Typesetting Algorithms Under LaTeX

Author
Andrew Lincoln Burrow
albcorp@gmail.com

Abstract
The AlbAlgorithms package provides a single alb-algorithms LaTeX package to typeset algorithms. While the algorithmic package is already available, alb-algorithms is designed to rectify certain deficiencies. In particular, it improves line numbering for groups of procedures in an algorithm, matches the markup structure to the algorithm's block structure, supports cross-referencing with AUCTeX and RefTeX, and uses the alb namespace. The package is supported by an emacs lisp file customising AUCTeX and RefTeX which automates the markup process of algorithm floats and block structure, and constructs labels in the file namespace.

Version Information
Revision
Date

1 Introduction

The `alb-algorithms` LaTeX package is designed to typeset algorithms. This system necessarily promotes a certain view of algorithms and data structures. In this case, algorithms are modelled by side effect producing functions termed *procedures*, and data structures are modelled by *accessor* functions.

In typographic terms, `alb-algorithms` extends special treatment to algorithms with respect to page layout and notation. In terms of page layout, algorithms are handled like figures. Namely, they are typeset as floats so that algorithms appear at the top of pages, and no algorithm is split across pages. While this means that an algorithm will occasionally be located a few pages from the accompanying text, it ensures each algorithm can be studied as a textual unit free of page breaks. In terms of notation, algorithms are subject to a number of conventions intended to improve their readability and to highlight certain facts. An algorithm is presented as a function prototype followed by a line numbered body. The function prototype is typeset in typewriter-style characters, as are calls to all other functions also defined by algorithms. The remainder of the text in the body is structured around keywords typeset in boldface, and typeset mathematics. The body is divided into lines which are numbered, and these numbers are used to reference details in the algorithm. Therefore, the line numbers are made unique within a float by appending a letter — a new letter for each procedure in the algorithm. These considerations lead to a collection of LaTeX commands and environments.

The `alb-algorithms` package provides:

- a float environment `algorithm` for algorithms, and a command `listofalgorithms` for generating front matter;
- commands `albNewAccessorIdent` and `albNewProcedureIdent` to declare new LaTeX commands to typeset accessors and procedures respectively, both as identifiers and as prototypes;
- master environments `albAlgorithmic` and `albAlgorithmic*` to typeset block structured pseudo-code where block structure is indicated typographically by indentation, and in mark up by the `albBlock` environment; and
- commands `albLet`, `albAssign`, `albWhile`, `albForAll`, `albForSequence`, `albIf`, `albElseIf`, `albElse`, and `albReturn` commands to typeset the keywords of both simple and compound statements.

An important feature of `alb-algorithms` is the support for labels. This makes it possible to refer to exact line numbers in the analysis of algorithms. Each line in an algorithm corresponds to an `item` in either an `albAlgorithmic` environment or a nested `albBlock` environment, and each such `item` can be referred to via the LaTeX `label` and `ref` mechanism. For this reason, the AUCTeX extensions of `alb-algorithms` provide support for automatically generating useful labels and providing useful context for labels during cross-referencing.

2 Using the Commands and Environments

The environments of `alb-algorithms` place certain syntactic restrictions on their use. This section considers these restrictions and the typographic meaning of

the markup. Section 3 provides a worked example. Section 4 describes how the AUCTeX customisation eases the burden of entering syntactically correct LaTeX code.

2.1 List of Algorithms

The `alb-algorithms` package provides a new command to generate a list of algorithms for inclusion with the table of contents. Otherwise, this command is analogous to the `listoffigures` command.

`\listofalgorithms` Generate a list of the figure float environments in the document. The command is analogous to the built-in LaTeX command `\listoffigures`.

2.2 Algorithm Floats

The `alb-algorithms` package provides the `algorithm` environment to present an algorithm as a single, unbroken typographic structure. The most visible benefit is that AUCTeX is automatically extended to provide improved prompting, so that the identifier in the first prototype is translated into the float's label.

