汎用的な証明図作成支援ソフトの構築

Constraction of Varsatile Proof Figure Drawing Tools

小川美樹
奈良女子大学大学院
人間文化研究科
Miki Ogawa
Graduate School of
Humanities and Sciences,
Nara Woman's University

鴨浩靖
奈良女子大学 理学部
Hiroyasu Kamo
Faculty of Science,
Nara Woman's University

新出尚之
奈良女子大学 理学部
Naoyuki Nide
Faculty of Science,
Nara Woman's University

概 要

数理論理学では、議論の対象としての証明を扱う。証明を表示するには、証明図というものがよく使われる。したがって、数理論理学での文献を執筆するには証明図の作成が必須となる。証明図を書くためのツールとしては、X window system 上で動作する xpe がある。しかし、大規模な証明図を柔軟にレイアウトする方法がないなど欠点もある。我々は xpe の欠点を克服した Jpre という証明図作成支援ソフトを開発中である。Jpre の実現にあたっては、今後加えられそうな機能を予測し、拡張性に配慮したデータ構造の設計を行った。また、xpe と異なり、半自動証明機能を目指すため、入力の構文解析が必要となるので、それらを行いやすいように木構造によるデータ表現を行う。本論文はこれらに触れつつ Jpre の設計について述べる。

1 はじめに

本論文では、記号論理学における証明図を作成支援するソフトを提案する。

推論のような人間の知的活動の分析や実現において記号論理学は重要である。また、記号論理学は知識データを記号化して表現する。よって計算機上で扱うことができる。したがって、計算機上で人の知的活動を表現したり、代行したりするような研究分野において、記号論理学は非常に重要な役割を果たす。例えば、状況を判断して目的を達成する手段を選択するような自律エージェントの基本設計は、記号論理学によるモデルの上で行うことが可能であり、そのような例として BDI エージェントが知られる。

記号論理学では、ある事柄についての証明が存在するかどうかや、証明するにはどんな手順が必要なのかなどを議論し、それについての証明を行っている。記号論理学において証明を表示するには、証明を記号で表したものである証明図というものがよく使われる。したがって、記号論理学での文献を執筆するには証明図の作成が必須となる。

記号論理学はそのように重要な分野であるので、証明図の構築や変換は非常に重要である。数学の論文を書くのに最も一般的に使われているのが TEX であるので、記号論理学の論文も TEX で書かれることが多い。

TEX で証明図を書くときは、多くの場合、TEX のマクロ proof.sty[1] を用いて行われる。このマクロにより証明図をたやすく作成できるようになった。しかし、もともと証明図は木構造を持つが、proof.sty ではグルーピングを用いて構造を一次元で表現するため、証明図が大きくなっていくとプログラムの可読性や運用性にやや難があった。それを補うツールとして、X Window System 上の GUI によって証明図を入力し、proof.sty の形式に変換する xpe が知られる。しかし xpe には、2.3 章で述べるように、Xlib を直接利用する古い設計であること、大規模な証明図の柔軟なレイアウト能力を欠くなどの問題もある。

我々は xpe の欠点を克服するため Jpre という証明図作成支援ソフトを開発中である。Jpre は、JAVA および Swing で構築することによって移植性および拡張性の高いものにするとともに、証明図の柔軟なレイアウト機能や、証明図作成支援としての半自動証明機能などを実現する計画である。

2 xpe

2.1 xpe の概要

xpe(X window system Proof Editor)[2] は、元々は TEX の証明図マクロ proof.sty と高い互換性を持つ証明図作成ソフトとして開発された。TEX のコマンドを解釈・表示し、必要に応じて証明図に論理式の挿入を行うことが出来る。また、xpe をユーザインターフェースとし、定理自動証明プログラムを呼び出す証明支援システムも開発されている [3]。

xpe は、GUI で証明図を作成し、それを proof.sty 形式で出力できる。このため、作成した証明図を TEX で書かれた論文等に取り込むことが容易になっている。xpe の画面表示においても TEX のプレビューアと同等の出力を持ち、可読性に優れたシステムとなっている。入力も TEX のマクロを流用しているため、TEX に慣れた利用者には xpe の操作の修得も容易である。

2.2 ユーザインターフェース

xpe の起動画面上部にはメニューが並び、中央部には黒く塗りつぶされた部分がある。そこに論理式を入力する。論理式としては、TEX のコマンドを入力する。例えば、A \lor \lnot A と入力すると、$A \vee \neg A$ が画面に出力される。

