

SATYSFI の 基本型とプリミティヴ

Takashi SUWA

目次

1. 型一覧	1
2. プリミティヴ一覧	2
2.1. 基本演算	2
2.2. 文字列演算	3
2.3. インライン方向に関する処理	4
2.4. ブロック方向に関する処理	7
2.5. テキスト文脈に関する処理	7
2.6. 数式に関する処理	9
2.7. 画像に関する処理	10
2.8. グラフィックスに関する処理	11

この文書では、SATYSFI によって提供される型とプリミティヴを掲げる。特に根幹となるのは第 2.3 節（4 ページ）である。

1. 型一覧

unit ユニット型。値では唯一 [\(\)](#) にのみこの型がつく。

`bool` 真偽値型. 値では `true` と `false` にのみこの型がつく.

`int` 符号つき整数型. 内部表現は 32 ビット環境では 31 ビット, 64 ビット環境では 63 ビットである.

`float` 浮動小数点数型. 諸演算が IEEE754 に従う.

`length` 寸法型.

`string` (純粋な) 文字列型. 原則として Unicode コードポイントの列として扱われるが, 一部の (古い, 使用非推奨な) プリミティヴでは UTF-8 バイト列表現を直接扱うインターフェイスになっている. インラインテキスト型との混同に注意されたい.

`inline-text` インラインテキスト型. 文書の文字データのうち “文字の進む方向” の部分であるインラインテキストにこの型がつく. インラインテキストとは, 典型的には`{...}`の形をしている部分である. `{it}` はそれ自体が値であって, コマンド適用なども構造的に保持しており, “評価されると書き換えられる式” ではないことに注意せよ. `it` に記述されているコマンド適用は `read-inline` で読まれたときにはじめて評価されてインラインボックス列となる.

`inline-boxes` インラインボックス型. インラインボックス列につく型である. インラインボックス列とは大雑把にいえば文字列のようなものだが, 文字列だけでなく, どのフォントで組まれるかや「空白がどの程度伸縮できるか」「どこで改行してよいか」といった行分割に関する情報も埋め込まれている.

2. プリミティヴ一覧

2.1. 基本演算

`+, -, *` : $(\text{int} \rightarrow (\text{int} \rightarrow \text{int}))$

整数の加算・減算・乗算.

`/, mod` : $(\text{int} \rightarrow (\text{int} \rightarrow \text{int}))$

整数の除算と剰余. 現在の仕様では, 第 2 引数に `0` を与えて評価した場合の結果は未定義. 現在の実装では実行時エラーとなり処理を中止する. [今後仕様変更の可能性あり]

$\text{==}, \text{<>}, \text{<}, \text{>}, \text{<=}, \text{>} = : (\text{int} \rightarrow (\text{int} \rightarrow \text{bool}))$

整数の比較演算.

$\text{&&}, \text{||} : (\text{bool} \rightarrow (\text{bool} \rightarrow \text{bool}))$

真偽値の連言・選言.

$\text{not} : (\text{bool} \rightarrow \text{bool})$

論理否定.

$+, -, . : (\text{float} \rightarrow (\text{float} \rightarrow \text{float}))$

浮動小数点数の加算・減算. IEEE754 に従う.

$+', -' : (\text{length} \rightarrow (\text{length} \rightarrow \text{length}))$

長さの加算・減算. 内部的には PDF ポイント単位の浮動小数点数で扱われ, IEEE754 に従う.

$*' : (\text{length} \rightarrow (\text{float} \rightarrow \text{length}))$

長さのスカラー演算.

$/' : (\text{length} \rightarrow (\text{length} \rightarrow \text{float}))$

長さの割合.

$<', >' : (\text{length} \rightarrow (\text{length} \rightarrow \text{bool}))$

長さの比較.

$\text{float} : (\text{int} \rightarrow \text{float})$

整数を浮動小数点数に変換する.

2.2. 文字列演算

文字列操作のための簡単なプリミティヴがいくつか用意されているが, 現状これらのうちの一部は開発初期の名残といった様相であり, 積極的な使用を推奨しない. Unicode 正規化の指定などはおろか Unicode コードポイント単位の扱いをサポートせず, 直接 UTF-8 バイト列を扱うなど, 低級な処理として形式化されているためである.

