

## 分散する証明推論エンジンのWEB上での結合について

佐塚秀人 長沢武司 廣川佐千男  
久留米工業大学 九州大学 九州大学

我々は型推論と証明探索エンジンをWebブラウザ上で3D GUI表示で利用できるシステムを公開している (<http://whale.i.kyushu-u.ac.jp/prover-j.html>). この経験に基づき現在、複数の推論エンジンのWeb上での結合とGUIのレイアウトを柔軟に記述できるようなLisp風言語を開発した. そのスクリプト言語をサーバ、ブラウザの双方で用い、JavaによるAppletも自然に組み込める枠組を与えた.

### 1. はじめに

我々は型推論と証明探索エンジンをWebブラウザ上で3D GUI表示で利用できるプロトタイプ・システムを公開している (図1).

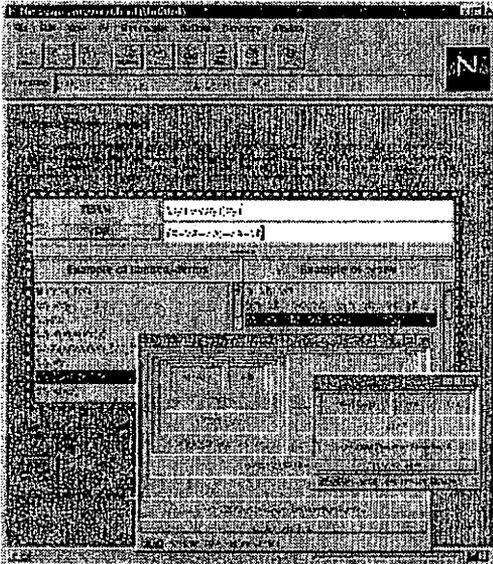


図1: インターネット・ブルーバ

我々の研究目標は、インターネット上に機能を提供するための枠組であって、ブラウザ上のみで動くJava Appletなどではない。ネットワーク検索エンジンは現在インターネットで提供される一番便利な機能ではあるが、基本的には静的なデータが対象でその上のサービス機能でしかない。我々が行っているのはこのような静的データやその加工ではなく、利用者の要求に応じた知的計算機能のインターネット上での提供である。本論文では、証明推論システムの機能をインターネット上に提供する方式について述べる。

2章では、インターネット対応となる以前の推論システムからインターネット版へどのように移行したかを述べる。3章では、現在公開しているインターネット版の実験システムの実現方式と評価を述べる。4章ではこの実験システムの経験を踏まえ、現在作成中の新しい方式について述べる。推論エンジン自体には一切手を加えない点は、新しい方式でも守っている。現在の実験システムとの大きな違いは分散する複数の推論エンジンの結合ができることと、GUIや複数のエンジンの結合を柔軟に記述できるスクリプト言語の導入である。

---

A Framework for Distributed Provers on WEB System,  
Hideto Sazuka · Kurume Institute of Technology,  
Takeshi Nagasawa · Kyushu University,  
Sachio Hiroakwa · Kyushu University

## 2. 証明推論システム

我々は具体的な対象を推論システムとして実験を始めた。これまでに多くの推論システムが開発され、現在ではそれらのシステムはインターネットから容易に持ってこられるようになった[2]。このような推論システムのホームページの中には、文字ベースで CGI により推論結果を返すものもある[1]。しかし、このような推論システムを実際に利用するにはシステムを自分の計算機環境に移植して再構築しなければならない、複数のシステムを統合的に利用はできないという問題点があった。このような環境構築はシステム開発者や管理者にはそれほど困難なことではなくても、利用者にとっては無関係な知識が要求されることになり、そのような障壁を乗り越えなければその推論システムが自分の本来の目的にどれだけ有効かを実感できない。

一方、インターネットとその上の WWW システムの急速な普及によって、図形などを含む文書を誰も自由に公開できるようになった。利用者側は専用のブラウジング・ツールを用意する必要はなく汎用のブラウザを用いて、世界中の情報を入手することができる。また、情報提供側も HTML 形式でドキュメントを記述し、HTTP サーバを立ち上げるだけで、特別な登録手続きもなく情報を公開することができる。このような手軽な環境の中に、機能や GUI を提供し、興味をもってくれる人に容易に公開できる方式の検討を開始した。その後、証明エンジンのためのユーザ・インターフェースのワークショップも開かれるようになってきている[3]ことや、その中には、我々と同様に WWW への対応の研究[4]もあることが分かってきた。

