Mikiβ におけるユーザ定義ファイルの生成に向けて

理学専攻・情報科学コース　中野　祥

1　はじめに

Mikiβ とは証明木を書くための汎用的な GUI ライブラリであり、ユーザが利用したいと考える推論システムを与えることで、その推論システムに相当する GUI を提供する。推論システムの導入にはその推論システムに相当するユーザ定義ファイルを作成する必要がある。ユーザ定義ファイルは OCaml 言語を用いて実装されており、字句解析器である lexer.mll、構文解析器である paser.mly、推論規則や判断の情報を解析する user.ml の 3 つのファイルから成り立っている。この、ユーザ定義ファイルの記述については、特に user.ml においてユーザの負担が大きいという指摘が以前よりなされている。Mikiβ の user.ml についての詳細は先行研究である [2] を参照されたい。

本研究では、ユーザの与える最低限の情報を基本データとして用いて user.ml における記述の大半を自動生成することを目標とする。

2　自動生成器の紹介

2.1　自動生成器の実行例

現在、本研究では user.ml の一部のソースコードの自動生成に成功している。自動生成器は OCaml によって開発され、JavaScript プログラムとしてブラウザ上で稼働する。以下に自動生成プログラムの実行例を示す。

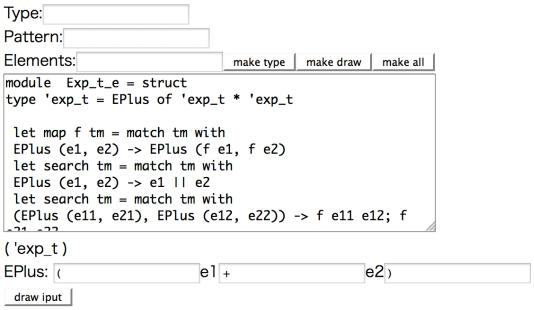

図 1: 自動生成プログラムの実行例

この実行例では、ユーザの入力として $(e + e)$ の構文を示す入力をしている。

2.2　ユーザの入力と自動生成器の出力

2.2.1　ユーザの入力

ユーザが入力する基本データとは、導入する推論システムの定義の構文定義 (型宣言) である。算術式に対する推論システム `EvalML1`[1] において、$e + e$ という式は $e \in exp\_t ::= e + e$ という構文であると判定される。これを user.ml に導入する場合、OCaml の構文定義に則って変換し、`type exp_t = EPlus of exp_t * exp_t` となる。ユーザが入力するべきは、`type` ① `=` ② `of` ③ `*` ③について、①:式がどの構文集合に含まれるかを示す “データ型名”、②:式の種類を示す “コンストラクタ名”、③:式を構成する引数 “コンストラクタの要素” となる。

また、最終的なテキストの作成のためには “式の描画情報” をいずれかの段階で入力する必要がある。これについては 3.2.2 節にて述べる。

2.2.2　自動生成器の出力

現在、自動生成可能なソースコードは user.ml の一部分のみである。自動生成器はユーザの入力に従って、以下のようなシグネチャをもったモジュールのソースコードを出力する。

```
module type STRUCTIRE_SIG = sig
  (* データ型 *)
  type 'a t
  (* 4 つの関数：map,search,unify,draw*)
  val map : ('a -> 'b) -> 'a t -> 'a t
  val search : bool t -> bool
  val unify : ('a -> 'a -> unit) -> 'a t ->
      'a t -> unit
  val draw : id_t t -> id_t
end
```

データ型については OCaml の構文定義をメタ変数を用いて定義したものである。4 つの関数は、データ型がもつコンストラクタすべてに特定の操作を行うためのもので、それぞれ OCaml のパターンマッチ構文を利用して作成された関数である。具体例として、$e + e$ に対する map 関数を示す。

```
let map f x = match x with
              EPlus(e1, e2) -> f e1; f e2
```

2.3　自動生成器の使用法

自動生成器の使用法は容易である。

- `Type` の入力フォームにデータ型名、`Pattern` にコンストラクタ名、`Elements` に 1 つ目の要素のデータ型を入力して、`make type` ボタンを押すことでデータ型を登録する。
- データ型を複数入力する際はデータ型名を変更し、`make type` ボタンを押すことでそれぞれのデータ型を登録できる。
- データ型が複数のコンストラクタを持つ場合、データ型の名前は変更せず、コンストラクタ名及び要素の型名を変更し、`make type` ボタンを押す。
- コンストラクタが複数の要素を持つ場合は、データ型名、コンストラクタ名の入力を変更せず、適切な要素の型名を入力し再び `make type` ボタンを押すことで要素の追加を行う。
- データ型の登録を終えた後、`make draw` ボタンを押すと描画情報の入力フォームが下部に表示される。適切な描画情報を入力し、`draw input` ボタンを押すことで描画情報が登録される。
- 描画情報を登録した後、`make all` ボタンを押すことで、登録した全てのデータ型について、求めるソースコードのテキストが生成される。(図 1 の状態)

