平成 13 年度

筑波大学第三学群情報学類

卒業研究論文

題目:3 次元ビジュアルプログラムの表現と編集方法に関する研究

主専攻	情報科学
著者名	岡 村 寿 幸
指導教員	電子・情報工学系 田中二郎

要旨

我々は、並列論理型言語 KL1 を 3 次元空間と 3 次元オブジェクトを用いて視覚化する、3 次元ビジュアルプログラミングシステム 3D-PP の研究を進めている。

従来の 3D-PP では、引数の表現法に曖昧な点が存在し、プログラムを読む際に誤解を招く場合があった。また、ゴールと節の対応関係を表す表現が不足していたため、大規模なプログラムの可読性が低下するという問題が存在した。

本研究では、これらの問題点を解決する手法として、1) 引数を表現するオブジェクトを付加し、引数の対応関係を明確にする手法、および、2) ゴールと節の対応関係を明確にするための入れ子表示を用いた手法を提案する。また、提案手法を 3D-PP に実装した。実装したシステムでは、プログラムを分かりやすく視覚化することができ、プログラマはプログラムの編集を容易に行えるようになった。

目次

第 1 章

はじめに

近年の計算機の性能向上に伴い、画像、図形を扱うことが容易になったため、ビジュアルプログラミングシステムの研究が数多く行われている [1, 5, 4]。ビジュアルプログラミングシステムは図形、画像等の視覚的な情報を用いてプログラムを表現する手法であり、プログラムの要素間の関係、処理の流れ等をテキストによるプログラム記述よりも分かりやすく表現できるという利点がある。

多くのビジュアルプログラミングシステムでは、２次元空間と２次元図形を用いてプログラムの視覚化を行っている [5]。一方、我々が研究を進めている、３次元ビジュアルプログラミングシステム 3D-PP[2] は、大規模プログラミングへの対応、リアリティの向上、レイアウト自由度の向上の実現を目的として、３次元空間と３次元オブジェクトを用いてプログラムの視覚化を行っている。 3D-PP では、３次元オブジェクト同士をエッジで結線することによって作成する、３次元グラフを用いて、プログラムを表現する。

しかし、 3D-PP のプログラム表現は、引数の入出力を持つグラフのエッジの区別ができず、エッジでつながっているオブジェクト同士がどのような関係であるのかが分かりにくいと言う問題点がある。また、ゴールと節の関係が分かりにくく、プログラムが大規模になると、対応関係を把握することが困難になるという問題と、節の定義を表すグラフが大規模になり、画面表示空間を圧迫するという問題がある。

本研究では、 3D-PP の、引数の区別ができないという問題点を解決するために、引数を間違えずに区別できる様にするための引数オブジェクトの追加の手法を提案する。また、ゴールと節の関係を明確にし、空間を有効に利用するための入れ子表示の２つの手法を用いたプログラム表現を提案する。そして、提案するプログラム表現手法を用いるシステムを実装し、表現手法に適合するプログラム編集機能を考案し、システムへの実装を行った。

本論文の構成は以下の様になっている。まず、 2章で 3D-PP の扱うプログラミング言語である、並列論理型言語 KL1[3] の紹介をし、 3D-PP の従来のプログラム表現とその問題点について述べる。次に、 3章では、従来のプログラム表現の問題点を解決するためのプログラム表現を提案する。その後、 4章では、提案するプログラム表現に沿って、プログラム編集を行うための操作方法についての解説を行い、 5章では、本システムの実装について述べる。 6章で関連研究について述べ、 7章で結論を述べる。

第 2 章

従来のプログラム表現と問題点

2.1 並列論理型言語 KL1

並列論理型言語 KL1[3] は並列論理型言語 Guarded Horn Clauses[7] に基づいて設計された、並列実行に適したプログラム言語であり、論理型言語としての特徴を備えている。論理型言語のプログラムは、述語論理の公理を記述することによって表され、実行は、述語論理をルールにしたがって書き換える簡約 (リダクション) によって行われる。

KL1 のプログラムは図 2.1のように記述される。以下に KL1 プログラムを構成する要素の説明を行う。

2.1.1 ゴール

ゴールは実行時の述語の呼び出しを表すものであり、いくつかの引数を持っている。ゴールは C 言語における関数のようなもので、与える引数によって異なった動作をし、また、引数を介して処理結果を返すことができる。

図 2.1の例では記号 “:-” の右側に現れる “name(X,Y,...)” という形をした記述がゴールである。ゴールは述語名と引数の数によって区別され、述語名が同じでも引数の数が異なれば別のゴールとして扱われる。

```
append(List1,List2,Out) :- List1 = []    | Out = List2.
append(List1,List2,Out) :- List1 = [H|T] |
                           Out = [H|O2], append(T,List2,O2).
```

図 2.1: append の定義

2.1.2 節

節は、ゴールのリダクションのルールを記述する役割をもっており、KL1 プログラムの基本単位である。節は、

```
name(Arg1,Arg2,...) :- guard | body.
```

のような形をしていて各部分には次のような意味がある。

ヘッド

冒頭の ":-" までの部分をヘッドと呼ぶ、この節の定義がどの述語に関わるかを表し、この節での引数の数と引数の呼び名を示している。"name" は述語名であり、"Arg1", "Arg2" は個々の引数の名前である。引数は "," の記号で区切る。

ガード

":-" から "|" までの部分はガードと呼ばれ、ゴールの書き換え時にその節を選んで良いかの条件となる。ガードにはユニフィケーションの他に、比較演算等の決まった種類のゴールのみが書ける。

ボディ

"|" から "." まではボディと呼ばれ、節が選ばれたときに何をするかを定義する部分である。ボディにはユニフィケーションの他に、他のゴール、あるいは自分自身を呼び出すゴールが書ける。

2.1.3 ユニフィケーション

KL1 プログラムにおいて、"=" 記号の両辺に値を書いた形をしたものと、節に存在する変数と同名の変数が、ガードまたはボディに現れた形をしたものを、ユニフィケーションと呼ぶ。ユニフィケーションはどのような場合でも、対応する２つの値が等しいという意味を表している。また、ユニフィケーションによって、２つの値を等しいものにすることをユニファイと呼ぶ。ただし、"=" 記号によるユニフィケーションは、ガード部に出てきたときと、ボディ部に出てきたときで意味が異なる。ガード部でのユニフィケーションは両辺が等しいという条件を表し、ボディ部でのユニフィケーションは両辺の値を等しいものにするという意味を持つ。また、節に存在する変数と同名の変数が、ガードまたはボディに現れた形のユニフィケーションでは、同名の変数は全て同じ値であるという意味を持つ。

2.1.4 データと論理変数

KL1 のデータには内部構造を持たず、その値だけに意味があるアトムと、構造を持ったデータがある。

アトムには文字列アトム、記号アトム、数値アトムがあり、文字列アトムは任意の文字列からなる識別子である。記号アトムは任意の文字列、または記号からなる識別子であり、演算記号等に用いられる。数値アトムは数字からなり、整数値を表す。

構造を持ったデータの代表的なものにはファンクタとコンスがある。ファンクタは

```
name(a,b,...)
```

という形をしていて、引数を持つことができる。引数はどのようなデータでも良く、また、入れ子構造をしていても良い。下の式はファンクタの例である。

```
f(g(a,b), h(c))
```

コンスは２つの引数を持つ構造データであり、

```
[Car|Cdr]
```

という形をしている。“|” は引数の区切りである。

コンスそれ自体は、引数を２つだけ持つ構造データであるが、コンスを組み合わせて、リストのような構造データをつくることができる。リストは、Car(Head) を表す第１引数とリストを構成する要素とをユニフィケーションし、Cdr(Tail) を表す第２引数と別のコンスの参照、またはリストの終端を表すアトム “[]” とをユニフィケーションすることで構築される構造データである。終端を表すアトムは “nil” と記述される場合もある。

コンスはファンクタの特殊な例と考えることもできるが、頻繁に使用される構造データであるので、本論文ではファンクタとコンスをわけて扱う。

論理型言語における変数は、論理変数とも呼ばれ、手続き型言語で言う変数とは異なり、値が不定である、値が決定している、という２通りの状態のみが存在し、一度値が決定すると二度と変化しない (単一代入)。

また、KL1 では変数に型はない。よって変数の型宣言は行わない。

2.1.5 整数演算、比較演算

KL1 では整数演算、比較演算は、始めから用意されている組み込み述語を用いて行うが、通常は “X operator Y” という形のマクロ記法を用いる。本論文では KL1 における整数演算、比較演算をまとめてオペレータと呼ぶことにし、“X operator Y” という形のみを扱うこととする。

オペレータのoperatorの部分には、整数演算では “+”,“-”,“*”,“/”,“mod” の計算に用いる記号または文字列が入り、比較演算では、“<”,“>”,“=<”,“>=”,“=:=”, “=\=” の比較に用いる記号が入る。ここで、“=:=” 記号は相等比較、“=\=” 記号は不等比較の記号である。

整数演算はガード、またはボディに現れ、整数の計算を行うことができ、“:=” という記号を用いて

```
Z := X operator Y
```

と記述することで計算結果を値として得ることができる。ここで、“:=” の代わりに “=” 記号を用いた表記では “operator(X,Y)” というファンクタとユニフィケーションされるが、この表記は本論文では扱わない。