## 3. プロトタイプ

ラムダ項の型推論、論理式の証明探索プログラムに GUI を与えるプログラムを Tcl/Tk を用いて作成した。このシステムの機能を

そのままインターネットで公開することを第一目標としてプロジェクトをスタートさせた。

Tcl/Tk を用いたシステムは、KCL(Kyoto Common Lisp) で書かれた証明システムをエンジンとして、UNIX のプロセス間通信を用いて Tcl/Tk で書かれた証明図示プログラムと通信するように作られていた。KCL, Tcl/Tk 共に List を扱う機能と、その入出力機能をもっているため、通信は単純な Lisp の List 形式を採用した。

Web ブラウザで、何らかのプログラムを動かす方式としては、ブラウザ側で Helper Application を登録し、データタイプに従ってブラウザから Helper Application を自動起動する方式が用意されている。Tcl のプログラムも MIME タイプとして application/x-tcl 登録されており、Tcl/Tk のインタプリタである wish を Helper Application に登録することによって、図形表示機能を提供することができた。Web の CGI 機能と組合せ、サーバ側で KCL を起動しその結果を Tcl/Tk プログラムに埋め込んで送ることによって、最初のプログラムを Web 上で提供できることを確認できた。

しかし、この方式は以下の面で問題があることが最初からわかっていた。

1. Tcl/Tk のインストールの必要性
2. セキュリティの問題
3. 対話処理機能を提供できない

1 はマルチプラットフォームでの利用に難がある。ブラウザ側の Tcl/Tk のバージョンの問題も出るだろうと予想できる。2 はこの時点では致命的と思われた、ファイル入出力やプロセス起動を自由にできる Tcl/Tk はこのような目的には使えない(後に Safe Tcl があることを知る)。3 も CGI で利用できる通信が 1 回だけの問い合わせ機能だけであるため、対話的な GUI は提供できない。以上の理由から Tcl/Tk を継続して利用していくことは断念した。

Helper Application の形態でなく、ブラウザに直接プログラムの処理と画像表示,GUI を提供できるものとして,Java 言語による Applet がある。Applet は中間言語に翻訳されたプログラムをサーバからダウンロードし,Web のページ内に表示させることができる。Tcl/Tk ほど簡単ではないが,AWT(Abstract Window Toolkit) と呼ばれる Toolkit を持ち,OS や Window System に関係なく GUI を提供することができる。

Java を用いることによって上記の問題 1, 2 はほぼ完全に解決できることがわかった。3 は Java では HTTP とは関係なく,独立に socket インターフェースが提供でき,通信は Applet プログラムのダウンロード先以外とはできないという制限はあるが TCP による stream を利用することができる。

次のプロトタイプを Web で公開できることを目標として次のステップで作成した(図 2)。

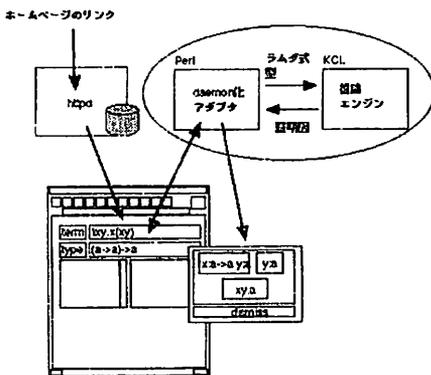


図 2: プロトタイプの構成

1. Lisp や Tcl/Tk の List 処理機能の Java での実現
2. Tcl/Tk の GUI 機能の Java への移植
3. stream による通信機能の実現
4. KCL エンジンのサーバ化フロントエンドの実現

1 は最初の Tcl/Tk を利用したプロトタイプが, List 処理を中心に作られていたため, まず最初に Lisp 流の List 表現機能を用意した。Java にはガベージ・コレクション機能もあり, 動的なデータ構造はオブジェクトを単位として比較的自由に扱うことができる。Lisp, Tcl/Tk 両方の入出力形式で文字列から内部構造への双方向の変換をできるような機能は最初から用意しておいた。