入力された論理式をさらに編集することも可能である。xpe の画面では、論理式の間に小さな四角の点があるのでそれをクリックしてから、順次挿入したい論理式を TEX 表現で入力すればよい。既に入力されている論理式を書き換えることもできる。

xpe の出力はそのまま TEX に取り込める。エディタで予め TEX のファイルを開いておき、xpe の上部メニューの whole → copy を選択し、ウィンドウマネージャのペースト機能を用いて TEX のファイルに張り付ける。この操作により xpe で表示している証明図をそのまま TEX の出力として得られる。逆に、TEX のファイル中の proof.sty 形式で書かれた証明図をこの xpe を用いて編集することも可能である。

このように xpe では可読性の極めて高い形で証明図を編集でき、証明図のエディタとしても充分なユーザインターフェースを備えている。

2.3 xpe の問題点

しかしながら、xpe の問題点として、Xlib を直接用いているためプログラムの設計が古く、安定版の更新が長期間止まっていること、また、X window system 上でしか動かないということなどが挙げられる。また、大規模な証明図を限られた紙面の中で作成するにはしばしば、以下のような xpe よりも柔軟なレイアウト能力を必要とする。

- 一部のシーケントや論理式をメタ変数に置き換える機能
- 証明図の一部を移動させる機能

例をあげると、以下のような証明図

$$\dfrac{\dfrac{\dfrac{\dfrac{B,q \to q}{B,q,\neg q \to}\,(\neg左) \quad \dfrac{\dfrac{B \to q,r}{B,\neg q \to r}\,(\neg左)}{B,\neg r,\neg q \to}\,(\vee左)}{B,q \vee \neg r,\neg q \to} \quad \dfrac{\dfrac{B,r \to r}{B,\neg r,r \to}\,(\neg左)}{B,q,r \to}\,(\vee左)}{B,q \vee \neg r,r \to}\,(\vee左)}{B,q \vee \neg r,\neg q \vee r \to}\,(\wedge左)}{B \wedge (q \vee \neg r),\neg q \vee r \to}\,(\wedge左)}{\dfrac{A \to}{\to \neg A}\,(\neg右)}$$

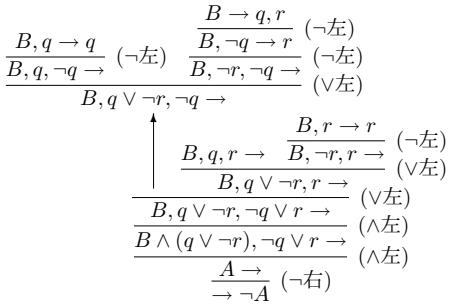
ただし

$$A = B \wedge (q \vee \neg r) \wedge (\neg q \vee r)$$

$$B = p \wedge \neg q \wedge \neg r \vee \neg p \wedge q \wedge \neg r \vee \neg p \wedge \neg q \wedge r$$

において、我々は一部の論理式を A か B のようなメタ変数に置き換えている。この方が図全体が簡潔で見やすくなるが、このためには A や B というメタ変数とそれが指し示す論理式を同一に捉える機能がいる。

また、この例でもわかるように、メタ変数で簡略化してもなお、ある程度以上の規模を持つ証明図はすぐ横に広がり、限られた紙面内にレイアウトすることは困難で、縮小するなどの措置を要する。しかし次の図のように、証明図の一部を移動させるようなレイアウト変更が可能であれば、縮小を伴わずに紙面にレイアウトでき、論文の可読性も上がることが期待される。



3 Jpre

我々はこれらの欠点を克服するため、Jpre という証明図作成支援ソフトを開発中である。概要として Jpre は、JAVA および Swing で構築することによって移植性および拡張性の高いものにする。 また、証明図の一部の枝を他の場所に移動させるなど、柔軟にレイアウトする機能を充実させたい。さらに、証明図作成支援の一環として、3.4 で述べる半自動証明の機能の追加を目指す。

