```
new Presentation{
  title:
```

実用水準の
プログラミング言語を
個人規模でつくる

```
  date:      2023-10-07
  event:     情報科学若手の会 2023
```


```
  author:  諏訪 敬之 (Takashi Suwa)
}
```

背景

- 多くの方が言語処理系を自作するようになった
  - 素晴らしいこと！
  - 目的・こだわりポイントは人それぞれ
    - 自分の理解のため，ちょっとした実用のため，ロマン，……
    - 生成コードの最適化，構文，型システム，セルフホスティング，……

一方で ……

- **明確な用途がある言語を，自分以外でも実用できるくらいに**
  ゼロから作り込もうとする人は案外少ないかも

概要

- 講演者，どんな人？
  - 実用水準の言語処理系の制作経験がある
    - **SATγSFI**：“**静的型つき函数型組版処理システム**”（2017年度未踏事業）
      - 広く使われていると言うには程遠いが，数百人ほどユーザがいる様子
      - 自身の修士論文や技術同人誌の執筆に使用
    - **SesterI**：“**静的型つき Erlang**”
      - 自分自身でオンライン盤上ゲームサーバの実装に使用

- 発表のねらい：
  - **実用水準の言語をつくる楽しさ・意義を感じてもらう**（最重要！）
    - SATγSFI を具体例にしていろいろ紹介します
  - **言語を作る上で経験した難しい点・上手くいかないところを共有し，これから取り組む人の転ばぬ先の杖にしてもらう**

目次

- **SATySFi の紹介**
- Sesterl の簡単な紹介
- 自作言語の意義・楽しさ
- 自作言語の始め方
- 私の場合に功を奏したこと
- プロジェクトが進むにつれ悩ましくなること・私の反省点
- まとめ

SATYSFI（サティスファイ）とは

- 講演者が 2017 年度 IPA 未踏事業の1プロジェクトとして
  開発した**組版処理システム**
- **静的型つき**のいわゆる**函数型**のプログラミング言語で
  パッケージや文書本体が書ける
  - 一般的なプログラムと同じ要領で読み書きできるので
    カスタマイズ性が高い
    - cf. TEX/LATEX： 複雑な意味論ゆえにパッケージ作成には鍛錬を要し，
      熟練者でも他人の書いた実装は読んで改変するのが難しい
  - 誤った記述をしたときも，型検査によって**迅速**（典型的には 0.1 秒）
    かつ原因の解りやすいエラー報告を出してくれる傾向にある
    - cf. TEX/LATEX： 些細なミスでも十数秒待ってから不可解なエラーが出がち

文書作成ソフトウェアの形態

大別して 2 種類：

マークアップ言語＋処理系

TEX・LATEX

[Knuth 1978] [Lamport 1985]

前処理用途

RE:VIEW

[青木 et al. 2002]

[Gruber 2004]

troff

[Osanna 1973]

SATySFi

WYSIWYG エディタ

[Microsoft 1983]

[Adobe 1991]

[Quark 1987]

マークアップ言語方式の性質

WYSIWYG エディタ方式と比べて，傾向として：

- **差分管理**が単純
- **ユーザ定義コマンド**により：
  - **複雑な自動処理**が実現可能
  - 文書の体裁が
    後から柔軟に変更可能

この特徴を活かしたい人が想定ユーザ

- **ユーザが不適格なコードを入力として与えやすい**

どのようにエラーが報告されるかが執筆効率に影響

既存システムの弱点

例： TEX/LATEX

- **エラー報告が不親切で遅い**傾向にある

```
! Undefined control sequence.
```

```
! Missing $ inserted.
```

```
! Missing number, treated as zero.
```

  - これゆえに，少し凝った処理を定義しようとすると**デバッグが困難**
  - 処理系の実装が怠惰なのではなく，TEX のもつ意味論の都合上
    不親切なエラーにならざるを得ないという側面が大きい

手の込んだ自動処理が定義できつつ，“明らかなミス”には親切なエラーが出せるような構文・意味論の言語にしたい

最初の案： 型つき wrapper をつくる

静的型つきで，TEX/LATEX コードを出力するような
言語をつくればよいのではないか？

2015, 6 年頃に **Macrodown** という OCaml 風の言語として実装し，
それなりに使えるものにはなった

顕在化した弱点：

- ページ数や紙面上の幅といった組版上の概念に依存する処理は
  wrapper 側で実装しにくく，**依然として LATEX 側での実装が必要**
  - 特にクラスファイルの実装などは wrapper 側では到底完結しないはず

目的意識

LaTeX バックエンドを wrap した言語で済ませずに,
自前で組版処理まで行なえる言語と処理系を設計・実装したい

主要な要件：

- **本質的に組版処理に関わるような複雑な自動処理ができること**
  - 他人の書いた自動処理の実装も大きな支障なく読んで改変しやすいことも含む
- **明らかなエラーについては静的に**（＝組版処理を始めるより前に）**検知し, その原因をなるべく特定しやすい形で報告できること**

蛇足： もっと雑な問題意識

/*

- TEX/LATEX はもちろん或る面では素晴らしいソフトウェアだが,
  これだけ言語設計の知見が溜まっている現在でも
  扱いにくい意味論をもつ TEX にずっと依存し続けるのは
  工学的に健全とは言いにくいのではないか？
  - cf. 汎用の言語では様々な言語設計が日夜提案されている

- 日々プログラムの性質を静的に保証することを熱心に考えているのに,
  いざ論文を書く際にはその結果を享受せず
  不親切なエラーばかり見ているのは，紺屋の白袴ではないか？

