

# viewPP ユーザーズマニュアル

for viewPP version1.00

・目次

- 1. はじめに
- 2. 本ソフトウェアの構成
- 3. インストールおよび実行
- 4. viewPP の操作
- 5. 入力プログラムファイルの文法
- 6. 実際の操作例

・付録: viewPPに関する用語集

---



この記号で示される段落で、現段階でのバグその他の原因による不具合についての説明をしています。

---

## 1. はじめに

“viewPP”は、並列論理型言語GHCの実行を可視化し、実行の過程をわかりやすく表現することができるビジュアルプログラミングシステムである。

GHCには、「実行順序がプログラム記述順序に依存しない」「並列性を素直に記述できる」などの特徴があるが、「論理変数が増えると、それらに適切な名前を付けづらくなる」「実際に並列に動作している様子は目に見えにくい」といった短所もある。これらの短所は、ビジュアル化によって解決できるものである。

実際に“viewPP”では、

- プログラムはグラフ構造で表現すること
- 論理変数は、グラフ構造の「辺」によって表現すること
- 複数の図形(グラフ構造)の変形によってプログラム実行を表現すること

によって、これらの短所を克服している。

現在の実装では、述語の定義自体は、GHCのサブセットのような文法を持ったプログラムファイルをviewPP上でロードして、その述語を実行フィールド(後述)上に配置し、それに入力データを接続して実行させることによって、プログラムを動作させる。

---

## 2. 本ソフトウェアの構成

viewPP は、以下の3つの実行モジュールから構成される。

viewpp.tcl

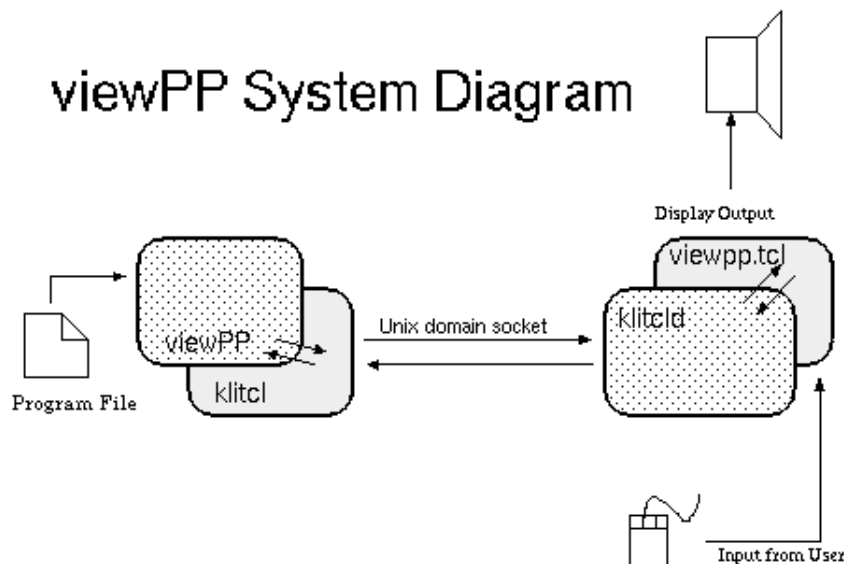
GUI およびガードユニフィケーション実行部であり、Tcl/Tk で記述されている。

viewpp

viewPP 本体であり、ファイルロードおよび、入力ファイルの構文解析をするルーチンが含まれる。klic で記述されている。実行ファイルは klitcl.kl1 と共にコンパイルされる。つまり、viewpp 本体は viewPP 処理部と klitcl の2つの部分から成っている。

klitcl

Tcl/Tk と klic のインタフェースをとるプログラム klitcl のコアとなるプログラムであり、Tcl/Tk インタープリタを保持し、klic プログラムからの要請によって(Tkを通じて)画面描画などを行う。C で記述されている。



これらのうち、viewpp および klitcl の2つは実行可能なバイナリであるので、コンパイルによって生成する必要がある。これらの生成法およびプログラムの実際の実行方法については、次章で説明する。