3.1 設計する上で実現する機能

Jpre では以下のような機能を実現する予定である。

1. TEX の記号を随時 GUI 上にアイコンとして生成でき、システムが持っている TEX の記号が全て使える機能。例えば、StMaryRoad フォントなどもそのシステムに存在すれば使える。
2. 証明図内に [1] や A_1 などの添字など、TEX のコマンドが使えて TEX と同じように表示される機能
3. 証明図の部分的な移動やコピーが容易に行える
4. GUI 上で証明図の一部を探索する機能
5. 論理式の構文が合っているかどうかチェックする機能 (3.3 節で述べる)
6. 半自動証明の機能 (3.4 節で述べる)
7. 数学記号の入力は TEX コマンドでなくボタンでも可能であり、TEX コマンドを覚えていなくてもよい

3.2 インターフェース

xpe は証明図のエディタとして優れたものであったため、Jpre も基本的な操作方法は xpe のものを踏襲している。キーボードから論理式を入力すると、画面中央部に論理式が表示される。そのコマンドで表される文字を画像として画面に出力する。入力された論理式を編集するには、論理式の上下左右もしくは小さな四角の点をクリックして新たに論理式を書き加えるか、論理式自身をクリックして書き直すことができる。上部にメニューバーがあり、ここまでは xpe と同じ設計になっている。

論理式の一部を移動させる機能については、移動させたい部分をマウスでドラッグして囲み、囲んだ部分を切り離して表示することによって独立した証明図として編集することができる。上部には選択した論理式の構文チェックの結果を表示するバーがある。

3.2.1 記号の表示

\ で始まる文字列を入力すると、TEX のコマンドを入力した時と同じ記号を画面に表示する。その他の文字列はそのまま表示される。

3.3 論理式の構文チェックの機能

論理式の構文解析を行えば、入力された論理式が構文的に正しいかどうかのチェックも可能である。構文チェックのボタンを押すと、間違いがあるかどうかの結果を画面のバーに表示する機能も実装する。また、間違っていたところを色をつけて表示するなどの機能もあると便利と考えられる。

3.4 半自動証明の機能

ここで言う半自動証明とは、あるシーケントに対し適応する推論規則を表示すると、それを適応した結果を自動生成するものであり、これによって証明図の作成の手間が軽減する。

例えば、シーケント計算での証明図の作成中に、証明図中のある論理オペレータをマウスで指示することによって、その論理オペレータに着目した推論規則を自動的に適用し、結果として得られるシーケントを画面上で生成する。

4 設計方針

Jpre は Java で構築し、GUI は Java の Swing を用いて構築している。半自動証明には J-Prolog を用いる予定である。Jpre の内部設計について以下に述べる。

4.1 設計のポイント

Jpre の実現にあたっては、今後加えられそうな機能を予測し、拡張性に配慮したデータ構造の設計を行った。

Java は型付けの強い言語であるため、プログラム中のデータの不整合などによるバグを、型チェックによってかなり防げる利点がある。

Java で実装することにより、Java の動くシステムであれば移植性が確保でき、X Window System に依存しない利点も生まれる。さらに高レベルの GUI ライブラリも利用でき、多言語にも対応できる。これに対し xpe は実現が古いため低レベルライブラリである Xlib で実装しており、今後の拡張が難しい可能性もある。

また、C 言語などと異なりメモリマネジメントを自分で書くことなくデータの動的生成や廃棄ができる。

論理式を表現するため、どうしても TEX の数式記号のフォントを持つことが必要となる。数式記号のフォントは CM フォント、AMS フォント、StMaryRoad フォントなど様々であり、人によっては特殊な記号を用いる可能性もあるため、全てを予め用意しておくことは難しい。そこで、予め持っている記号以外の記号が必要になったらそのアイコンを動的に生成することにした。現在は、TEX の命令から LATEX のコマンド、dvips や gs コマンドを呼び出して自動的にビットマップを生成する、ltx2png というシェルスクリプトを用意して対応している。ただ、これだと例えば Windows に移植したい場合は Cygwin の助けが要ることになる。Jpre が汎用性を目指している点からはこれは課題の 1 つである。

4.2.1 節で述べる画像と文字列の列や、原子論理式の引数の列などは、主に ArrayList を用いて格納する。理由は、List が順序づけのインターフェースをもつことと、その中でも ArrayList はランダムアクセスに適しているためである。

4.2 内部設計

4.2.1 記号の表示

Jpre では、\ で始まる文字列は TEX のコマンドとして認識する。論理式中に現れる TEX の記号文字は、画像 (アイコン) として表示している。従って、論理式の表示には文字と画像が混じることになる。この 2 つを統一的に扱うため、両者を DisplayString、DisplayImage というそれぞれ String と ImageIcon を 1 つ持つようなクラスで表現し、それらの共通の親クラスとして DisplayElement というクラスを設けた。これにより、画像と文字列が任意に混じった列を DisplayElement という単一のクラスの ArrayList として表現できる。

