SATySFi v0.1.0 開発進展の現状報告

2022 年 9 月 24 日

SATySFi Conf 2022

Takashi Suwa

@gfngfn

@bd_gfngfn

概要

主に v0.1.0 の開発の進展報告です

- 既に実装した言語機能：
  - 具象構文の細かい変更
  - 列多相による**ラベルつきオプション引数**とレコード
  - **F-ing modules** に基づくモジュールシステム
    - 多段階計算とも共存可能
  - 数式コマンドの新しい意味論
    - ようやく数式がテキスト出力モードにも対応
- 近いうち（今年中くらい）にできそうなこと：
  - **パッケージシステム**の実装
  - 縦書きサポートのための一般化
  - フォントデコーダ・エンコーダの切替え・機能拡充

目次

- **具象構文などの細かい変更**
- ラベルつきオプション引数とレコード
- F-ing modules によるモジュールシステム
- 数式コマンドの新しい意味論

} 実装済

- パッケージシステム
- 縦書きサポートのための一般化

} 近いうちに実装予定

- まとめ

具象構文の細かい変更

```
module Tree :> sig
  type tree ‘a
  val dfs ‘a : tree ‘a -> list ‘a
  …
end = struct

  type tree ‘a =
    | Empty
    | Node of ‘a * tree ‘a * tree ‘a

  val rec dfs t =
    match t with
    | Empty -> []
    | Node(x, t1, t2) ->
        List.concat
          [ [ x ], dfs t1, dfs t2 ]
    end
  …
end

val inline ctx \emph it =
  let ctx =  …  in
  read-inline ctx it
```

- 型引数は後置
  - 例： `list int`, `option length`
- コマンド型も引数を後置
  - 例： `inline [int, inline-text]`
- 型変数の全称量化の明示

具象構文の細かい変更

```
module Tree :> sig
  type tree 'a
  val dfs 'a : tree 'a -> list 'a
  …
end = struct

  type tree 'a =
    | Empty
    | Node of 'a * tree 'a * tree 'a

  val rec dfs t =
    match t with
    | Empty -> []
    | Node(x, t1, t2) ->
        List.concat
          [ [ x ], dfs t1, dfs t2 ]
    end
  …
end

val inline ctx \emph it =
  let ctx =  …  in
  read-inline ctx it
```

- 型引数は後置
  - 例： `list int`, `option length`
- コマンド型も引数を後置
  - 例： `inline [int, inline-text]`
- 型変数の全称量化の明示
- トップレベルの **`let`** は **`val`** に変更
- **`let-rec`** → **`let rec`** / **`val rec`**
- **`let-inline`** → **`val inline`**
  - block や math も同様

具象構文の細かい変更

```
module Tree :> sig
  type tree 'a
  val dfs 'a : tree 'a -> list 'a
  …
end = struct

  type tree 'a =
    | Empty
    | Node of 'a * tree 'a * tree 'a

  val rec dfs t =
    match t with
    | Empty -> []
    | Node(x, t1, t2) ->
        List.concat
          [ [ x ], dfs t1, dfs t2 ]
    end
  …
end

val inline ctx \emph it =
  let ctx =  …  in
  read-inline ctx it
```

- 型引数は後置
  - 例： `list int`, `option length`
- コマンド型も引数を後置
  - 例： `inline [int, inline-text]`
- 型変数の全称量化の明示
- トップレベルの **`let`** は **`val`** に変更
- **`let-rec`** → **`let rec`** / **`val rec`**
- **`let-inline`** → **`val inline`**
  - block や math も同様
- **`match`** … **`with`** … **`end`**
  - ぶら下がりによる曖昧性防止
- リストやレコードの区切り記号は `;` ではなく `,` に変更

その他の非互換な変更

- `direct` 廃止
- `.satyh` ファイルの内容は 1 つのモジュールの定義のみに限定
- `page` 型の廃止
  - `A4Paper` などのコンストラクタではなく
    単に `PaperSize.a4 : length * length` などによる指定に変更
- `graphics` 型の変更
  - 従来 `graphics list` 型だった値が `graphics` 型になる
    - `graphics list` 型の値と `graphics` 型の値の違いが曖昧だったため解消
  - `unite-graphics : list graphics -> graphics` で重ね合わせる
- 数式文字クラスの追加
  - `MathSansSerif` や `MathTypewriter` など 5 種類，Unicode 準拠