2 は Tcl/Tk の機能に依存して作られていた点や, Tcl/Tk のほうが高度なレイアウト機能をもっていたため, それを補う機能を Java 側で作成し, ほぼ同様の出力をさせることができた。しかし, この段階で専用 GUI の作成を Java で行うことは, X Window や Macintosh, Windows でプログラミングすることに比べれば容易ではあるが, GUI プログラミング経験のない人にはそれなりの作業量になるものであることを感じた。コード量も Tcl/Tk で作成したものの数倍になっている。

3 の socket による TCP 通信については, Java が容易に使えるライブラリを用意している。サーバ側も Java で容易に書くことが可能である。このプロトタイプでは, 4 の機能を容易に実現することを考え Perl で作成している。Java Applet を用いることができるブラウザでは, セキュリティ面の配慮から Applet コードのダウンロード先と, socket のよる通信先が同一ホストでなければならない制限がある。

4 は, KCL で書かれたプログラム (エンジン) には一切手を入れないという方針で機能を用意した。このようなプログラムは既に作成されているという仮定のもとに, エンジン側から見た時, キャラクタ端末に対して出力しているようにみなせるようにするために, ブラウザ側から見た場合にサーバとして振る舞うようなアダプタとして作成した。

KCL 側には UNIX の仮想端末をインターフェースを提供し, KCL は端末を相手だと思って入出力を行う。アダプタでは, 出力結

果からパターンマッチにより必要な情報を切り出し、ブラウザ側に送る。このプログラムは Perl で書かれた標準的なサーバプログラムである。socket への接続要求により、コネクションを確立し、仮想端末を用意した後、KCL プロセスを起こし必要なプログラムをロードさせる。後は、コマンド入力状態で要求に応える形になっている。KCL と Perl のフィルタプロセスは、1 要求ごとに fork するようになっているために、一度に多くの要求が来た場合の負荷が問題になるが、このプロトタイプで特に考慮しなかった。

#### 4. プロトタイプの評価

##### 4.1. コメントから

推論エンジン自身には全く手を加えてないので、短期間でプロトタイプを公開できた。6月下旬から数カ月運用を継続しているが9月中旬の時点で、アクセスした件数は表1の通りである。

表 1: アクセス・カウント

6633	jp	53	au	17	ca
213	edu	42	dk	12	gr
173	uk	39	pl	11	nz
115	de	29	org	11	in
113	com	26	it	11	ie
105	se	25	es	9	gov
99	fr	22	kr	6	no
73	ch	21	net	3	pt
71	nl	17	fi		

メール等で以下のようなコメントや意見、提案を得ることができた。ブラウザのバグにより表示できなかつたり、ハングするという苦情もあった。また、Firewall が存在するために Applet との通信ができないという苦情は多数寄せられた。利用できた人のログを観察すると、大半は入力例のメニューか

らデータを選択し実行していた。確かに、メニューが存在しなければ、利用者は具体的に何を入力すればいいのかわからない場合がほとんどであったはずである。

Kansas State University の A. Stoughton からは、彼の Porgi システム [5] が同様な証明システムであるというメールがあった。それで、実際その Porgi システムを ftp で取りインストールして、我々のプロトタイプの枠組により Web ブラウザから利用できるようになった。

以上のこれらのコメントから、以下の3つの開発項目が浮かび上がった。

1. GUI の部品化
2. 他のエンジンとの結合
3. http で保障されるネットワーク機能の利用

1 については、本質的な機能を活かすために、例題のメニューの表示のような補助的な機能が重要であることがわかった。Porgi システムにおいて、その本質的な機能は通常の CGI により実現したが、この GUI と組み合わせることにより、使いやすいものに容易に仕上げることができた。1,2 いづれにせよ、部品化とその結合方式がシステムの柔軟性を左右するといえる。

計算エンジンとなるものも、中心となるエンジン機能以外にも、汎用的なデータの変換機能や、データの蓄積機能、ライブラリ機能などを考えることもできるであろう。

3 は、現在のインターネットの環境を考えると大きな問題であることがわかった。ブラウザのページ内の Applet と、サーバは HTTP による通信とは異なるポートを用いて通信を行うようにしているが、Firewall によってその通信はできない。Applet は表示されるが、その機能を使うことができないというメッセージがメールで数多く寄せられた。インターネットで機能を公開する場合、特殊なポートを利用する通信は最近のネットワーク事情を考えると実用的ではないことがわかった。