4.2.2 字句解析

Jpre では現在、論理式を以下のような字句の連続として表現している。

- 空白の連続
- 英字 1 字
- バックスラッシュに続いての英字の列 (TEX のコマンド)
- コンマか括弧 1 字の字句

この字句を正規表現で記述し、正規表現マッチング処理によって字句解析を行っている。

また、4.2.3 で述べるが、以下のものは正規表現を定義したクラスを作り、設定が変更できるようにした。

```
MVPredicateFormula ::= "[A-H][A-Za-z]*"
MVAtomicFormula ::= "[I-O][A-Za-z]*"
PredicateSymbol ::= "[P-Z][A-Za-z]*"
FunctionSymbol ::= "[a-t][A-Za-z]*"
TruthValue ::= "True | False"
Negation ::= "not"
Conjunction ::= "and"
Disjunction ::= "or"
Implication ::= "supset"
Universal ::= "forall"
Existential ::= "exists"
IndividualVariable ::= "[u-z][A-Za-z]*"
```

MVPredicateFormula は述語論理式のメタ変数、AtomicFormula は原子論理式のメタ変数、PredicateSymbol は述語記号、FunctionSymbol は関数記号、TruthValue は真偽値、Negation は否定、Conjunction は連言、Disjunction は選言、Implication は含意、Universal は全称記号、Existential は存在記号、IndividualVariable は個体変数を表す。

例えば、$\forall x.P(c, f(x, a)) \land Q(x, y)$ という論理式を入力したい場合、\forall x. P(c, f(x, a)) \land Q(x, y) と入力すればよいが、これを字句解析すると'\forall'、'x'、'.'、'P'、'('、'c'、','、'f'、'('、'x'、','、'a'、')'、')'、'\land'、'Q'、'('、'x'、','、'y'、')' に分けられる。

4.2.3 構文解析

xpe は論理式の意味まで考えた編集は行わないため、入力された論理式の構文解析は行っていない。これに対し、Jpre では半自動証明の実現を目指すため、論理式の構文解析は必要となる。入力の構文解析に、LL 構文解析で行っており、解析の結果、論理式・シーケントや証明図を木構造のデータとして構築する。

現在、各構文要素に対応する以下のようなクラスがある。

- 命題論理の式を解析するディレクトリ
- 一階述語論理の式を解析するディレクトリ
- , のクラス
- →のクラス
- シーケントのクラス
- 木構造のクラス

シーケントは、「,」で区切られた論理式の列をオペレータ→の左右に 1 つずつ持つデータである。木構造は、シーケントをノードに持つ木であり、親ノードのシーケントと子ノード以外の部分木の List として表現している。

実装の構文規則は以下のようになっている。

一階述語論理の場合
ただし、全称記号と存在記号の後に書くことのできるのは個体変数、すなわち、一階の変数に限られている。

```
PredicateFormula ::= MVPredicateFormula
  | Fromula5
AtomicFormula ::= PredicateSymbol
  | PredicateSymbol '(' Term(',' Term)* ')'
  | TruthValue
Function ::= FunctionSymbol
  | FunctionSymbol '(' Term(',' Term)* ')'
Term ::= IndividualVariable
  | Function

Formula5 ::= Formula4
  | Formula4 Implication Formula5
Formula4 ::= Formula3 (Disjunction Formula3)*
Formula3 ::= Formula2 (Conjunction Formula2)*
Formula2 ::= Formula1
  | Negation Formula2
  | Universal IndividualVariable Formula2
  | Existential IndividualVariable Formula2
Formula1 ::= '(' Formula5 ')'
  | AtomicFormula
```

PredicateFormula は述語論理式、AtomicFormula は原子論理式、Term は項、Function は関数を表す。

各述語記号の引数の数は、その述語記号のアリティと一致していなくてはならないが、引数の数が違う場合述語記号が同じでもお互い違う関数であることは直観的に認識できるので、実装の際は区別しない事にした。各関数記号についても同様。

例をあげると、$\forall x P(a, f(x, c)) \land Q(x, y)$ という原子論理式の場合、\forall x P(a,f(x,c)) \land Q(x,y) と入力するが、構文解析は図 1 のようになる。変数記号と定数記号は区別する必要があるので、どの記号が変数記号かは与えておく。例えば、変数の設定に x, y と指定しておくと、残りの a, c は定数と認識する。

