Algorithm description macros

Anders Eriksson

1998-09-27

1 Introduction

This is an introduction to my macro package (file `algo.mac`) for algorithm description. Generally, I will give an expression, as typed in emacs (in `typewriter` font), followed by the same expression, as typeset by LaTeX. To use the macros, simply use the `\input{algo.mac}` command before the `\begin{document}` statement (if you have the the file in the same directory as your document file).

The types of expressions supported by the macro package are:

- Functions and procedures (sect. 2).
- Loops (sect. 3).
- If – then statements (sect. 4).
- The column format Col (sect. 5).
- A pair of larger examples (sect. 6).

The general principles behind the macros are:

- The macros should be easy to use, and reflect the structures they represent.
- There should be as few restrictions as possible on what is possible to typeset in an algorithm. E.g. when typesetting an if–then block, inside the block there should be no notion of the typesetting of the head (locality).
- Finally, the final result should be easy to read, and look good!

A final remark: the macros are all written from my point of view of how to typeset algorithms. Doubtless other people will have other preferences, and I'm certain there are more elegant ways of writing these macros. I strongly encourage tinkering with the macros!

2 Functions and Procedures

```
\Procedure{name}{variables}{body}
```

procedure name (variables) **is**
begin
body
end name

```
\Procedure{name}{var1 &: type1\\& var2 &: type2}{body}
```

procedure name (var1 : type1
var2 : type2) **is**
begin
body
end name

```
\Procedure{name}{var1 : type1; var2 : type2}{body}
```

procedure name (var1 : type1; var2 : type2) **is**
begin
body
end name

```
\Procedure{name}{}{body}
```

procedure name () **is**
begin
body
end name

```
\Function{name}{variables}{type}{body\\ \Return result}
```

function name (variables) **return** type **is**
begin
body
return result
end name

3 Loops

```
\Forloop{condition}{body 1\\ \Break\\ body 2}
```

for condition **loop**
  body 1
  break
  body 2
end loop

```
\Forloopr{condition}{body}\\
```

for condition **loop** body

```
\Whileloop{condition}{body}
```

while condition **loop**
  body
end loop

```
\Whileloopr{condition}{body}\\
```

while condition **loop** body

```
\Doloop{condition}{body}
```

do loop
  body
while condition

```
\Whileloop{incredibly long condition 1 or\\&
  extra complicated condition 2\\}{body}
```

while incredibly long condition 1 or
    extra complicated condition 2
loop
  body
end loop

4 If – Then Statements

```
\Procedure{IfThens}{}{
  \Ifthen{cond}{a}
  \Endif \\

  \Ifthen{cond}{a}
  \Else{b}
  \Endif \\

  \Ifthenr{cond} a\\
  \Ifthenr{cond} a \Elser\ b\\ \\

  \Ifthen{cond1}{a}
  \Elser \Ifthen{cond2}{b}
  \Else{c}
  \Endif\\

  \NIfthen{either long condition 1 \\&
    or long condition 2}{a}
  \Endif
  }
```

procedure IfThens () **is**
begin
  if cond **then**
  | a
  end if

  if cond **then**
  | a
  else
  | b
  end if

  if cond **then** a
  if cond **then** a **else** b

  if cond1 **then**
  | a
  else if cond2 **then**
  | b
  else
  | c
  end if

  if either long condition 1
    or long condition 2
  then
  | a
  end if
end IfThens

5 Column Format Col

This is simply a two-column table, very handy to align columns of text, such as assignments or type declarations. The example below illustrate the principle. This is, of course, a matter of taste and readability.

```
\Procedure{Cols}{}{
  \Col{ : }{
    T       & Tree \\
    E$_l$, E$_r$ & Examples \\
    p       & Plane \\
    }
  \\

  \Col{ $\leftarrow$ }{
    T        & NewTreeNode()\\
    T.plane & p \\
    T.left  & EstimateBSP(E$_l$)\\
    T.right & EstimateBSP(E$_r$)\\
    }
  \\

  T       $\leftarrow$ NewTreeNode()\\
  T.plane $\leftarrow$ p \\
  T.left  $\leftarrow$ EstimateBSP(E$_l$)\\
  T.right $\leftarrow$ EstimateBSP(E$_r$)\\
  }
```

procedure Cols () **is**
begin

T	:	Tree
E_l, E_r	:	Examples
p	:	Plane

T	$\leftarrow$	NewTreeNode()
T.plane	$\leftarrow$	p
T.left	$\leftarrow$	EstimateBSP(E_l)
T.right	$\leftarrow$	EstimateBSP(E_r)

T $\leftarrow$ NewTreeNode()
T.plane $\leftarrow$ p
T.left $\leftarrow$ EstimateBSP(E_l)
T.right $\leftarrow$ EstimateBSP(E_r)

end Cols

6 Examples

```
procedure Estimate (E : in Examples
                    P : out Array of real numbers) is
begin
  C, N : Array of natural numbers, one index for each x.

  -- Clear counters
  C(x) := 0 for all x
  N(x) := 0 for all x

  do loop
    -- create a new leaf node
    T           ← NewLeafNode()
    T.examples  ← E
    T.p         ← c(E)/n(E)
  while T.x > 0

  -- Count
  for all examples e ∈ E loop
    if e.IS_GOOD then C(e.x) := C(e.x) + 1
    N(x) := N(x) + 1
  end loop

  for each parameter configuration x loop
    if N(x) > 0 then
      P(x) := C(x)/N(x)
    else if x < 5 then
      -- Don't know symbol
      P(x) := -1
    end if
  end loop
end Estimate
```

function EstimateBSP (E : in Examples) **return** Tree **is**
begin

T : Tree
E_l, E_r : Examples
p : Plane

(1) Choose a plane p in the parameter space, s.t.
$I(p_l) + I(p_r)$ is made as large as possible.

(2) Let $E_l = \{e \in E \mid e.x\}$ is on the same side as the
plane normal points to, and let $E_r = E \setminus E_l$.

– – I define $p_l = c(A_l)/n(A_l)$ and $p_l = c(A_r)/n(A_r)$ where
– – $A_l = \{x \in A \mid$ the normal of the separatation plane points to $x\}$
– – and $A_r = A \setminus A_l$.
if both E_l and E_r are acceptable from a
statistical point of view *and* $I(p_l) + I(p_r) > I(p)$
then
– – create a new tree node
T := NewTreeNode()
T.plane := p
T.left := EstimateBSP(E_l)
T.right := EstimateBSP(E_r)
else
– – create a new leaf node
T := NewLeafNode()
T.examples := E
T.p := $c(E)/n(E)$
end if
return T
end EstimateBSP