比較演算はガードにのみ現れ、条件として用いることができる。比較演算は演算の結果を返すことはないため、“:=” の記号を用いて計算結果を値として得ることはできない。

2.1.6　実行機構

KL1 プログラムの実行は、節が定める書き換えルールに従ってゴールを書き換えるリダクションによって進められる。ゴールはこれ以上書き換えることのできないゴール “true” になった時点でリダクションを終了する。プログラムの実行はすべてのゴールが “true” に書き換えられた時点で終了する。

2.2　従来のプログラム表現

3D-PP は KL1 プログラムのゴール、アトム、コンス等の要素を 3 次元オブジェクトで表し、 3 次元オブジェクト同士をエッジで結線した 3 次元グラフを用いて KL1 プログラムの視覚化を行うビジュアルプログラミングシステムである。

本節では、 3D-PP における従来のプログラム表現手法と編集方法の解説を行う

2.2.1　３次元オブジェクト

３次元空間上に配置される 3 次元オブジェクトは、それぞれがゴール、ファンクタ、アトム等の KL1 プログラムの要素を表している。図 2.2はプログラム要素を表す 3 次元オブジェクトの一覧である。左から、緑色の円柱がゴール、紫色の立方体が組み込み述語、肌色の球形が数値アトム、青色の円錐形が文字列アトム、赤色の四角錐が結果の出力をそれぞれ表している。

それぞれのオブジェクトには固有のラベルが与えられており、ラベルはオブジェクトの手前に表示することができる。ラベルはゴールならゴール名、アトムなら値、オペレータなら演算記号といった、そのプログラム要素固有の情報を表している

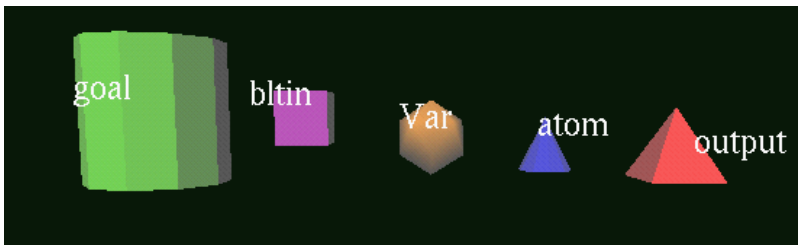


図 2.2: 3 次元オブジェクト

2.2.2　エッジとグラフ

3D-PP のプログラム表現では各オブジェクトをエッジで結線しグラフを作成する (図 2.3)。オブジェクトを結ぶエッジはプログラム要素同士の関係を表していて、プログラムの動作を決定する。 KL1 プログラムとエッジの対応関係としては、ゴール、ファンクタ等の引数とのユニフィケーション、データとのユニフィケーションが挙げられる。

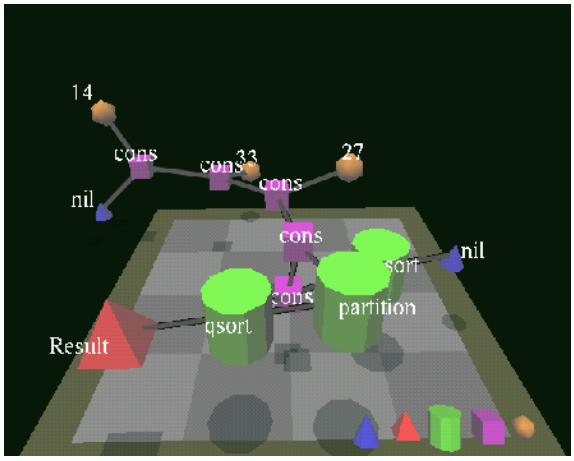


図 2.3: グラフの例

2.3 従来のプログラム表現の問題点

2.3.1 ユニフィケーション表現の問題

従来の 3D-PP のプログラム表現では、オブジェクト同士のエッジ結線はユニフィケーションを表しており、それはデータ同士のユニフィケーションを表す場合と、データと引数との間のユニフィケーションを表す場合とに分けることができる。

エッジがデータ同士のユニフィケーションを表す場合は、両端にあるものは互いに同じものであると言うことを表している。そのため、仮に一つのデータに複数のユニフィケーションがあったとしても、それらはすべて同じものであるので、つながっている順番や数は問題にはならない。 (図 2.4)。

しかし、ゴール等が複数の引数を持っている場合のデータと引数との間のユニフィケーションでは問題が起きる。データと引数との間のユニフィケーションの場合、グラフは図 2.5に示すように データ同士のユニフィケーションの例 (図 2.4) と似た形となる。ところが、データのオブジェクトとゴールのオブジェクトの結線は、データのユニフィケーションのときと違い、データとゴールは同じものであるという意味ではない。データのオブジェクトとゴールのオブジェクトの結線は、実際には、データと、ゴールの持っている引数のうちの一つとの間にユニフィケーションがあり、データと引数は同じものであるということを表している。

ゴールは与える引数によって動作を変えるため、ゴールの引数が何とユニファイされているかは、プログラムの動作を決める重要な要素である。従って、複数の引数がある場合には、どの引数とのユニフィケーションであるかは明確に区別される必要がある。

しかし、従来のプログラム表現では、図 2.5の様になり、プログラマは、どのデータがどの引数と対応しているのかを区別することができない。引数が区別できないことにより、プログラマは引数の対応を見失う可能性がでてくる。引数の対応が分からない場合、プログラマはプログラムを理解するのに多大な労力を必要とするようにな

る。その結果として、プログラムの処理の流れを見失ったり、誤解したりしやすくなる。

図 2.5の例で言うと、ゴール “append” は３つの引数を持っている。ゴール “append” は第１引数、第２引数に与えられたリストを結合し、第３引数を介して、結合されたリストを返すという処理を行うが、図 2.5の従来のプログラム表現では、引数がどのデータとユニファイしているか区別できず、処理が分かりにくくなってしまっている。

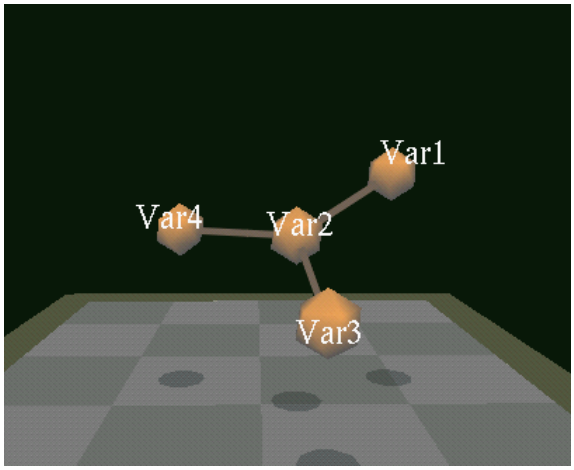


図 2.4: データ同士のユニフィケーションの例

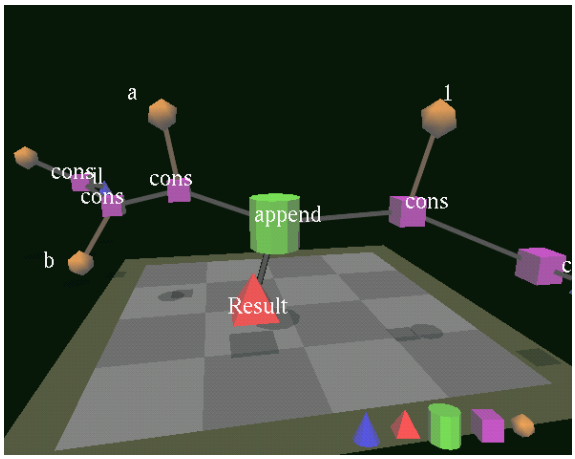


図 2.5: 誤解を招く例

2.3.2　ゴールと節の対応関係表現の問題

KL1 プログラムは、ゴールに与えられた引数等の条件によって処理を分岐させる方法として、同名、同引数でガード部の条件が異なる節を複数作成するという方法をとる。そのため同名、同引数の節が複数存在する場合がほとんどである。

しかし、従来のプログラム表現では、ゴールと節の対応関係を記述するための方法が存在しないため、対応関係を表すことができず、対応するゴールと節のオブジェクトもお互い全く関連のない位置に表示されることになる。このため、プログラマがゴールと節の対応関係を知る方法は、ラベルと引数の数をみて判断する方法のみとなる。

すべてのゴールと節について対応関係を判断することは、ゴールの数が増えてくると、プログラマにとって、非常に大きな負担となり、プログラムの可読性を悪化させる要因となる。

2.3.3 オブジェクト表示空間の問題

従来の表現では、節がどのような処理を行うかを定義する場合、節のオブジェクトの外に定義を表すオブジェクトの組がむきだしの状態で表示せざるを得なかった。よって、節が増えてくると、表示されるオブジェクトが非常に多くなり、オブジェクトを表示するの場所が足りなくなるため、オブジェクト同士の重なり合い、エッジの交差等が起こり、可読性が低下する。

従来の 3D-PP では、オブジェクトのレイアウトを自動的にを行うことで、オブジェクト同士の重なり合いを軽減する方法がとられたが、表示されるオブジェクトが減るわけではないため、オブジェクトが多くなったときの可読性の低下を防ぐには十分と言えない (図 2.6)。