$\text{~} : (\text{string} \rightarrow (\text{string} \rightarrow \text{string}))$

文字列の結合.

arabic : $(\text{int} \rightarrow \text{string})$

受け取った整数の十進文字列を返す.

string-unexplode : $((\text{int}) \text{ list} \rightarrow \text{string})$

受け取った整数列を Unicode コードポイント列と看なして対応する文字列を返す. Unicode コードポイントとして不適切な整数が含まれていた場合の動作は未定義. [今後仕様変更の可能性あり]

string-same : $(\text{string} \rightarrow (\text{string} \rightarrow \text{bool}))$

文字列が UTF-8 のバイト列として等価かの判定. [今後仕様変更の可能性あり] [使用非推奨]

string-sub : $(\text{string} \rightarrow (\text{int} \rightarrow (\text{int} \rightarrow \text{string})))$

string-sub $s i l$ で文字列 s の第 i 番目の文字を先頭とする長さ l の部分文字列を取り出す. 最初の文字は 0 番目, また長さは UTF-8 表現でのバイト長であって Unicode コードポイントの個数ではないことに注意. [今後仕様変更の可能性あり] [使用非推奨]

string-length : $(\text{string} \rightarrow \text{int})$

受け取った文字列の UTF-8 表現のバイト長を返す. [今後仕様変更の可能性あり] [使用非推奨]

2.3. インライン方向に関する処理

read-inline : $(\text{context} \rightarrow (\text{inline-text} \rightarrow \text{inline-boxes}))$

read-inline $ctx it$ で文脈 ctx を用いてインラインテキスト it を変換したインラインボックス列を返す.

inline-skip : $(\text{length} \rightarrow \text{inline-boxes})$

inline-skip l で長さ l の（伸縮しない）インライン方向の空白を返す.

inline-glue : $(\text{length} \rightarrow (\text{length} \rightarrow (\text{length} \rightarrow \text{inline-boxes})))$

inline-glue $l_0 l_1 l_2$ で自然長 l_0 , 縮小基準長 l_1 , 伸長基準長 l_2 のインライン方向の空白を返す.

inline-fil : inline-boxes

自然長 0, 罰則なしで任意有限長に伸長できるインライン方向の空白. 左揃え, 右揃え, 中央揃えなどに有用である. 特に段落の整形を目的として **line-break true true** (*ib* ++ **inline-fil**) の形で使う場面が多い.

++ : $(\text{inline-boxes} \rightarrow (\text{inline-boxes} \rightarrow \text{inline-boxes}))$

2つのインラインボックス列を結合して返す.

inline-nil : inline-boxes

長さ 0 のインライン方向の空白. より正確に言えばこれは空白ではなく, 任意のインラインボックス列 *ib* に対して **ib** ++ **inline-nil** が *ib* と全く同様に振舞うようになっている.

embed-string : $(\text{string} \rightarrow \text{inline-text})$

文字列をインラインテキストに変換する.

embed-math : $(\text{context} \rightarrow (\text{math} \rightarrow \text{inline-boxes}))$

数式をインラインボックス列に変換する.

discretionary : $(\text{int} \rightarrow (\text{inline-boxes} \rightarrow (\text{inline-boxes} \rightarrow (\text{inline-boxes} \rightarrow \text{inline-boxes}))))$

line-break による行分割の候補位置をつくる. **discretionary p ib₀ ib₁ ib₂** で「行分割されなかったときは *ib₀* を出力し, 行分割されたときは分割位置の直前に *ib₁* を挿入し直後に *ib₂* を挿入する」という効果をもつインラインボックス列を返す. なお, この位置で行分割することになるか否かにかかわらず, 各 *ib_i* 内にあるすべての行分割候補位置は行分割しない箇所として扱われる. *p* はペナルティ値であり, “どの程度行分割してほしくないか”の指標である. **10000** 以上で「最悪」, すなわち「本当に行分割することが避けられない場合を除いてなんとしてもここで行分割しないでほしい」ことを指し, **0** で「行分割の抑制も促進もしない」を指す. **0** 未満は「行分割しないよりも行分割する方が望ましい」ことを意味し, よりその位置での行分割が促進される.

script-guard : $(\text{script} \rightarrow (\text{inline-boxes} \rightarrow \text{inline-boxes}))$

script-guard script ib で, インラインボックス列 *ib* を文字体系 *script* の単語として扱う. これは和欧間空白など異なる文字体系間のスペースの挿入の有無に影響を与える.