3 実装手法

自動生成器の実装は、ユーザの入力した情報を格納するデータ構造を作成し、そのデータ構造に従って求めるテキストの生成を行うという手法で行った。本節では、ユーザの入力した情報を格納するデータ構造の紹介を行った後、テキストの生成を行う手法の概略を示す。

3.1 ユーザ入力情報を格納するデータ構造

ユーザの入力した情報の受け皿として basic record である base_decl_t を定義する。

```
(* base_decl_t *)
type base_decl_t = {
  (* データ型の名前 *)
  base_name: string;
  (* コンストラクタとその要素 *)
  base_cons: (string * string list) list }
```

base_name はデータ型を区別するものである。base_cons はコンストラクタの集合をリストとして実装する。リストの要素はコンストラクタとその要素のリストを組にしたものである。ユーザの入力した情報が base_decl_t 型のレコードに格納されることでデータの登録が行われる。

3.2 ソースコードのテキスト生成手法（概略）

ソースコードのテキストの生成はユーザが入力した情報を base_decl_t に格納したデータを基本データとして行われる。

3.2.1 パターンマッチ構文に対する考察

期待する出力結果のうち、2.2.2 節で紹介した 4 つの関数は OCaml のパターンマッチ構文を利用して作成されている。

パターンマッチ構文は、記号->の左側がパターン部、右側がパターン対する処理部となっている。パターン部は bese_decl_t 型レコードに格納したコンストラクタ名とその要素によって簡単に作成することが出来る。処理部に関しては、それぞれの関数の表現規則に従ってテキストを生成する必要があるが、その表現はパターン部の引数、すなわちコンストラクタの各要素に従う。よって処理部はパターン部を元に作成すればよい。パターン部の引数と処理部の引数は完全に対応させる必要がある。今回の実装では、パターン部、処理部共に、同じ長さをもつリスト持つ構造のデータを基本データから作成し、パターン部のリスト、処理のリストを組にして処理することで引数の対応を実現した。

3.2.2 描画情報への対応

加算を表す EPlus(e1, e2) を描画するとき、$e1 + e2$ とするか優先度を示すために $(e1 + e2)$ とするかはユーザの意向によって異なってくる。描画を行う関数 draw はこのような描画の仕様を決定するが、3.1 節で紹介した base_decl_t 型のレコードには “演算子” などの描画のための情報を格納できない。よって、登録されたデータ型に対応する描画情報入力 GUI の生成と、描画情報を格納するデータ構造が必要となる。

描画情報を格納するデータ構造 dr_decl_t は base_decl_t を元に作成される。base_decl_t 型はコンストラクタが引数のリストを持っていたのに対し、dr_decl_t 型は描画情報のリストを持っている。

```
(* dr_decl_t *)
type dr_decl_t = {
  (* データ型の名前 *)
  base_name: string;
  (* コンストラクタの要素は描画情報となる *)
  base_cons: (string * string list) list }
```

また、描画情報を入力するための GUI は dr_decl_t 型のもつ描画情報のリストに対応するように生成される。base_decl_t 型の情報から描画情報が空である dr_drcl_t 型のレコードを生成し、各コンストラクタ毎にその要素のリストに対応した個数のインプットボックスのリストを生成し、入力情報をリストの順番通りに回収することで正しい描画情報を作成する。描画情報入力 GUI は base_decl_t 型レコードに登録されたデータを利用して作成されるため、自動生成器の実行中に別途新たな GUI を作成、追加表示する構成になっている。

3.3 複数の登録データからのテキスト生成

3.1、3.2 節におけるテキスト生成のための手法は 1 つのレコードに対しての処理である。現実には 1 つの推論システムに対しデータ型の登録は複数行われる。本研究では登録された dr_decl_t 型レコードデータを要素とする参照リストを作成し、1 レコードに対するテキスト生成手法を、リストの各要素にマッピングすることで、複数の登録データに対するテキスト生成を可能にした。

4 まとめ

Mikiβ のユーザ定義ファイルのうち user.ml について、その一部のテキスト自動生成手法の確立と実装を行った。ユーザ入力情報を基本データとして取り扱うためのデータ構造を設定した。OCaml のパターンマッチ構文のテキスト生成に対する考察を行った。描画情報を得るために、ユーザの入力情報から新たな入力フォームを生成し、ブラウザ上に表示することを可能にした。期待するソースコードのテキスト生成を実現した。

今後の課題として、自動生成器の完成及び出力したテキストによる user.ml の稼働確認が挙げられる。現在、SK 論理、EvalML などの比較的小規模な推論システムについては当稿で紹介した自動生成器の出力するテキストの有効性を確認した。しかし、より複雑で大規模な推論システムについての検証は行われていない。自動生成器の完成を進めると共に、出力結果の検証を進めていきたい。

参考文献

[1] 五十嵐 淳 “プログラミング言語の基礎概念”, サイエンス社, 2011/07/25.

[2] Sakurai, K., and K. Asai “MikiBeta: A General GUI Library for Visualizing Proof Trees”, In M. Alpuente, editor, *Logic-Based Program Synthesis and Transformation (LNCS 6564)*, pp. 84–98 (April 2011).