目次

- 具象構文などの細かい変更
- **ラベルつきオプション引数とレコード**
- F-ing modules によるモジュールシステム
- 数式コマンドの新しい意味論

} 実装済

- パッケージシステム
- 縦書きサポートのための一般化

} 近いうちに実装予定

- まとめ

従来のオプション引数

従来の SATySFi にもラベルなしオプション引数 `?:(...)` はあった

```
+section?:(`sec:sample`)?:(`Sample`){Sample}<
  （章の本文）
>
```

- `(`sec:sample`)` : LaTeX の \label 相当
- `(`Sample`)` : PDF の outline に章題として出す文字列
- `{Sample}` : 版面上に組む章題

cf. LaTeX： 慣習的に角括弧で与える

```
\parbox[c]{6zw}{..}, $\sqrt[3]{2}$
```

不便な点：

- 順番に意味があり，どの指定を何番目にすべきか解りづらい
- 手前のものを省略するときは `?*` で “明示的に省略” する必要あり

```
+section?*?:(`Sample`){Sample}<
  （章の本文）
>
```

ラベルつきオプション引数

```
+section?(ref = `sec:sample`, outline = `Sample`){Sample}<
  (章の本文)
>
```

- ?($label_1$ = arg_1, ..., $label_n$ = arg_n) の形で使用
  - ひとつも指定しない場合は全体を省略
- 順番は自由

美点：

- ラベル名がつくので単純に何に関する指定かわかりやすい
- 順不同なので一部省略時にもぎこちない方法をとる必要なし
- 相変わらず静的に型がつき，想定していないラベル名を
  指定した場合や引数の型がおかしい場合は型エラー
  - `+section : block [ ?(ref : string, outline : string) inline-text, block-text ]`

オプション引数に対する多相性

- コマンドだけでなく一般の函数もオプション引数を取れる

```
let increment ?(diff = diff-opt) n =
  match diff-opt with
  | None    -> n + 1
  | Some(d) -> n + d
  end
```

```
– increment : ?(diff : int) int -> int
```

オプション引数に対する多相性

- コマンドだけでなく一般の函数もオプション引数を取れる

```
let increment ?(diff = diff-opt) n =
  match diff-opt with
  | None    -> n + 1
  | Some(d) -> n + d
  end
```

```
– increment : ?(diff : int) int -> int
```

- 高階函数のために，**列多相** [Gaster & Jones 1996] の仕組みがある
  - ちょっとした独自の型システム

```
let use-optional f = f ?(foo = 42) 57 + 1
```

```
. use-optional : ∀ρ. (?(foo : int, ρ) int -> int) -> int
```

. ρ が**列変数**

型　$\tau ::= ?(r)\ \tau \to \tau \mid \{r\} \mid \cdots$

列　$r ::= l : \tau, r \mid \cdot \mid \rho$

空列

レコード型も列

目次

- 具象構文などの細かい変更
- ラベルつきオプション引数とレコード
- **F-ing modules によるモジュールシステム**
- 数式コマンドの新しい意味論

} 実装済

- パッケージシステム
- 縦書きサポートのための一般化

} 近いうちに実装予定

- まとめ

従来のモジュールシステムの課題

`struct` ... `end` によるモジュールのみを扱う従来のSATYSFI のモジュールシステムではカプセル化能力に限界があった

- 例： マップ（＝辞書，連想配列）のカプセル化が困難
- 例： コードブロックを組むコマンドを提供するモジュール
  - コマンドの形で API を提供しようとすると，フォント設定や背景色もモジュール内で決め打ちになってしまい，ユーザ側から変更しづらい

```
@require: code
  ⋮
  +Code.code(```
    print(“Hello!”)
  ```);
```

```
print("Hello!")
```
```

新しいモジュールシステム

- **F-ing modules** [Rossberg, Russo & Dreyer 2014] という理論に準拠したモジュールシステムを採用
  - **ファンクタ**と呼ばれる “モジュールを受け取ってモジュールを返す大きな関数” の機能が入る
  - 実体を公開した型の定義ができるようになる

```
@require: code
  ⋮
module TerminalSettings = struct
  val font-family = `Menlo`
  val text-color  = RGB(0.0, 1.0, 0.0)
  val fill-color  = Gray(0.25)
end

module Terminal = Code.Make TerminalSettings
```