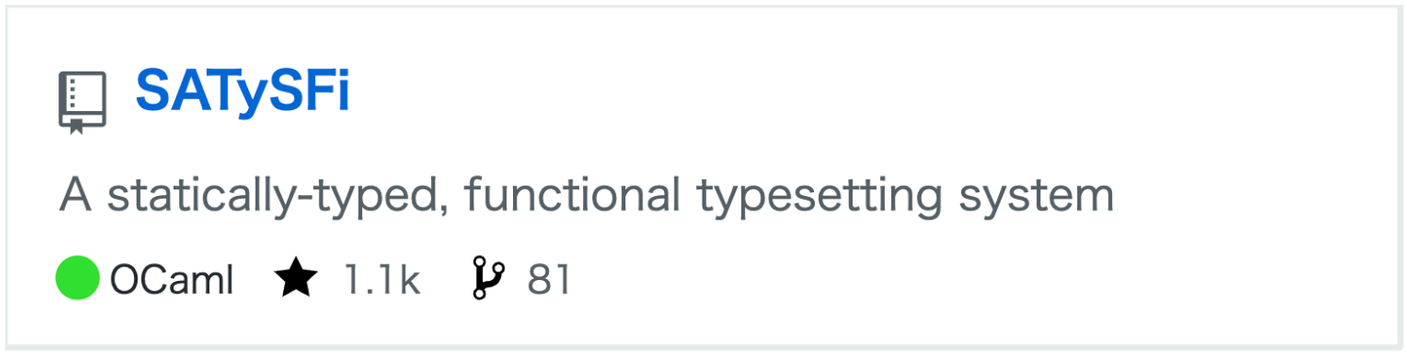
*/

成果物

新しい組版処理システム

SATySFi

https://github.com/gfngfn/SATySFi



Static **A**nalysis-based **TY**pesetting **S**ystem
for **F**unctional **I**mplementation

（かなり強引なこじつけ）

特徴

構文が大別して 2 層に分かれている

“マークアップ層”

```
+section{\SATySFi;の原稿例}<
  +p{
    若手の会\emph{初参加}です.
    お世話になります.
  }
>
```

- LaTeX 風
- インライン { … } と
  ブロック < … > の区別がある

“プログラム層”

```
let-inline ctx \emph it =
  let ctx =
    ctx |> set-font
      Latin italic-font
  in
  read-inline ctx it
```

- OCaml 風

特徴

“プログラム層” ではコマンド定義が OCaml 風の言語で書ける

- “グローバルな状態” をあまり気にせずに処理が書ける
  - ※ 破壊的に変更可能な変数も止むを得ず入れており，採番などに使用
- **型システム**を用いた静的な（＝組版処理に先立った）エラー報告能力
  - Hindley–Milner 多相（いわゆるジェネリクスをもつ体系）＋α 程度
    - レコード計算 [Ohori 1995] [Rémy 1993] [Gaster & Jones 1996]
    - オプション引数
  - 依存型などの複雑な仕組みは無し
  - **組版処理やドキュメント処理に関わる多数の基本型と組込み函数**

動作デモ１： 正常系

- デモ映像：
  - https://drive.google.com/file/d/1muaWGgyAGfIYJNyKfKvvAzN8vur1sTG3/view?usp=drive_link

動作デモ２： 異常系

以下のような「いろは歌を幅 5cm の枠で囲って組む」出力をしようとして
やりがちな不適格なコードを書いてしまったときのエラー報告を
LaTeX と SATySFi で比較

いろは歌：

色は匂へど散りぬるを，吾が世誰ぞ恒ならむ。有為の奥山今日越えて，浅き夢見じ，酔いもせず。

LaTeX の場合

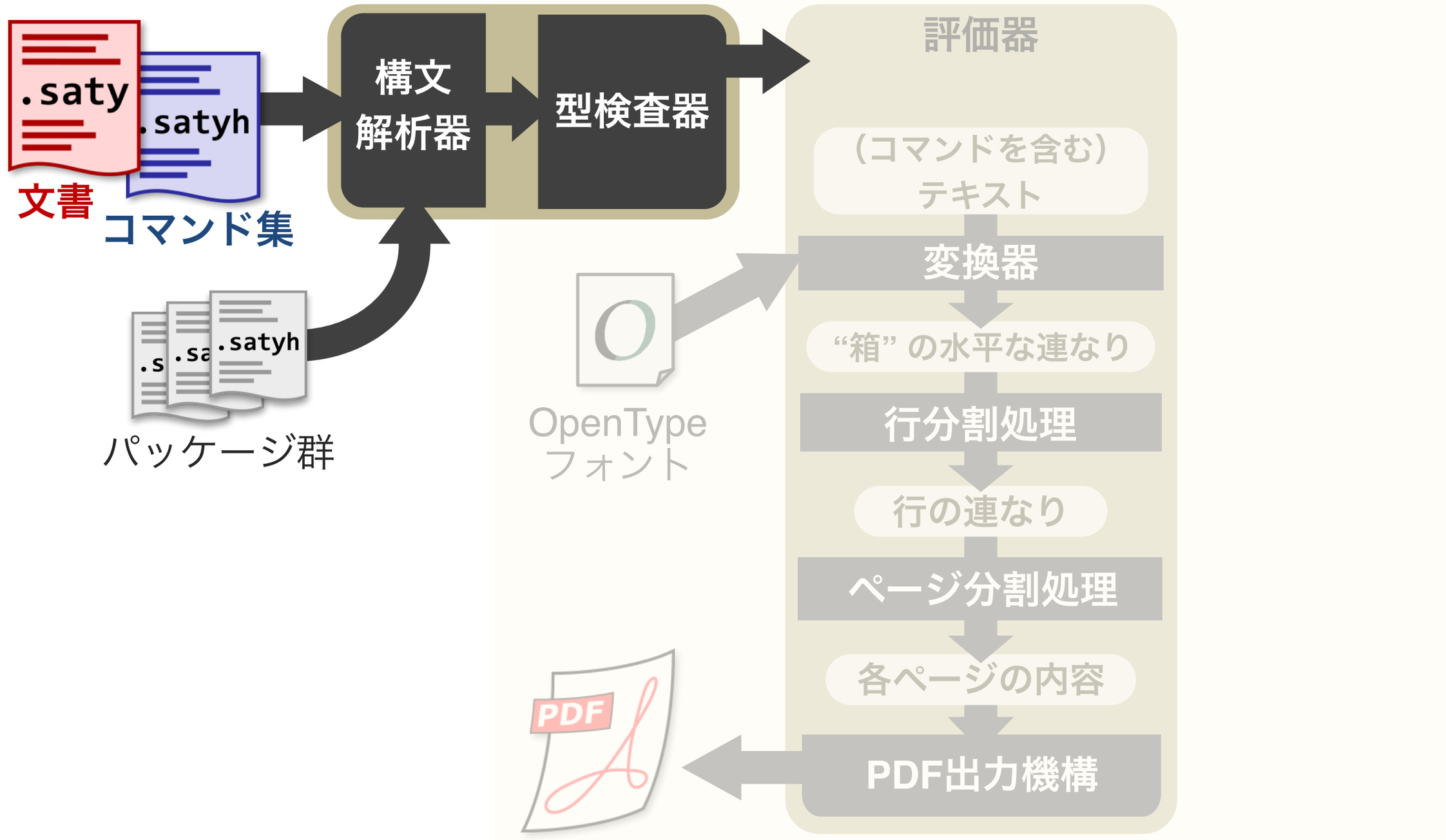
- デモ映像：
  - https://drive.google.com/file/d/1r14gHwCUxe3DQjoxkGHDWWkwtHWqMwaY/view?usp=drive_link
  - 最初に正常に処理できる場合を実行し,
    次に \parbox の幅指定の引数を書き忘れた場合に
    どんなエラーが出るかを見て,
    最後に \parbox のオプション引数 [t] を
    必須の引数だと勘違いした場合にどんなエラーが出るかを見ています