`get-natural-width` : $(\text{inline-boxes} \rightarrow \text{length})$

インラインボックス列を受け取り、その自然な幅を返す。

`inline-graphics` : $\left(\text{length} \rightarrow \left(\text{length} \rightarrow \left(\text{length} \rightarrow \left(\text{inline-graphics} \rightarrow \text{inline-boxes} \right) \right) \right) \right)$

`inline-graphics w h d igr` で幅 `w`、高さ `h`、深さ `d` の領域にインライングラフィックス `igr` を描画したものをインラインボックス列として返す。

`inline-frame-outer`, `inline-frame-inner` : $\left(\text{paddings} \rightarrow \left(\text{deco} \rightarrow \left(\text{inline-boxes} \rightarrow \text{inline-boxes} \right) \right) \right)$

`inline-frame-outer p d ib` でパディング指定 `p`、装飾指定 `d`、内容 `ib` の、途中で行分割不可能なフレームを返す。`inline-frame-outer` は外側の都合に合わせて内側の空白などが伸縮するのに対し、`inline-frame-inner` は内側の自然な長さのみにより組まれる。すなわち、後者は枠で囲われた部分全体が“1つの文字”のように振舞う。

`inline-frame-breakable` : $\left(\text{paddings} \rightarrow \left(\text{deco-set} \rightarrow \left(\text{inline-boxes} \rightarrow \text{inline-boxes} \right) \right) \right)$

`inline-frame-breakable p ds ib` でパディング指定 `p`、装飾4つ組指定 `ds`、内容 `ib` の、途中で行分割可能なフレームを返す。

`embed-block-top`, `embed-block-bottom` : $\left(\text{context} \rightarrow \left(\left(\text{length} \rightarrow \left(\text{context} \rightarrow \text{block-boxes} \right) \right) \rightarrow \text{inline-boxes} \right) \right)$

`embed-block-top ctx l k` で文脈 `ctx` をテキスト幅に関して `l` に変更して継続 `k` に渡し、その結果のブロックボックス列をインラインボックス列内に埋め込む。高さと深さは中身の最初の行のベースラインが外のベースラインと一致するように決められる。`embed-block-bottom` は最後の行のベースラインが外のベースラインと一致することを除いて `embed-block-top` と同様。

`line-stack-top`, `line-stack-bottom` : $\left((\text{inline-boxes}) \text{ list} \rightarrow \text{inline-boxes} \right)$

複数のインラインボックス列をブロック方向に積む。全体の幅は最も長い行の幅となる。`line-stack-top` は最初の行のベースラインが外のベースラインと一致するよ

うに位置が決められ、`line-stack-bottom` は最後の行に合わせて決められる。

2.4. ブロック方向に関する処理

`read-block` : $(\text{context} \rightarrow (\text{block-text} \rightarrow \text{block-boxes}))$

`read-block ctx bt` で文脈 `ctx` に従ってブロックテキスト `bt` を変換したブロックボックス列を返す。

`line-break` : $(\text{bool} \rightarrow (\text{bool} \rightarrow (\text{context} \rightarrow (\text{inline-boxes} \rightarrow \text{block-boxes}))))$

`line-break b1 b2 ctx ib` で文脈 `ctx` にしたがってINLINEボックス列 `ib` を適切に行分割して段落の形に組んだブロックボックス列を返す。`b1` が `true` のときはその段落の直前での改ページを許し、`false` のときは許さない。`b2` も同様に段落の直後で改ページを許すかの指定である。[今後仕様変更の可能性あり]

`form-document` : $(\text{context} \rightarrow (\text{block-boxes} \rightarrow \text{document}))$

与えられたINLINEボックス列を適切にページ分割して文書に整形する。

`+++` : $(\text{block-boxes} \rightarrow (\text{block-boxes} \rightarrow \text{block-boxes}))$

2つのブロックボックス列を結合して返す。

`block-nil` : `block-boxes`

高さ 0 のブロックボックス列。より正確には、任意のブロックボックス列 `bb` に対して `bb` と `bb ++ block-nil` が全く等価に振舞うようになっている。

`block-frame-breakable` : $(\text{context} \rightarrow (\text{paddings} \rightarrow ((\text{deco-set} \rightarrow (\text{context} \rightarrow \text{block-boxes})) \rightarrow \text{block-boxes})))$

`block-frame-breakable ctx pads ds k` は文脈 `ctx` をテキスト幅に関して `pads` を用いて変更して継続 `k` に渡し、その結果のブロックボックス列を装飾 `ds` のフレームで囲んだものを返す。この処理でつくられるフレームは途中で改ページ可能である。

2.5. テキスト文脈に関する処理

`set-space-ratio` : $(\text{float} \rightarrow (\text{context} \rightarrow \text{context}))$

