end of the line on which they appear. For example,

repeat {this is comment number one}
  if condition one is met **then** {this is comment number two}
    do something
  else if condition two is met **then** {this is comment number three}
    do something else
  else {this is comment number four}
    do nothing
  end if
until hell freezes over

3.13 An Example

The following example demonstrates the use of the `algorithmic` environment to describe a complete algorithm. The following input

```
\begin{algorithmic}
\REQUIRE $n \geq 0$
```

```
\ENSURE $y = x^n$
\STATE $y \leftarrow 1$
\STATE $X \leftarrow x$
\STATE $N \leftarrow n$
\WHILE{$N \neq 0$}
\IF{$N$ is even}
\STATE $X \leftarrow X \times X$
\STATE $N \leftarrow N / 2$
\ELSE[$N$ is odd]
\STATE $y \leftarrow y \times X$
\STATE $N \leftarrow N - 1$
\ENDIF
\ENDWHILE
\end{algorithmic}
```

will produce

Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow N/2$
    else {N is odd}
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

which is an algorithm for finding the value of a number taken to a non-negative power.

3.14 Options and Customization

There is a single option, `noend` that may be invoked when the `algorithmic` package is loaded. With this option invoked the *end* statements are omitted in the output. This allows space to be saved in the output document when this is an issue.

3.14.1 Changing Indentation

In the spirit of saving vertical space (which is especially important when submitting a paper for a journal, where space is frequently limited for authors), the `algorithmic` environment offers, beginning with the version released in 2005-

05-08, a way to control the amount of indentation that is used by a given algorithm.

The amount of indentation to be used is given by the command

```
\algsetup{indent=lenght}
```

where *length* is any valid length used by TEX. The default value of the indentation used by the `algorithmic` environment is 1 em (for "backward compatibility reasons"), but a value of 2 em or more is recommended, depending on the publication. For example, the snippet

```
\algsetup{indent=2em}
\begin{algorithmic}[1]
  \STATE $a \leftarrow 1$
  \IF{$a$ is even}
  \PRINT ``$a$ is even''
  \ELSE
  \PRINT ``$a$ is odd''
\end{algorithmic}
```

produces

1: $a \leftarrow 1$
2: **if** a is even **then**
3: **print** "a is even"
4: **else**
5: **print** "a is odd"
6: **end if**

while

```
\algsetup{indent=5em}
\begin{algorithmic}[1]
  \STATE $a \leftarrow 1$
  \IF{$a$ is even}
  \PRINT ``$a$ is even''
  \ELSE
  \PRINT ``$a$ is odd''
\end{algorithmic}
```

would produce

1: $a \leftarrow 1$
2: **if** a is even **then**
3: **print** "a is even"
4: **else**
5: **print** "a is odd"
6: **end if**

The intended use of this option is to allow the author to omit the *end* (see Section 3.14 for details) statements without loosing readability, by increasing the amount of indentation to a suitable level.

3.14.2 Changing Line Numbering

As mentioned in Section 3 and illustrated in Section 3.14.1, `algorithms` already provides you with the possibility of numbering lines.

Starting with the version released in 2005-07-05, you can now change two aspects of line numbering: the size of the line numbers (which, by default, is `\footnotesize`) and the delimiter used to separate the line number from the code (which, by default, is `:`, i.e., a colon).

You can change the size of the line numbers using the command:

```
\algsetup{linenosize=size}
```

where *size* is any of the various commands provided by LaTeX to change the size of the font to be used. Among others, useful values are `\tiny`, `\scriptsize`, `\footnotesize` and `\small`. Please see the complete list of sizes in your LaTeX documentation.

As another frequently requested feature, you can change the delimiter used with the line numbers by issuing the command:

```
\algsetup{linenodelimiter=delimiter}
```

where *delimiter* is any "well-formed" string, including the empty string. With this command, you can change the colon to a period (`.`) by issuing the command

```
\algsetup{linenodelimiter=.}
```

or even omit the delimiter, by specifying the empty string or a space (`\ `), whatever seems best for your document.