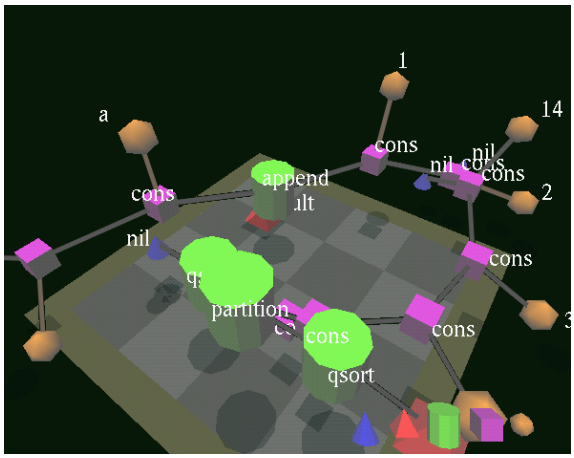


図 2.6: オブジェクトが非常に多い例

第 3 章

プログラム表現の提案

3.1　KL1 プログラムと 3 次元オブジェクトの対応

提案するプログラム表現では、KL1 プログラムを、図 3.1に示す 3 次元オブジェクトとエッジによって表現する。図 3.1の各オブジェクトは、緑色の円柱形がゴール、青色の円柱形が節、黄色の逆さになった円錐形がオペレータ、白色の高さの低い円柱形がファンクタ、紫色の直方体がコンス、水色の球形がアトムをそれぞれ表している。また、オブジェクト同士を結ぶエッジは灰色の線であり、エッジはユニフィケーションを表している。

プログラム要素を表すオブジェクトには、文字列からなるラベルが付加される。ラベルはオブジェクトの手前に表示され、ゴールと節では述語名が、オペレータでは “+” や “<” 等の整数演算、比較演算を表す記号が、アトムではそのアトムの表す値がつけられる。図 3.1において、一部のオブジェクトに付加されている円錐形の小さなオブジェクトは引数を表す引数オブジェクトである。

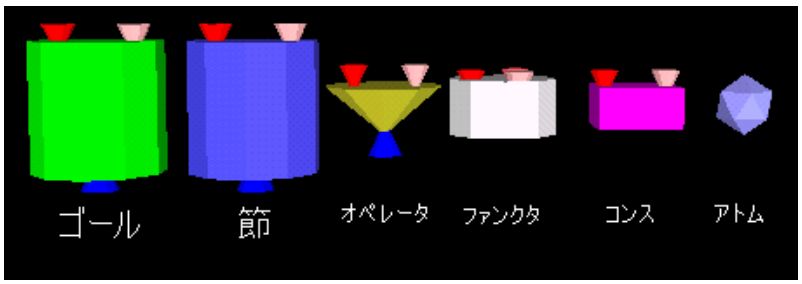


図 3.1: KL1 プログラム要素と 3 次元オブジェクトの対応

3.2　引数の表現

3.2.1　引数を表す引数オブジェクトの付加

引数を持っている、ゴール、ファンクタ、コンス、オペレータを表す各オブジェクトに対して、引数の数に等しい数の小さな円錐形のオブジェクトを付加し、引数を表す (図 3.2)。以下、引数を表す小さな円錐形のオブジェクトを引数オブジェクトと呼ぶこととする。

引数オブジェクトを介して、引数として与えるオブジェクトを結線することによって、オブジェクトと引数との対応関係を明確にする。引数オブジェクトと他のオブジェクトのエッジ結線は、引数とのユニフィケーションを表している。

3.2.2　引数オブジェクトによる引数モードの明示

KL1 プログラムにおいて、それぞれのゴールの引数が入出力のどちらとして使われるかは、陽には記述しないが、プログラマは通常、引数の使い方についてあらかじめイメージを持っている [1]。よって、ゴールの引数オブジェクトの入出力モードをプログラマが明示できることは、プログラマの理解を助ける働きがあると考える。引数オブジェクトを付加する際に、引数の入出力モードを明示するための手法として次のような方法を提案する。

- 入力の引数オブジェクトは引数を持つオブジェクトの上の位置にある面に配置し、引数オブジェクトの色は赤とする
- 出力の引数オブジェクトは引数を持つオブジェクトの下の位置にある面に配置し、引数オブジェクトの色は青とする

引数オブジェクトの位置は図 3.2の様になる。引数オブジェクトの位置を別けることによってプログラマは引数の入出力モードを見分けることができる。また、色の違いは入出力モードの区別だけでなく、後述する引数同士の区別、引数の順番の明確化に重要な役割を果たす。

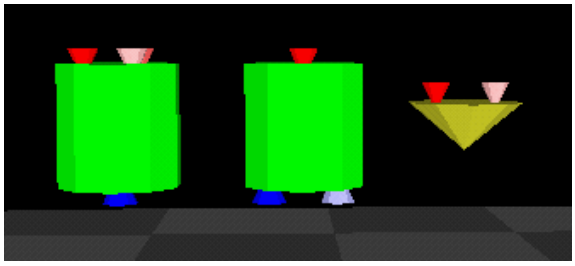


図 3.2: 引数オブジェクトの位置の違い

3.2.3　引数同士の区別と順番の明確化

ゴール、オペレータ、コンス、ファンクタの引数は、引数の順番が大きな意味を持つ場合がある。例えば、オペレータ “-” は減算数と被減算数の２つの引数を持っている。もし、この２つの引数が取り違えられるとオペレータ “-” の計算結果が違ってしまい、プログラムの処理に重大な影響を及ぼす。

我々は、ゴール、オペレータ、コンス、ファンクタの引数は、引数同士が明確に区別される必要があると考える。そこで、入力引数同士、出力引数同士で区別でき、順番を明確するための方法として、次の様な方法を提案する。

1. ゴール等、入力引数または出力引数が複数になる場合、引数を持つ側のオブジェクトは対応する引数オブジェクトを並べるための面が存在する

2. 引数オブジェクトが一つの面に複数並ぶ場合、引数オブジェクトは引数を持つ側のオブジェクトの面上に時計回りに引数の順番通りに並ぶ

3. 同一面に並んだ引数オブジェクトの一番濃い色 (入力なら濃い赤、出力なら濃い青) の引数オブジェクトが第１引数であり、第２引数、第３引数 · · · となるにしたがって薄い色となる

例を挙げると、図 3.3の様になる。ゴールの持つ３つの入力引数の引数オブジェクトが時計回りの順番で並び、第１引数を表す引数オブジェクトがもっとも濃い赤をしている。

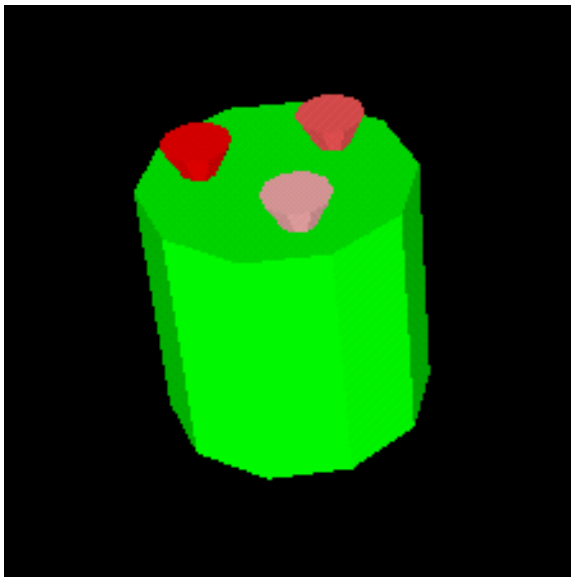


図 3.3: ３つの入力引数を持つゴールの場合

3.3　ゴールと節の表現

3D-PP においてプログラマは、いくつかの節を定義し、それらをいくつかまとめてゴールを定義する。本節では、ゴールと節の関係と節の定義を表す手法として、オブジェクトの内部に、それと関連を持つプログラムを表すオブジェクトを配置する入れ子表示を提案する。

3.3.1　ゴールと節の入れ子

ゴールを表すオブジェクトは緑色の円柱形をしており、緑色の円柱の内部にそのゴールで定義されている節の数と同数の節のオブジェクトが入っている。これらの節のオブジェクトは後述する節を表すオブジェクトと同じもので、互いに重なり合わないようにゴールのオブジェクトの中に配置されている。

プログラマは内部にある節のオブジェクトより、そのゴールに存在する節の数を知ることができる (図 3.4)。

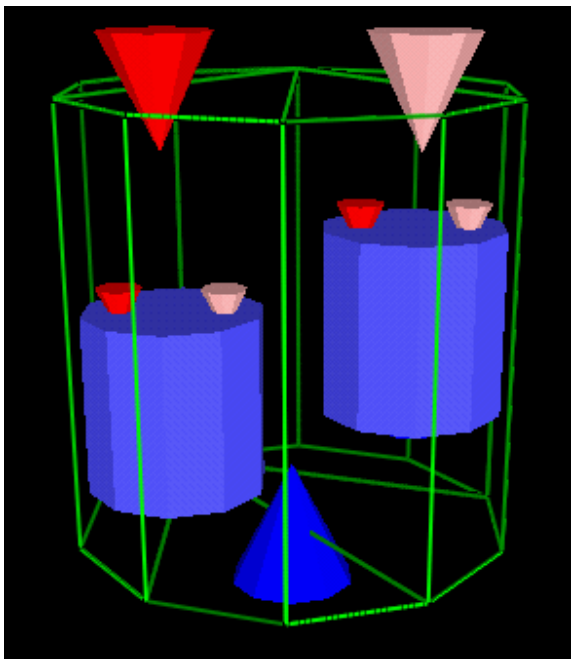


図 3.4: ゴールのオブジェクト内部に配置された節のオブジェクト

3.3.2　節とガード部、ボディ部の入れ子

節を表すオブジェクトの内部には、その節のガードとボディに相当するプログラムを表すオブジェクトの組が入っている。節を半透明表示、ワイヤフレーム表示、または、拡大表示したときに、内部の定義を見ることができる (図 3.5)。