`ctx |> set-space-ratio r` で単語間空白の幅をフォントサイズの `r` 倍に変更した

テキスト文脈を返す。[今後仕様変更の可能性あり]

set-font-size : $(\text{length} \rightarrow (\text{context} \rightarrow \text{context}))$

ctx |> **set-font-size** *s* でフォントサイズを *s* に変更したテキスト文脈を返す。

get-font-size : $(\text{context} \rightarrow \text{length})$

テキスト文脈が保持しているフォントサイズを返す。

set-font : $(\text{script} \rightarrow (\text{font} \rightarrow (\text{context} \rightarrow \text{context})))$

ctx |> **set-font** *script* *font* で文字体系 *script* の文字に対して使うフォントを *font* に変更した文脈を返す。

set-language : $(\text{script} \rightarrow (\text{language} \rightarrow (\text{context} \rightarrow \text{context})))$

ctx |> **set-language** *script* *lang* で文字体系 *script* に対して言語システム *lang* を割り当てた文脈を返す。

get-language : $(\text{script} \rightarrow (\text{context} \rightarrow \text{language}))$

ctx |> **get-language** *script* で文脈 *ctx* に於いて文字体系 *script* に割り当てられている言語システムを返す。

set-math-font : $(\text{string} \rightarrow (\text{context} \rightarrow \text{context}))$

ctx |> **set-math-font** *fname* で数式フォントを *fname* に変更した文脈を返す。

set-dominant-wide-script : $(\text{script} \rightarrow (\text{context} \rightarrow \text{context}))$

ctx |> **set-dominant-wide-script** *script* で East_Asian_Width プロパティが W (wide) , F (fullwidth) のいずれかである文字を文字体系 *script* の文字と看なす文脈を返す。

get-dominant-wide-script : $(\text{context} \rightarrow \text{script})$

受け取ったテキスト文脈に於いて East_Asian_Width プロパティが W , F のいずれかである文字がどの文字体系に属すると看なされているかを返す。

set-dominant-narrow-script : $(\text{script} \rightarrow (\text{context} \rightarrow \text{context}))$

ctx |> **set-dominant-wide-script** *script* で East_Asian_Width プロパティが Na (narrow) , H (halfwidth) , A (ambiguous) , N (neutral) のいずれかである文字を文字

体系 `script` の文字と看なす文脈を返す.

`get-dominant-narrow-script` : $(\text{context} \rightarrow \text{script})$

受け取ったテキスト文脈に於いて East_Asian_Width プロパティが Na , H , A , N のいずれかである文字がどの文字体系に属すると看なされているかを返す.

`set-text-color` : $(\text{color} \rightarrow (\text{context} \rightarrow \text{context}))$

`ctx |> set-text-color color` で文字色を `color` に変更した文脈を返す.

`set-leading` : $(\text{length} \rightarrow (\text{context} \rightarrow \text{context}))$

`ctx |> set-leading l` で行送りを `l` に変更した文脈を返す. これはフォントサイズに対する比での指定ではなく, 直接長さを指定するプリミティヴであることに注意. すなわち, フォントサイズを変更しても標準の行送りの長さは変更されない.

`set-manual-rising` : $(\text{length} \rightarrow (\text{context} \rightarrow \text{context}))$

`ctx |> set-manual-rising l` で文字全体を長さ `l` だけ持ち上げて組む文脈を返す.

`get-text-width` : $(\text{context} \rightarrow \text{length})$

テキスト文脈が保持している段落幅を返す. `line-break` はこの長さにしたがって行分割を行なう.

2.6. 数式に関する処理

`math-char` : $(\text{math-class} \rightarrow (\text{string} \rightarrow \text{math}))$

`math-char mathcls s` で文字列 `s` を数式中の文字として使えるようにしたものを作成する. `mathcls` はその文字をスペーシングに関してどのように扱ってほしいかの指定である.

`math-big-char` : $(\text{math-class} \rightarrow (\text{string} \rightarrow \text{math}))$

`math-char` の大型演算子版.

`math-char-with-kern` : $(\text{math-class} \rightarrow (\text{string} \rightarrow (\text{math-kern-func} \rightarrow (\text{math-kern-func} \rightarrow \text{math}))))$

`math-char` と同様だが, 添字や上附をつけるためのカーニング量を `y` 座標に応じて長

さを返す函数で指定できる。

math-big-char-with-kern : $(\text{math-class} \rightarrow (\text{string} \rightarrow (\text{math-kern-func} \rightarrow (\text{math-kern-func} \rightarrow \text{math}))))$

math-char-with-kern の大型演算子版。

math-sup, **math-sub**, **math-upper**, **math-lower** : $(\text{math} \rightarrow (\text{math} \rightarrow \text{math}))$