### 3. インストールおよび実行

viewPP を利用するには、利用しようとする環境に以下のツール類がインストールされている必要がある。

- klic (version 2.002)
- Tcl (version 7.5)
- Tk (version 4.1)

ただし、以下のことに注意されたい。

1. バージョン番号については、作者が動作試験を行った環境のものであり、これよりも古い/新しいものでも動作する可能性は大きい。ただし、「Tcl/Tkのバージョンが古い(特に、Tkが3.6以前である)」場合は、動かない可能性が大きいと思われる。
2. 作者が動作試験で使ったOSはSunOS 5.4/5.5.1である。他のOSであっても、上記の環境が用意されていれば動作する可能性は大きいと思われる。
3. klicについては、コンパイル時に「非同期I/O機能」を有効にしておく必要がある(Configure時の "Does your system support I/O ready signals?" に yes と答えていること)。
4. klicが動く環境であればANSI Cコンパイラ(大抵はGCC)も入っているはずである。viewPPを動作させるにはANSI Cコンパイラが必要である。

まず、展開されたファイル群に以下のファイルが含まれていることを確認する。

- Makefile
- klitcld.c
- klitcl.kl1
- viewpp.tcl
- viewpp.kl1
- list.kl1
- viewpp\_fileload.tcl
- viewpp\_gardpred.tcl

参考までに、先に述べた3つの実行モジュールが、どのファイルから生成されるかを以下に記しておく。

viewpp.tcl

これはTcl/Tkのプログラムであるので、klitcldによって保持されているインタプリタからそのまま実行される。ただし、内部でviewpp\_fileload.tcl viewpp\_gardpred.tclの2つのプログラムを読み込んでいる。

klitcld

klitcld.c および、各自の環境にインストールされているTcl/Tkライブラリ(libtcl.aおよびlibtk.a)から生成される。

viewpp

viewpp.kl1, list.kl1, klitcl.kl1の3本のプログラムをklic処理系でコンパイルすることによって生成される。

## コンパイル

コンパイルの前に、Makefileの修正を行なう必要がある。Makefileの先頭付近に書かれた変数の変更が主である。変更の可能性のある変数名とその内容は以下の通りである。

CC

使用するCコンパイラ

XINCLUDE

X Window関係のインクルードファイルの位置

XLIB

X Window関係のライブラリファイルの位置

XRLIB

(同上)

TCLTKINCLUDE

Tcl/Tk 関係のインクルードファイルの位置

TCLTKLIB

Tcl/Tk 関係のライブラリファイルの位置

これらの変数を各自の環境に応じて適切に設定した上で、シェルのプロンプトから

```
% make
```

と入力すれば、2つの実行バイナリ `klitcld` と `viewpp` が生成される(GCCを使う場合途中で Warningが出るが気にしなくてよい)。

## viewPP を動作させる

viewPP の動作にあたって、以下の10個のファイルが実行時のカレント ディレクトリに置かれている必要がある。

- `klitcld`
- `viewpp`
- `viewpp.tcl`
- `viewpp_fileload.tcl`
- `viewpp_gardpred.tcl`
- `bitmap/clear.ppm`
- `bitmap/exec.ppm`
- `bitmap/exit.ppm`
- `bitmap/file.ppm`
- `bitmap/replace.ppm`

この他、実際に使うためには、viewPP 用のプログラムが書かれた 入力ファイルが必要である。この記述方法については後述する。

viewPP を動かすには、まず `klitcld` を常駐させる。シェルプロンプト から以下のように入力して、`klitcld` をバックグラウンド実行させる。

```
% ./klitcld &
```