`\begin{algorithm}` Float an algorithm. The procedure prototype should be the first typeset material. See Section 3 for a worked example.

2.3 Accessor and Procedure Definition

The `alb-algorithms` LaTeX package supports the typesetting of the accessor and procedure functions that define algorithms. To this end it provides two commands that create new commands: `albNewAccessorIdent` creates new accessor functions, and `albNewProcedureIdent` creates new procedures. Again, AUCTeX is extended to prompt input of these commands. However, it is further extended to parse these commands so that the newly defined accessor and procedure typesetting commands are also available to tab completion.

`\albNewAccessorIdent{`*identcmd*`}{`*protocmd*`}{`*name*`}{`*args*`}` Create two LaTeX commands called *identcmd* and *callcmd* to respectively typeset an accessor's identifier and prototype. The typeset identifier is *name* and the number of arguments in the prototype is *args*.

For example,

```
\albNewAccessorIdent%
  {accFeaturesId}{\accFeatures}%
  {features}{2}
```

produces the command `\accFeaturesId` to typeset the accessor's identifer `features` and the command `\accFeatures{`*arg1*`}{`*arg2*`}` to typeset a call to the accessor `features`(*arg1*, *arg2*).

The *callcmd* command must only be used in math mode, while the *identcmd* command may be used outside of math mode. Outside of math mode, special consideration is given to hyphens in *name*. Namely, *name* may contain hyphens. These will be typeset as hyphens rather than minuses, and will be used in the LaTeX line breaking algorithm when *identcmd* is used outside of math

mode. The use of hyphens in names is in keeping with practice in languages like lisp, and improves the line breaking in text containing long identifiers.

`\albNewProcedureIdent{`*identcmd*`}{`*protocmd*`}{`*name*`}{`*args*`}` Create two LaTeX commands *identcmd* and *callcmd* to respectively typeset a procedure identifier and prototype. The typographical meaning is identical to that of `albNewAccessorIdent`. The markup distinction allows AUCTeX customisation to support the semantic difference in the denoted objects.

2.4 Block Structured Algorithms

The final piece in `alb-algorithms` is the support for typesetting line numbered, block structured pseudo-code. This support is provided by nesting the environments `algorithm`, `albAlgorithmic`, and `albBlock`. This section describes the syntax and typographical meaning of these structures.

An `algorithm` float contains one or more `albAlgorithmic` or `albAlgorithmic*` environments. These correspond to the procedure definitions comprising the algorithm. Behind the scenes the a counter is kept for the number of procedures in the float, so that line numbers are prefixed by an alphabetic sequence letter. Line numbers in the each procedure are prefixed by an uppercase letter.

An instance of the `albAlgorithmic` list environment typesets a line per `item` command. However, some items may be nested within parasitical `albBlock` list environments. These environments do not contain any of their own counters, but simply indent the nested items. Thus, `albBlock` environments correspond to blocks in a block structured language. The pseudo-code keywords introducing blocks and mathematical entities in a line are typeset by separate LaTeX commands.

This structure provides the following benefits.

- Line numbers are named apart into procedures when an algorithm is implemented by a number of procedures.
- The indentation in the LaTeX source matches the indentation in the typeset algorithm.
- The `albAlgorithmic` and `albBlock` list environments do not disguise their `item` and `label` components, making AUCTeX and RefTeX cross-referencing support much more effective.
- The system is extensible to different keywords without involving the structural aspects.

`\begin{albAlgorithmic}` **and** `\begin{albAlgorithmic*}` The outermost list environment for statements in a procedure. Each `item` is associated with a statement, and may have a `label` for cross-referencing of the statement. Compound statements introduce nested `albBlock` list environments.

The un-starred version must occur within an `algorithm` environment. The starred version is used outside the `albAlgorithmFloat` environment, and does not implement procedure numbering in the line numbering scheme.

`\begin{albBlock}` A list environment for nesting blocks of lines associated with compound statements. Each `item` is associated with a statement, and may have a `label` for cross-referencing of the statement. Compound statements introduce nested `albBlock` list environments.

