Mikiβ における相互再帰の実装と web ユーザインターフェースの改善

理学専攻　情報科学コース　2240643　大石 美緒　(指導教員：浅井 健一)

1　はじめに

証明木を用いた計算は、プログラミング言語学における型推論、論理学や自然言語学などで用いられている。手計算にて証明木を用いた計算を行うと、何度も項を書き写したり書き換えたりしながら進める必要があり、非常に手間がかかる。このような問題を解決するために、Mikiβ [3] を用いた計算が提案されていた。Mikiβ とは、GUI 上で証明木の構築や操作を行うシステムのことで、ユーザが型定義や推論規則などの証明木に関する情報を OCaml プログラミングにて記述すると、対応した GUI を構築することができる。

先行研究において、大きな問題点が 2 つ挙げられていた。1 点目は相互再帰をする型システムでは Mikiβ を用いて GUI を構築できない点、2 点目はユーザが記述するプログラミング量が多いという点である。これらの問題点に対し、ユーザが OCaml の型を用いて証明木に関する情報を記述していたところを、本研究ではデータとして定義するよう変更することで解決した。さらに本研究では 2 点目の問題点に対して、ユーザは必要な情報を web インターフェース上から入力することで、証明を行いたい論理体系を入力するようにし、ユーザはプログラミングをせずに Mikiβ を用いて証明木の GUI を構築できるようになった。

2　Mikiβ の概要

先行研究において、Mikiβ はライブラリとユーザファイルから構成されていた。ライブラリでは GUI を構成するための関数を提供し、ユーザファイルでは推論規則に依存する『構文の定義・推論規則・出力関数・単一化関数・lexer・parser』などをユーザが OCaml プログラミングで記述していた。ユーザファイルを作成しコンパイルをすると、ユーザが記述した証明木の体系に特化した GUI が構築される。

2.1　Two-level Types

GUI に関するプログラムを意識せずユーザが証明木に関する定義のみを記述できるよう、Two-level Types [4] を用いて実装されていた。Two-level Types を用いたプログラミングでは、再帰的な構造をしている一つのデータ型を、再帰的な構造を司る部分 (generic type) と、再帰以外の構造を表す部分 (specific type) の二つに分ける。generic type と specific type は互いに依存せず独立して記述されるため generic type についての関数は specific type の内容を知らなくても記述できる。それを利用し、GUI に関する部分を generic type に対して記述しておき、ユーザが specific type のみを記述することで、GUI に関する部分を意識せずにプログラミングを行うことができる。

3　データによるユーザの入力

先行研究においては、specific type はユーザが OCaml プログラミングで型定義を与え、generic type は Mikiβ 上のライブラリで与えることで Two-level Types を実現していた。本研究では、specific type をユーザが web インターフェース上からデータを入力することで与えるように変更した。

図 1: 型の定義の入力

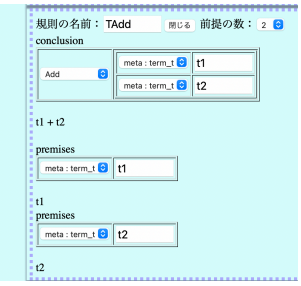

図 2: 推論規則の定義

3.1　型定義

型定義を入力する画面は図 1. のようになっている。型の名前、コンストラクタ名、引数についての情報、画面上の証明木にどのように描画するかの情報を入力する。先行研究においてユーザファイルで与えていた情報のうち、「型定義」と「出力関数」の二つと同じ情報をデータとして保持している。

3.2　推論規則の定義

推論規則の入力画面は図 2. のようになっている。規則名、各規則の前提の数、結論、前提条件を入力する。結論は conclusion、前提は premises に入力する。先行研究においてユーザファイルで与えていた情報のうち、「推論規則」と同じ情報をデータとして保持している。

3.3　Two-level Types

Mikiβ 内部では Two-level Types を用いて処理を行う。specific type は 3.1 節で述べたユーザの入力したデータを利用し、generic type は先行研究と同様に Mikiβ 上のライブラリで提供する。

specific type をデータにして与えたことで、相互再帰する型も定義することが可能になった。OCaml の型定義を用いて定義していた際は、定義が終わっていない再帰先は記述することができなかったが、本研究では、文字列のデータとして与えるだけで再帰先の指定を行えるため、エラーが起きずに記述できる。

3.4　ユーザ定義の関数

先行研究では、単一化関数や出力関数をユーザが記述していた。出力関数は 3.1. 節で述べた通り、データとして与えるため、記述する必要がなくなった。

一方、単一化関数は specific type の型について記述されるが、specific type は OCaml の型からデータに変更された。パターンマッチに型特有の情報が含まれなくなったことで、型定義が変更されても同じ関数で記述できるようになった。その結果、ユーザが記述す