節のオブジェクトの内部は、円盤状のオブジェクトで上下に分けられており、それぞれ上の部分はガード部、下の部分はボディ部の定義が入っている。円盤状のオブジェクトは KL1 プログラムにおける節 “|” 記号に相当する。

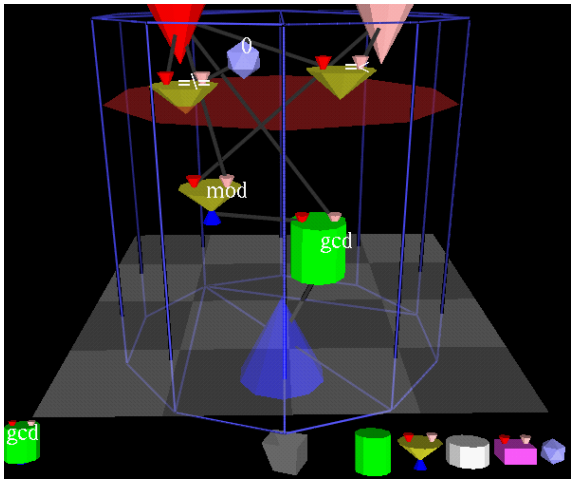


図 3.5: 拡大表示状態の節

3.4 データとオペレータの表現

3.4.1 アトム

アトムは水色の球形のオブジェクトで表現する。アトムは引数を持たないため、従来のプログラム表現と異なる部分はない。すなわち、アトムは値のみを持っていて、アトムの値はラベルを用いて表される。

3.4.2 ファンクタ、コンス

ファンクタは白色の円柱形のオブジェクトで表現し、コンスは紫色の直方体で表現する。

ファンクタ、コンスは引数を持っているが引数は何らかの値を返す目的で使用されることはないため、引数の入出力モードを区別する必要はない。しかし、ファンクタ、コンスは引数が変われればデータ内容が変わってしまうため、引数の対応関係は明確に区別する必要がある。そこで、引数オブジェクトを用いて、ファンクタ、コンスのための次のような表現方法を提案する。

1. 引数オブジェクトはオブジェクトの上の位置に配置する赤い色の円錐 (ゴールの場合では入力を表す) のみを用いる
2. 引数同士の区別は、ゴールの場合と同様に色の濃淡によって行う
3. ファンクタの引数の数に制限はないが、コンスの引数の数は 2 で固定する
4. データの構築はゴールに引数を設定するときと同じく、他のデータを引数オブジェクトと結線しユニフィケーションを表すことで行う

5. ファンクタ、コンスによって構築されるデータを参照する場合は、ファンクタ、コンスのオブジェクトにエッジ結線を行う

6. 構造データの終端は”nil” というラベルを持ったアトムで表す

提案するプログラム表現において、コンスを用いてリストを表す場合は、図 3.6のような、延々とコンスがつながった形となる。リストの終端のコンスは第 2 引数に “nil” のラベルを持つアトムが結線されている。

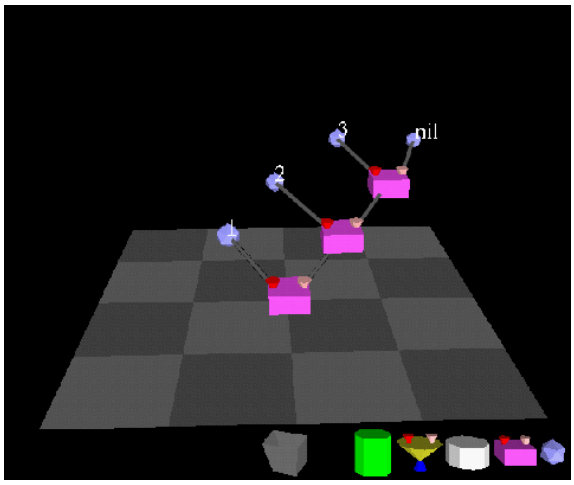


図 3.6: リストのプログラム表現

3.4.3 オペレータ

オペレータは黄色の逆さまになった円錐形で表現する (図 3.7)。オペレータは演算数、被演算数という 2 つの入力引数をもった述語であると考えることができる。よって、引数オブジェクトを用いて、次のような表現方法とる。

- 入力引数の数は常に 2 で固定であり、入力引数オブジェクトはオペレータのオブジェクトの底面に配置する
- 被演算数 (第 1 引数) は濃い赤の入力引数オブジェクトで表し、演算数 (第 2 引数) は薄い赤の入力引数オブジェクトで表す
- 演算の種別はラベルの違いで表す。ラベルには演算の種別を表す記号、文字列を用い、言語上扱わない記号、文字列をラベルをつけることはできない

2.1.5節で説明したように、オペレータには整数演算と比較演算があり、整数演算は“:=”の記号で計算結果を得ることができるが、比較演算では結果を返すことはないという違いがある。提案手法では、この違いを出力引数オブジェクトを用いて次のように表現する。

- 整数演算では、出力引数オブジェクトを一つ付加する
- 比較演算では、出力引数オブジェクトは付加しない

整数演算オペレータの出力引数オブジェクトに対する結線は“:=”を用いて計算結果を得ることに相当する。比較演算は結果を返すことはないため、出力引数オブジェクトは付加しない。

編集操作により、オペレータが整数演算から比較演算に変化した場合は出力引数オブジェクトが一つ減少し、0個になる。比較演算から整数演算に変化した場合は出力引数オブジェクトが一つ増加し、1個になる。

オペレータ“mod” オペレータ“>=”

図 3.7: オペレータのプログラム表現

3.5 提案手法を用いたプログラムの例

本章で提案した、ゴールと節の入れ子表示、引数オブジェクトの付加という２つの手法を用いて、図 3.8に挙げる最大公約数 (GCD) を求める KL1 プログラムの視覚化を行った例を図 3.9, 3.10に示す。

```
main :- gcd(10,4,Z).

gcd(X,Y,Out) :- X =:= 0          | Out = Y.
gcd(X,Y,Out) :- Y =:= 0          | Out = X.
gcd(X,Y,Out) :- X =\= 0, X =< Y | gcd(X1,X,Out),X1 := Y mod X.
gcd(X,Y,Out) :- Y =\= 0, X >= Y | gcd(Y,Y1,Out),Y1 := X mod Y.
```

図 3.8: 最大公約数を求める KL1 プログラム例

3.5.1 ゴール “gcd” と節

図 3.9では、引数 “X”,“Y”、 “Out” の３つの引数を持つゴール “gcd” が定義されている。ここでは、引数 “X”,“Y” は入力モードの引数として扱われており、 “X” を表す引数オブジェクトは第１引数であるため濃い赤色、 “Y” を表す引数オブジェクトは第２引数であるため薄い赤色をしている。引数 “Out” は出力モードの引数として扱われており、引数オブジェクトの色は青となっている。

図ではゴール “gcd” のオブジェクトの各引数オブジェクトに、それぞれ、 10,4,Z の値を持つアトムのオブジェクトが結線されている。これは 節 “main” の定義にある ゴールの呼び出し “gcd(10,4,Z)” に相当する。また、ゴールのオブジェクトの内部にある節のオブジェクトの数から、ゴール “gcd” には４つの節が存在していることが分かる。

以後この４つの節を図 3.8の定義の上から順に、節１、節２、節３、節４と呼ぶことにする。

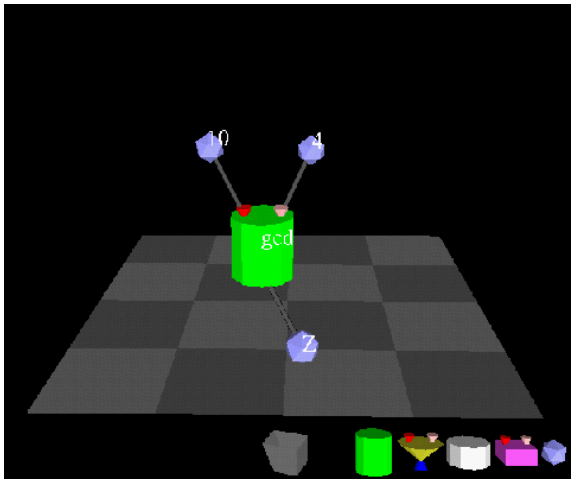


図 3.9: 図 3.8のプログラムを視覚化した例

3.5.2 各節の定義の表現

図 3.10の各図は、図 3.9にある 4 つの節のオブジェクトをそれぞれ拡大表示状態にした場合の図である。以下では各節のプログラム表現の解説を行う。

節 1 の定義の表現

図 3.10(a) は節 1 を表すプログラム表現である。節のオブジェクトのガード部にはオペレータ “=:=” を表すオブジェクトがある。オペレータの第 1 引数オブジェクトに節の第 1 引数 “X” のオブジェクトが、第 2 引数オブジェクトにアトム “0” のオブジェクトが結線されている。この表現は “X =:= 0” に相当し、 “X” が “0” に等しいという条件を表している。