ファンクタ Code.Make に
スタイル設定のモジュールを渡し,
設定が反映された結果を
Terminal という
モジュールとして使う

```
  ⋮
  +Terminal.show(```
    $ gcc hello.c -o hello
  ```);
```
```

最近の差分： モジュールと多段階計算との共存

- モジュール自体はステージをもたない（従来の `@stage:` は廃止）
- 1 つの `struct ... end` 中に複数ステージの値が共存できる
- 実は論文が 1 つ書けるくらいの工夫が必要だった

```
module Timestamp :> sig
  type t
  val advance_by_seconds : int -> t -> t
  ~val generate : string -> code t
  …
end = struct
  type t = int % 内部表現は Unix time
  val advance_by_seconds n t = …
  ~val generate s =
    match parse_iso8601_datetime s with
    | …
end
```

絶対時刻を扱うモジュール
Timestamp に，時刻文字列から
Timestamp.t 型の値を生成する
マクロを，抽象化を壊さず
自然に追加できる

```
val beginning_of_satysfi_conf_2022 =
  ~(Timestamp.generate
    `2022-09-24T17:30:00+09:00`)
```

目次

- 具象構文などの細かい変更
- ラベルつきオプション引数とレコード
- F-ing modules によるモジュールシステム
- **数式コマンドの新しい意味論**

} 実装済

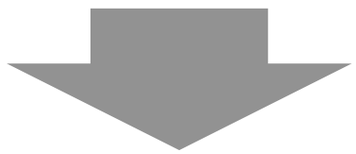
- パッケージシステム
- 縦書きサポートのための一般化

} 近いうちに実装予定

- まとめ

数式コマンドの意味論の変更

- 従来の数式コマンド：
  - インライン/ブロックコマンドとは異なり，context を受け取らない
  - これはこれで悪くないが，テキスト出力モードに自然に拡張できなかった



- 新しい数式コマンド：
  - **context を受け取る**
  - **`math-text`** 型と **`math-boxes`** 型とを区別
    - `${ ... }` に `math-text` 型がつく
  - **`read-math`** `: context -> math-text -> math-boxes` で変換
  - **テキスト出力モードを自然にサポート**

新しい数式コマンドの定式化

- **context を受け取る**
- **`math-text`** 型と **`math-boxes`** 型とを区別
  - `${ ... }` に `math-text` 型がつく
- **`read-math : context -> math-text -> math-boxes`** で変換
- **`embed-math : context -> math-boxes -> inline-boxes`** で埋め込む

```
val math ctx \frac (mt1 : math-text) (mt2 : math-text) =
  math-frac ctx (read-math ctx mt1) (read-math ctx mt2)

val math ctx \Gamma =
  let s =
    match get-math-char-class ctx with
    | MathItalic -> `Γ` % U+1D6E4
    ...
    end
  in
  math-char ctx MathOrd s
```

分数のプリミティヴ

```
math-frac :
  context ->
  math-boxes -> %分子
  math-boxes -> %分母
    math-boxes
```

添字・指数を扱えるコマンド

- **val math** *ctx* *\cmd* x_1 … x_n **with** *sub sup* = *e* という構文で
  \cmd を定義すると，添字・指数がついた場合に
  それらが *sub*, *sup* : option math-text に渡されて使える

- 従来は math-pull-in-scripts というプリミティブで実現

```
val math ctx \sum with sub sup =
let mb-op = math-big-char ctx MathOp `∑` in
let mb =
  match sub with
  | None       -> mb-op
  | Some(mt1) -> math-lower ctx mb-op (fun ctx -> read-math ctx mt1)
  end
in
match sup with
| None       -> mb
| Some(mt2) -> math-upper ctx mb (fun ctx -> read-math ctx mt2)
end
```

テキストモードでの数式コマンド定義

- math-boxes の代わりに string がコマンド定義の戻り値の型となる
- **stringify-math** : text-info -> math-text -> string で文字列化

```
val math tinfo \frac mt1 mt2 =
  let f = stringify-math tinfo in
  `\frac{` ^ (f mt1) ^ `}{` ^ (f mt2) ^ `}`

val math tinfo \Gamma =
  `\Gamma `#
```