SATySFi の場合

- デモ映像：
  - https://drive.google.com/file/d/1DyKB0V2xIKz6SMYNGdEWnupPAxXkWCeE/view?usp=drive_link
  - 最初に LaTeX の場合と等価な SATySFi での実装を見せて
    それが正常に処理できることを確認し,
    次に `\parbox` の幅指定の引数を書き忘れた場合に
    どんなエラーが出るかを見て,
    最後に `\parbox` のオプション引数 `?:(Top)` を
    必須の引数だと勘違いした場合にどんなエラーが出るかを見ています
  - `\fbox` や `\parbox` はわかりやすさのために名前を揃えているだけで,
    裏で LaTeX が動いているなどではなく SATySFi のライブラリとして
    実装されているものです

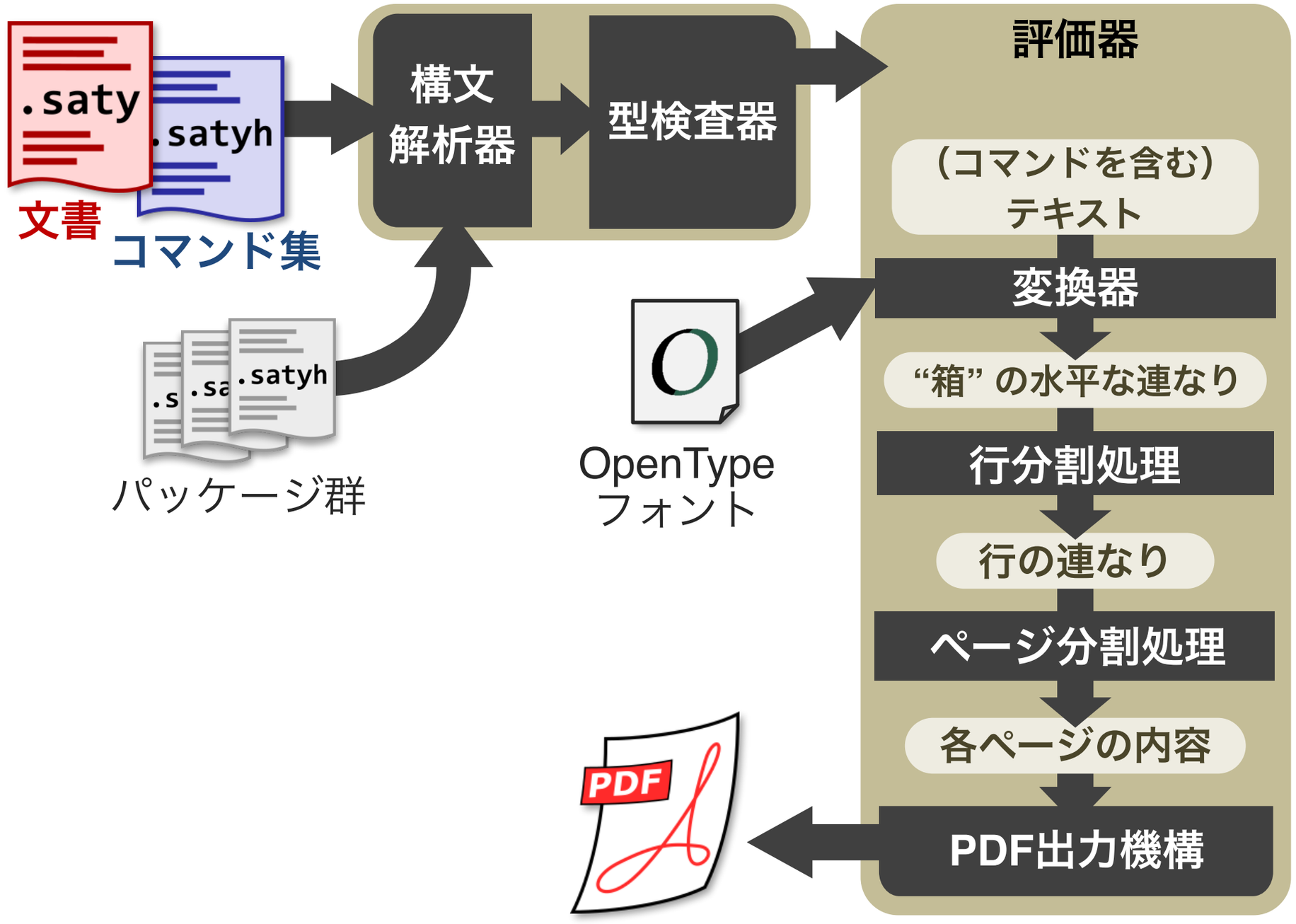
処理系の構成の模式図

フロントエンド（検査機構） ＋ バックエンド（インタプリタ）

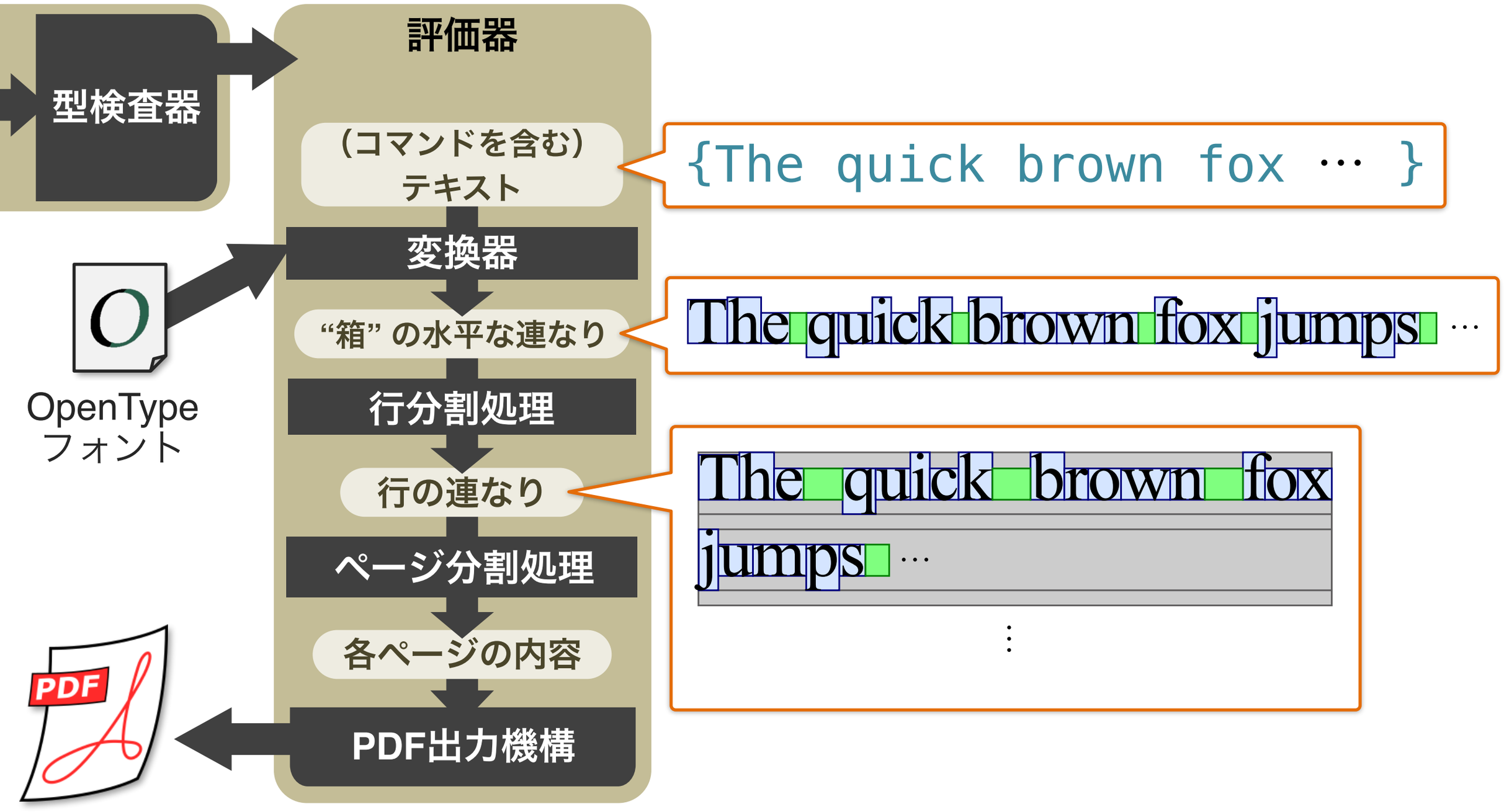


処理系の構成の模式図

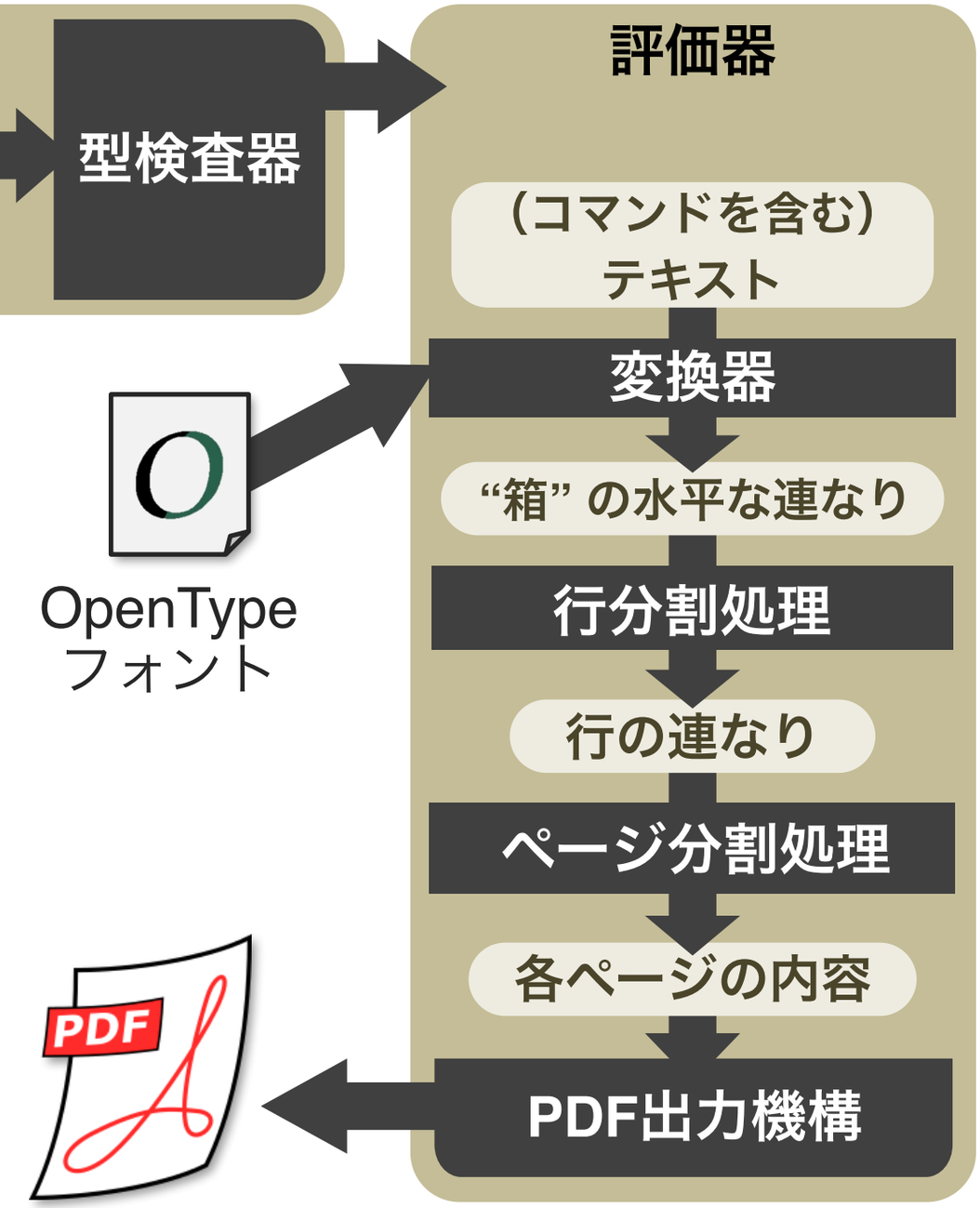