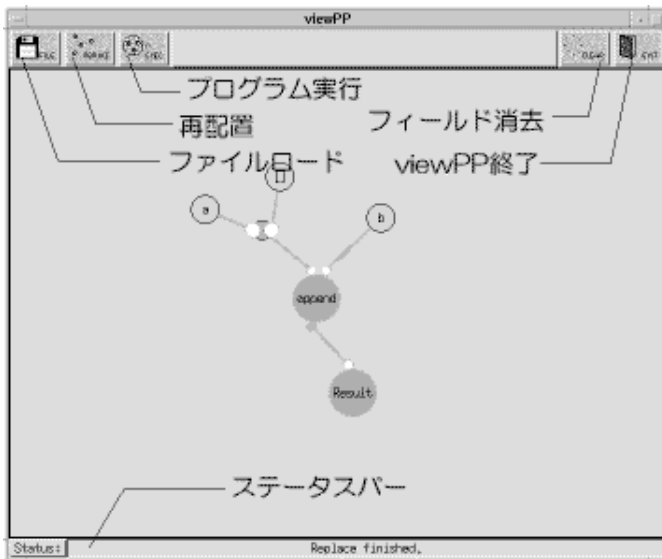
この後、viewPP 本体を実行させる。

```
% ./viewpp
```

少し待つと、viewPP の実行画面が表示されるはずである。実際の viewPP の操作については、次章で説明する。

---

## 4 . viewPP の操作



viewPPを実行すると，上のような画面が現れる．

viewPPの画面は，大きく3つの部分に分けることができる．

- 機能アイコンが並ぶアイコンバー
- 実際にプログラムを動かす場である実行フィールド
- viewPPの状態などを表示するステータスバー

以下で，それぞれについて説明する．

## アイコンバー

ここには5つのアイコンがあり，左から「ファイルロードアイコン」「再配置アイコン」「プログラム実行アイコン」「フィールド消去アイコン」「viewPP終了アイコン」である．

### ファイルロードアイコン

外部プログラムファイルをロードして，viewPP上で使えるようにする

### 再配置アイコン

実行フィールド上のプログラムを再配置して見やすくする

### プログラム実行アイコン

実行フィールド上に作られたプログラムを実行する

### フィールド消去アイコン

実行フィールド上のプログラムをすべて消去する

### viewPP終了アイコン

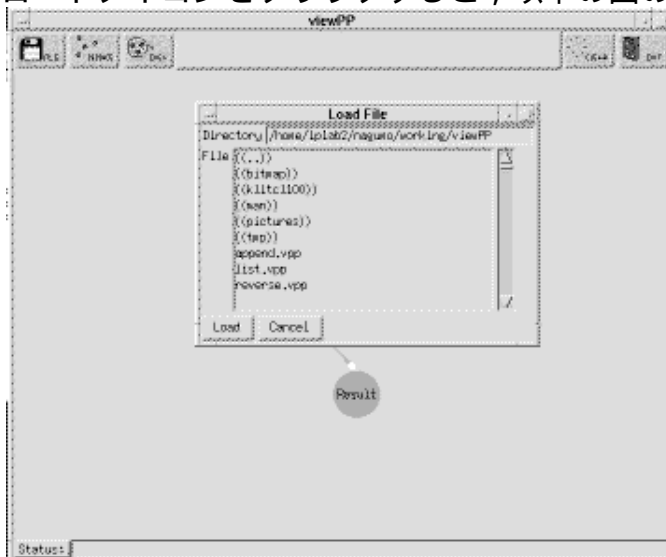
viewPPの実行を終了する

これらのうち，「再配置アイコン」「プログラム実行アイコン」「フィールド消去アイコン」は，マウスでクリックすることにより即座にそれぞれの動作をする．

残りの「ファイルロードアイコン」「viewPP終了アイコン」については，以下で説明する．

## ファイルロードアイコン

ファイルロードアイコンをクリックすると、以下の図のようなファイル選択ウィンドウ




が開く。

ウィンドウの一番上にはカレントディレクトリ名が表示されており、その下に、カレントディレクトリにあるサブディレクトリ名と viewPP入力ファイル名(拡張子が “.vpp” であるもの)が表示されている。