節 1 のボディにある “Out = Y” は図 3.10(a) では引数 “Y” と引数 “Out” の結線で表しており、 “Out” が “Y” とのユニフィケーションによって “Y” と等しくなることを表している。

節 2 の定義の表現

図 3.10(b) は節 2 を表すプログラム表現である。節 2 は節 1 の定義の “X” と “Y” をちょうど入れ替えた形をしている。よってプログラム表現も節 1 の表現の “X” と結線されていたものが “Y” と、 “Y” と結線されていたものが “X” と結線された表現となっている。

節 3 の定義の表現

図 3.10(c) は節 3 を表すプログラム表現である。節のオブジェクトのガード部にオペレータ “=\=”、 “=<”、アトム “0” を表すオブジェクトがあり、 それぞれ “X =\= 0”、

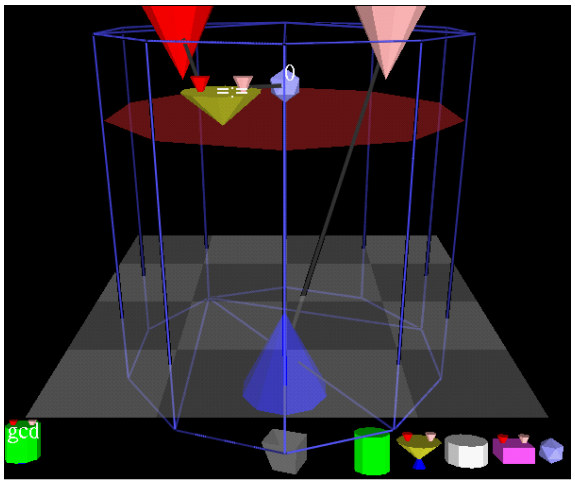
“X =< Y” を表すように結線されている。

ボディ部にはオペレータ “mod” のオブジェクトと、ゴール “gcd” のオブジェクトがある。オペレータ “mod” の各入力引数オブジェクトに “Y”、 “X” を結線することで “Y mod X” を表しており、出力引数オブジェクトと結線することで “:=” を用いて演算結果を得ることを表している。また、ボディ部に現れるゴール “gcd” は節 “gcd” と同名、かつ同じ引数を持っている。これはゴール “gcd” を再起的に呼び出すことを表している。

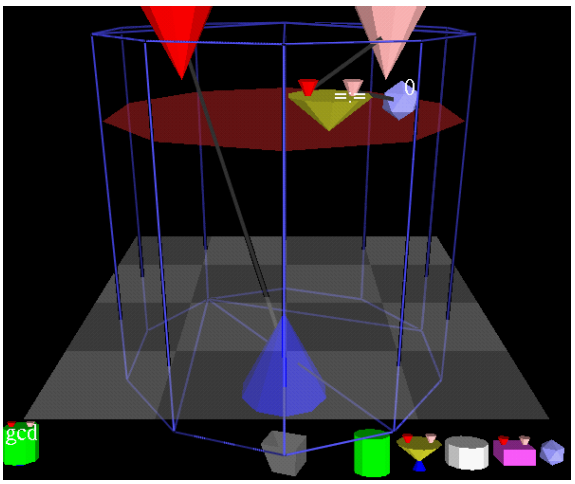
演算結果をゴール “gcd” の第 1 引数に与える方法として、図 3.8のテキストでは変数 “X1” を “X1 := Y mod X”、 “gcd(X1,X,Out)” のように共有し、ユニフィケーションを表すことで行っている。提案手法では変数を共有する形のユニフィケーションもエッジ結線で表すため、オペレータの出力引数オブジェクトがゴール “gcd” のオブジェクトの第 1 引数オブジェクトに結線し、これらがユニファイすることを表している。

節 4 の定義の表現

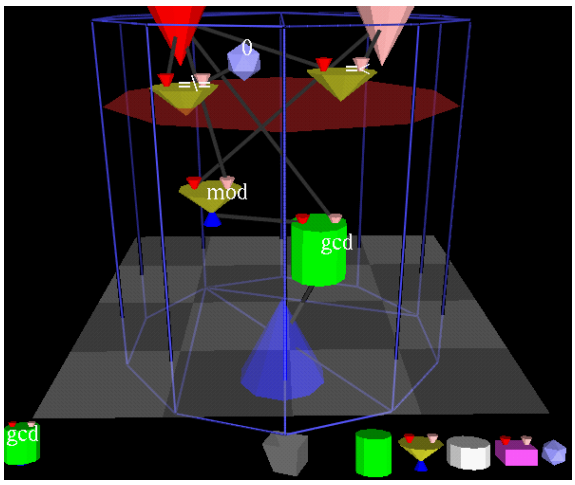
図 3.10(d) は節 4 を表すプログラム表現である。節 4 の定義は節 3 の定義と似た形をしており、提案手法を用いたプログラム表現も節 3 の表現と似たような形となっている。すなわち、ガード部にある 2 つのオペレータ “=\=”,“>=” と節の引数 “X”,“Y” の結線によりガードの条件 “ Y =\= 0”,“X >= Yを表現し、ボディ部のオペレータ “mod”、ゴール “gcd” の各引数と節の引数の結線により、 “X mod Y” の演算結果を引数として与えたゴール “gcd” の再起呼び出しを表現している。



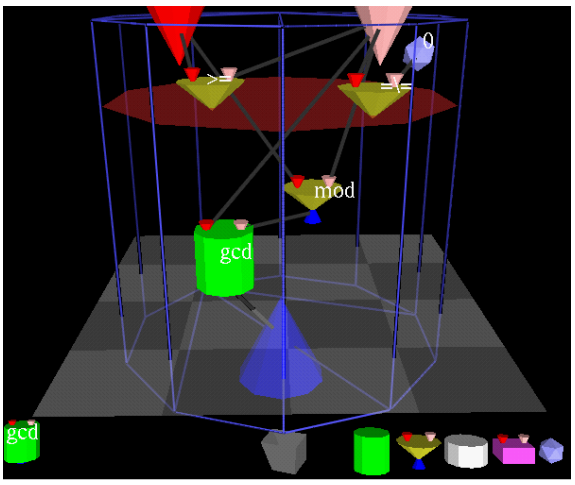
節 1 の定義 (a)



節 2 の定義 (b)



節 3 の定義 ((c)



節 4 の定義 ((d)

図 3.10: 図 3.9のゴールの各節を拡大表示した様子

第 4 章

プログラム編集方法

4.1 3D-PP の基本的な操作方法

3D-PP におけるプログラム編集には、３つのボタンを持つマウスを用いる。オブジェクトはマウスを用いて、直接操作することができる。直接操作とは、オブジェクト上で各ボタンのクリック、ダブルクリック、ドラッグ & ドロップの操作をすることで、あたかもマウスでオブジェクトに触れる、オブジェクトをつまみ上げるといった様に、オブジェクトの選択、移動等の操作が行える操作方法である。

本論文におけるドラッグ & ドロップ操作は、神谷によって提案された、２次元空間上でのドラッグ & ドロップを３次元空間に適する様に拡張した、拡張ドラッグ & ドロップ手法 [13] を用いている。拡張ドラッグ & ドロップ手法では、オブジェクトの移動は画面上のマウスポインタに追随するように行う、ドロップ時はマウスポインタの位置をユーザの視線とし、視線上にあるもっとも近いオブジェクトをドロップ対象とする等、３次元空間上のオブジェクトを２次元的なドラッグ & ドロップ操作で扱うための拡張を行っている。

また、画面上には常に、地面のオブジェクトが存在する。これは大芝が提案した、強化された直接操作手法 [12] に基づいている。強化された直接操作手法では、オブジェクトと共にそれらの形状、位置を把握しやすくするための付加情報を表示する。本システムの場合、地面は、プログラマがオブジェクトの位置と上下関係を把握するための指針として存在する。

