SATYSFI の基本型とプリミティヴ

Takashi SUWA

目次

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（7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．

regexp　内部で正規表現を表すデータを保持する型．

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 {*it*} 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連する情報も埋め込まれている．

block-text　ブロックテキスト型．文書の文字データのうち “段落の進む方向” の部分であるブロックテキストにこの型がつく．ブロックテキストは典型的には ⟨*bt*⟩ の形をしており，インラインテキストと同様これ自体が値である．⟨*bt*⟩ に記述されているコマンド適用は `read-block` で読まれたときにはじめて評価されてブロックボックス列となる．

block-boxes　ブロックボックス型．ブロックボックス列につく型である．ブロックボックス列とは大雑把にいえば段落の連なりであり，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”・“どこで改ページしてよいか”といった情報も保持している．

math　数式を扱う型．

math-class　数式のスペーシングのために必要な“どの種類の数学記号なのか”を表す型．`MathOrd`（通常）・`MathBin`（2 項演算子）・`MathRel`（関係子）・`MathOp`（作用子）・`MathPunct`（パンクチュエーション）・`MathOpen`（開き括弧類）・`MathClose`（閉じ括弧類）・`MathPrefix`（接頭類）・`MathInner`（部分式）の 9 つの値構築子をもつ代数的データ型である．

math-char-class　数式中に書かれている文字を数式用文字書体に基づいてグリフに変換するときに使用する型．`MathItalic`・`MathBoldItalic`・`MathRoman`・`MathBoldRoman`・`MathScript`・`MathBoldScript`・`MathFraktur`・`MathBoldFraktur`・`MathDoubleStruck` の 9 つの値構築子をもつ代数的データ型である．

math-char-style　どの数式用文字書体にどの文字列が対応するのかを定める際に使用される．$\langle$italic = string; bold-italic = string; roman = string; bold-roman = string; script = string; bold-script = string; fraktur = string; bold-fraktur = string; double-struck = string$\rangle$ に等しい．

math-kern-func　カーニング量を調節するための関数．length → length → length に等しい．

paren　内容の高さ・内容の深さ・軸の高さ・文字サイズ・文字色を受け取り，片側の括弧の中身と添字・指数をカーニングするための詰め幅を返す関数である．数式の括弧を描画するために使われる．length → length → length → length → color → $\big($inline-boxes $*$ $\big($length → length$\big)\big)$ に等しい．

context　コンテキスト型．組版処理に必要な情報のうちテキスト以外のものをすべて保持している．

text-info　テキスト出力モードで使用されるコンテキスト型．

point　(x, y) で座標を表す．左下が原点で，右に行くほど x の値が大きくなり，上に行くほど y の値が大きくなる座標系である．

path　曲線や閉曲線および，それらが占める領域の情報を保持している．

graphics　`path` の情報に加え，“どのような色で塗られるか”・“どのような線で描かれるのか”といった情報を保持する．

paddings　余白の大きさを表現するために使用されるもので，その中身は単純に length を 4 つ組にしたものである．それぞれの値の役割は順に左・右・上・下である．

deco　point → length → length → length → graphics list である．枠の指定に使われる．

deco-set　deco の 4 つ組である．行分割・ページ分割可能な枠に使われる．

script　“どの文字体系に属するか”を表す型．`HanIdeographic`・`Kana`・`Latin`・`OtherScript` の 4 つの値構築子をもつ代数的データ型である．

language　どの言語システムに基づいているのかを表す型．現在のところ `Japanese`・`English`・`NoLanguageSystem` の 3 つの値構築子をもつ代数的データ型である．

color　色を表す型．`Gray of` *float*・`RGB of` (*float* ∗ *float* ∗ *float*)・`CMYK of` (*float* ∗ *float* ∗ *float* ∗ *float*) の 3 つの値構築子をもつ代数的データ型である．

document　`page-break` などによってしか生成することができない．PDF に出力するデータなどを格納する．

page　ページサイズを表す．`A0Paper`・`A1Paper`・`A2Paper`・`A3Paper`・`A4Paper`・`A5Paper`・`USLetter`・`USLegal`・`UserDefinedPaper of` (*length* ∗ *length*) の 9 つの値構築子をもつ代数的データ型である．〔今後仕様変更の可能性あり〕

page-count-func　ページ番号を基に“そのページのテキストの領域の段落方向の長さ”と“そのページの中身を配置するときの左上の座標”を返す関数である．(| page-number = int |) → (| text-height = length; text-origin = point |) に等しい．

page-parts-func　ページ番号を基にヘッダーとフッターの中身とその左上の座標を返す関数である．(| page-number = int |) → (| header-content = block-boxes; header-origin = point; footer-content = block-boxes; footer-origin = point |) に等しい．

2. プリミティヴ一覧

2.1. 基本演算

+, -, * : int → int → int
整数の加算・減算・乗算.

