js_of_ocaml を用いた Mikiβ の実装に向けて

中野祥　(指導教員：浅井健一)

1　はじめに

Mikiβ は証明木を描くための汎用的な GUI ライブラリであり，証明木の定義を与えることでその GUI を得ることができる．従来の Mikiβ には OCaml 及び LablTk が用いられており，ユーザの使用環境として OCaml 及び X11 を要求している．

本研究では Mikiβ に対し，OCaml プログラムを JavaScript へ変換するコンパイラである js_of_ocaml を用いることで，LablTk の提供する API に依存して構築していた GUI を HTML5 準拠の API で再構築し，JavaScript を用いた Web アプリケーションとして実装した．また，証明木のデータ構造を変更することによって，推論規則適用領域の拡大や適用規則の除去といった新しい機能も追加した．

なお, 先行研究 [1] で開発された Mikiβ は `http://pllab.is.ocha.ac.jp/~asai/MikiBeta/` で確認出来る．

2　js_of_ocaml とは

本研究に用いるプログラムである『js_of_ocaml』は OCaml による JavaScript の記述を可能にする．これは OCaml の全言語とおおよその標準ライブラリをサポートしている. 元となる OCaml プログラムは特別な操作をせずコンパイルでき, 生成したバイトコードを js_of_ocaml でコンパイルすることで JavaScript へ変換する. JavaScript の Object であったものには相応の型が付けられるので OCaml プログラムで静的型安全に呼び出すことが可能である．作成した JavaScript は JavaScript を利用できる任意の Web ブラウザで動作可能となる．

また js_of_ocaml における API の関数名及び用法は JavaScript のものと酷似しているため，既存の JavaScript 資源を用いることに対する障害は少ない．

js_of_ocaml は `http://ocsigen.org/js_of_ocaml/` にて公開, 配布されている．

3　実装

先行研究で開発された Mikiβ の構成ファイルは GUI 部とユーザ定義部に大別出来る．GUI 部はその構築に OCaml プログラムから Tk へのアクセス法を提供するライブラリである LablTk に依存する一方で，ユーザ定義部においてはこれに対する依存はない．これが意味することは, Mikiβ を JavaScript に移行する際に変更を要する部分は GUI 部のみであり，ユーザ定義部は従来のものをそのまま使用出来るということである．

GUI 部を構成するのは，主に証明木を描画する gui.ml, ユーザ定義部と GUI 部を繋ぐ fix.ml, Mikiβ の挙動を制御する miki.ml である．本稿では主に gui.ml 及び miki.ml に言及する．

3.1　描画のためのデータ構造

従来の Mikiβ では証明木のデータ構造は Tk の提供する API によって制御され, 座標の取得は容易だったため, データ構造の中に表示位置を制御する情報を含む必要は生じなかった. しかし, js_of_ocaml の提供する API には描画領域内の座標を自動取得する等といった機能は存在しないため, 描画されるそれぞれの要素は自分が表示されるべき位置情報を保持することが必要となった. この位置情報を coord_t とし, これを用いて描画のための証明木データ構造 id_t は gui.ml において以下のように定義される．

```
type coord_t = float * float * float * float
type id_t = String of string * coord_t
          | H of id_t list * coord_t
          | V of id_t * id_t * coord_t
               * string * float
```

coord_t は表示領域の左上角の (x,y) 座標と右下角の (x,y) 座標を情報として持ち, この情報を適切に計算することによって項やラインは正しい位置に描画される．

また, String は文字列, H は同一ライン上に横並ぶ id_t, V は仮定部と結論部によって構築される部分木を表している．V については, 適用規則をライン右端に描画するために, 規則名 (string) と規則名の長さ (float) も保持している．

SK 論理による $a \supset a$ の証明木は図 1 のようになる．

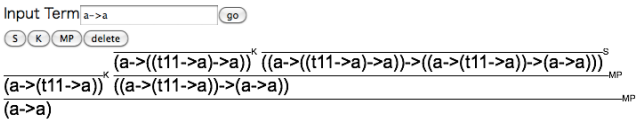

図 1: SK 論理による $a \supset a$ の証明木

3.2　内部データとしての証明木のデータ構造

miki.ml において，証明木の部分木の選択や仮定部にユーザ定義の規則を適用するといった操作を行うためのデータ構造 tree_t が必要となる. tree_t は描画のためのデータ構造 (id_t) を持つことによってマウスイベントによる証明木の座標情報取得などの操作を可能にしている．

```
type tree_t =
    Axiom of Node.fix_t * string * id_t
  | Infer of Node.fix_t * tree_t list
          * string * id_t
```

Axiom は公理を意味し, 公理 (Node.fix_t) と規則名 (string), 描画のためのデータ構造 (id_t) を持つ. Infer は推論規則を意味し, その結論 (Node.fix_t) と前提部 (tree_t list), 規則名 (string) 及び描画のためのデータ構造 (id_t) を持つ．

3.3　座標の取得

Canvas 上を click すると, Canvas 要素の onmouse メソッドによってマウスイベント event が発生する. event.clientX 及び event.clientY はウィンドウ座標を取得するが, この座標はウィンドウのドキュメントを表示している部分の左上を原点とするため, Canvas 左

上からの相対座標にはならない. これを解消するために getBoundingClientRect() を用いる. getBoundingClientRect() は対象の要素の領域の上下左右の座標を持つ. Canvas に対してこれを用い, ウィンドウのドキュメントと Canvas 位置の差分を取り払うことで, click した場所の Canvas 左上からの相対座標を取得出来る.