## 4.2. 部品の細分化と Web 上で結合

Java は Web 上に機能を提供するための強力な機能をもっている。GUI パーツや通信を記述するために十分な機能を提供してくれる。しかし、コンパイラ言語であり、HTML を記述するような手軽さはない。前述の3つの資源の結合を記述できるスクリプト言語を検討するに至った。

Netscape Navigator では、Web ブラウザの資源をコントロールするスクリプト言語として JavaScript を用意している。JavaScript は HTML を拡張する形で、Web ブラウザをより細かく制御できる。Netscape Navigator 3.0 では Java Applet のメソッドを呼び出す LiveConnect 機能が提供されている。LiveConnect を利用することによって、Web ページ上の Applet を JavaScript でコントロールすることができる。

しかし、JavaScript が呼び出せるものは、既にブラウザ側にロードされている class に限られる。JavaScript からネットワーク機能呼び出すことができないために、ネットワークからダイナミックに class ファイルを読み込む ClassLoader を呼び出すことができない。そのため単純に Applet にパラメータを渡す程度の目的には利用できるが、Applet の機能呼び出し結合するような目的には適さないことがわかった。

このことから上記の3つの資源をコントロールするスクリプト言語を Java で記述し、Web 環境を利用した、分散処理環境のフレームワークを構築することにした。

## 5. 機能部品化 Web システム

### 5.1. 利用者、開発者の分類

まずここで、本システムにかかわる人の立場を明確にするために、以下の4種類に立場を分類する。

#### 1. ブラウザ利用者

Netscape Navigator 等の Java Applet

が動作するブラウザを用いて情報を見ようとする利用者。この利用者は Web 上のドキュメントをブラウジングする感覚で、ネットワーク上でサービスされる機能を利用できる。プログラムをダウンロードして、自分の利用するシステムにインストールしたり、機能提供側の計算機に利用者登録をしたりしなくても自由に利用できる。

#### 2. 環境構成者 (ページ作成者)

各種のツールを組み合わせ、Web ページとして提供する人。Network 上に提供される3の立場の人が提供する機能を既存の HTML ドキュメントと組み合わせ提供して提供される。この立場の人は Java でプログラミングすることはせず、HTML を拡張する Script 言語を利用して Web ページの形にエンジンの機能ツールと GUI を組み合わせてレイアウトする。

#### 3. 資源提供者

エンジンまたは GUI の部品を提供する。計算を中心とする機能を提供するエンジン、Web 上で GUI を提供する GUI パーツを資源と考える。プロトタイプとして作成した、証明図のグラフィック機能は Logic の分野で汎用的に使用することができる。グラフの表示や入力ツールはより汎用的に使用できる。計算エンジンは、ネットワーク上でサーバ化するアダプタを用意する必要がある。テキストを出力を基本とする既存のツールは、仮想端末に出力した結果からパターンマッチや出力のタイミングを利用して結果を切り出し、サーバと通信するアダプタでの対応を考えている。既存のエンジンはこのアダプタを作成することによって、本システムの資源として提供できる。

#### 4. フレームワーク提供者

部品間のネットワーク接続を提供する通信環境、3のツールの結合を提供するスクリプト言語を用意する立場。すな

わち本システムの開発者.

## 5.2. システム全体の構成

次に4の立場で提供する枠組について説明をする. 図3にシステム全体の概念図を示す.

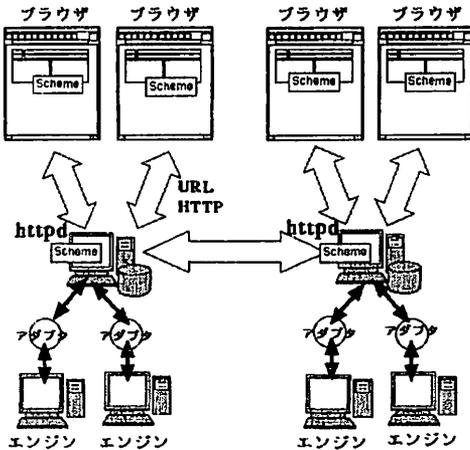


図3: システム概念図

サーバが提供するコントロールの枠組として下層機能から以下の5ステップで考える.

