

Algorithm

pickoba

Algorithm パッケージは、 $\text{\texttt{SATySFi}}$ 上で擬似コードを記述するためのパッケージです。 $\text{\texttt{LATEX}}$ における $\text{\texttt{algorithmicx}}$ パッケージ等に相当する機能を提供することを目指しています。

1. 基本的な使い方

以下を記述して Algorithm パッケージを読み込みます。

```
@require: algorithm/core
@require: algorithm/style/default
```

ここでは標準 (default) のスタイルを指定していますが、Python 風、C 風のスタイルを選択することも可能です (後述)。

実際に擬似コードの組版をしてみましょう。以下はユークリッドの互除法の記述例です。
($\text{\texttt{\backslash bmod}}$, $\text{\texttt{\backslash gets}}$ 等は適切に定義されていると仮定)

```
+algorithmic<
+Procedure{Euclid}{ $\{a, b\}$ }<
+State{ $\{r \text{\texttt{\backslash gets}} a \text{\texttt{\backslash bmod}} b\}$ }
+While{ $\{r \text{\texttt{\backslash neq}} 0\}$ }<
+State{ $\{a \text{\texttt{\backslash gets}} b\}$ }
+State{ $\{b \text{\texttt{\backslash gets}} r\}$ }
+State{ $\{r \text{\texttt{\backslash gets}} a \text{\texttt{\backslash bmod}} b\}$ }
>
+State{\Return{ $\{b\}$ }}
>
>
```

```
1: procedure EUCLID( $a, b$ )
2:    $r \leftarrow a \bmod b$ 
3:   while  $r \neq 0$  do
4:      $a \leftarrow b$ 
5:      $b \leftarrow r$ 
6:      $r \leftarrow a \bmod b$ 
7:   end while
8:   return  $b$ 
9: end procedure
```

擬似コードは $\text{\texttt{+algorithmic}}$ で括り、その中で $\text{\texttt{+Procedure}}$ や $\text{\texttt{+State}}$ を使って記述していきます。これらのコマンド名はおおよそ $\text{\texttt{LATEX}}$ の $\text{\texttt{algorithmicx}}$ パッケージと揃えてあります。次ページから利用可能なコマンドを紹介します。

2. 定義されている命令一覧

2.1. 基本

行番号がつく行

```
+State{body}
```

1: body

行番号がつかない行

```
+Statex{body}
```

body

2.2. ループ

```
+For{cond}<  
  +State{body}  
>
```

1: **for** cond **do**
2: body
3: **end for**

```
+ForAll{something}<  
  +State{body}  
>
```

1: **for all** something **do**
2: body
3: **end for**

```
+While{cond}<  
  +State{body}  
>
```

1: **while** cond **do**
2: body
3: **end while**

```
+RepeatUntil<  
  +State{body}  
>{cond}
```

1: **repeat**
2: body
3: **until** cond

```
+Loop<  
  +State{body}  
>
```

1: **loop**
2: body
3: **end loop**

2.3. 条件分岐

else 節が存在し得るため、+If は自動では閉じられないことに注意。

```
+If{cond}<
  +State{body}
>
+ElsIf{cond}<
  +State{body}
>
+Else<
  +State{body}
>
+EndIf;
```

```
1: if cond then
2:   body
3: else if cond then
4:   body
5: else
6:   body
7: end if
```

2.4. Procedure / Function

関数名は自動的にスモールキャピタルになる。

```
+Procedure{name}{params}<
  +State{body}
>
```

```
1: procedure NAME(params)
2:   body
3: end procedure
```

```
+Function{name}{params}<
  +State{body}
>
```

```
1: function NAME(params)
2:   body
3: end function
```

2.5. コメント

コメントは挿入したい行に対応する `inline-text` 内に `\Comment` を用いて記述する。`end while` 等のブロックの終端行にコメントを挿入する場合は `+Comment` を用いる。スタイルは `+algorithmic` の引数で調整可能。詳細は第5章を参照。

```
+While{cond\Comment{1st comment}}<
  +State{body\Comment{2nd comment}}
  +Comment{3rd comment}
>
```

```
1: while cond do    ▷ 1st comment
2:   body           ▷ 2nd comment
3: end while        ▷ 3rd comment
```