As an example of such commands, the code produced by

```
\algsetup{
   linenosize=\small,
   linenodelimiter=.
}
\begin{algorithmic}[1]
   \STATE $i \leftarrow 10$
   \RETURN $i$
\end{algorithmic}
```

would be something like

1. $i \leftarrow 10$
2. **return** i

3.14.3 Customization

In order to facilitate the use of this package with foreign languages, all of the words in the output are produced via redefinable macro commands. The default definitions of these macros are:

```
\newcommand{\algorithmicrequire}{\textbf{Require:}}
\newcommand{\algorithmicensure}{\textbf{Ensure:}}
\newcommand{\algorithmicend}{\textbf{end}}
\newcommand{\algorithmicif}{\textbf{if}}
\newcommand{\algorithmicthen}{\textbf{then}}
\newcommand{\algorithmicelse}{\textbf{else}}
\newcommand{\algorithmicelsif}{\algorithmicelse\ \algorithmicif}
\newcommand{\algorithmicendif}{\algorithmicend\ \algorithmicif}
\newcommand{\algorithmicfor}{\textbf{for}}
\newcommand{\algorithmicforall}{\textbf{for all}}
\newcommand{\algorithmicdo}{\textbf{do}}
\newcommand{\algorithmicendfor}{\algorithmicend\ \algorithmicfor}
\newcommand{\algorithmicwhile}{\textbf{while}}
\newcommand{\algorithmicendwhile}{\algorithmicend\ \algorithmicwhile}
\newcommand{\algorithmicloop}{\textbf{loop}}
\newcommand{\algorithmicendloop}{\algorithmicend\ \algorithmicloop}
\newcommand{\algorithmicrepeat}{\textbf{repeat}}
\newcommand{\algorithmicuntil}{\textbf{until}}
\newcommand{\algorithmicprint}{\textbf{print}}
\newcommand{\algorithmicreturn}{\textbf{return}}
\newcommand{\algorithmictrue}{\textbf{true}}
\newcommand{\algorithmicfalse}{\textbf{false}}
```

If you would like to change the definition of these commands to another content, then you should use, in your own document, the standard LaTeX command `renewcommand`, with an usage like this:

```
\renewcommand{\algorithmicrequire}{\textbf{Input:}}
\renewcommand{\algorithmicensure}{\textbf{Output:}}
```

About the Way Comments Are Formatted The formatting of comments is implemented via a single argument command macro which may also be redefined. The default definition is

```
\newcommand{\algorithmiccomment}[1]{\{#1\}}
```

and another option that may be interesting for users familiar with C-like languages is to redefine the comments to be

```
\renewcommand{\algorithmiccomment}[1]{// #1}
```

Comments produced this way would be like this:

$i \leftarrow i + 1$ // Increments i

This second way to present comments may become the default in a future version of this package.

4 The `algorithm` Environment

4.1 General

When placed within the text without being encapsulated in a floating environment `algorithmic` environments may be split over a page boundary, greatly detracting from their appearance.[2] In addition, it is useful to have algorithms numbered for reference and for lists of algorithms to be appended to the list of contents. The `algorithm` environment is meant to address these concerns by providing a floating environment for algorithms.

4.2 An Example

To illustrate the use of the `algorithm` environment, the following text

```
\begin{algorithm}
\caption{Calculate $y = x^n$}
\label{alg1}
\begin{algorithmic}
\REQUIRE $n \geq 0 \vee x \neq 0$
\ENSURE $y = x^n$
\STATE $y \leftarrow 1$
\IF{$n < 0$}
\STATE $X \leftarrow 1 / x$
\STATE $N \leftarrow -n$
\ELSE
\STATE $X \leftarrow x$
\STATE $N \leftarrow n$
\ENDIF
\WHILE{$N \neq 0$}
\IF{$N$ is even}
\STATE $X \leftarrow X \times X$
\STATE $N \leftarrow N / 2$
\ELSE[$N$ is odd]
\STATE $y \leftarrow y \times X$
\STATE $N \leftarrow N - 1$
\ENDIF
\ENDWHILE
\end{algorithmic}
\end{algorithm}
```

[2] This is the expected behaviour for floats in LaTeX. If you don't care about having your algorithm split between pages, then one option that you have is to ignore the `algorithm` environment.

produces Algorithm 1 which is a slightly modified version of the earlier algorithm for determining the value of a number taken to an integer power. In this case, provided the power may be negative provided the number is not zero.

Algorithm 1 Calculate $y = x^n$

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  if $n < 0$ **then**
    $X \leftarrow 1/x$
    $N \leftarrow -n$
  else
    $X \leftarrow x$
    $N \leftarrow n$
  end if
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow N/2$
    else // N is odd
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

The command `\listofalgorithms` may be used to produce a list of algorithms as part of the table contents as shown at the beginning of this document. An auxiliary file with a suffix of `.loa` is produced when this feature is used.