The environment must occur within an `albAlgorithmic` or `albAlgorithmic*` environment.

`\albLet{`*var*`}{`*val*`}` Command to typeset a simple statement introducing the variable typeset by *var* and assigning it the value typeset by *val*. For example,

```
\albLet{$A \subseteq X$}{$\emptyset$}
```

produces **let** $A \subseteq X \longleftarrow \emptyset$.

`\albAssign{`*var*`}{`*val*`}` Command to typeset a simple statement assigning the value typeset by *val* to variable typeset by *var*. For example,

```
\albAssign{$A$}{$A \cup \{x\}$}
```

produces $A \longleftarrow A \cup \{x\}$.

`\albReturn{`*val*`}` Command to typeset a simple statement returning the value typeset by *val*. For example,

```
\albReturn{$A \cup \{x\}$}
```

produces **return** $A \cup \{x\}$.

`\albWhile{`*cond*`}` Command to typeset the test part of a while statement. The condition is typeset by *cond*. It should be followed by an `albBlock` environment listing the statements in the body of the loop. For example,

```
\albWhile{$A \neq \emptyset$}
```

produces **while** $A \neq \emptyset$.

`\albForAll{`*var*`}{`*set*`}` Command to typeset the loop definition part of a for statement where the order is undetermined. The loop variable is typeset by *var*, and the collection is typeset by *set*. It should be followed by an `albBlock` environment listing the statements in the body of the loop. For example,

```
\albForAll{$x$}{$A$}
```

produces **for all** $x \in A$.

`\albForSequence{`*var*`}{`*seq*`}` Command to typeset the loop definition part of a for statement where the order is explicit. The loop variable is typeset by *var*, and the sequence is typeset by *seq*. It should be followed by an `albBlock` environment listing the statements in the body of the loop. For example,

```
\albForSequence{$x$}{$0, 1, \ldots n$}
```

produces **for** x **in** $0, 1, \ldots n$.

`\albIf{`*cond*`}` **and** `\albElseIf{`*cond*`}` **and** `\albElse` Commands to typeset if statements. The condition is typeset by *cond*. Each form should be followed by an `albBlock` environment listing the statements in the body. For example,

```
\albElseIf{$i \neq 0$}
```

produces **else if** $i \neq 0$.

3 Worked Example

This section contains a worked example. The algorithm is a depth-first traversal over a typed feature structure. It is depicted in Algorithm 1.

The first step is to define the accessor functions used to describe the data structure. The `albNewAccessorIdent` command is used. AUCTeX customisation simplifies this process for the case where identifiers are constructed from hyphenated phrases.

```
\albNewAccessorIdent{\accRootId}{\accRoot}{root}{1}
\albNewAccessorIdent{\accTypeId}{\accType}{type}{1}
\albNewAccessorIdent{\accValueId}{\accValue}{value}{3}
```

Next, define the functions that comprise the algorithm. In this example, the algorithm is implemented by two functions. The first function declares a new variable to hold the result and initialises this variable. It then calls the second function, which performs the depth-first recursive traversal of the typed feature structure.

```
\albNewProcedureIdent%
{\prcDepthFirstSearchId}{\prcDepthFirstSearch}%
{Depth-First-Search}{5}
\albNewProcedureIdent%
{\prcDepthFirstSearchVisitId}{\prcDepthFirstSearchVisit}%
{Depth-First-Search-Visit}{6}
```

The algorithm is also parameterised on the actions to take at various stages in the traversal. Therefore, we must also define names for these functions in the parameter list. This is achieved as for the names of the functions implementing the traversal.

```
\albNewProcedureIdent{\prcPreId}{\prcPre}{Pre}{2}
\albNewProcedureIdent{\prcPostId}{\prcPost}{Post}{2}
\albNewProcedureIdent{\prcFwdId}{\prcFwd}{Fwd}{3}
\albNewProcedureIdent{\prcBckId}{\prcBck}{Bck}{3}
```

Algorithm 1 shows the constructed algorithm, and Figure 1 contains the souce code used to generate the float.

