XPath パッケージ

The XPath Contributors

1. 基本的なコマンド

1.1. パスの組み立て

satysfi-xpath は、端的に言えばパスを表す独自の型である XPath.t 及び未完パスを表す XPath.pre を扱うためのライブラリです。これらは SATYSFI における path 及び pre-path に対応します。

satysfi-xpath は SATYSFI が持つパス操作コマンドと互換性のあるコマンドをサポートしており、opan XPath とすることにより、SATYSFI の提供するパス操作コマンドを置き換えることもできます (推奨はしない)。

具体的には、以下のコマンドをサポートします。

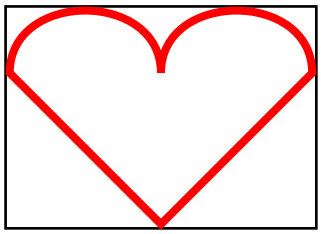
```
XPath.unite-path : XPath.t -> XPath.t -> XPath.t
XPath.shift-path : point -> XPath.t -> XPath.t
XPath.linear-transform-path : float -> float -> float -> float -> XPath.t
-> XPath.t
XPath.get-path-bbox : XPath.t -> point * point
XPath.start-path : point -> XPath.pre
XPath.line-to : point -> XPath.pre -> XPath.pre
XPath.bezier-to : point -> point -> point -> XPath.pre -> XPath.pre
XPath.terminate-path : XPath.pre -> XPath.t
XPath.close-with-line : XPath.pre -> XPath.t
XPath.close-with-bezier : point -> point -> XPath.pre -> XPath.t
XPath.stroke : length -> color -> XPath.t -> graphics
XPath.fill : color -> XPath.t -> graphics
XPath.dashed-stroke : length -> length * length * length -> color ->
XPath.t -> graphics
```

1.2. 例

以下のコードは、簡単な図形描画の例です. (SATYSFI 組み込みのコマンドを使った場合と同様の結果が得られる。)

```
XPath.(
    start-path (0cm, 0cm)
        |> bezier-to (0cm, 1.1cm) (-2cm, 1.1cm) (-2cm, 0cm)
        |> line-to (0cm, -2cm) |> line-to (2cm, 0cm)
        |> close-with-bezier (2cm, 1.1cm) (0cm, 1.1cm)
        |> stroke 3pt (Color.red)
)
```

この出力は以下のようになります。



1.3. 組み込み型との変換

`XPath.t`, `XPath.pre` を SATYSFI に組み込みの `path`, `pre-path` に変換するため、以下のコマンドが用意されています。[*1]

```
XPath.to-embedded-path : XPath.t -> path
XPath.to-embedded-prepath : XPath.pre -> pre-path
```

逆に、SATYSFI の組み込み型から `satysfi-xpath` の型に変換する方法はありません。

2. 便利な機能

ここでは、SATYSFI の組み込み `path` 機能にはない、`satysfi-xpath` 独自の便利な機能

1 これらのコマンドは通常使用する必要はありませんが、他の `path` を操作するライブラリ等と連携させる場合に使用できます。

について説明します。

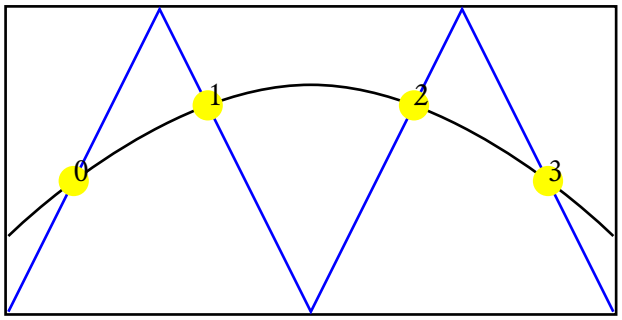
2.1. 交点を求めるコマンド

交点を求める機能として以下が提供されています。

```
XPath.get-intersections : length -> XPath.t -> point list
XPath.get-intersections-with : length -> XPath.pre -> XPath.t -> point
list
```