フロントエンド（検査機構） ＋ バックエンド（インタプリタ）



処理系の構成の模式図


評価器
型検査器
（コマンドを含む）
テキスト
{The quick brown fox … }
変換器
“箱” の水平な連なり
The quick brown fox jumps …
行分割処理
行の連なり
The quick brown fox
jumps …
ページ分割処理
各ページの内容
PDF出力機構
OpenType
フォント
PDF

処理系の構成の模式図



ユーザ（パッケージ開発者）は
実装でこのあたりの組版処理に
介入できる

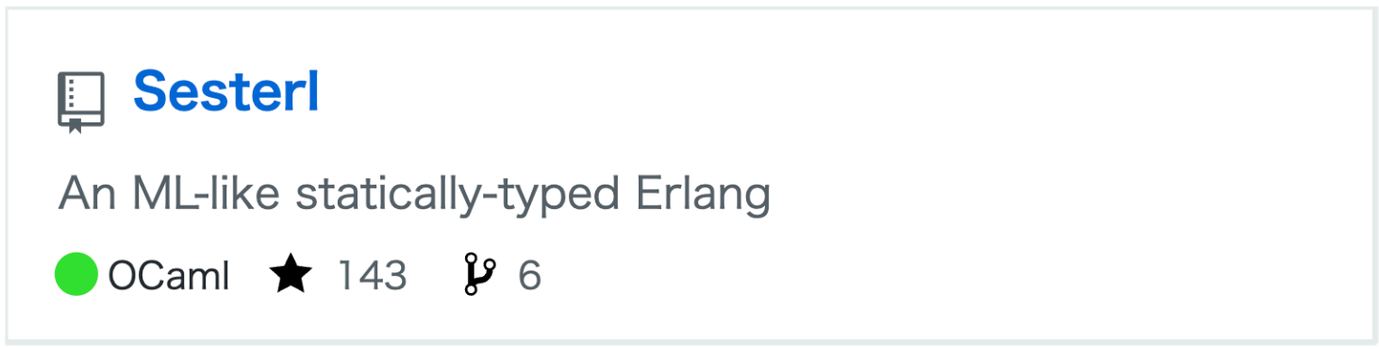
自動処理の定義が型検査に通れば,
評価時にここで
データの不整合が起きないような
型システムになっている

目次

- SATySFi の紹介
- **Sesterl の簡単な紹介**
- 自作言語の意義・楽しさ
- 自作言語の始め方
- 私の場合に功を奏したこと
- プロジェクトが進むにつれ悩ましくなること・私の反省点
- まとめ

Sesterl

https://github.com/gfngfn/Sesterl

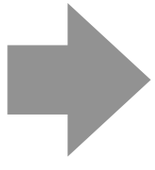


- **Erlang** をラップした OCaml 風の型つき言語
