簡約過程の一般的可視化システムの実装

石川ちひろ　(指導教員：浅井健一)

1　はじめに

本研究では、簡約過程の可視化を目標としている。その際、GUI プログラム部分と簡約規則まわりの関数を分離し、それぞれ Reduce モジュール,Term モジュールとした。今回の研究では、Term モジュール内の項の定義やそれに関する関数を書き換えるだけで、ユーザの定義した簡約規則にしたがって簡約をし、さらにその過程を GUI 上で目視できるシステムの構築を行った。

GUI 上のプログラム部分をほとんど意識せずに本質的な項の定義などを考えるだけで GUI システム構築を可能にする。さらに、ユーザが選んだ部分を簡約できるようにした。

なお、GUI システム部分には先行研究である、証明木作成 GUI を使用した。[1]

2　研究背景

GUI システムを作る際の問題点として、システムを作ってもその内部の簡約規則に依存した 1 つのシステムができるだけで、それを再利用して新しいシステムを作るのは難しいという点があげられる。

そこで、GUI システム部分のプログラムと簡約規則のプログラム部分を分離し、GUI プログラム部分をほとんど意識せずに、簡約規則部分である Term モジュールを書き換えるだけで、GUI アプリケーションが構築できる、再利用性の高いシステムの開発を目標とした。今回は、その一例として λ 計算の簡約規則を実装した。

3　λ 計算

λ 計算とは、文字 λ を使用した式によって表記した関数を扱う計算である。λ を使用した式によって表記された関数を λ 項といい、λ 項は以下のように定義されている。

- $M(\lambda\text{ 項}) = x(\text{変数}) \mid \lambda x.M\ (\lambda\text{ 抽象})$
  $\mid MM(\text{関数適用})$
- $V(\text{値}) = x \mid \lambda x.M$
- $E(\text{評価文脈}) = [] \mid []M\ \mid V[]$

λ 計算では、関数適用の形の式を変換していくことで式を簡単にしていくことが可能である。それを簡約と言う。

以下に λ 計算の簡約規則を示す。[2]

- $(\lambda x.M)V\ \rightarrow\ M[x\ :=\ V]$
- $E[M]\ \rightarrow\ E[M']\qquad if\ (M\ \rightarrow\ M')$

1 番目の規則は、項 M の中の x に V を代入するということを意味している。2 番目の規則は、評価文脈中の項 M が項 M' に簡約されるならば、評価文脈中の項 M を項 M' にしてもよいという意味である。簡約を行う場合には、ある式中の簡約できる部分 M と評価文脈 E に分離し、1 番目の規則で簡約して、もとの評価文脈に挿入する。

4　実装 (Term モジュール)

ユーザに用意してもらう関数は Term モジュール内にすべて記述する。GUI システム部分は Reduce モジュール (次節) に用意してある。

システムの構成は以下の通りである。

gui.ml → term.ml → parser → lexer → reduce.ml

ユーザが各自定義しなければならない関数は以下の表の通りである。

term_t	項の型の定義
context_t	評価文脈の型の定義
is_value	ある項が value かそうでないか判断する関数
is_redex	ある項が redex かそうでないか判断する関数
decompose	簡約できる部分とそれ以外を分離する関数
reduce	redex を簡約する関数
plug	簡約した項を評価文脈に挿入する関数
getid	項それぞれがもつ id を取ってくる関数
draw	GUI 上での表示に関する関数

表を見ればわかる通り、GUI 部分のプログラムを書かなければならないのは、draw 関数のみで、その他は項の定義や簡約するのに必要な関数ばかりである。

ここで、いくつかの関数について説明をする。

- term_t:項それぞれを区別するために id_t という型が gui.ml 内に用意してあるため、Term モジュール内で、term_t を定義する際にこの id_t をそれぞれ付け加える必要がある。その id_t を取り出す関数は、getid 関数で定義してもらう。term_t の構造に応じてこの関数も書き換える必要がある。以下に term_t の定義例を示す。

```
type term_t =
   Var of string * Gui.id_t
 | Abs of string * term_t * Gui.id_t
 | App of term_t * term_t * Gui.id_t
```

- decompose:引数 p と引数 term をもらい、p を満たす redex とその評価文脈に分離する関数。Reduce モジュール内で、p に「カーソルが redex 部分に入っているかどうかを判定する関数」を渡すことで、ユーザが簡約したい場所を選んで簡約するという機能を実現している。
- reduce：受け取った term を簡約する関数。これは、3 節の 1 番目の簡約規則をプログラムしたものである。
- plug：評価文脈と term を受け取り、評価文脈に term を挿入する関数。
- draw：GUI 上にどう表示するかを指定する関数。あらかじめ用意された「create」,「combine」と