1. エンジン, ブラウザ間の仮想コネクション
2. 部品間の標準データ形式としての List
3. 部品間のコーディネイトするインタプリタ
4. スクリプト言語としての Scheme
5. 拡張 HTML としての WKML

エンジンやブラウザの GUI 部品 HTTP を用いて, 仮想的なコネクションをもつ. 部品間の通信は Lisp の List 形式を用いて行われ, 各部品にはコーディネータとして小さな Scheme インタプリタを貼り付けることができる. 各部品のコントロールは, Scheme をスクリプト言語として, 部品間のやりとりを記述する. さらに, サーバ側ではそのスク

リプト記述のためのマクロ機能を用意し 拡張 HTML として提供する.

## 5.3. エンジン, ブラウザ間の仮想コネクション

プロトタイプ・システムでは, 通信に Java が提供する stream を用いたが, それでは Firewall を越えることができない. そのため, サーバとの通信は HTTP を用いる.

HTTP は, ブラウザ側からデータを要求する GET, データを送る POST コマンドが基本であり, 永続的にコネクションの維持はできないため, エンジン側との対話的な処理には向かない.

このために, サーバ側で通信のコンテキストを一時的に保持することによって, エンジンとの間では stream で接続し, ブラウザとの間は HTTP を用いて接続する. 通信のコンテキストに ID を割り当て, それをやりとりすることにより仮想的にコネクションを維持する.

ブラウザの利用者は, 正常な終了をすることは限らないが, コンテキストは通信に矛盾が生じた場合や, ある一定の時間が過ぎても通信の継続がない場合にエンジンとの通信を破棄することにする.

このメカニズムの欠点として, 次の2つが考えられる.

1. 全二重の通信は実現できない.
2. CGI によるプログラム起動のオーバーヘッドが大きい.

1 について, 現状では全二重の通信を必要とするような, リアルタイム処理は対象としない. 必要な場合は, 特別なポートを用意する方法, FTP や TELNET のポートを利用する手法を検討する.

2 については従来からサーバでは CGI によるプログラム起動をする必要があるが, 最近ではサーバにモジュールを追加する形や, システムを動的拡張する形で CGI リクエストを処理することができる. また, w3c の Jigsaw や Sun Microsystems の Jeeves で

は、Java を用いて機能拡張することができる。これによって処理の高速化が可能となる。

この処理は現在 Java で記述をしており、今後 Java で書かれたサーバへの拡張として実現することを準備している。

#### 5.4. オブジェクト間のデータ形式

汎用的な HTTP を用いるため、URL を用いてネットワーク上の資源を指定することが可能である。エンジンは daemon 化アダプタを介するか、あるいは直接インターフェースを用意し通信する。サーバは、エンジンとの stream による接続を HTTP 化し、ブラウザや、他のサーバに機能を提供する。

サーバとの通信は、ページを記述する HTML 以外に Lisp の S 式を基本にした List データ形式を用意する。各オブジェクト間は Lisp の List 形式を利用して通信する。内部では、直接 List の内部表現 (Java オブジェクト) を、ネットワークを介した通信では List を S 式表現した文字列により通信する。それには HTTP の MIME type として “application/x-list” を用いる。

#### 5.5. 部品間のコーディネータ

ネットワーク上に分散する、サーバ、ブラウザ上のオブジェクトのコーディネータ用に拡張性をもった小さな Scheme (Lisp) インタプリタを Java で記述した。このインタプリタを各所に配置することによって、コーディネーションを行う。

このインタプリタは Java で記述され、それ自身 Java の Object として Java で書かれた部品からは自由に呼び出すことも、機能の一部として組み込むことも可能である。また Java で書かれた class の Object を動的に Lisp のオブジェクトとして扱うこともできる。各部品のフロントエンドとして、その接続を受け持つ能動的インターフェースとして機能する (図 4)。