  - Erlang： **並行処理・耐障害**を得意とする言語
- 特徴：
  - 意味論はほぼ Erlang のままで，Erlang との FFI が実現しやすい
    - 既存の Erlang 実装からゆるやかに移行できる
    - 型がつけにくい部分だけ直接 Erlang で書いたりできる
  - モナドによる純粋計算・並行処理の区別
  - ML 系のモジュールシステムによる **OTP ライブラリ**の定式化
    - OTP： 成熟したライブラリ，大抵の並行処理はこれで書ける

Erlang の利点・弱点（→問題意識）

- **函数型で並行並列機能がプリミティヴとして備わった意味論**
  - 競合状態が生じにくい
- **耐障害性の追求しやすさ**
  - link/monitor によりプロセスの生死を監視する機能
  - OTP という成熟したプラットフォームの存在

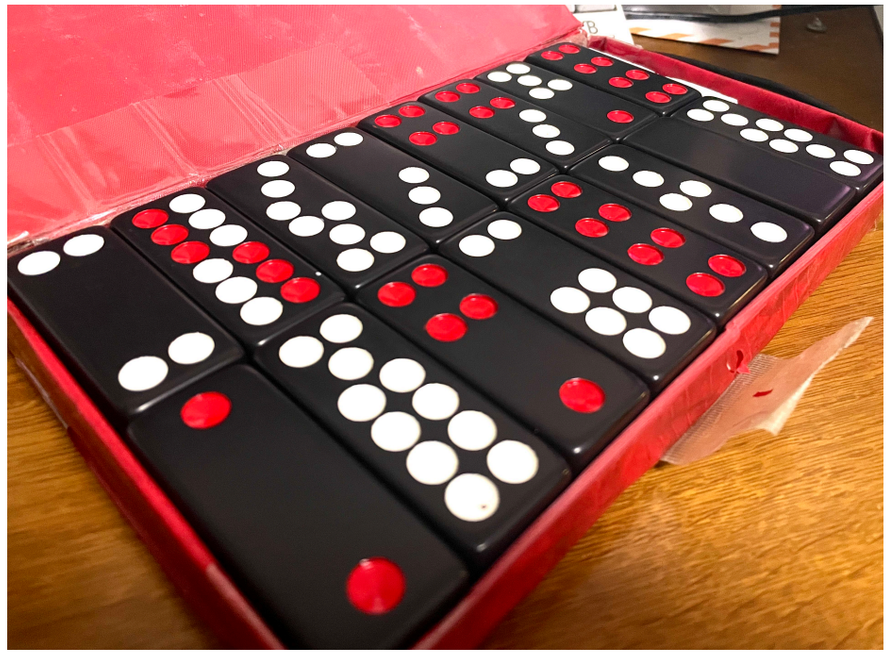
- **実装の正しさを静的に保証する手段の不足**
  - なにしろ型が事実上ない（unsound な型つけは一応ある）
- **言語機能上の細かい厄介さ**
  - 一部の状況に関しては競合状態が防げていない
    - 非同期的にプロセスを終了させ同名で再起動すると確率的にダブるなど
  - … etc.



