証明木作成のための GUI 構築

櫻井加奈子　(指導教官:浅井健一)

1　はじめに

証明木は論理学、自然言語学の推論だけでなく、プログラミング言語学の型推論等でも用いられている。証明木の記法を示す。

- 命題 S_1 から命題 S_2 が推論される時

$$\frac{S_1}{S_2}$$

- 複数の命題 $(S_1, S_2, ..., S_n)$ から命題 S_{n+1} が推論される時

$$\frac{S_1\ S_2\ ...\ S_n}{S_{n+1}}$$

- 命題 S が前提なしに成り立つ時 (公理)

$$\overline{S}$$

例えば、シーケント計算で、$a \vee b \wedge c \Rightarrow (a \vee b) \wedge (a \vee c)$ が正しいことを示す証明木は以下のようになる。

$$\dfrac{\dfrac{\dfrac{\overline{a \Rightarrow a}}{a \Rightarrow a \vee b} \quad \dfrac{\overline{b \Rightarrow b}}{b \Rightarrow a \vee b}}{a \vee b \wedge c \Rightarrow a \vee b} \quad \dfrac{\dfrac{\overline{a \Rightarrow a}}{a \Rightarrow a \vee c} \quad \dfrac{\overline{c \Rightarrow c}}{c \Rightarrow a \vee c}}{b \wedge c \Rightarrow a \vee c}}{a \vee b \wedge c \Rightarrow (a \vee b) \wedge (a \vee c)}$$

最上段の式が全て成り立っているので、最下段の式が正しいことが証明される。

2　研究背景

手で証明木を描く場合の主な問題点を挙げる。

- 描こうとする証明木がどれだけのスペースを必要とするのかを推論する前に考えなければならない。
- 証明に試行錯誤が必要な時に、何度も描き換えなくてはならない。
- 単一化する際に型推論等では証明木の至るところを書き換えなくてはならない。

証明木を描くことはそれ自体が目的ではなく、証明木の本質にあるものを読み取ることが目的である。そのためには証明木を簡単に描けること、証明木を描く十分なスペースが確保されていることが必須となってくる。以上を考慮に入れた上で、本研究では証明木作成のための GUI 構築を行う。今回は、GUI 構築の手法を示すための一例としてシーケント計算の推論規則を用いた。

3　シーケント計算

本節では、Wadler [2] の定式化を使ったシーケント計算を導入する。シーケント計算の推論規則の一部を示す。

Formula

$$A, B ::= X \mid A \wedge B \mid A \vee B \mid \neg A \mid A \supset B$$

Antecedent　$\Gamma ::= A_1, ..., A_m$

Succedent　$\Theta ::= B_1, ..., B_n$

Sequent(式)　$\Gamma \Rightarrow \Theta$

$$\frac{}{A \Rightarrow A}\ \texttt{Id} \qquad \frac{\Gamma \Rightarrow \Theta, A \quad \Gamma \Rightarrow \Theta, B}{\Gamma \Rightarrow \Theta, A \wedge B}\ \wedge\texttt{R}$$

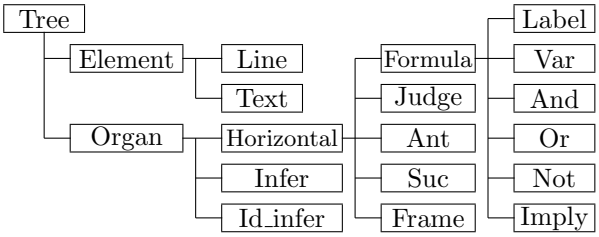
$$\frac{A, \Gamma \Rightarrow \Theta}{A \wedge B, \Gamma \Rightarrow \Theta} \quad \frac{B, \Gamma \Rightarrow \Theta}{A \wedge B, \Gamma \Rightarrow \Theta}\ \wedge\texttt{L}$$

4　LablTk

本研究では OCaml のプログラムから Tk [1, 3] へのアクセス法を提供する LablTk ライブラリを用いた。Tk を利用する利点は、容易に GUI の構築ができること、他言語への移植が比較的容易なこと、OS への依存が低いことが挙げられる。

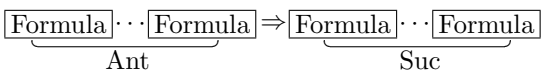
5　実装

OCaml のオブジェクトを用いて実装を行った。クラス階層を以下に示す。

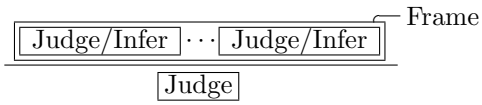


証明木を作成するための基本的な要素を Element クラスとし、必要最低限の機能を与えた。証明木を構成するための道具として、Organ クラスを作成し、機能別にクラスを作成した。Organ クラスに属する各クラスは次のように証明木を構成する。