このとき、サブディレクトリ名が “ ( ( ) ) ” で囲まれて表示される ことにより、ファイル名とは区別される。

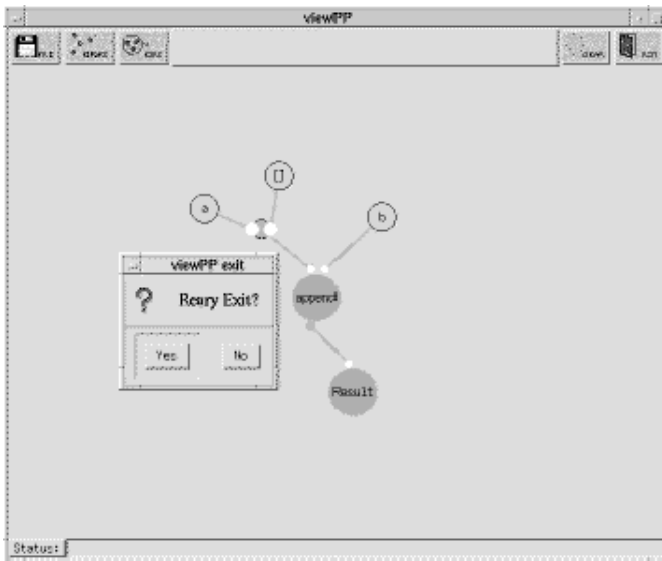
これらの名前をマウスの左ボタンでクリックして選択し、ウィンドウ下部にある “ Load ” ボタンを押す、あるいは、名前をマウスの左ボタンでダブルクリック することによりファイルのロード、あるいはディレクトリの移動ができる。  
ファイルのロードが終了すると、ファイル選択ウィンドウは閉じる。

“ Cancel ” ボタンを押せば、ファイルをロードせずに ファイル選択ウィンドウを閉じることが出来る。

 klitclとviewPPの通信の不具合により、まれにロードがうまくいかないことがあります。viewPPを起動した(仮想)端末上にロード中の ファイル名や内部的に使われる述語の番号が表示されていきますが、この述語番号(カッコで括られて表示されています)が連番でない場合にはロードが失敗しています。

## viewPP終了アイコン

viewPP終了アイコンをクリックすると、以下の図のような終了ダイアログが開く。



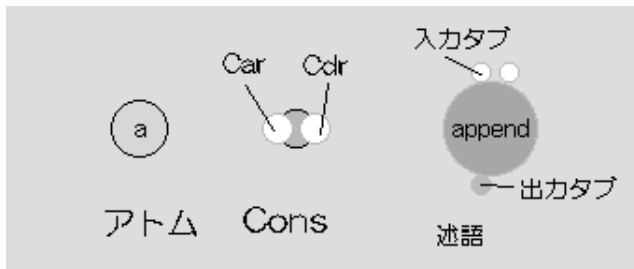
“ Yes ” をクリックすれば viewPP は終了する． “ No ” をクリックすれば，終了ダイアログが閉じて，viewPPの操作を続けることができる．

viewPPの実行終了後に，klitcd のプロセスがシステムに残っていないかを ps コマンドや jobs コマンドで確認して，もしプロセスが残っていた場合には確実に消去しておくこと．

## 実行フィールド

ユーザは，このフィールド上に述語やデータ構造を配置 することでプログラムを作る．

viewPPでは，プログラムは「節」と「辺」からなる グラフ構造として表現される．viewPP で使用できる プログラム要素のうち，アトム・CONS・述語は「節」に対応し，論理変数 およびCONSとアトムの関係付けは「辺」に対応する．以下で，「辺」および「節」の配置の仕方について説明する．



### 節 (述語・アトム・CONS) の配置

実行フィールド上の，まだ何も配置されていない部分で マウスの左ボタンをクリックすると，メニューが現われるので，そこで「述語，アトム，CONS のうちのどれを置くか」の選択をする．