適切な型システムを設計し，
利点を活かしつつ弱点を補えるようにしたかった

実際に自分で実用してみた

https://github.com/gfngfn/game_tianjiupai



game_tianjiupai

A Tian Jiu Pai (天九牌) game server written in Sesterl & Elm

Elm ★ 15 2

- **天九**のオンライン対戦サーバ
  - 天九 (Tian Jiu, Tien Gow)：
    - **天九牌**という 32 枚の牌を使う 4 人対局のトリックテイキングゲーム
    - 1 局 5 分未満，半荘または全荘やる
    - なかなか中毒性があるが，日本では多分数百人の物好きしかやってない
  - HTTP/WebSocket 部分は既存ライブラリ Cowboy を FFI で型つけして使用
  - フロントエンドは Elm 製
  - AWS EC2 にデプロイして動作済

様子

localhost:8080

天九 Online | gfn さん

My Room

東1局・1倍場

中断して退室

東 gfn
得点： 6

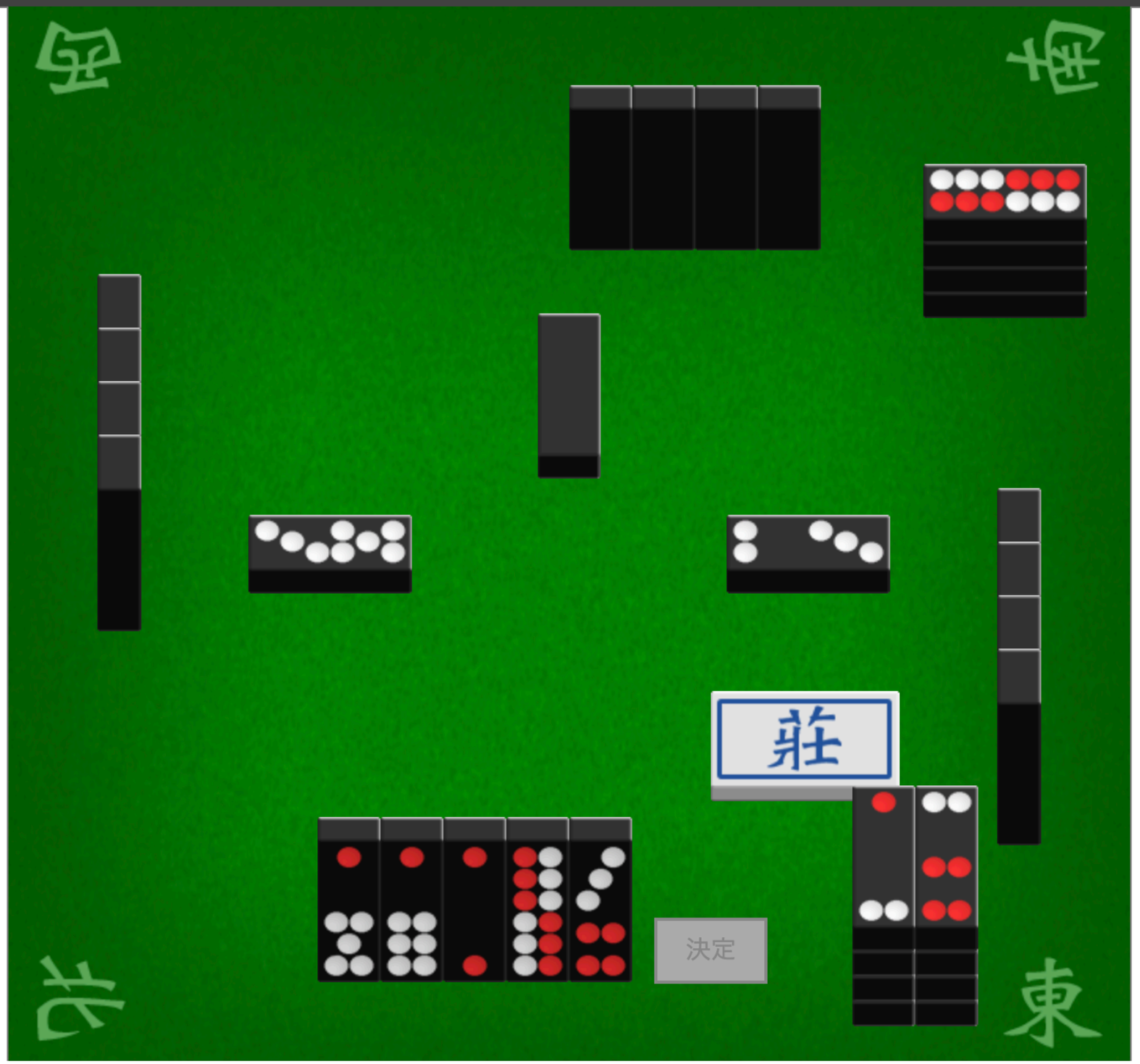
南 taro
得点： -2

西 jiro
得点： -2

北 hanako
得点： -2

debug info

- user ID: 68f8d747-c965-4f36-989e-0455556c51a2
- room ID: 625e7e93-3c50-4840-9fa9-b794058ea612
- snapshot ID: f1d08141-5549-4856-babc-e6051814f10f
- synchronizing: N
- your turn: Y



gfn さんが参加しました
taro さんが参加しました
jiro さんが参加しました
hanako さんが参加しました
東1局・1倍場 開始！
6, -2, -2, -2（至尊）

目次

- SATγSFI の紹介
- Sesterl の簡単な紹介
- **自作言語の意義・楽しさ**
- 自作言語の始め方
- 私の場合に功を奏したこと
- プロジェクトが進むにつれ悩ましくなること・私の反省点
- まとめ

なぜ私が言語をつくりたがるか

- 言語という対象自体が好き
- “木こりであるなら斧も研ぎたい”
- **問題解決の容易さはインターフェイスの適切さに大きく左右される**と考えているから
  - “スープをフォークで飲みたくはない”
  - **意識的に技巧を凝らさねばならない箇所を減らせるような良い道具をもつことが問題解決への近道**
    - 私の場合は特に型システムを重視しがち
  - ある意味では他責的な態度の方が創造的
    - 「うまくプログラムを書けないのは言語設計が悪いからだ！」