- get-intersections delta pat… 与えられたパス pat の内部の交点 (point 型) のリストを得ます。delta は精度です。
- get-intersections-with delta ppat pat… 与えられた未完パス ppat がパス pat と交わる点のリストを得ます。得られる点列は ppat 上で順番になっています。

例えば、以下の図は黒線で示されるパス pat 及び青線で示される未完パス ppat が与えられた時に、これらの交点 (黄色) を get-intersections-with コマンドで取得する例です。このとき得られる交点の配列は、未完パス ppat 上での順になっています (わかりやすいように番号を表示しています)。

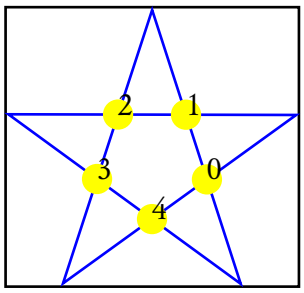


コードを以下に示します。

```
let pat = XPath.(
    start-path (4cm, -1cm) |> line-to (2cm, 3cm)
    |> line-to (0cm, -1cm) |> line-to (-2cm, 3cm)
    |> line-to (-4cm, -1cm) |> terminate-path
) in
let ppat = XPath.(
    start-path (-4cm, 0cm)
```

```
    |> bezier-to (-4cm, 0cm) (-2cm, 2cm) (0cm, 2cm)
    |> bezier-to (2cm, 2cm) (4cm, 0cm) (4cm, 0cm)
) in
let pts = XPath.get-intersections-with 0.01cm ppat pat in
let labels = pts |> List.mapi (fun i p -> (
    [
        Gr.circle p 0.2cm |> fill (Color.yellow);
        arabic i |> embed-string |> read-inline ctx |> draw-text p
    ]
)) |> List.concat in
List.append [
    pat |> XPath.stroke 1pt (Color.blue);
    ppat |> XPath.terminate-path |> XPath.stroke 1pt (Color.black);
] labels
```

一方で get-intersections を用いることにより、パス pat の内部の交点 (すなわち自分自身との交点) を得ることができます。この場合に得られる交点の配列の順番は非自明です。



コードを以下に示します。

```
let pat = XPath.(
    [2.; 4.; 1.; 3.] |> List.fold-left (fun pp r -> (
        let theta = r *. 6.28 /. 5. +. 1.57 in
        pp |> line-to (2cm *' (cos theta), 2cm *' (sin theta))
    )) (start-path (0cm, 2cm)) |> close-with-line
) in
let pts = XPath.get-intersections 0.01cm pat in
let labels = pts |> List.mapi (fun i p -> (
```

```
    [
        Gr.circle p 0.2cm |> fill (Color.yellow);
        arabic i |> embed-string |> read-inline ctx |> draw-text p
    ]
)) |> List.concat in
(pat |> XPath.stroke 1pt (Color.blue))::labels
```

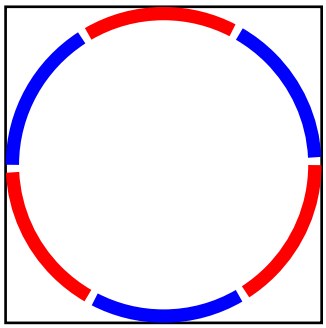
2.2. 長さに関するコマンド

ここでは、 曲線の長さを取得したり、 点の曲線上における位置を取得したり、 長さを用いて曲線を操作するコマンドを紹介します。

```
XPath.get-rough-length : length -> XPath.pre -> length
XPath.get-point-of-len : length -> length -> pre -> point
XPath.get-projection : length -> point -> pre -> point
XPath.get-projection-length : length -> point -> pre -> length
XPath.get-derivative : length -> length -> pre -> point
XPath.split : length -> length -> pre -> pre * pre
```