この Scheme インタプリタは、ブラウザ

側、サーバ側双方で動作し、全体のコーディネーションはその Scheme スクリプトを用いてすべて記述する。

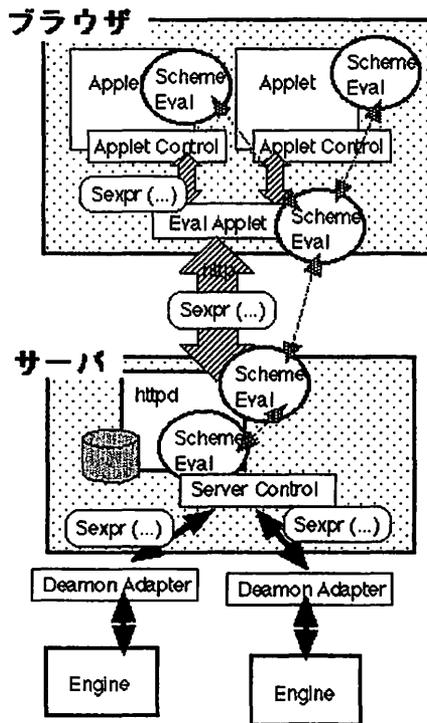


図 4: 部品間のコーディネーション

#### 5.6. Scheme スクリプト言語

Scheme の処理系は Java を用いて記述しており、Java のオブジェクトを扱うことができる。手続きやオブジェクトは Java で書かれた class をファイルを動的に組み込むことができる。

Scheme で Java のオブジェクト扱うためには、各種のインターフェースをあらかじめ実装しておく。基本的な Scheme のオブジェクトは LispObject インターフェース、ブラウザ上の Applet のコントロールには AppletControl インターフェースを実装しておく。

このスクリプト言語の処理系の中核となる Eval 自体が Java のクラスとして書かれている。新しく Eval のインスタンスを生成することによって、自由にインタプリタの機能を付加することができる。Eval の中からさらに入れ子に Eval を生成し呼び出すことも可能であり、その構成は十分な柔軟性をもつ。

## 5.7. 拡張 HTML

HTTP サーバでは Scheme スクリプトを記述するために HTML を拡張した WKML を用いる。HTML にサーバ側マクロ処理記述を加えたものである。JavaScript はクライアント側だけのスクリプト言語であるが、本システムで用意した Scheme はサーバ側の処理も記述する必要がある。ブラウザ側の処理は Applet のパラメータとして記述する。サーバ側の処理は新たにマクロタグを追加、直接 HTML の中にエンジンやブラウザの Applet との通信を記述できるようにする。

## 6. おわりに

ラムダ項の型推論と直観主義論理の証明探索のシステムを Web ブラウザから利用できるようにして公開した。これは、単独の推論エンジンにアダプタを付けることによりサーバ化し、ネットワークによる論理式や証明図の通信には、これらデータ構造を Lisp の S 式としてソケットで通信した。ブラウザには入力パーザや S 式から証明図形を 3D 表示する GUI パーツを Java Applet として実現したものである。この経験を踏まえて、現在我々はネットワーク上に分散する複数の推論エンジンを Web ブラウザから GUI を通して利用できるシステムを構築中である。このシステムで提供する資源は

1. ネットワーク上に分散するエンジン、
2. ネットワークによる通信環境、
3. Web 上の Java Applet による GUI パーツ

の3つである。構成等のコンフィギュレーションや GUI のレイアウト、そしてこれら資源の接続を柔軟に記述できる Lisp 風言語を開発している。この言語は、HTML 中では新しいタグとして使われる。またサーバ、クライアントの両方のためのスクリプトとして使われ、Java によるアプレットも自然に組み込める。

この枠組を利用し、推論エンジン開発者と共同し、その機能を表示する GUI 部品とともに新しい Web のページ公開の予定である。

## 参考文献

- [1] A. Heurding, G. Jager, S. Schwendimann, M. Seyfried, Propositional logics on the computer, in: proceedings of Tableaux 95, Springer LNCS 918, 310 - 323, 1995.  
<http://lwbwww.unibe.ch:8080/LWBInfo.html>
- [2] M. Kohlhase, C. Talcott, Automated reasoning Systems Database のページ,  
<http://www-formal.stanford.edu/clt/ARS/ars-db.html>
- [3] N. Merriam, Electric proceedings of UITP '96, (User Interfaces for Theorem Provers)  
<http://dpcpu1.cs.york.ac.uk:6666/~nam/uitp/proceedings.html>
- [4] J. Pitt, A WWW Interface to a Theorem Prover for Modal Logic, in: proceedings of Workshop on User Interface Design for Theorem Proving Systems 1996.
- [5] A. Stoughton, Porgi: a Proof-Or-Refutation Generator for Intuitionistic propositional logic, proceedings of CADE-13 Workshop on Proof Search in Type-Theoretic Language, pp. 109-116, 1996.  
<http://www.cis.ksu.edu/~allen/porgi.html>