/, mod : int → int → 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〕

==, <>, <, >, <=, >= : int → int → bool
整数の比較演算.

<<, >>, bxor, band, bor : int → int → int
整数のビット演算.

bnot : int → int
整数のビット NOT 演算

&&, || : bool → bool → bool
真偽値の連言・選言.

not : bool → bool
論理否定.

+., -. : float → float → float
浮動小数点数の加算・減算．IEEE754 に従う.

*., /. : float → float → float
浮動小数点数の乗算・除算.

+', -' : length → length → 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.

*' : length → float → length
長さのスカラー演算.

/' : length → length → float
長さの割合.

<', >' : length → length → bool
長さの比較.

`float` : int → float
　整数を浮動小数点数に変換する．

`round` : float → int
　浮動小数点数を整数に変換する．

`ceil`, `floor` : float → float
　`ceil` で浮動小数点数の切り上げを，`floor` で浮動小数点数の切り捨てを行う．

`sin`, `asin`, `cos`, `acos`, `tan`, `atan` : float → float
　三角関数の演算を行う．

`atan2` : float → float → float
　三角関数の演算を行う．

`log`, `exp` : float → float
　指数と対数の演算を行う．

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．

`^` : string → string → string
　文字列の結合．

`arabic` : int → string
　受け取った整数の十進文字列を返す．

`show-float` : float → string
　受け取った浮動小数点数の十進文字列を返す．

`string-unexplode` : int list → 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-explode` : string → int list
　受け取った文字列を Unicode スカラー値のリストとして返す．

`string-same` : string → string → 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` : string → int → int → 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．文字は Unicode コードポイント単位．*i* と *l* による指定が部分文字列の範囲でない場合は実行時エラーが出る．

`string-length` : string → int

受け取った文字列の Unicode コードポイントでの長さを返す．

`split-into-lines` : string → (int ∗ string) list

`split-into-lines` *s* で，*s* が改行文字で分割され，分割後の文字列の先頭部分の半角スペースが取り除かれる． 返り値の整数と文字列のペアの内， 整数部分は取り除かれたスペースの数で，文字列部分はスペースが取り除かれた後の文字列である．

`regexp-of-string` : string → regexp

`regexp-of-string` *s* で文字列から正規表現型を生成する．使える正規表現の構文は OCaml の `Str.regexp` で使えるものと等しい．正規表現として不適当な文字列が与えられた場合は実行時エラーが出る．

`string-scan` : regexp → string → (string ∗ string) option

`string-scan` *re s* で *s* の *re* が一致する部分文字列と， 残りの文字列を返す． 一文字も一致しなかった場合は `None` が返る．

`string-match` : regexp → string → bool

`string-match` *re s* で，*s* が *re* にマッチするかを判定する．

`split-on-regexp` : regexp → string → (int ∗ string) list

`split-on-regexp` *re s* で，*re* がマッチする部分で *s* が分割され，分割後の文字列の先頭部分の半角スペースが取り除かれる．返り値の整数と文字列のペアの内，整数部分は取り除かれたスペースの数で，文字列部分はスペースが取り除かれた後の文字列である．

`read-file` : string → string list

`read-file` *path* で文書ファイルから見て *path* の位置にある外部ファイルの中身を改行文字で分割したリストで返す．`..` を含む *path* は使用できない．

2.3. インライン方向に関する処理

`extract-string` : inline-boxes → string

インラインボックス列を無理やり文字列に変換する．

`read-inline` : context → inline-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` : length → 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` : length → length → length → 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．

`++` : inline-boxes → inline-boxes → 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` : string → inline-text
文字列をインラインテキストに変換する．

`embed-math` : context → math → inline-boxes
数式をインラインボックス列に変換する．

`discretionary` : int → inline-boxes → inline-boxes → inline-boxes → inline-boxes
`line-break` による行分割の候補位置をつくる．`discretionary` p ib_0 ib_1 ib_2 で「行分割されなかったときは ib_0 を出力し，行分割されたときは分割位置の直前に ib_1 を挿入し直後に ib_2 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` : script → inline-boxes → 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` : inline-boxes → 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．