2.6. その他

+Require, +Ensure には行番号がつかない。

```
+Require{something}
```

Require: something

```
+Ensure{something}
```

Ensure: something

3. コマンドの定義

Algorithm モジュール内の `inline-scheme`, `block-scheme` を利用することで独自のコマンドが定義できます。以下では Algorithm モジュールが `open` されていることを仮定しています。

3.1. 一行のコマンドを定義する場合

`inline-scheme : context -> bool -> inline-text -> block-boxes` を使えばよいです。第二引数は行番号を付与するかどうかを示します。例えば、

```
let-block ctx +Assert it =
  inline-scheme ctx true {\keyword{assert}\ #it;}
```

と定義することで

```
+Assert{body}
```

1: assert body

のように使えます。なお `\keyword` も Algorithm モジュール内で定義されているコマンドです。

3.2. ブロックを作るコマンドを定義する場合

`block-scheme : context -> inline-text -> block-text -> inline-text ->`

`block-boxes` を使えばよいです。例えば、

```
let-block ctx +Block inner =  
  block-scheme ctx  
    {\keyword{begin}}  
    inner  
    {\keyword{end}}
```

と定義することで

```
+Block<  
  +State{body}  
>
```

```
1: begin  
2:   body  
3: end
```

のように使えます。なお、第二・第四引数の `inline-text` は空にすることで対応する行を非表示にすることが可能です。

4. レイアウトのカスタマイズ

Algorithm パッケージでは、`@require` で読み込むファイルを変更することで異なる表記 (L^AT_EX の `algorithmicx` にならいレイアウトと呼称します) に変更することができます。

4.1. Python 風のレイアウト

読み込み時に

```
@require: algorithm/core  
@require: algorithm/style/python
```

と記載することで以下の出力を得られます (コードは第1章を参照)。

```
1: def EUCLID( $a, b$ ):  
2:    $r \leftarrow a \bmod b$   
3:   while  $r \neq 0$ :  
4:      $a \leftarrow b$   
5:      $b \leftarrow r$   
6:      $r \leftarrow a \bmod b$   
7:   return  $b$ 
```

4.2. C 言語風のレイアウト

読み込み時に

```
@require: algorithm/core
@require: algorithm/style/c
```

と記載することで以下の出力を得られます。

```
1: EUCLID( $a, b$ ) {
2:    $r \leftarrow a \bmod b$ ;
3:   while ( $r \neq 0$ ) {
4:      $a \leftarrow b$ ;
5:      $b \leftarrow r$ ;
6:      $r \leftarrow a \bmod b$ ;
7:   }
8:   return  $b$ ;
9: }
```

5. スタイルのカスタマイズ

`+algorithmic` に与える第一オプショナル引数のレコードによってスタイルをカスタマイズできます。以下のデフォルト値は `Algorithm.default-config` として与えられています。

- `font-set : fss-font-set` (デフォルト値: `FssFonts.default-font-set`)
 - コード中で利用するフォントセット。詳細は `fss` パッケージを参照。
- `keyword : style list` (デフォルト値: `[bold]`)
 - プログラム中のキーワード (予約語) に対するスタイルを指定。詳細は `fss` パッケージを参照。
- `line-start : int` (デフォルト値: `1`)
 - 行の開始番号。
- `show-linenummer : bool` (デフォルト値: `true`)
 - 行番号を表示するか。
- `indent-scale : float` (デフォルト値: `1.`)
 - フォントサイズに対するインデントサイズの比

- `prefixf : context -> int -> inline-boxes` (デフォルト値: `default-prefixf`)
 - 各行のプレフィックス (行番号表示) をカスタマイズするためのフック。
- `commentf : context -> inline-text -> inline-boxes` (デフォルト値: `default-commentf`)
 - コメントをカスタマイズするためのフック。

例えば以下のようにカスタマイズできます。

```
+algorithmic?:(|
  Algorithm.default-config with
    keyword = [bold; italic];
    line-start = 10;
    indent-scale = 2.;
    commentf = fun ctx it ->
      (inline-skip 15pt ++ read-inline ctx {// #it;});
|)<
+State{${s \leftarrow 0}\Comment{comment}}
+For{${i \leftarrow 1, n}}<
  +State{${s \leftarrow s + i}}
  >
>
```

```
10:  $s \leftarrow 0$  // comment
11: for  $i \leftarrow 1, n$  do
12:    $s \leftarrow s + i$ 
13: end for
```