4.3 Options

The appearance of the typeset algorithm may be changed by use of the options: `plain`, `boxed` or `ruled` during the loading of the `algorithm` package. The default option is `ruled`.

The numbering of algorithms can be influenced by providing the name of the document component within which numbering should be recommenced. The legal values for this option are: `part`, `chapter`, `section`, `subsection`, `subsubsection` or `nothing`. The default value is `nothing` which causes algorithms to be numbered sequentially throughout the document.

4.4 Customization

In order to facilitate the use of this package with foreign languages, methods have been provided to facilitate the necessary modifications.

The title used in the caption within `algorithm` environment can be set by use of the standard `\floatname` command which is provided as part of the `float` package which was used to implement this package. For example,

```
\floatname{algorithm}{Procedure}
```

would cause **Procedure** to be used instead of **Algorithm** within the caption of algorithms.

In a manner analogous to that available for the built in floating environments, the heading used for the list of algorithms may be changed by redefining the command `listalgorithmname`. The default definition for this command is

```
\newcommand{\listalgorithmname}{List of Algorithms}
```

4.4.1 Placement of Algorithms

One important fact that many users may not have noticed is that the `algorithm` environment is actually built with the `float` package and `float`, in turn, uses David Carlisle's here style option. This means that the floats generated by the `algorithm` environment accept a special option, namely, **[H]**, with a capital 'H', instead of the usual 'h' offered by plain LaTeX.

This option works as a stronger request of "please put the float here": instead of just a suggestion for LaTeX, it actually means "put this float HERE", which is something desired by many. The two algorithms typeset in this document use this option.

Warning *You* can't *use the 'H' positioning option together with the usual 'h' (for "here"), 'b' (for "bottom") etc. This is a limitation (as far as I know) of the* `float.sty` *package.*

5 Labels and References in Algorithms

With the release of 2005-07-05, now `algorithmic` accepts labels and references to specific lines of a given algorithm, so you don't have to hardcode the line numbers yourself when trying to explain what the code does in your texts. Thanks to Arnaud Legrand for the suggestion and patch for this highly missed feature.

An example of its use is shown in Algorithm 2.

Algorithm 2 Calculate $y = x^n$

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$
1: $y \leftarrow 1$
2: **if** $n < 0$ **then**
3: $X \leftarrow 1/x$
4: $N \leftarrow -n$
5: **else**
6: $X \leftarrow x$
7: $N \leftarrow n$
8: **end if**
9: **while** $N \neq 0$ **do**
10: **if** N is even **then**
11: $X \leftarrow X \times X$
12: $N \leftarrow N/2$
13: **else**
14: $y \leftarrow y \times X$
15: $N \leftarrow N - 1$
16: **end if**
17: **end while**

See that, in line 10, we deal with the case of N being even, while, in line 13, we give treatment to the case of N being odd. The numbers you see on this document were generated automatically from the source document.

6 Issues Between `algorithms` and `tocbibind` or `memoir`

It has been discussed in late 2005 that `algorithms` may have bad interactions with the `tocbibind` or the `memoir` package (which includes `tocbibind`).

A workaround has been suggested for the problem. After including something like

```
\usepackage[nottoc]{tocbibind}
```

in the preamble of your document, you can put, after `\begin{document}`, the following snippet of code:

```
\renewcommand{\listofalgorithms}{\begingroup
  \tocfile{List of Algorithms}{loa}
\endgroup}

\makeatletter
\let\l@algorithm\l@figure
\makeatother
```

which should make the command `\listofalgorithms` work as expected.

7 Hints for Typesetting Algorithms

Here are some short hints on typesetting algorithms:

- Don't overcomment your pseudo-code. If you feel that you need to comment too much, then you are probably doing something wrong: you should probably detail the inner workings of the algorithm in regular text rather than in the pseudo-code;
- Similarly, don't regard pseudo-code as a low-level programming language: *don't pollute your algorithms* with punctuation marks like semicolons, which are necessary in C, C++ and Java, but not in pseudo-code. Remember: your readers *are not* compilers;
- Always document what the algorithm receives as an input and what it returns as a solution. Don't care to say in the `\REQUIRE` or in the `\ENSURE` commands *how* the algorithm does what it does. Put this in the regular text of your book/paper/lecture notes;
- If you feel that your pseudo-code is getting too big, just break it into subalgorithms, perhaps abstracting some tasks. Your readers will probably thank you.

Of course, you should follow those hints with common sense. Well, anything should be done with common sense.