`get-natural-metrics` : inline-boxes → (length * length * length)

インラインボックス列を受け取り，その自然な幅・高さ・深さを返す．

`inline-graphics` : length → length → length → (point → graphics list) → inline-boxes

`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` : paddings → deco → inline-boxes → inline-boxes

`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` : paddings → deco-set → inline-boxes → inline-boxes

`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` : context → length → (context → block-boxes) → inline-boxes

`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` : inline-boxes list → inline-boxes

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．

`embed-block-breakable` : context → block-boxes → inline-boxes
`embed-block-breakable` *ctx bb* で *bb* をインラインボックス列に“擬態”させて型検査を通し，インラインボックス列を組む時になってその“擬態”を解き，ブロックボックス列として振る舞うようにする．

2.4. ブロック方向に関する処理

`read-block` : context → block-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` : bool → bool → context → inline-boxes → block-boxes
`line-break` b_1 b_2 *ctx ib* で文脈 *ctx* にしたがってインラインボックス列 *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．b_1 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．b_2 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〕

`+++` : block-boxes → block-boxes → block-boxes
2 つのブロックボックス列を結合して返す．

`block-skip` : length → block-boxes
`block-skip` *l* で長さ *l* のブロック方向の空白を返す．

`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` : context → paddings → deco-set → (context → block-boxes) → 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．

`get-natural-length` : block-boxes → length
ブロックボックス列を受け取り，そのブロック方向の長さを返す．

2.5. テキスト文脈に関する処理

`get-initial-context` : length → math inline-cmd → context
デフォルトのテキスト文脈を返す．第 1 引数は段落の幅を指定し，第 2 引数はインラインテキスト中に現れた数式に適用するコマンドを与える．

`set-space-ratio` : float → context → 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-space-ratio-between-scripts` : float → float → float → script → script → context → context

ctx `|> set-space-ratio-between-scripts` $script_1$ $script_2$ r_0 r_1 r_2 で前方が文字体系 $script_1$ の文字，後方が文字体系 $script_2$ の文字，の隣接箇所で自然長・収納基準長・伸長基準長の文字サイズに対する比率がそれぞれ r_0・r_1・r_2 になるようなグルーを挿入するように変更したテキスト文脈を返す．

`set-font-size` : length → context → 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` : context → 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` : script → font → context → 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．

`get-font` : script → context → font

`get-font` *script ctx* で文字体系 *script* の文字に対して使うフォントを返す．

`set-language` : script → language → context → 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` : script → context → 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` : string → context → 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` : script → context → 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` : context → 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` : script → context → 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` : context → 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` : color → context → context

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

`get-text-color` : context → color

テキスト文脈が保持している文字色を返す．

`set-leading` : length → context → 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-min-gap-of-lines` : length → context → context

ctx `|>` `set-min-gap-of-lines` l で行間の最小値を引数の値に変更した文脈を返す．

`set-paragraph-margin` : length → length → context → context

ctx `|>` `set-paragraph-margin` l_1 l_2 で段落の上のマージンを l_1 に，段落の下のマージンを l_2 にするよう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-manual-rising` : length → context → 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` : context → 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．

`set-word-break-penalty` : int → context → context

`set-word-break-penalty` p *ctx* で“そこで行分割された場合に加算されるペナルティ値”を p に設定したテキスト文脈を返す．

`set-every-word-break` : inline-boxes → inline-boxes → context → context
`set-every-word-break` ib_1 ib_2 ctx で行分割候補箇所の“そこで行分割された場合に直前の行末に入る内容”を ib_1 に，“そこで行分割された場合に直後の行頭に入る内容”を ib_2 に設定したテキスト文脈を返す．

`get-every-word-break` : context → (inline-boxes ∗ inline-boxes)
`set-every-word-break` で設定されている ib_1 と ib_2 の内容を取得する．

`set-hyphen-penalty` : int → context → context
ハイフネーションによるペナルティを設定する．デフォルトは 100 である．

`set-hyphen-min` : int → int → context → context
ctx `|> set-hyphen-min` n_1 n_2 でハイフネーションによって単語が分割されるときの左側の最小文字数を n_1 に，右側の最小文字数を n_2 にそれぞれ設定したテキスト文脈を返す．

2.6. 数式に関する処理

`math-char` : math-class → string → 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返す．$mathcls$ 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` : math-class → string → math
`math-char` の大型演算子版．

`math-char-with-kern` : math-class → string → math-kern-func → math-kern-func → 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` : math-class → string → math-kern-func → math-kern-func → math
`math-char-with-kern` の大型演算子版．