#### 述語(Predicate)の配置

メニューから “ Predicate ” を選択すると，その時点で 登録されている述語名が並んだメニ

ユーが現われるので、その中から配置したい述語を選択する。  
選択すると、その場所に述語アイコンが置かれる。

## CONSの配置

メニューから“ Cons ”を選択すると、3つの円が組みあわされた形状の CONSアイコン が置かれる。

左側の白い円が Car を表わし、右側の白い円が Cdr を表わす。

## アトム of 配置

メニューから“ Atom ”を選択すると、水色の円の中にアトムの名前が書かれた Atomアイコン が置かれ、そのアトムの名前を決定するためのダイアログが現われる。

このダイアログ内でアトム of 名称を入力してリターンキーを押すとアトム of 名称が確定する。

一旦確定したアトム of 名称を変更したい場合は、そのアトムをマウスの右ボタンでクリックする。アトム of 名称を編集するためのダイアログが現われるので、適宜編集する。

## 辺 (論理変数) of 配置

辺は、接続される対象 (常に2つである) を順にマウスでクリックすることで配置される。接続対象となるのは、以下のものである。

- Atom
- Consおよびそれに付随する Car,Cdrタブ
- 述語の入出力タブ

辺 of 起点となる接続対象をマウスの左ボタンでクリックすると、それが黒い四角形で囲まれ、選択されたことを示す。この状態から 辺 of 終点となる接続対象をクリックすることで、辺が配置される (実際は無向グラフなので起点・終点 of 区別はない)。

一度選択されて黒い四角形で囲まれたアイコンをもう一度 マウス左ボタンでクリックすると、選択が取り消されて、黒い四角形は消える。

## 節 of 移動・削除

節を移動させるには、移動させたい節をマウスの 中ボタンでドラッグする。

節を削除するには、削除したい節をマウスでポイントした状態でキーボード of Deleteキーを押す。

削除された節に継がっている辺もこのとき一緒に削除される。



現在の実装では、辺を削除することはできません。辺の上にマウスカーソルがある状態で Deleteキーを押すとエラーとなります。



CONS of 移動について、CarおよびCdrタブをドラッグして of 移動はできません。



## ステータスバー

viewPP実行中の様々な情報が表示される。この部分をユーザが明示的に操作することはない。

---

## 5. 入力プログラムファイルの文法

viewPP に入力するためのプログラムファイルは、GHC のサブセットのような文法を持ったテキストファイルであり、“.vpp” という拡張子を持つ (言うまでもなく、ViewPP の略である)。

通常のGHCと大きく異なるのは以下の点である。

- ヘッド部における、入力引数と出力引数を分けるためのダミーの引数 “:” の存在
- 論理変数は1から始まって連続する整数で表現する
- (これに伴って整数アトムの変換が変更される(未実装))
- 無名変数 “\_” には対応していない

例えばこれらの制限の下で、リストの連結を行なう `append` という述語を記述してみると、以下ようになる。

```
append(1,2,:,3) :- 1=[]|3=2.  
append(1,2,:,3) :- 1=[4|5]|append(5,2,6),3=[4|6].
```

viewPP は、実際にこのように書かれたプログラムを正常に受け付ける。



現在の実装では、ガード部に書ける述語は1つだけです。

### viewPPに組込まれているガード述語

viewPP には、ガード部で使うために以下の述語が用意されている。

`integer/2`

入力が整数アトムであれば成功する。

`atomic/2`

入力がアトムであれば成功する。

`atom/2`

入力が記号アトムであれば成功する。

`not_integer/2`

入力が整数アトムでなければ成功する。

`not_atomic/2`

入力がアトムでなければ成功する。

`not_atom/2`

入力がアトムでなければ成功する。

これらはあくまでガード述語であるため、ユーザが実行フィールドに直にこれらを配置

することはできない。

引数が2つあるが、第1引数が入力で第2引数が出力であり、出力には入力と同じものが返る。これは、現在の実装では

```
foo(1) :- atomic(1) | foo(1).
```