実用言語の自作による恩恵・楽しさ

- **思い通りの言語を使って実装できる！**
  - もちろんこれが最大の恩恵
- **言語機能に不足を感じたら簡単に追加できたりする**
  - ドッグフーディングによって言語がさらに実用的になっていく
- **研究になるテーマが見つかることも**
  - 実用のための実装があると，紙面上だけでは気づかなかった課題が浮き彫りになってくる
- **ユーザが自分の想像を超えた使い方をしてくれる**
- 自作の言語処理系が型エラーなどで
  自分のミスを指摘してくれるとなんだか嬉しい

目次

自作言語の始め方

1. ひとまず最初は何か読んだりして基礎的内容を知る
   - 大学では B3 頃に言語処理系の講義がありがち
     - **良い講義資料はわりと学内のみの公開だったりする**
       - 私が最も理解を促進された文献も学内向けの講義資料でした
   - 主な文献：
     - A. W. Appel『最新コンパイラ構成技法』(Tiger Book)
     - 五十嵐 淳『プログラミング言語の基礎概念』
     - Rui Ueyama「低レイヤを知りたい人のためのCコンパイラ作成入門」
     - …
   - 多くの人はよく最初の構文解析の章で詰まってしまうが,
     実は構文解析は真面目に取り組まなくてよい
     - 構文解析の理論も面白いが，**実際に言語処理系をつくるときは**
       パーサジェネレータやパーサコンビネータをいじれば事足りる
     - とはいえ，LR(1) で曖昧性のある文法を定義してしまったときに
       shift/reduce conflict がなぜ生じたかなどを理解するには大事かも

自作言語の始め方

2. 最小限の言語処理系の実装にとりかかる
   - どんな実行形態の言語にするか決める
     - インタプリタや高級言語を吐くコンパイラ（＝トランスパイラ）は特に取り組みやすい
   - 言語設計をする
     - こだわりがなければ，ひとまず好きな言語の小さいサブセットで OK
   - 言語処理系の実装に使う言語を決める
     - OCaml, Haskell, Rust あたりは特に処理系の実装に便利

3. 好きなだけ作り込む
   - **ひとまず動くものができると面白いうえに，どんな機能を入れたいか想像しやすくなる**
   - 一旦できてしまえば方針転換も容易

目次

SATγSFI の開発初期

2014 年 4 月

B3 の 1 年間で言語処理系に関する講義 2 つを他学部履修

2015 年 3 月

春休みに「HTML・LaTeX に変換できてマークアップも自分で拡張可能な簡単な言語をつくろう」と思い立ち，実装して **Macrodown** と命名

- 函数定義・リスト処理など最低限の機能のみに制限した言語だった

2015 年 4 月

`@zr_tex8r` さんにより Macrodown の Turing 完全性が示されたので，
根本的に方針転換して OCaml 風の言語にすることを決め，
院試勉強から逃避し型推論などを実装（面接の待ち時間にも実装していた）

2015 年 8 月

B4 〜 M1 の頃，ドッグフーディングでいろいろな機能を追加

- かなり OCaml に近い言語に

2017 年 2 月

M1 の終盤，Macrodown を組版処理システムにすることを決め，未踏事業に応募

2017 年 6 月

採択され，特に 10 月からは休学して開発に打ち込み，
9 ヵ月後に組版処理システムとして一旦完成

- 毎日寝食を忘れて実装でき，猛烈に楽しかった期間

2018 年 3 月

私の場合に功を奏したこと

1. 未踏事業に応募し採択されたこと
   - **応募を契機として問題意識を分析し整理できる**
     - 結果的に不採択でも有用
     - 実用志向で言語を作りたいなら何かしら新規性・改善点があるはず
       - **既存言語に対するフワッとした現状の不満感を分析し，
何が真に新しいウリになるかを吟味して言語化し提案するとよさそう**
   - 既に形になっている部分があると歓迎されやすいかも
     - SATYSFI も Macrodown が既にあった
   - やや品のない表現だが，採択が或る種の “お墨付き” となり
多くの人に関心を抱いてもらうきっかけになった

私の場合に功を奏したこと

2. 自分自身で積極的に実用したこと
   - SATγSFl： 修士論文，The SATγSFlbook，技術同人誌，etc.
   - Sesterl： オンライン天九サーバ
   - 改善すべき点が肌身に沁みてわかり，どんどん機能が追加できた
   - **実用に即した課題から研究テーマになるものも出てきた**

3. **言語を 2 つ作ったこと**
   - **2 回目の設計・実装の方がスムーズにいくので，**
     1 つ目の言語の改善にもその経験を活かせる
   - Sesterl の開発のおかげで SATγSFl も改善できた

目次

悩ましくなること・私の反省点

1. **理論的に堅牢な言語の実現には時間・労力を要する**
   - 「論文に載っていた型システムを自前で拡張したはいいが,
     本当に型安全性を壊していないかな？」
   - なんなら新たな論文が書ける（良いことでもある）

2. リリース後に（あーこの言語仕様よくなかったな）と
   思うところはどうしても出てくる
   - ユーザがいると非互換なリリースのハードルが上がる
   - 言語自体のバージョン切替えがエコシステムでできると理想的
     - 例： rustup, ghcup

悩ましくなること・私の反省点

3. 全部を 1 人でやるのは大変，無理が出てくる
   - **昨今は “エコシステムまで含めて言語” で，実用上の要請が肥大化**
   - 設計・実装すべきもの：
     - 言語処理系本体（もちろんこれが主軸）
     - language server などのエディタ支援
     - フォーマッタ
     - パッケージマネージャ・ビルドシステム
     - テストツール
     - ドキュメント生成の仕組み
     - マニュアル・解説書・公式 Web サイト
     - …

熱心なコミッタの方々と協業したいが……

悩ましくなること・私の反省点

4. どこかの段階で**自分以外にも読める実装にしないと作業が分担できない**
   - 私の場合：**†*NO TEST 開発*† を継続してしまったのがダメ**
     - 開発当初の私はテストという概念に馴染んでいなかった
       - プログラムの性質保証といえば形式検証だろうと思っていた
     - そもそも未踏事業の限られた期間ではテスト整備の余裕がなかったかも
     - テストなしだとモジュール分割も複雑化しやすい
     - **せめて結合テストは早くから用意したい**