上附, 添字, 真上, 真下。

math-frac : $(\text{math} \rightarrow (\text{math} \rightarrow \text{math}))$

分数。

math-radical : $(\text{math} \rightarrow \text{math})$

根号。

math-paren : $(\text{paren} \rightarrow (\text{paren} \rightarrow (\text{math} \rightarrow \text{math})))$

中身に応じて自動で大きさが調整される括弧で囲う。

text-in-math : $(\text{math-class} \rightarrow ((\text{context} \rightarrow \text{inline-boxes}) \rightarrow \text{math}))$

数式中にインラインボックス列を埋め込む。

math-variant-char : $((\text{math-class} \rightarrow \text{math-char-style}) \rightarrow \text{math})$

数式文字クラス指定（イタリック, ボールドローマン, スクリプトなど）に応じて変化する文字を定義する。

math-color : $(\text{color} \rightarrow (\text{math} \rightarrow \text{math}))$

数式の文字色を変更する。

math-char-class : $((\text{math-char-class} \rightarrow \text{math}) \rightarrow \text{math})$

数式文字クラスを変更する。

2.7. 画像に関する処理

外部の画像ファイルを読み込んで用いるためのプリミティヴを（まだ少数ながら）用意している。現状では PDF と JPEG のみをサポートしている。

load-pdf-image : $(\text{string} \rightarrow (\text{int} \rightarrow \text{image}))$

外部 PDF ファイルのパスとページ番号 n (最初のページを 1 ページと数える) を受け取り, その PDF の n ページ目を画像情報として返す. 指定されたファイルが存在しない場合の動作は未定義. 現在の実装では実行時エラーとなり処理を中止する. [今後仕様変更の可能性あり]

load-image : $(\text{string} \rightarrow \text{image})$

外部の画像ファイルのパスを受け取り, その内容を画像情報として返す. 現状では色空間がグレイスケールまたは RGB の JPEG ファイルのみをサポートする. 指定されたファイルが存在しない場合の動作は未定義. 現在の実装では実行時エラーとなり処理を中止する. [今後仕様変更の可能性あり]

use-image-by-width : $(\text{image} \rightarrow (\text{length} \rightarrow \text{inline-boxes}))$

use-image-by-width $img w$ で画像 img を幅 w の大きさで描画したものインラインボックス列として返す.

2.8. グラフィックスに関する処理

start-path : $(\text{point} \rightarrow \text{pre-path})$

点を受け取り, その点からパスを開始する.

line-to : $(\text{point} \rightarrow (\text{pre-path} \rightarrow \text{pre-path}))$

prepath |> **line-to** v で未完パス **prepath** を終点から点 v へと線分で延長したものを返す.

bezier-to : $(\text{point} \rightarrow (\text{point} \rightarrow (\text{point} \rightarrow (\text{pre-path} \rightarrow \text{pre-path}))))$

prepath |> **bezier-to** $u_1 u_2 v$ で未完パス **prepath** を終点から点 v へと Bezier 曲線で延長したものを返す. u_1 と u_2 は制御点である.

close-with-line : $(\text{pre-path} \rightarrow \text{path})$

未完パスを受け取り, 起点と終点を線分で結んで閉じてできるパスを返す.

close-with-bezier : $(\text{point} \rightarrow (\text{point} \rightarrow (\text{pre-path} \rightarrow \text{path})))$

prepath |> **close-with-bezier** $u_1 u_2$ で未完パス **prepath** の起点と終点を制御点 u_1 , u_2 の Bezier 曲線で結んで閉じてできるパスを返す.

terminate-path : $(\text{pre-path} \rightarrow \text{path})$

未完パスを受け取り、開いたままのパスとして返す。

unite-path : $(\text{path} \rightarrow (\text{path} \rightarrow \text{path}))$

2つのパスを統合して1つにする。これはドーナツ形など中空のパスをつくるのに必須である。

fill : $(\text{color} \rightarrow (\text{path} \rightarrow \text{graphics}))$

fill color path でパス path の内側を色 color で塗ったグラフィックスを返す。パスのどこが内側であるかは偶奇則によって決められる。

stroke : $(\text{length} \rightarrow (\text{color} \rightarrow (\text{path} \rightarrow \text{graphics})))$

stroke t color path でパス path を幅 t 、色 color の線として描いたグラフィックスを返す。【今後仕様変更の可能性あり】

draw-text : $(\text{point} \rightarrow (\text{inline-boxes} \rightarrow \text{graphics}))$

draw-text v ib で位置 v をベースラインの左端としてINLINEボックス列 ib を置いたグラフィックスを返す。