のような記述ができないせいである。このよう場合には

```
foo(1) :- atomic(1,2) | foo(2).
```

と書くことによって目的を達成できる。

---

## 6. 実際の操作例

この章では、viewPPの使い方の例を示す。

アーカイブの展開からコンパイルまでが終了している ものとして、カレントディレクトリ以下に次のファイルがあることを確認してほしい。

- klitcld
- viewpp
- viewpp.tcl
- viewpp\_fileload.tcl
- viewpp\_gardpred.tcl
- sample/append.vpp
- sample/reverse.vpp

すべて揃っていたら、viewPPを起動してみる。シェルプロンプトから、

```
% ./klitcld &  
% ./viewpp
```

と順に入力すると、viewPPの画面が現われる。

この時点では、特定の動作をする述語がまったく定義されていない。よって、外部ファイルから述語を読みこむ必要がある。そのために、まずファイルロードアイコンをクリックし、ファイル選択ウィンドウを表示させる。

マウスで((sample))をマウスの左ボタンでダブルクリックするとカレントディレクトリが移動し、sampleディレクトリ内にあるファイルを選択できる状態になる。ここでは、append.vppをダブルクリックしてロードする。この操作によって、appendという述語が登録され、利用可能となる。

次に、appendプログラムとそれに与える引数を実行フィールドに配置してみる。

実行フィールド内の適当な所(中心付近が良いだろう)でマウスの左ボタンを押し、メニューを表示させて、その中から Predicate を選ぶ。このメニューはカスケードメニューに

なっているので、さらにそこからどの述語を置くかを選択する。

選択肢には `result/1` と `append/3` の2つがあるはずなので、ここでは `append/3` の方を選択する。

入力引数に必要なリストと、出力を保持するための特殊な述語である `result` を配置しよう。

`viewPP`には、あるプログラムを実行させた結果として得られるデータを保持するための特別な述語である `result/1` が用意されている。これを、先に `append` を配置したのと同様な方法で配置する。位置は、`append` よりも下が良い。これは、`viewPP`の述語アイコンでは入力タブがアイコン上部に、出力タブがアイコン下部に配置されていることによる。これによって、「入力 出力」が「上 下」という方向の自然な流れにマッピングされることになるということによる。

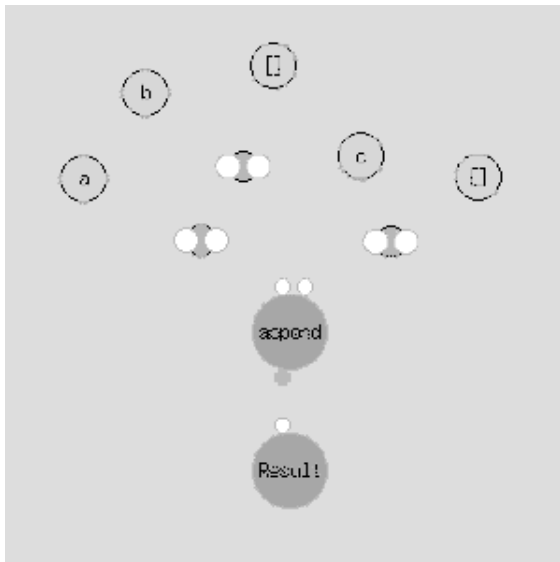
次に、入力引数を配置しよう。ここでは、第1引数には `[a,b]`、第2引数には `[c]` を入れるようにする。

これらを作るには、`CONS`アイコンが3つと`Atom`が5つ必要である(リスト終端の `[]` を考慮すること)。まず、先に配置した `append/3` の上部の適当な位置に `CONS` を2つ置く。実行フィールド内の何も配置されていない部分でマウスの左ボタンを押し、出てきたメニューから `Cons` を選択すればよい。