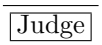
Judge： 各段の式を作成する。最下段の式はユーザが入力した論理式をもとに作成されるが、推論規則を適用すると上段の式を作成し、Infer クラスを作る。他の推論規則を導入する際には、Judge クラスの Apply メソッドに加えれば良い。



Infer : 3 つのオブジェクト (Judge、Line、Frame) を縦に並べる。上段の Frame では複数の Judge もしくは Infer を配置することができる。Infer クラスは再帰的な構造をしており、証明木を上に積み上げる。



Id_infer : 公理を適用した Judge に上線を付け、推論規則をそれ以上適用できなくする。



6 挙動

推論を行いたい論理式を Antecedent(空でも良い) と Succedent に入力し (図 1)、Start ボタンを押すと最下段の式が作成される。各機能の中から Apply を選択し、推論規則を適用したい論理式をクリックすると、適用可能な論理式に青枠が付く。Go ボタンのクリックにより、上段に規則適用後の論理式が生成される。3 節の ∧L のように、同じ推論規則でも 2 項ある Formula の内どちらかを選択しなければならない場合はポップアップウインドウが表示されるので、選択する (図 2)。これを繰り返し、証明木を完成させる (図 3)。

各機能を表にまとめる。

Apply	推論規則を適用。
Id	公理を適用し、不要な Formula を削除。
All Id	可能な限り、全ての式に Id を適用。
Highlight	同期している Formula に青枠表示。
Tex	TEX コードを別ファイルに出力。 ♣ 1 節の証明木はこの機能を用いて作成したものである。
Remove	証明木の一部分を消去。

図 3 では、Highlight 機能によって同じ意味を持つ Formula が青枠で囲まれている。型付き言語においては表現が異なるものであっても同じ意味を持つことがある。この機能は今後型付き言語を導入する際、ユーザが視覚的に捉えやすいように考慮したものである。しかし、シーケント計算でも論理式がどのように変化していくのかを見るためには有益であると思われる。

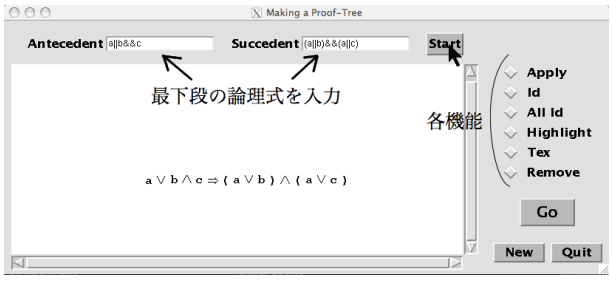


図 1: 初期画面

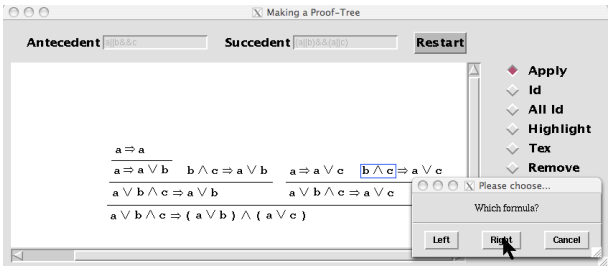


図 2: 証明木作成画面

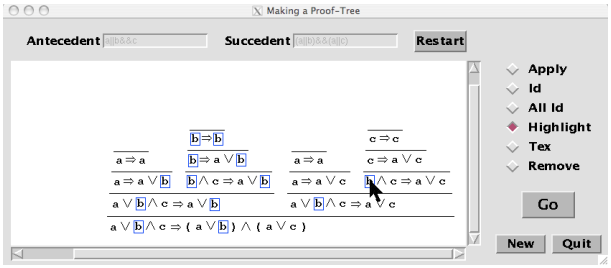


図 3: 証明木完成画面

7 まとめ

本稿では証明木作成のための GUI 構築の基礎を述べた。構想としては汎用性のあるものとなっている。簡単な操作で証明木が描けるため、研究目的だけでなく、初めて証明木に触れる学生の教育支援としても利用できると考える。

8 今後の課題

Id_infer オブジェクトを生成する際に、Judge の Ant と Suc にある論理式を同じものとして見なしたい (単一化、Unification)。しかし、完全に同じものとしてしまうと、Remove する際に元に戻すことができなくなってしまうので、Unification のための構造を見直す必要がある。

また、より広範な使い方ができるように機能を追加していくこと、他の推論規則を組み込むために、ユーザがマクロを書けるようにしていくことが今後の課題である。

参考文献

[1] Mark Harrison and Michael McLennan, *Effective Tcl/Tk Programming*, Massachusetts : Addison Wesley (1998).

[2] Philip Wadler, "Call-by-Value is Dual to Call-by-Name", In *ICFP'03*, pp. 189–201 (August 2003).

[3] http://homepage3.nifty.com/kaku-chan/tcl_tk/, Tcl/Tk で Windows プログラミング.