4 AUCTeX Customisations

Under AUCTeX the file `alb-algorithms.el` is automatically loaded whenever the `alb-algorithms` package is used. The customisation adds support for the commands and environments from `alb-algorithms`. In addition, the `reftex-label` command (`C-c (`) constructs labels that indicate the containing procedure, and the `reftex-reference` command (`C-c )`) determines that a line number is being referred to when the preceding text is `Line` or `Lines` and formats the context lines to better indicate the typeset output.

```
\begin{algorithm}
  $\prcDepthFirstSearch%
  {F}{\prcPreId}{\prcFwdId}{\prcBckId}{\prcPostId}$
  \begin{albAlgorithmic}
  \item \label{ln:depth-first-search-1}
    \albLet{$Q \subseteq \mathsf{Node}$}{$\emptyset$}

  \item \label{ln:depth-first-search-2}
    $\prcDepthFirstSearchVisit%
    {F}{\prcPreId}{\prcFwdId}{\prcBckId}{\prcPostId}%
    {\accRoot{F}}$
  \end{albAlgorithmic}

  \medskip{}

  $\prcDepthFirstSearchVisit%
  {F}{\prcPreId}{\prcFwdId}{\prcBckId}{\prcPostId}{q}$
  \begin{albAlgorithmic}
  \item \label{ln:depth-first-search-visit-1}
    $\prcPre{F}{q}$

  \item \label{ln:depth-first-search-visit-2}
    \albAssign{$Q$}{$Q \cup \{q\}$}

  \item \label{ln:depth-first-search-visit-3}
    \albForAll{$f$}{$\accFeatures{F}{q}$}

    \begin{albBlock}
    \item \label{ln:depth-first-search-visit-4}
      $\prcFwd{F}{f}{q}$

    \item \label{ln:depth-first-search-visit-5}
      \albIf{$\accValue{F}{f}{q} \not\in Q$}

      \begin{albBlock}
      \item \label{ln:depth-first-search-visit-6}
        $\prcDepthFirstSearchVisit%
        {F}{\prcPreId}{\prcFwdId}{\prcBckId}{\prcPostId}%
        {\accValue{F}{f}{q}}$
      \end{albBlock}

    \item \label{ln:depth-first-search-visit-7}
      $\prcBck{F}{f}{q}$
    \end{albBlock}

  \item \label{ln:depth-first-search-visit-8}
    $\prcPost{F}{q}$
  \end{albAlgorithmic}
  \caption{Depth first search in a typed feature structure.}
  \label{alg:depth-first-search}
\end{algorithm}
```

Figure 1: Source code to `Depth-First-Search`

```
Depth-First-Search(F, Pre, Fwd, Bck, Post)
A1: let Q ⊆ Node ⟵ ∅
A2: Depth-First-Search-Visit(F, Pre, Fwd, Bck, Post, root(F))

Depth-First-Search-Visit(F, Pre, Fwd, Bck, Post, q)
B1: Pre(F, q)
B2: Q ⟵ Q ∪ {q}
B3: for all f ∈ features(F, q)
B4:     Fwd(F, f, q)
B5:     if value(F, f, q) ∉ Q
B6:         Depth-First-Search-Visit(F, Pre, Fwd, Bck, Post, value(F, f, q))
B7:     Bck(F, f, q)
B8: Post(F, q)
```

Algorithm 1: Depth first search in a typed feature structure.

5 Makefile Targets

The `AlbLaTeXDocumentTemplate` makefile provides a target to relabel the lines in the algorithm environment of this package. The `alb-relabel-ln` target edits the LaTeX source in an attempt to match theorem labels to theorem numbers. Labels of the form `ln:`*identifier*`:`*identifier*`:`*line* `ln:`*identifier*`:`*identifier*`-`*line* are processed, where *line* is a number or lowercase letter followed by a number. *line* is rewritten to the line number, and the preceeding separator is rewritten to a colon. This is helpful in proof reading.