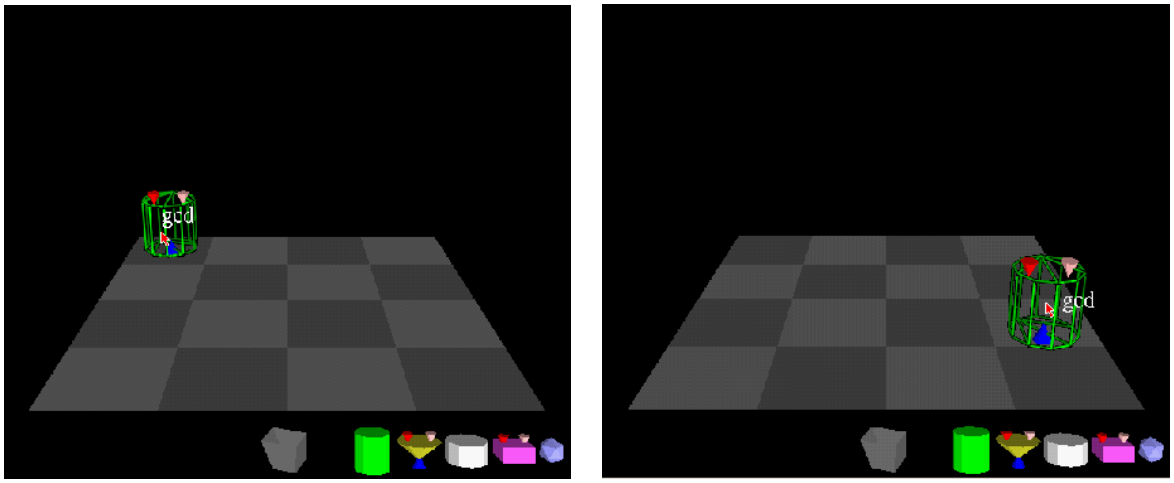
3D-PP の操作の内、すべてのオブジェクトに共通する操作として、３次元空間内の移動、オブジェクト同士のエッジによる結線、オブジェクトの削除、ラベルの変更の各操作があり。オブジェクトに依存しない操作として視点の変更がある。ただし、地面のオブジェクトはプログラマがオブジェクトの位置と上下関係を把握するための指針あるので、プログラマが編集することはできないようになっている。

4.1.1 オブジェクトの生成

3D-PP の編集画面では、画面右下に、３次元アイコンが常に配置されている。オブジェクトは節の場合を除いて、生成しようとしているオブジェクトと同じ形の３次元アイコンを左クリックすることによって生成され、画面中央に出現する。節のオブジェクトの生成方法については後述する。

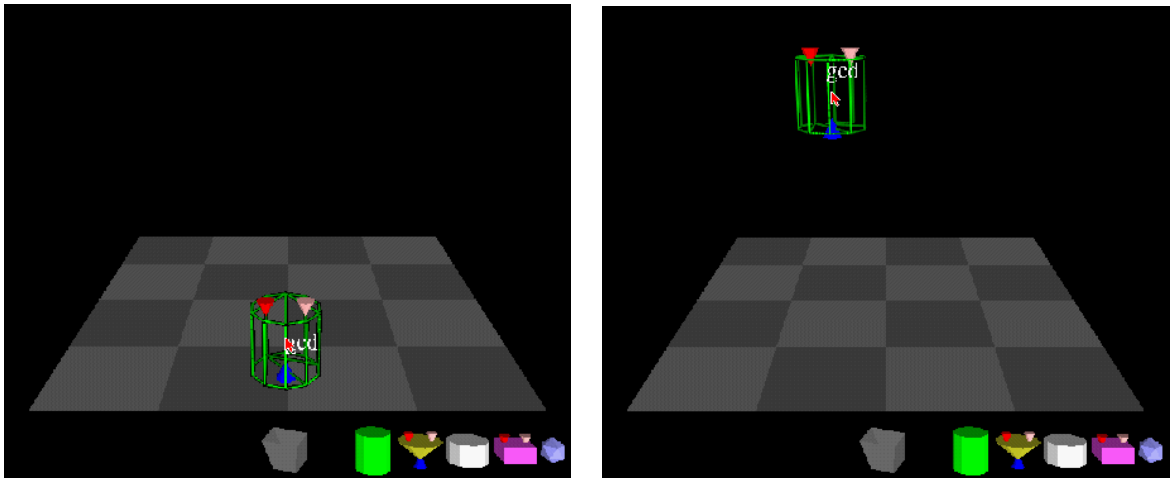
4.1.2　オブジェクトの３次元空間内の移動

３次元空間内に表示されたオブジェクトを左ボタンドラッグ操作、または右ボタンドラッグ操作することによって、オブジェクトの表示位置を変更することができる。左ボタンドラッグ操作では、地面との距離を変えること無く移動し (図 4.1)、右ボタンドラッグ操作では、現在見ている画面に対して平行に移動する (図 4.2)。移動する際、オブジェクトは常にマウスポインタに重なるように移動する。また、移動中のオブジェクトは他のオブジェクトと区別するため、ワイヤフレーム表示となる。



移動操作前 (a)　　移動操作後 (b)

図 4.1: 地面に対して平行なオブジェクト移動



移動操作前 (a)　　移動操作後 (b)

図 4.2: 画面に対して平行なオブジェクト移動

4.1.3　エッジ結線、結線解除

エッジ結線および解除は中ボタンのドラッグ & ドロップ操作で行う。結線する２つのオブジェクトのどちらか一方を始点として中ボタンドラッグ操作し、他方のオブジェクトにドロップする。

２つのオブジェクトの間にエッジが無い場合はエッジ結線が行われ、オブジェクト間にエッジが張られる (図 4.3)。既にエッジが存在する場合はエッジ結線の解除が行われ、張られていたエッジは消滅する。

始点となったオブジェクトと結線できるかどうかは、対象とするオブジェクトにポインタが触れたときの、表示状態の変化で見分けることができる (図 4.4)。結線可能な場合は、対象となるオブジェクトは半透明表示となる。 KL1 プログラム上でユニフィケーションが不可能であり、結線不可能なオブジェクトが対象となっている場合には、表示状態は変化しない。

エッジ結線前 (a)　　エッジ結線後 (b)

図 4.3: エッジ結線

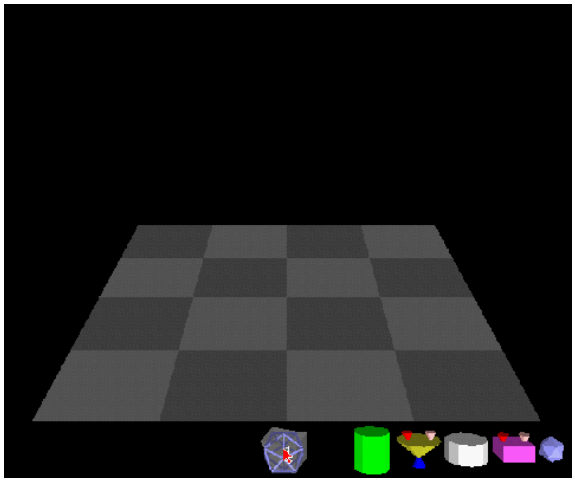
ポインタがオブジェクト上にないとき (a)　　ポインタがオブジェクト上に来たとき (b)

図 4.4: エッジ結線時の表示状態の変化

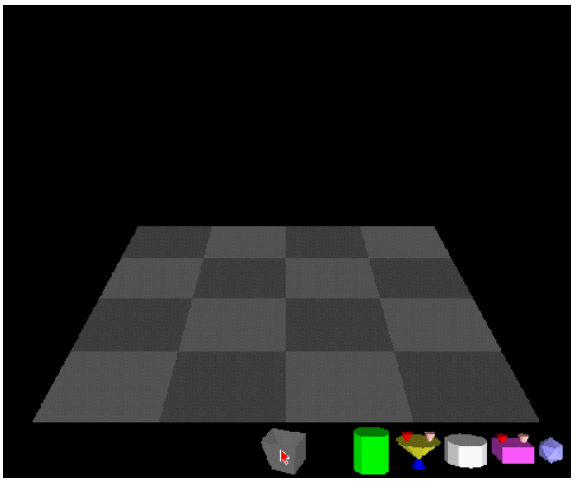
4.1.4　オブジェクトの削除

画面の下の位置には、オブジェクト作成のための３次元アイコンだけでなく、ゴミ箱の形をしたゴミ箱アイコンも存在する。オブジェクトの削除は、削除したいオブジェクトを移動 (4.1.2節) と同様の操作でゴミ箱アイコンの上へ ドラッグ & ドロップする、または、削除したいオブジェクトを始点とした中ボタンドラッグ & ドロップ操作でドロップ対象をゴミ箱アイコンとすることによって行う (図 4.5)。

移動と同様の操作の場合は、ゴミ箱アイコンの上 でない位置でドロップするとオブジェクトはその位置に移動してしまうが、中ボタンドラッグ & ドロップ操作の場合はオブジェクトが移動することはない。



ゴミ箱上にドラッグ (a)



ゴミ箱にドロップ (b)

図 4.5: オブジェクトの削除

4.1.5 ラベル変更

オブジェクト上で、中ボタンダブルクリック操作を行うとオブジェクトのすぐ下の位置に、ラベル入力用の小さなウィンドウが現れる。プログラマはラベル入力用のウィンドウを用いて、オブジェクトに付加するラベルを変更することができる。

ウィンドウ内にはラベルを入力するための部分と、“ok”、“cancel” の 2 つのボタンから構成されている。最初、ラベル入力の部分には現在のラベルの文字列が表示されているが、キーボードから入力を行うことによってこの文字列を書き換えることができる。入力が終了した後、“ok” ボタンを押すとウインドウが閉じ、ラベル変更が終了する。