- get-rough-length delta ppat… 与えられた未完パス ppat の長さを得ます。 これは厳密なアルゴリズムではなく、精度を delta で指定します。
- get-point-of-len delta len ppat… 与えられた未完パス ppat 上で始点から長さ len の地点を取得します。
- get-projection delta p ppat… 与えられた未完パス ppat 上で与えられた点 p に最も近い点を取得します。
- get-projection-length delta p ppat… 与えられた未完パス ppat 上で与えられた点 p に最も近い点の位置を表す長さを取得します。
- get-derivative delta len ppat… 与えられた未完パス ppat 上で位置により与えられた点 len 上での未完パスの微分をベクトルとして取得します。
- split delta len ppat… 与えられた未完パス ppat を長さ len の位置で分割します。

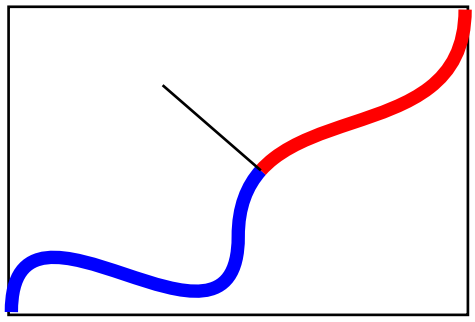
以下の例では、円形の未完パス ppat が与えられた時に、get-rough-length を用いてその長さを取得し、 その長さを 6 等分した値を用いて split を用いてパスを分割します。 わかりやすさのために各パス片を青と赤に色分けし、間に隙間をいれています。



この出力は以下のコードで得られます。

```
let ppat = XPath.(
    start-path (-2cm, 0pt)
    |> bezier-to (-2cm, 1.1cm) (-1.1cm, 2cm) (0cm, 2cm)
    |> bezier-to (1.1cm, 2cm) (2cm, 1.1cm) (2cm, 0cm)
    |> bezier-to (2cm, -1.1cm) (1.1cm, -2cm) (0cm, -2cm)
    |> bezier-to (-1.1cm, -2cm) (-2cm, -1.1cm) (-2cm, 0pt)
) in
let len = XPath.get-rough-length 0.01cm ppat in
let m = 0.1cm in
let slen = len *' (1. /. 6.) -' m in
let (_, gs) = [0; 1; 2; 3; 4; 5] |> List.fold-right (fun i (acc, gs)
-> (
    let (pp, acc) = acc |> XPath.split 0.01cm slen in
    let (_, acc) = acc |> XPath.split 0.01cm m in
    let clr = if i mod 2 == 0 then Color.red else Color.blue in
    let g = pp |> XPath.terminate-path |> XPath.stroke 5pt clr in
    (acc, g::gs)
)) (ppat, []) in gs
```

get-projection-length を使用した例を示します。下図では与えられた未完パス ppat 上において、特定の点 p から最も近い点の位置を get-projection-length を用いて取得しています。また、get-point-of-len を用いて座標を取得し点 p から直線を引いています。またわかりやすいように未完パス ppat を分割し色分けして表示します。



コードを以下に示します。

```
let ppat = XPath.(
    start-path (0cm, 0cm)
        |> bezier-to (0cm, 2cm) (3cm, -1cm) (3cm, 1cm)
        |> bezier-to (3cm, 3cm) (6cm, 2cm) (6cm, 4cm)
) in
let p = (2cm, 3cm) in
let len = XPath.get-projection-length 0.1cm p ppat in
let proj = XPath.get-point-of-len 0.1cm len ppat in
let (ppat1, ppat2) = XPath.split 0.1cm len ppat in
XPath.([
    ppat1 |> terminate-path |> stroke 5pt Color.blue;
    ppat2 |> terminate-path |> stroke 5pt Color.red;
    start-path p |> line-to proj |> terminate-path |> stroke 1pt Color.black
])
```