目次

- 具象構文などの細かい変更
- ラベルつきオプション引数とレコード
- F-ing modules によるモジュールシステム
- 数式コマンドの新しい意味論

} 実装済

- **パッケージシステム**
- 縦書きサポートのための一般化
- まとめ

} 近いうちに実装予定

パッケージシステムの必要性

- 従来はパッケージの概念が曖昧
  - `.satyh` ファイルを「パッケージファイル」と呼んでいるものの,
    実態はパッケージという感じでもない
  - **Satyrographos** が扱う単位（≒ OPAM パッケージ）が
    事実上 SATYSFI のパッケージになっており，こちらの方が妥当そう

- モジュール/パッケージとは何か？
  - モジュール：
    - 実装の抽象化に関する単位
  - パッケージ：
    - リリースに関する単位をなすモジュールの集まり
    - どのモジュールを外部に公開するかも制御

- リリースの制御は Satyrographos との分担が非自明だが,
  少なくとも**各パッケージがどのモジュールを公開するかの制御は必要**

パッケージシステムの設計案

- **各パッケージは 1 つだけモジュールを公開する**
  - 原則としてパッケージと同名を推奨
  - 他のモジュールはその入れ子のモジュールとして公開すれば OK
    - 例： `satysfi-base` パッケージの `Int` モジュールなら `Base.Int`
    - 使う側で適宜 `open Base` などとする
- `~/.satysfi` 等以下ではパッケージごとにディレクトリを分けるとする
  - 現行の Satyrographos と同様
- 以下の内容を含む**設定ファイル**をパッケージのディレクトリ直下に置く
  - ソースファイルのあるディレクトリ
  - 公開するただ 1 つのモジュール
  - 依存する他のパッケージ

```
packages/
  base/
    satysfi.yaml
    src/
      base.satyh
      int.satyg
      …
  easytable/
    satysfi.yaml
    src/
      …
```

Satyrographos との責務の境界はどこか？

- 責務の境界を決めるのが難しい
- 現状の想定：
  - モジュールの公開/非公開単位としてのパッケージは SATγSF| が担う
  - リリースの単位としてのパッケージは Satyrographos が担う
    - `satysfi.yaml` も `Satyristes` から生成されるとよいのかも
- しかし，例えば軽量な文書ではファイル冒頭で
  パッケージのバージョン制約が指定できると嬉しいかも？
  - 実現するには SATγSF| がリリースの単位まで意識する必要あり

```
#[dependencies([
  (`base`, `~1.0.0`),
  (`std-ja-report`, `>= 0.5.0 && < 0.5.5`),
])]
require open Base
require open StdJaReport
```

目次

- 具象構文などの細かい変更
- ラベルつきオプション引数とレコード
- F-ing modules によるモジュールシステム
- 数式コマンドの新しい意味論

} 実装済

- パッケージシステム
- **縦書きサポートのための一般化**
- まとめ

} 近いうちに実装予定

縦書きや LTR/RTL のための一般化

- 文字体系の方向性 (directionality) の定式化の前例：
  - UTN #22: Robust Vertical Text Layout [Etemad 2005]
  - TEX エンジンの拡張の一種 **Omega** [Plaice & Haralambous 1994–2000]
- いずれも以下の 3 要素によって方向性を分類：
  - **Beginning of the page**
    - 段落の開始側が紙面上の絶対方向として上下左右のどれか
    - 欧文・アラビア文字： 上，縦書き和文： 右，モンゴル文字： 左
  - **Beginning of the line**
    - 行の開始側が紙面上の絶対方向として上下左右のどれか
    - 必ず beginning of the page と 90 度をなす 2 通りのうちどちらか
    - 欧文： 左，アラビア文字： 右，縦書き和文・モンゴル文字： 上
  - **Top of the line**
    - 行中の各文字がどちらを上側とみなして並ぶか
    - 基本的には上だが，例えば縦書き和文中の欧文は右倒しになるので右