`math-sup`, `math-sub`, `math-upper`, `math-lower` : math → math → math
上附，添字，真上，真下．

`math-frac` : math → math → math
分数．

`math-radical` : math option → math → math
根号．第 1 引数には現在のところ `None` しか渡すことは出来ない．

`math-paren` : paren → paren → math → math
　中身に応じて自動で大きさが調整される括弧で囲う．

`math-paren-with-middle` : paren → paren → paren → math list → math
　第 3 引数で与えた括弧で第 4 引数の数式のリストを区切って出力する．第 1 引数と第 2 引数の扱いは `math-paren` を同じである．

`text-in-math` : math-class → (context → inline-boxes) → math
　数式中にインラインボックス列を埋め込む．

`math-variant-char` : math-class → math-char-style → math
　数式用文字書体の指定（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` : color → math → math
　数式の文字色を変更する．

`math-char-class` : math-char-class → math → math
　数式用文字書体を変更する．

`math-concat` : math → math → math
　数式を結合する．

`math-group` : math-class → math-class → math → math
　`math-group` cls_1 cls_2 m で数式 m 内は通常通りのスペーシングで組むが，m 全体の左側のスペーシングについては cls_1 を，右側のスペーシングについては cls_2 を持っているかのように扱わせる．

`math-pull-in-scripts` : math-class → math-class → (math option → math option → math) → math
　添字と指数がつけられたときに，“それらを内側に取り込んで使える”数式を作成する．`math-pull-in-scripts` cls_1 cls_2 f で m 全体の左側のスペーシングについては cls_1 を，右側のスペーシングについては cls_2 を持っているかのように扱わせる．“中身の数式”は使われた箇所で，

- 添字も指数も付かなかった場合：f `None None`
- 添字 m_1 だけついた場合：f `Some(`m_1`) None`
- 指数 m_2 だけついた場合：f `None Some(`m_2`)`
- 添字 m_1 も指数 m_2 もついた場合：f `Some(`m_1`) Some(`m_2`)`

　を評価した結果の数式となる．

`get-left-math-class`, `get-right-math-class` : context → math → math-class option
　第 1 引数のテキスト処理文脈に基づき，第 2 引数の数式の左端と右端がそれぞれどのよ

うな属性の“アトム”になっているか返す．数式が空の場合は `None` が返る．

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）用意してある．現状では PDF と JPEG のみをサポートしている．

`load-pdf-image` : string → int → 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` : string → 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` : image → length → 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を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` : point → pre-path
点を受け取り，その点からパスを開始する．

`line-to` : point → pre-path → 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` : point → point → point → pre-path → 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` : pre-path → 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` : point → point → pre-path → 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` : pre-path → 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` : path → path → 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．

`clip-graphics-by-path` : path → graphics → graphics
`clip-graphics-by-path pat gr` でグラフィクス `gr` をパス `pat` で切り抜く。

`fill` : color → path → 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．

`dashed-stroke` : length → (length ∗ length ∗ length) → color → path → graphics
`dashed-stroke` t $(len, space, start)$ *color path* でパス *path* を幅 t，色 *color* の破線として描いたグラフィックスを返す．このとき，*len* で破線一つ一つの長さを指定し，*space* で破線同士の間の長さを指定し，*start* で最初の破線の長さを指定する．

`stroke` : length → color → path → 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` : point → inline-boxes → graphics
`draw-text` v *ib* で位置 v をベースラインの左端としてインラインボックス列 *ib* を置いたグラフィックスを返す．

`shift-path` : point → path → path
path `|> shift-path` *pt* で *path* を *pt* 座標分だけ移動させる．

`shift-graphics` : point → graphics → graphics
gr `|> shift-graphics` *pt* で *gr* を *pt* 座標分だけ移動させる．

`get-path-bbox` : path → (point ∗ point)
`get-path-bbox` *path* で *path* をのバウンディングボックスの座標の内，x 座標が取る最小値と y 座標が取る最小値の組 (x_{min}, y_{min}) と，x 座標が取る最大値と y 座標が取る最大値の組 (x_{max}, y_{max}) を返す．

`get-graphics-bbox` : graphics → (point * point)

`get-path-bbox` と同様に `get-graphics-bbox` *gr* で *gr* をのバウンディングボックスの大きさを返す．ストロークの太さは考慮されない．