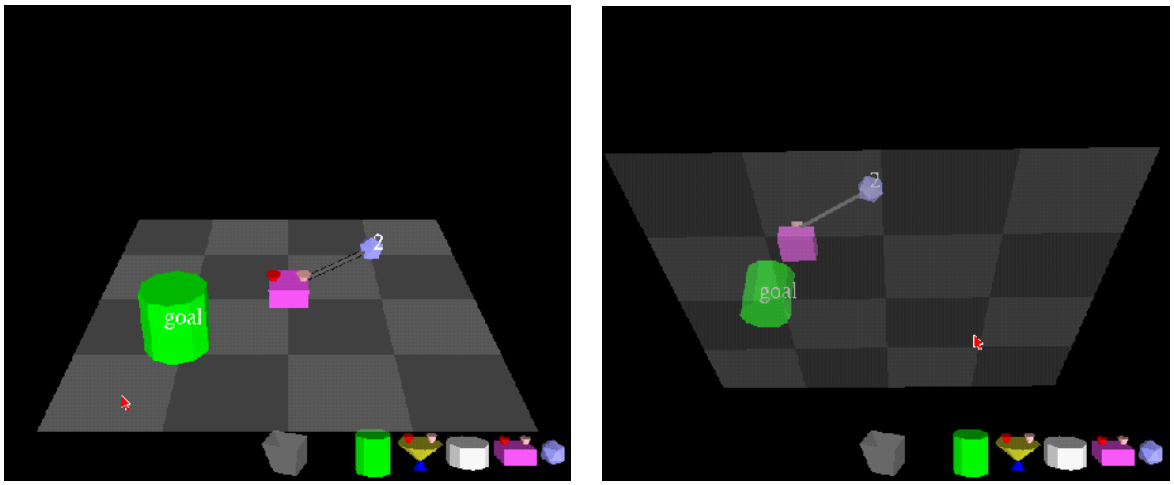
ただし、入力した文字列が、例えばオペレータに “+”、“-”、“*”、“/”、“mod”、“<”、“>”、“=<”、“>=”、“=:=”、“=\=” 以外のラベルをつけようとした場合のように、プログラム上ふさわしくない文字列が入力されていた場合は、もう一度ウインドウが開き、再入力を要求する。

ラベル変更操作は “cancel” ボタンを押すことによって、いつでも中止することができる。

4.1.6　視点の変更

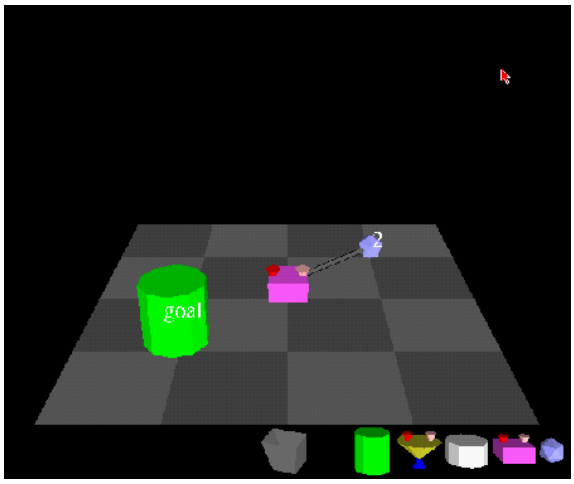
背景または地面に触れている状態で、左ボタンドラッグ操作を行うと画面中央を中心としてオブジェクトが回転するように視点が移動する。ドラッグを開始した位置とドラッグ中のマウスポインタの位置のずれに合わせて視点が変化する (図 4.6)。横方向のずれは横方向の回転となり、上下方向のずれは縦方向の回転となる。

また、背景または地面に触れている状態で、右ボタンドラッグ操作を行うと、視点位置を遠ざける、または近づけるパン・ズーム操作を行うことができる (図 4.7,4.8)。パン・ズーム操作の場合、視点はマウスポインタの上下の動きに合わせて移動する。マウスポインタを上方向に移動した場合、視点は表示オブジェクトから遠ざかる方向に移動し、下方向に移動した場合は近づく方向に移動する。

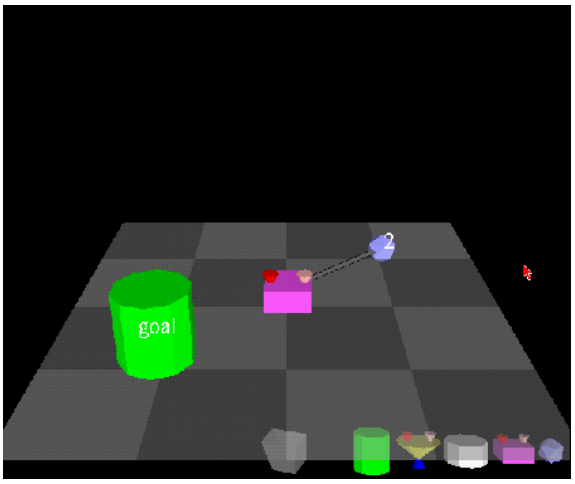


回転操作前 (a)　　回転操作後 (b)

図 4.6: 視点移動 (回転)

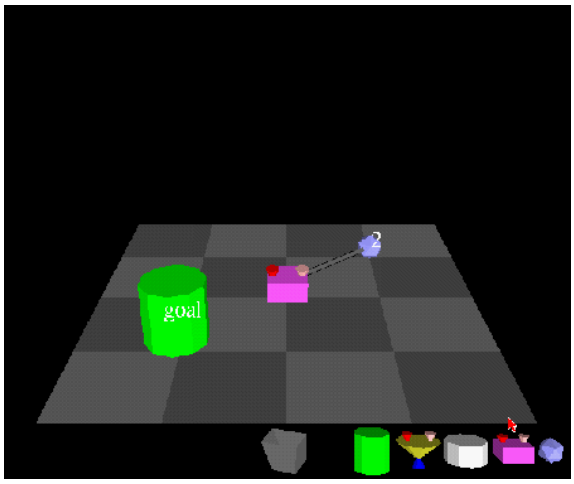


ズーム操作前 (a)

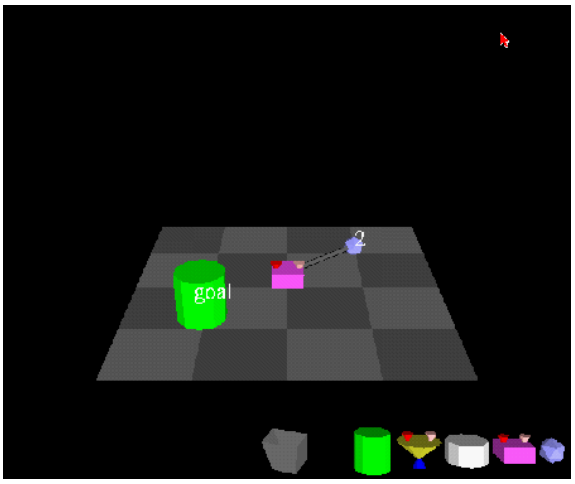


ズーム操作後 (b)

図 4.7: 視点移動 (ズーム)



パン操作前 (a)



パン操作後 (b)

図 4.8: 視点移動 (パン)

4.2　ゴールの編集

4.2.1　ゴールのオブジェクトの生成

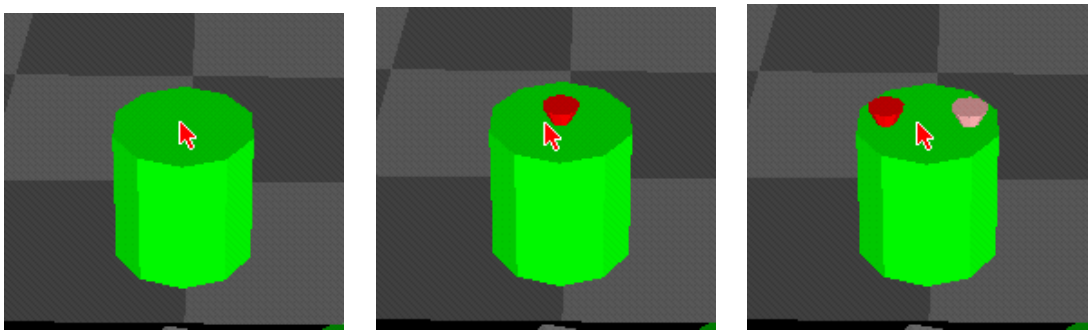
ゴールのオブジェクトは他のオブジェクトと同様に、画面右下の緑の円柱形 (ゴールの形) の 3 次元アイコンを左クリックすることによって生成する。生成されたばかりのゴールは引数の設定されていない状態 (引数オブジェクトが付加されていない状態) で、節も定義されていない状態 (内部に節のオブジェクトが一つもない状態) である 。

4.2.2　引数の数、入出力モードの変更

生成した直後のゴールは引数を一つも持っていない状態である。プログラマは必要に応じてゴールの持つ引数の数と、引数の入出力モードを変更することができる。

引数オブジェクトの追加

引数を持つオブジェクトには、入力引数オブジェクトを配置する面と、出力引数オブジェクトを配置する面が存在する (3.2.2節)。引数オブジェクトを配置する面の上で左ボタンダブルクリック操作を行うと、新しい引数オブジェクトがその面に追加される (図 4.9)。すなわち、入力の引数オブジェクトを追加したい場合は、引数を持つ側のオブジェクトの上の位置にある面を左ボタンダブルクリックし、出力の引数オブジェクトを追加したい場合は、引数を持つ側のオブジェクトの下の位置にある面を左ボタンダブルクリックする。



追加前 (a)　　一つ目を追加 (b)　　二つ目を追加 (c)

