Algorithm

pickoba

Algorithm パッケージは、SATySFI 上で擬似コードを記述するためのパッケージです。LaTeX における 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。（`\bmod`, `\gets` 等は適切に定義されていると仮定）

```
+algorithmic<
  +Procedure{Euclid}{${a, b}}<
    +State{${r \gets a \bmod b}}
    +While{${r \neq 0}}<
      +State{${a \gets b}}
      +State{${b \gets r}}
      +State{${r \gets a \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 `+algorithmic` で括り、その中で `+Procedure` や `+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 LaTeX の 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;
```

if cond **then**
    body
else if cond **then**
    body
else
    body
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}
>
```

procedure NAME(params)
    body
end procedure

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

function NAME(params)
    body
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

```
+State{\Return{something}}
```

1: **return** 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（LaTeX 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 ← r
6:        r ← a mod 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 ← a mod b;
3:    while (r ≠ 0) {
4:       a ← b;
5:       b ← r;
6:       r ← a mod 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関数によってスタイルをカスタマイズできます。調整できる項目は以下の通りです。

- `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を指定。詳細は fss パッケージを参照。
- `keyword-style : 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。
- `linenumber-start : int`（デフォルト値 : `1`）
  - 行の開始番号。
- `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。行番号を非表示にしたい場合はここに `Algorithm.null-prefixf` を指定。
- `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。

実際にはこれらに `set-` を前置した関数を連結することでスタイルを指定します。例えば以下のようにカスタマイズできます[*1]。ここでも `Algorithm` モジュールが `open` されていることを仮定しています。

```
+algorithmic?:(fun config -> config
  |> set-linenumber-start 10
  |> set-keyword-style [bold; italic]
  |> set-indent-scale 2.
  |> set-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***

1 SATγSFI v0.1.0 がリリースされた後に、ラベル付きオプショナル引数を利用する形に変更する予定です。