取得したイベントの座標を調整する関数 adjustXY は以下のように実装される.

```
let adjustXY event =
 let rect = Gui.canvas##
            getBoundingClientRect() in
  Gui.lx: = (float_of_int event##clientX)-.
            (Js.to_float rect##left);
  Gui.ly: = (float_of_int event##clientY)-.
            (Js.to_float rect##top)
```

3.4 部分木の選択

従来の Mikiβ における選択可能範囲は推論規則を適用していない項のみである. 本研究では規則適用済みの部分木を選択可能範囲に追加し, 拡大した. これによって部分木に対する規則の再適用・適用規則の削除が可能となる.

select_tree 関数は, click した座標を含む最小の部分木を選択し, その周囲に選択領域枠を描画する. 部分木の選択は証明木のデータ構造 tree_t の持つ描画情報 id_t から coord_t を取り出すことで可能となる. 選択領域枠の描画については HTML5 における Canvas 要素の 2D コンテキストオブジェクトである strokeRect を用いた.

3.5 内部の証明木データに対する操作

規則名の挿入・削除と項への推論規則の適用という 2 つの操作を証明木のデータに対して同時に行うことは難しい. そのため, これらを同時に行わず, データの操作を 2 段階に分けることにした.

3.5.1 規則名の挿入・削除

規則の適用や適用規則の除去を行うためには, 規則名は正しい位置において挿入・削除される必要がある. また, 規則適用の際に適用不可能な規則を選択した場合はその規則名はデータに挿入してはいけない. 適用する規則の持つ判定 (judge) と適用対象の項の単一化 (unfiy) が不可能である場合がこれにあたる.

規則を適用する場合は, 適用する規則名と, 元々データが保持している既存の規則名を正しい位置に挿入したデータを作成する. 規則を削除する場合は, 対象の項の持つ規則名と, その項の前提部が持つ規則名をすべて空文字列にリセットしたデータを作成する.

3.5.2 規則の適用と証明木の再構築

規則名を正しくセットしたデータとユーザが最初に入力した式を用いることで証明木のデータは再構築できる. データから規則名を取り出し, これを key とすることでユーザ定義の規則のリストから対応する推論定義を取り出す. これをユーザが最初に入力した式に単一化 (unify) することで結論部から木の再構築が可能である.

3.6 ユーザインターフェース (UI) の構築

UI の構築は miki.ml 内の launch_gui 関数で行われる. これはユーザが式を入力する input_box, 証明木が描画される Canvas, 証明木を操作するための各種 button で構成される.

従来は Tk の提供する API で構築されていたが, js_of_ocaml の提供する API によって代替, 実装することができた.

3.6.1 Input box

ユーザが証明すべき式を入力する input_box は, `Dom_html.window##createTextNode(js name)` によって生成する. `Dom_html.createInput ~_type:(js "text") document` の value メソッドに入力内容を代入し, onchange メソッドで実行することで入力内容を取り込むことができる.

3.6.2 Canvas

Canvas は gui.ml においてグローバル変数として定義される. Canvas の生成は `Dom_html.createCanvas document` によって行われ, 2D コンテキストオブジェクトを用いることで文字列や図形を描画する. 証明木の描画は gui.ml の display 関数で行われ, 3.1 節で述べた id_t に対してパターンマッチを行い, 対応する 2D コンテキストオブジェクトにそれぞれの要素と coord_t から座標を与えることで可能になる.

3.6.3 Button

`Dom_html.createInput` によって生成する submit タイプの button に対し, value メソッドに button 名称を, onclick イベントハンドラに実行内容を与えることで実装する. 推論規則の button はユーザ定義の規則のリストに再帰をかけることで生成出来る.

4 まとめと今後の課題

Tk への依存を排除し, js_of_ocaml の API を用いることによって Mikiβ を Web アプリケーションとして再実装した. それに伴い, 証明木のデータ構造を新たに定義した. データ構造に規則名をを追加したことで適用規則名の表示, 及び, 部分木に対する規則の再適用・適用規則削除を実装した.

推論規則の適用によって生成されたメタ変数の具体化をはじめとした未実装の機能が存在するほか, 証明木を左揃えでしか描画できない, Canvas サイズが可変でないため横に長い描画は途中で途切れてしまう, 証明木の作成を結論部からしか行えない, といった問題が生じている.

今後の研究では, 未実装の機能の実装と問題解決に加え, より利用しやすい GUI の構築を追求していく.

参考文献

[1] 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).