2.3. パスの変形

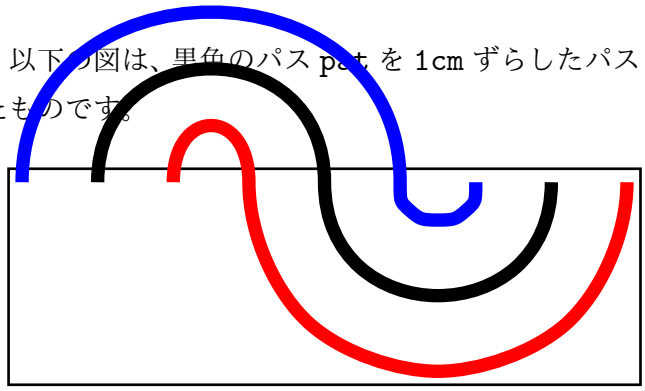
ここでは、パスの変形に用いるコマンドを説明します。

```
XPath.offset : length -> XPath.pre -> XPath.pre
XPath.offset-path : length -> XPath.t -> XPath.t
XPath.deformate-path : length -> pre -> XPath.t -> XPath.t
XPath.deformate-path-by-func : (point -> point) -> XPath.t -> XPath.t
```

`XPath.offset l ppat` 及び `XPath.offset-path l pat` はそれぞれ未完パス及びパスを与えられた長さ `l` 分左にずらします。長さとして負の値を指定すると、逆方向にずらしま

す。

以下の図は、黒色のパス pat を 1cm ずらしたパス (青色) 及び -1cm ずらしたパスを描画したものです。



コードを以下に示します。

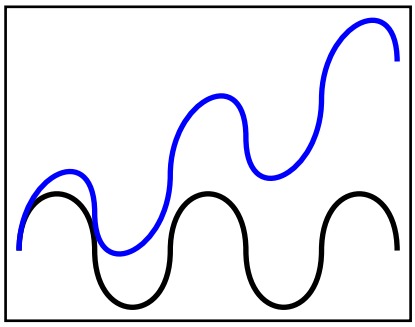
```
let pat = XPath.(
 start-path (0cm, 0cm)
    |> bezier-to (0cm, 2cm) (3cm, 2cm) (3cm, 0cm)
    |> bezier-to (3cm, -2cm) (6cm, -2cm) (6cm, 0cm)
    |> terminate-path
) in
[
 (pat, Color.black);
 (pat |> XPath.offset-path 1cm, Color.blue);
 (pat |> XPath.offset-path -1cm, Color.red);
] |> List.map (fun (pp, clr) -> (
 pp |> XPath.stroke 5pt clr
))
```

deformate-path-by-func f p を用いると、与えられた関数 f : point -> point を用いてパス p を変形することができます。これは linear-transform-path の一般化です。例えば以下は関数 (x, y) -> (x, y *' (x *' 0.5)') を用いて黒いパスを青いパスに変形する例です。

```
XPath.(
    let f = fun (x, y) -> (x, y +' (x *' 0.5)) in
    let p = start-path (0pt, 0pt)
       |> bezier-to (0cm, 1cm) (1cm, 1cm) (1cm, 0cm)
       |> bezier-to (1cm, -1cm) (2cm, -1cm) (2cm, 0cm)
```

```
        |> bezier-to (2cm, 1cm) (3cm, 1cm) (3cm, 0cm)
        |> bezier-to (3cm, -1cm) (4cm, -1cm) (4cm, 0cm)
        |> bezier-to (4cm, 1cm) (5cm, 1cm) (5cm, 0cm)
        |> terminate-path
    in
    [
        p |> stroke 2pt (Color.black);
        p |> deformate-path-by-func f |> stroke 2pt (Color.blue)
    ]
)
```