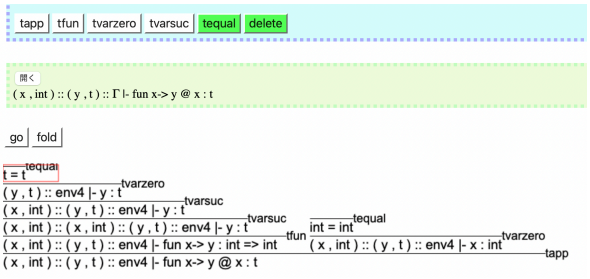

図 3: Mikiβ 上で構築した単純型付ラムダ計算の証明木

る必要はなくなった。

3.5 構築された証明木

Mikiβ 上で計算を行いたい、証明木の根となる項を入力し計算を開始する。単純型付きラムダ計算を例に構築された証明木を図 3. に示す。一番上の段には定義した推論規則のボタンが並んでいる。真ん中の段には Mikiβ 上で計算したい項が入力されている。「開く」ボタンを押すと、根となる項の入力画面が表示される。一番下には構築された証明木が表示されている。証明木をクリックすると、そのノードに適用可能な規則のボタンが緑色に光るようになっている。そのボタンをクリックすることで計算が進んでいく。

3.6 lexer, parser について

web インターフェース上から入力できるようになり、ユーザは OCaml でプログラミングをせずに Mikiβ を利用して証明木の GUI を構築できるようになった。今までは OCaml の型定義として構文を定義していたため、文字列での入力を読み取り構文解析していたが、本研究では画面上でセレクトボックスを用いて入力するため、lexer・parser が必要なくなった。その結果入力ミスによる Syntax エラーや、間違った構文を入力する事による型エラーも起きなくなった。

先行研究では型安全性が OCaml の型システムにより保たれていたが、本研究では、web インターフェース上からの入力を実装し、型が間違っている入力はできなくなったため、型安全性は引き続き保たれている。

4 諸機能の実装

4.1 xml ファイルへの入出力とダウンロード

入力した定義のダウンロード、アップロードを行う機能を実装した。ダウンロードを実行すると入力した定義を保存することができ、またそのファイルをアップロードすれば使用することができる。この機能は OCaml を JSON ファイルに出力、解析する `yojson` というライブラリを用いて実装した。

4.2 これまでに実装した体系

本研究において使用した定義をいくつか使用できるように実装してある。使用できる論理体系は、一階命題論理の体系 SK[5] を実装した `sk` と、それに諸規則を追加した `sk_plus`、自然演繹による最小論理の体系 NM を実装した `nm` と、それに諸規則を追加し NJ や NK まで拡張した `nm_plus`、単純型付きラムダ計算を実装した `lambda_caluculus` と、それぞれ限定継続 shift/reset を加えた `shift_reset`、限定継続 control/prompt を加えた `control_prompt`, 4 種の限定継続、shift/reset・shift0/reset0・control/prompt・control0/prompt0 を加えた `4typeControl0perators` がある。

4.3 tex への出力

利便性の向上のため、定義や完成した証明木の tex への出力を実装した。『tex を download する』というボタンをクリックすると tex のファイルがダウンロードされる。項の定義と推論規則の定義を新しいコマンドとして定義し、それを用いて計算した証明木を出力する。

5 まとめと今後の課題

本研究における成果は大きく分けて 2 つある。1 つ目は、ユーザの入力と specific type をデータにしたことにより、相互再帰する型をサポートできるようになったことである。例えば `control_prompt` 付き単純型付ラムダ計算 [1] や 4 種の限定継続付き単純型付ラムダ計算 [2] は相互再帰を含む複雑な体系であり、手計算で型推論を行うことは非常に困難であった。`compatible` や `id_cont_type` など、手計算では結果がわかりにくく複雑な規則を含むからである。しかし、Mikiβ を用いて GUI を構築できたことで、型推論の様子を (手動で) 観察できるようになった。

2 つ目は、ユーザが OCaml プログラミングをせずに、web インターフェース上からの入力のみで証明木を構築できるようになったことで、OCaml プログラミングが必要なくなり、より使いやすくなった。また、画面上で入力した定義を参照しながら推論規則を入力するため、再帰先が間違っている項は入力できなくなり、複雑な体系でも利用しやすくなった。

一方、相互再帰をサポートし、利用できる体系は大きく広がったが、まだ利用できない体系も多く存在する。例えば、集合を扱う体系は定義するのが難しい。理由としては、集合の中の任意のひとつをとってくるといった操作や、集合の中身の順番が変わっても同じ集合とみなすといった操作をどのように推論規則で表現したら良いか明らかではないためである。

今後は Mikiβ で多くの論理体系を実装してみることで、定義できる論理体系の例を集め、より多くの研究分野で利用できるようにしていきたいと考えている。

参考文献

[1] Youyou Cong, Chiaki Ishio, Kaho Honda, and Kenichi Asai. A functional abstraction of typed invocation contexts. *Logical Methods in Computer Science*, Vol. 18, No. 3, pp. 1–31, Sep 2022.

[2] C. Ishio and K. Asai. Type system for four delimited control operators. *Proceedings of the 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE 2022)*, pp. 45–58, 2022.

[3] Kanako Sakurai and Kenichi Asai. MikiBeta: A general GUI library for visualizing proof trees. *20th International Symposium on Logic-Based Program Synthesis and Transformation*, pp. 1–15, July 2010.

[4] Tim Sheard and Emir Pasalic. Two-level types and parameterized modules. *Journal of Functional Programming*, Vol. 14, No. 5, pp. 547–587, 2004.

[5] 戸次大介. 数理論理学. 東京大学出版会, 2016.