いう関数を使用する。create 関数は、string を受け取り、その string を表示する。combine 関数は、アイテムリストを受け取り、それらのアイテムを横に並べることで、図 3 のように表示させる。この関数を変えることで、GUI 上の表示をユーザの望むものにできる。例えば、現在は、「fun x → x」と box に入力すると、「λx.x」と表示されるようにしている。

5 実装 (Reduce モジュール)

Reduce モジュールには、GUI の起動や Term モジュール内で定義された関数を使った簡約動作などの定義が書かれている。Reduce モジュール内の関数は、Term モジュールが書き換えられても対応できるように、Term モジュール内の型構造に依存しないように記述している。Term モジュールで定義した reduce 関数や plug 関数は、Reduce モジュール側で以下のように使われる。

```
let double_act () = (*ダブルクリック時の動作*)
  match !data_r with
      None -> ()  (*redex が選択されていない*)
    |Some(c,redex) ->
  (let r = Term.reduce redex in
            (*redex を簡約し*)
   let t = Term.plug c r in
            (*結果を c に挿入*)
    Gui.x := 0;   (*x 座標に 0 をセット*)
    let _ = Gui.create "~>" in
      Gui.x := 30;  (*x 座標に 30 をセット*)
      main_draw t;  (*簡約結果を表示*)
      data_r := None )
    (*保存した redex と評価文脈を clear*)
```

6 実装 (その他の Term モジュール)

今回は、call-by-value を実装したが、どのくらい Term モジュールの書き換えが必要なのかを調べるために、Term モジュールを call-by-name でも実装した。(ここで、call-by-value とは、値渡し評価といわれ、引数の式を評価し、結果として得られた値で関数内の対応する変数を束縛する。call-by-name とは、名前渡し評価といわれ、関数の引数は全く評価されずに、関数本体内に直接置換される。)

以下に具体的な変更部分を示す。

- V(値) $= \lambda x.M$
- $(\lambda x.M)N \ \rightarrow \ M[x \ := \ N]$

上記変更部分を元の call-by-value のものと比べると、数行の書き換えで実現できることがわかった。term_t の構造を大きく変えた場合などは、手間がもう少しかかるが、それでも通常よりもそれほど手間をかけずに、実装が可能であると思われる。

7 動作

GUI を立ち上げると、図 1 のような画面が出る。はじめに、簡約したい式を box に入力し [図 2：①]、Go ボタン [図 2：②] を押すと、キャンバス上に式が表示される。表示された式にマウスカーソルを合わせると、簡約可能な部分には青い囲みが表示される。[図 3：③] 囲み部分をクリックすることによって該当部分が簡約される [図 3]。これを簡約可能な項がなくなるまで繰り返す。

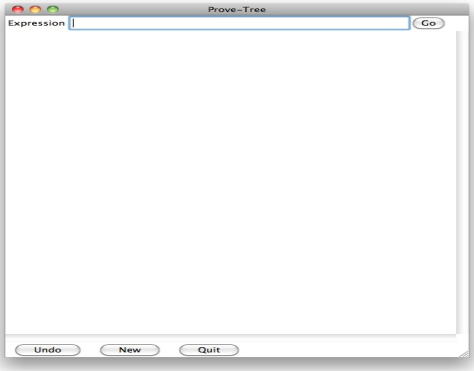

図 1：初期画面

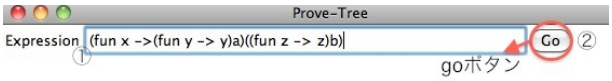

図 2：式を入力

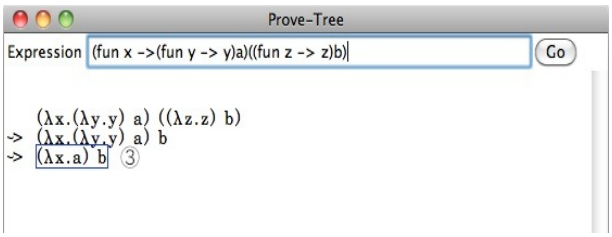

図 3：簡約中の画面

8 まとめ

本研究では、簡約過程の一般的な可視化システムを構築した。これにより、簡約規則に関わる定義などを必要に応じて適宜記述するだけで、その定義に沿った簡約過程の可視化を実現することができた。その結果、Term モジュールを書き換えるだけで新しいシステムを構築できるため、再利用性もあがったと考える。

9 今後の課題

今後は call-by-value や call-by-name だけでなく、その他の簡約規則を実装し、どの程度の書き換えで実装できるかの検証を行いたい。

また、さらに再利用性を高めるために工夫する点があるかどうかを再考していきたい。

参考文献

[1] 櫻井加奈子, 浅井健一 “証明木作成のための GUI 構築” 第 12 回プログラミングおよびプログラミング言語ワークショップ (PPL2010) 掲載予定.

[2] Hankin C. 2004.“An Introduction to Lambda Calculi For Computer Scientists”, King’s College Publications.