縦書きや LTR/RTL のための一般化

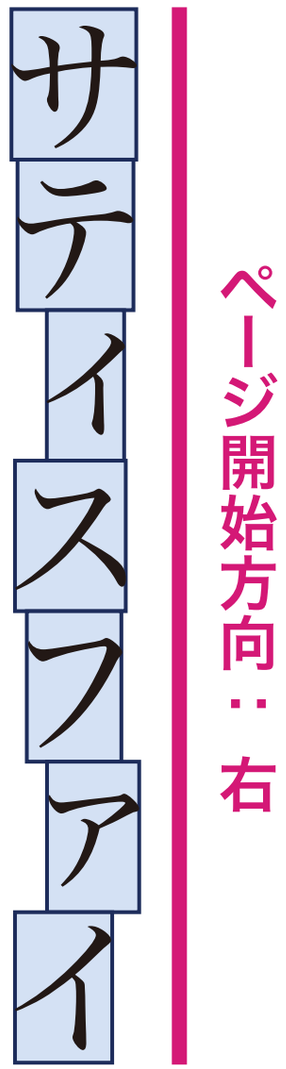
- 文字体系の方向性 (directionality) の定式化の前例：
  - UTN #22: Robust Vertical Text Layout [Etemad 2005]
  - TEX エンジンの拡張の一種 **Omega** [Plaice & Haralambous 1994–2000]

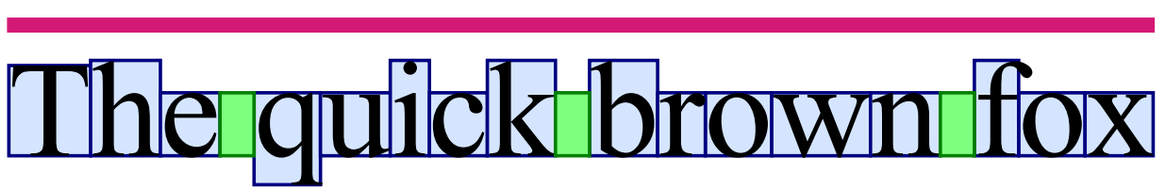
- いずれも以下の 3 要素によって方向性を分類：
  - **Beginning of the page**
    - 段落の開始側が紙面上の絶対方向として上下左右のどれか
    - 欧文・アラビア文字： 上，縦書き和文： 右，モンゴル文字： 左
  - **Beginning of the line**
    - 行の開始側が紙面上の絶対方向として
    - 必ず beginning of the page と 90 度を
    - 欧文： 左，アラビア文字： 右，縦書き
  - **Top of the line**
    - 行中の各文字がどちらを上側とみなして並ぶか
    - 基本的には上だが，例えば縦書き和文中の欧文は右倒しになるので右

これだけは型レベルでも区別し，
同じ beginning of the page をもつ
ボックス列でないと
結合できないようにする

縦書きの型レベルでの定式化

- **ページ開始方向** $d ::=$ **@T** | **@R** | **@L** | δ を導入
  - Beginning of the page を表す
  - δ は**ページ開始方向変数**, 多相性に使う
- テキストやボックスの型はページ開始方向を引数にもつように一般化
  - `inline-text` d や `block-boxes` d という具合
  - コマンドも `inline` d `[...]` 型などになる
  - 従来はすべて $d =$ **@T** に限定されていたと考える





: `inline-boxes` **@T**

: `inline-boxes` **@R**

縦書きの型レベルでの定式化

Beginning of the page が相等しいボックス同士しか
結合できないことを静的に保証するにはどうするか？

ページ開始方向変数で型を多相にする！

- `(++) : ∀δ. inline-boxes δ -> inline-boxes δ -> inline-boxes δ`

その他，欧文を縦書き和文中に埋め込む場合などは明示的に変換

- `top : bop @T`
- `right : bop @R`
- `rotate-to : ∀δ∀δ′. bop δ′ -> inline-boxes δ -> inline-boxes δ′`

`rotate-to right ib` で `ib : inline-boxes @T` を回転して `@R` にできる

目次

- 具象構文などの細かい変更
- ラベルつきオプション引数とレコード
- F-ing modules によるモジュールシステム
- 数式コマンドの新しい意味論

} 実装済

- パッケージシステム
- 縦書きサポートのための一般化

} 近いうちに実装予定

- **まとめ**

まとめ

SATγSFI v0.1.0 に入る予定の機能のうち,
実装が完了したものと近いうちに実装できそうなものを紹介しました

- 実装完了分：
  - 具象構文の細かい変更
  - 列多相による**ラベルつきオプション引数**とレコード
  - **F-ing modules** に基づくモジュールシステム
    - 多段階計算とも共存可能
  - 数式コマンドの新しい意味論
    - ようやく数式がテキスト出力モードにも対応
- 近いうち（今年中くらい）にできそう：
  - **パッケージシステム**の実装
  - 縦書きサポートのための一般化
  - フォントデコーダ・エンコーダの切替え・機能拡充