は以下のグラフィックを出力します。

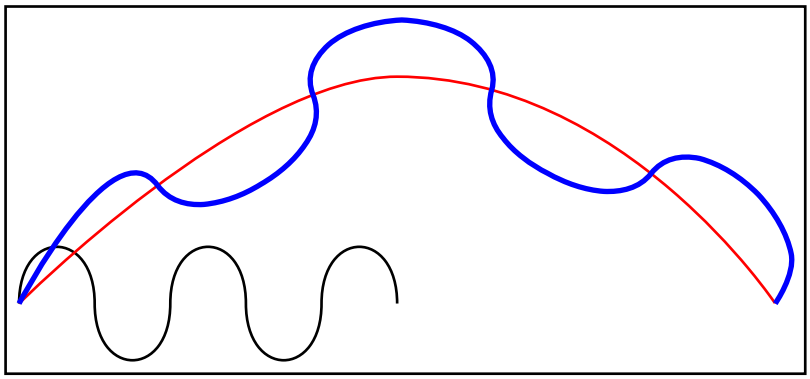


`deformate-path delta pp p` を用いると、与えられた未完パス `pp` に沿わせるようパス `p` を変形することができます。パス `p` は座標 `(0pt, 0pt)` から `x > 0` 方向に配置される必要があり、この時直線 `x = 0` を未完パス `pp` に合わせるように変形します。

以下は黒いパスを赤い未完パスを用いて変形し青いパスを得る例です。

```
XPath.(
    let pp = start-path (0pt, 0pt)
        |> bezier-to (0cm, 0cm) (3cm, 3cm) (5cm, 3cm)
        |> bezier-to (8cm, 3cm) (10cm, 0cm) (10cm, 0cm)
    in
    let p = start-path (0pt, 0pt)
        |> bezier-to (0cm, 1cm) (1cm, 1cm) (1cm, 0cm)
        |> bezier-to (1cm, -1cm) (2cm, -1cm) (2cm, 0cm)
        |> bezier-to (2cm, 1cm) (3cm, 1cm) (3cm, 0cm)
```

```
        |> bezier-to (3cm, -1cm) (4cm, -1cm) (4cm, 0cm)
        |> bezier-to (4cm, 1cm) (5cm, 1cm) (5cm, 0cm)
        |> terminate-path
    in
    [
        p |> stroke 1pt (Color.black);
        pp |> terminate-path |> stroke 1pt (Color.red);
        p |> deformate-path 1pt pp |> stroke 2pt (Color.blue)
    ]
)
```



3. xpath-gr の機能

パッケージ xpath-gr に含まれるモジュール XPathGr は、SATySFI 標準の Gr モジュールに相当する機能を提供します。シグネチャは Gr モジュールと同等であり、各関数の値のみが path ではなく XPath.t になります。

```
XPathGr.rectangle : point -> point -> XPath.t
XPathGr.rectangle-round       : length -> point -> point -> XPath.t
XPathGr.rectangle-round-left  : length -> point -> point -> XPath.t
XPathGr.rectangle-round-left-lower : length -> point -> point -> XPath.t
XPathGr.rectangle-round-left-upper : length -> point -> point -> XPath.t
XPathGr.rectangle-round-right : length -> point -> point -> XPath.t
XPathGr.poly-line : point -> point list -> XPath.t
XPathGr.polygon : point -> point list -> XPath.t
```

```
XPathGr.line : point -> point -> XPath.t
XPathGr.circle : point -> length -> XPath.t
XPathGr.text-centering : point -> inline-boxes -> graphics
XPathGr.text-leftward : point -> inline-boxes -> graphics
XPathGr.text-rightward : point -> inline-boxes -> graphics
XPathGr.arrow : length -> color -> length -> length -> length -> point
-> point -> graphics list
XPathGr.dashed-arrow : length -> length * length * length -> color -
> length -> length -> length -> point -> point -> graphics list
XPathGr.rotate-path : point -> float -> XPath.t -> XPath.t
XPathGr.scale-path : point -> float -> float -> XPath.t -> XPath.t
XPathGr.rotate-graphics : point -> float -> graphics -> graphics
cale-graphics : point -> float -> float -> graphics -> graphics
```

4. 今後追加する機能

以下のような機能を今後追加する予定です。

- 与えられたパスを与えられた未完パスを基準に変形する機能
- 与えられたパスのうち、内部のループを削除する機能
- 複数のパスの結合、交差、分割を生成する機能
- パスと点が与えられた時に、点が塗られる部分に属するか判定する機能
- パスと点が与えられた時に、点を取り囲む最小のパスを取得する機能
- パスのアウトライン化する機能
- ASCII 文字に対応するパスを生成する機能