5. サードパーティ製ツールをつくってもらえると，処理系本体とツールの責務の分担が難しくなる
   - ユーザや開発者の意見を聞く場を設けるのは案外難しい
     - meetup だけでは議論が尽くせない
     - “全員が準備して臨む会議” が必要かも

悩ましくなること・私の反省点

6. いただいた Pull Request の対応が後回しになりがち

- 反応が遅いと貢献してくれる方々の士気に影響
- 自分による既存実装がゴチャゴチャだとレビューもしにくい
- 気合いで好循環に入るしかない……？

7. 日本語圏以外へのアウトリーチを怠りがち

- 母語以外でドキュメントを整備したりユーザと意思疎通するのは
  時間と体力の消費がでかい
  - 文法はともかく，不自然な言い回しや変なニュアンスを防ぐのに神経を使う
- やっぱり気合い・気の持ちよう

目次

まとめ

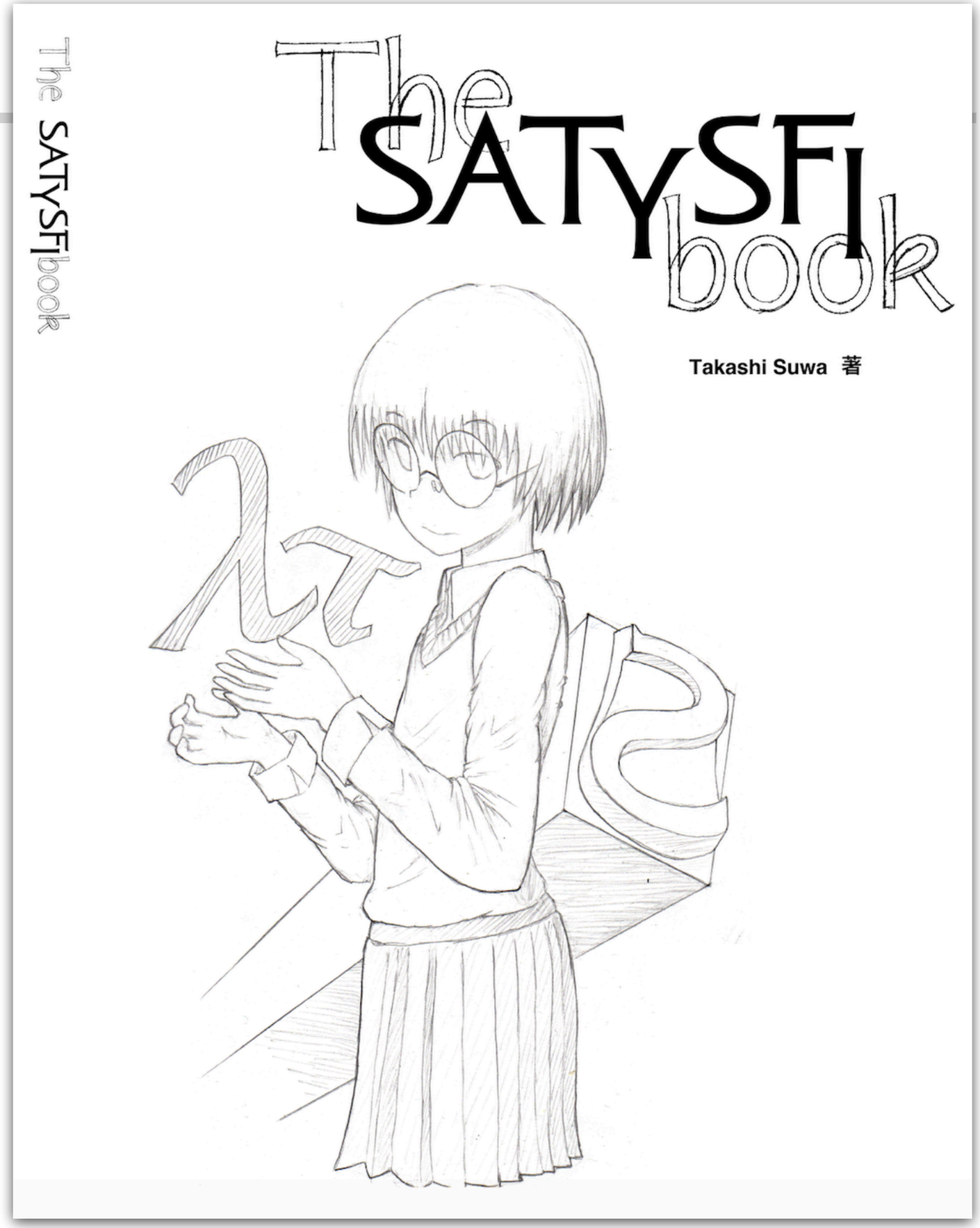
- 実用水準にまで言語をつくり込むのは,
  根気が要るがたいへん楽しく実り多い営み
  - 言語仕様を自分で好き勝手いじれることより自由なことはない
  - 是非やろう！
- 始め方・進め方：
  - まずは授業や文献などで最低限の素養を得る
  - とにかくミニマルに実装し始めてみる
  - 一旦動くものができれば，次にどう改善しようか想像がはたらく
  - 漠然とした問題意識を分析して要件にし，言語設計に落とし込む
    - 未踏事業など何かしらのコンペに提案するとよいかも
- 私からの教訓：
  - 開発を有志で分担できる状態を保つ
  - 日本語圏以外にもアウトリーチする

宣伝 1

The SATySFIbook

https://booth.pm/ja/items/1127224

- 公式解説書
- 無償公開！



宣伝 2

ヤバイテックトーキョー vol. 7

- https://booth.pm/ja/items/3518239
- 修士の同期と出している技術同人誌
  - ちなみに SATySFi 製です
- Sesterl で天九サーバを書いた
  今回の話の詳細な記事あり
- そのうち自分の Web ページで
  同様の内容を公開するかも
  しれませんが，PDF 版もぜひ！

宣伝 3

SATySFi Conf 2023

- https://connpass.com/event/295734/
- 2023 年 10 月 23 日 Cluster でオンライン開催
- Naoki Kaneko さん (@puripuri2100) 主催