`Atom`を配置するには、実行フィールド内でマウスの左ボタンを押すことによって出るメニューから `Atom` を選択する。すると、その位置に`Atom`が配置され、`Atom`の内容を変更するためのダイアログが出る。デフォルトは `[]` になっているので、適宜変更する。

一旦配置された`Atom`の内容は、その`Atom`の上でマウスの右ボタンを押すことによってダイアログが現われるので、そこで変更する。

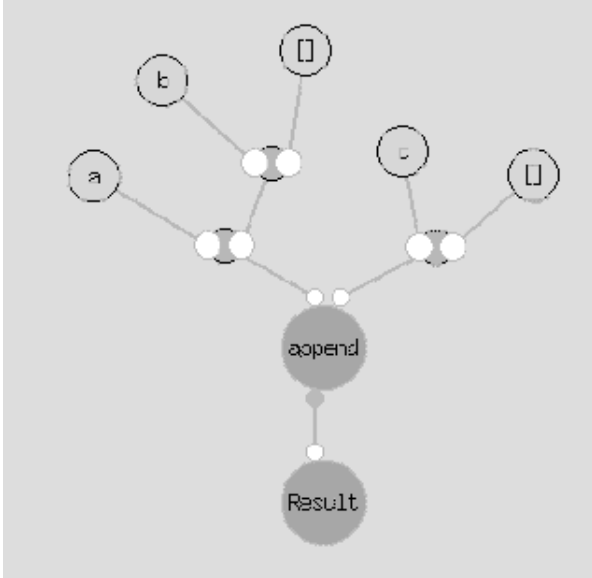
すべて揃ったら、各アイコンをマウスの中ボタンドラッグによって適当な位置に移動させる。以下の図のような感じになるだろう。



次に、これらのアイコンの間を線で継いでいく。

線によるアイコン間の接続は、始点と終点を順にマウスの左ボタンでクリックすることによって行う。  
片方をクリックすると、そのアイコンが黒い四角形で囲まれ、選択されていることがわかる。その状態でもう片方をクリックすると線が引かれる。

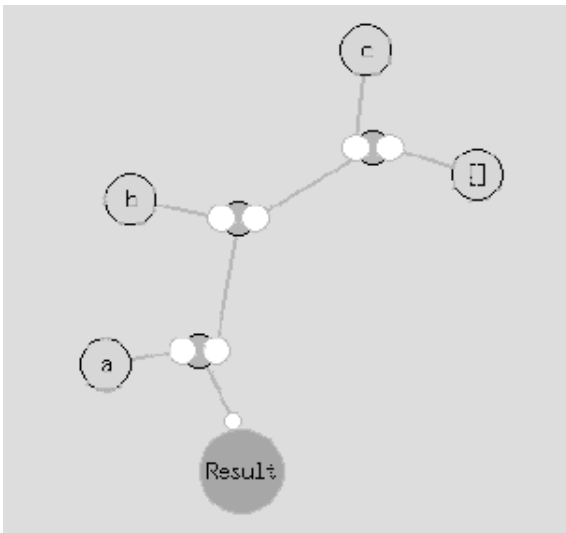
これを繰り返して、以下の図のように結線する。



append の左側の入力タブに接続されている部分が [a,b] というリストを示しており、右側に接続されている部分が [c] というリストを示している。

いよいよ、プログラムを実行させる。各アイコンが正しく接続されていることを確認したら、ウィンドウの上の方にあるプログラム実行アイコンを押すと実行が開始される。

しばらく画面内をアイコンが動きまわったあと、以下のような状態になってプログラムの実行が停止する。



Result に接続されている構造は, [a,b,c] というリストを示しており, これが [a,b] と [c] を append した結果となっている.

結果に満足したら, 右上のviewPP終了アイコンを押して viewPPを終了させる.

終了後, 念のためにバックグラウンドで実行させていた klitcld が 終了していることを

```
% jobs
```

によって確認し, プロセスが残っていたら kill コマンドによって確実に消去しておくこと.

---

*Copyright (C) 1997 Jun Nagumo*