また、命題論理式や述語論理式はは A_1、関数記号は t_1 のように添字をつけたりする事もあるので、これにも対応することは今後の課題である。

論理式の構文解析ができていれば、3.4 に述べた半自動証明の他、例えば論理式の省略形 ($\land$ を $\lor\neg$ による略記として扱うなど) の書き換えやドモルガン則の置き換

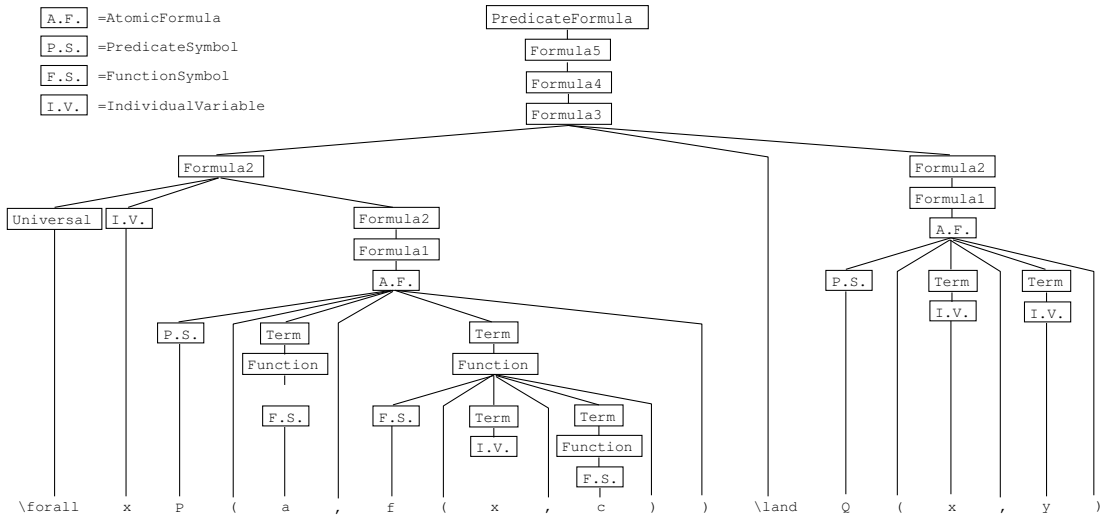


図 1: 構文解析の例

えなどもを可能になると考えられ、また、論理式を冠頭形に書き換える事ができるようになると、束縛変数と自由変数の違いを認識することで α 変換、スコーレム関数を導入する事によりスコーレム化ができると考えられる。また、命題論理においては命題記号の真偽の解釈を決定することにより、命題論理式の真偽を得ることができるし、一階述語論理においては述語記号と定数記号を含む関数記号の解釈を決定することにより述語論理式の真偽を得ることができる。この解釈をもとに計算を行う機能も実現できると考えられるので、将来実現したい。

5 考察

Jpre は現在、構文に対応したクラスの設計や、フォントの自動生成、基本的なキー入力や画面表示などの構築を終え、実装を進めている段階である。Jpre により、限られた紙面幅へ複雑な証明図を柔軟にレイアウトするようなことが可能になり、証明論など記号論理学の研究や、証明論を基盤として用いる知的エージェントシステムなどの進展に寄与すると期待される。また、大学などで記号論理学の基礎を履修するにあたって、古典論理などの具体的な論理体系について、その証明システムに関する理解を、Jpre の半自動証明による証明図記述を通して深めるなどの応用も考えられよう。

今後の課題としては、さらなる実装をすすめることや、異なる推論規則をもつさまざまな論理体系に容易に対処できるような機構の開発などが挙げられる。

参考文献

[1] Makoto Tatsuta: Proof Figure Macros for LaTeX, `http://research.nii.ac.jp/~tatsuta/proof-sty.html` (2005)

[2] 毛利元彦: xpe(X window system Proof Editor), `http://www.jaist.ac.jp/~mouri` (2000)

[3] 毛利元彦: 証明支援システム xpe, 電子情報通信学会技術研究報告. *AI,* 人工知能と知識処理, Vol. 102(91), pp. 55–59 (2002)

[4] 萩谷, 西崎: 論理と計算のしくみ, 岩波書店 (2007)