`linear-transform-path` : float → float → float → float → path → path

path `|>` `linear-transform-path` *a b c d* で *path* に対して線形変換を行う．

`linear-transform-graphics` : float → float → float → float → graphics → graphics

gr `|>` `linear-transform-graphics` *a b c d* で *gr* に対して線形変換を行う．

2.9. 相互参照に関する処理

`register-cross-reference` : string → string → unit

`register-cross-reference` *key value* で相互参照の ID *key* に内容 *value* を紐づけて登録する．既に *key* に内容が紐づけられていた場合は内容は上書きされ，内容が異なっていた場合は不動点に達していないとして再度 SATYSFI の処理が行われる．

`get-cross-reference` : string → string option

`get-cross-reference` *key* で相互参照の ID *key* に紐づけられた内容 *value* を取得する．何も紐づけられたいなかった場合は `None` を返す．

2.10. ページ分割に関する処理

`clear-page` : block-boxes

強制的な改ページを行う．

`add-footnote` : block-boxes → inline-boxes

`add-footnote` *bb* でブロックボックス列 *bb* を脚注として挿入する． 返り値のインラインボックス列は， これが挿入されたページに脚注が挿入されるようになる以外は `inline-nil` と同じ挙動をする．

`hook-page-break` : ((|page-number = int|) → point → unit) → inline-boxes

`hook-page-break` *f* で `inline-nil` と同じ挙動をするインラインボックス列を得ることができる．このインラインボックス列は，ページ分割処理の最中に与えた関数 *f* に対してそのインラインボックス列が挿入されたページのページ番号とページ内の座標を与える．

`hook-page-break-block` : ((|page-number = int|) → point → unit) → block-boxes

`hook-page-break` のブロック版である．

2.11. PDF の機能に関する処理

`register-link-to-uri` : string → point → length → length → length → (length * color) option → unit

`register-link-to-uri` *uri point w h d frame* で *uri* へのハイパーリンクを作成する．ハイパーリンクを設置する場所を座標 *point*・幅 *w*・高さ *h*・深さ *d* で指定し，PDF ビューワでの閲覧時にのみ表示される枠線の幅と色を *frame* で指定する．

`register-link-to-location` : string → point → length → length → length → (length * color) option → unit

`register-link-to-location` *key point w h d frame* で *key* へのハイパーリンクを作成する．*key* は `register-destination` で登録したものに対応する．

`register-destination` : string → point → unit

`register-destination` *key point* で *point* をジャンプ先に指定したリンクを *key* と紐づけて登録する．

`register-outline` : (int * string * string * bool) list → unit

`register-outline` *lst* で *lst* の中身を PDF のしおりとして登録する．タプルの中身は，しおりの深さ・しおりのタイトル・ジャンプ先のキー名・デフォルトの折りたたみ状態をそれぞれあらわす．

`register-document-information` : (title = string option; subject = string option; author = string option; words = string list) → unit

`register-document-information` *info* で *info* の中身を PDF の辞書情報として登録する．

2.12. document 型に関する処理

`page-break` : page → page-count-func → page-parts-func → block-boxes → document

ブロックボックス列をページ分割処理を施すための情報などが含まれたデータに変換する．一段組のみをサポートする．

`page-break-two-column` : page → length → (unit → block-boxes) → page-count-func → page-parts-func → block-boxes → document

二段組をサポートする．第 2 引数は一段目と二段目の *text − origin* の *x* 座標の差である．

`page-break-multicolumn` : page → length list → (unit → block-boxes) → (unit → block-

boxes) → page-count-func → page-parts-func → block-boxes → document
　多段組をサポートする．

2.13. テキスト出力モードに関する処理

`stringify-inline` : text-info → inline-text → string
　`stringify-inline` *tinfo it* で *tinfo* に基づいて *it* を処理し，テキストに変換する．

`stringify-block` : text-info → block-text → string
　`stringify-block` *tinfo bt* で *tinfo* に基づいて *bt* を処理し，テキストに変換する．

`deepen-indent` : int → text-info → string
　tinfo `|>` `deepen-indent` *i* で *i* 分だけインデントの量を大きくしたテキスト文脈を返す．

`break` : text-info → string
　改行文字を出力し，`deepen-indent` で設定したインデントを出力する．

`get-initial-text-info` : unit → text-info
　テキスト文脈を返す．

2.14. 文字列出力に関する処理

`display-message` : string → unit
　`display-message` *msg* で *msg* を標準出力に出力する．

`abort-with-message` : string → Any
　`abort-with-message` *msg* で *msg* を出力し，SATYSF_I の処理を中止する．型は任意の型に評価される．