図 4.9: 引数オブジェクトの追加

引数オブジェクトの削除

引数の数を減らすには、他のオブジェクトの削除の場合と同様に、削除したい引数オブジェクトをゴミ箱アイコンの 上にドラッグ & ドロップする。削除された引数オブジェクトにエッジ結線されていた場合、全て切断される。引数オブジェクトの数が変化した場合は、引数オブジェクトの配置、色は自動的に修正される。

入出力モードの変更

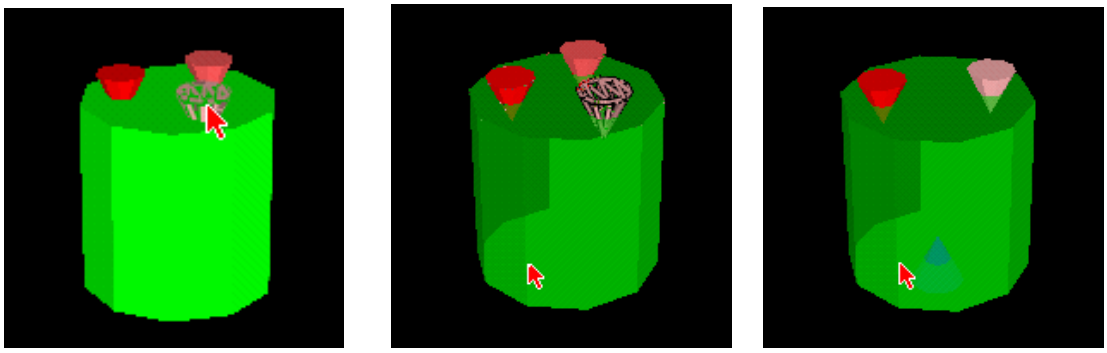
引数オブジェクトの入出力モードを入力から出力へ、または、出力から入力へと変更する操作には、オブジェクトの削除のときと同様にドラッグ & ドロップ操作を用いる。ただし、ドロップする対象が、引数オブジェクトが付加されているゴールのオブジェクトである点が、削除の操作と異なる。以下に、操作の手順の解説を行う。

入出力モードを変更する引数オブジェクトを始点とし、ドラッグ操作を行うと、左ボタンドラッグの場合は地面に並行に、右ボタンドラッグの場合は画面に水平に、引数オブジェクトがマウスポインタに重なるように移動する。中ボタンドラッグの場合、引数オブジェクトは移動しない。

その後、対象としている引数オブジェクトの入出力モードを入力とする場合は、入力の引数オブジェクトを配置する面に近い位置にドロップする。出力とする場合は、出力の引数オブジェクトを配置する面に近い位置にドロップする。引数オブジェクトが元々あった面にドロップされた場合には、入出力モードは変化しない。

また、引数オブジェクトが、引数オブジェクトを配置する面以外の場所にドロップされた場合、入出力モードは変化せず、引数オブジェクトは元の位置に戻る。入出力モードが変化した場合は、引数オブジェクトは変化した入出力モードに応じた面に移動する。また、配置、色は自動的に修正される。

図 4.10は中ボタンドラッグ & ドロップを用いて引数オブジェクトの入出力モード変更を行った例を示している。



引数オブジェクトを選択 (a)　　ドラッグ中 (b)　　ドロップ後 (c)

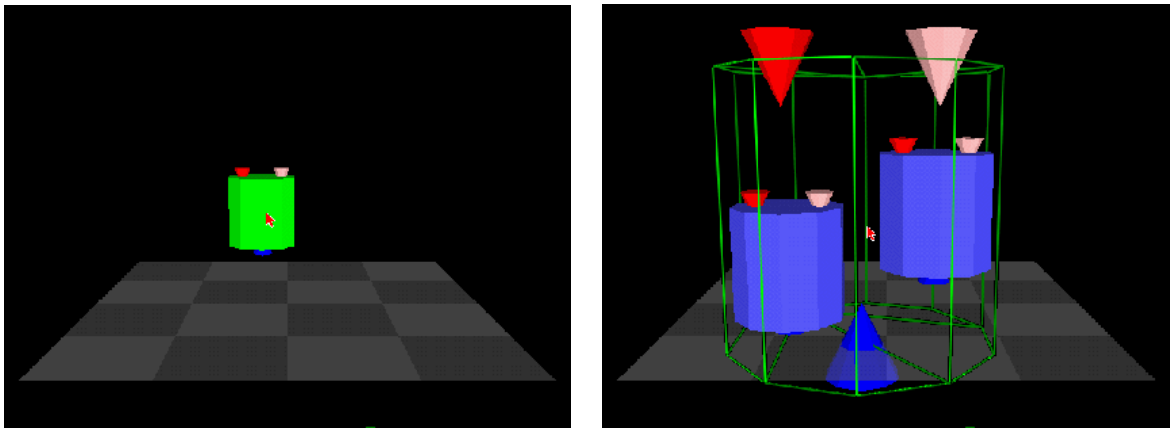
図 4.10: 引数の入出力モード変更 (中ボタンを使用した場合)

4.2.3　ゴールと関連する節の編集

ゴールのオブジェクトを左ダブルクリックすると、オブジェクトが画面の 1/2 を占める程度の大きさまで拡大され、ワイヤフレーム表示となる。同時に、内部に入っている節のオブジェクトも拡大される (図 4.11)。拡大表示されているときに、ゴールのオブジェクトを左クリックすると、拡大表示が解除され、元の大きさ、表示状態に戻る。

拡大表示状態では、内部にある節のオブジェクトを操作することができる。節のオブジェクトををゴミ箱アイコン上へドラッグ & ドロップし、節を削除したり、ゴールの内部にある節のオブジェクトを編集したりすることができる。

また、ゴールからは、そのゴールと関連する節のオブジェクトを生成することができる。



拡大表示前のゴール (a)　　拡大表示後のゴール (b)

図 4.11: 拡大表示されたゴールのオブジェクト

4.3　節の編集

4.3.1　節のオブジェクトの生成

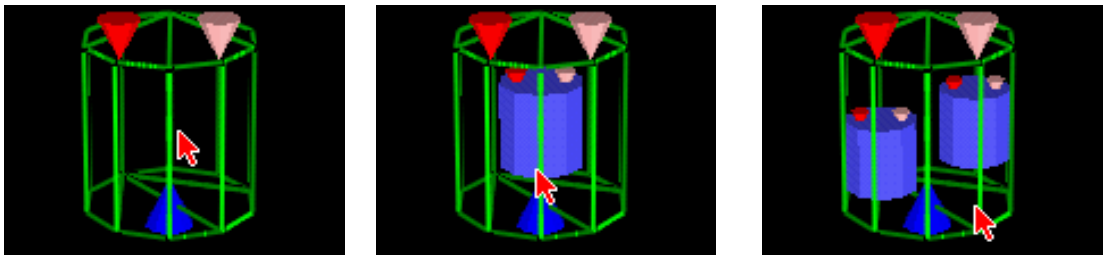
節のオブジェクトの生成は、ゴールのオブジェクトを右ダブルクリックすることによって行う。生成される節は、生成に用いたゴールと数、入出力モードがともに同じ引数を持った状態で生成され、ゴールの内部に配置される。新たに生成された節はガード部、ボディ部ともに何も定義されていない空の状態である。

図 4.12は節のオブジェクトの生成の様子を示した図であり、節の生成後の様子をゴールをワイヤフレーム表示にして示している。生成前には内部に一つも節がない状態であるが (図 4.12(a))、ゴールを右ダブルクリックするたびに節の数が増加する (図 4.12(b),(c))。

4.3.2　節の引数

節は、必ず対応しているゴールと同様の引数を持っている。仮に、節の引数が変更可能であるすると、節の引数がゴールの引数と異なる場合が起こってしまう。

我々は節の引数は変更すべきではないと考え、節の引数は生成時に決まり、以後の変更はできないようにしている。



節がない状態 (a)　　節を一つ生成した後 (b)　　二つ目の節を生成した後 (c)

図 4.12: 節のオブジェクトの生成

4.3.3　節の定義の編集

節のオブジェクトを左ダブルクリックすると、節のオブジェクトが画面の大部分を占める程度まで拡大し、ワイヤフレーム表示となる。節のオブジェクトの内部に、定義を表すオブジェクトが存在する場合、節と同様に拡大される (図 4.13)。

節が拡大された状態のときは、画面左下に、現時点で定義されている全てのゴールに対応する 3 次元アイコンが配置される。

節が拡大表示された状態では次のような編集が行える。

- 新しいゴール、オペレータ、ファンクタ、アトムの各オブジェクトの生成
  右下の 3 次元アイコンを左クリックして新たなオブジェクトを生成し、ガード部またはボディ部に配置することができる。また、オブジェクト同士または節の引数とエッジ結線を行うことができる。

- 現時点で定義されているゴールのコピーの生成
  前述の、現時点で定義されているゴールのアイコンを左クリックすることで、そのゴールのコピーを生成することができる。ここで作成されたゴールは対応するゴールと同じ数の引数を持っている。引数の入出力モードも同様である。生成されたゴールの引数に他のオブジェクトを結線し、引数として与えることができる。

節の定義編集中に現れるゴールは、新しく生成した場合には、引数の変更、ラベルの変更を行うことができる。現時点で定義されているゴールのコピーを生成した場合は、ラベル変更、引数変更を行うことはできない。

また、いずれの場合でも、ゴールを拡大して定義を編集することはできない。

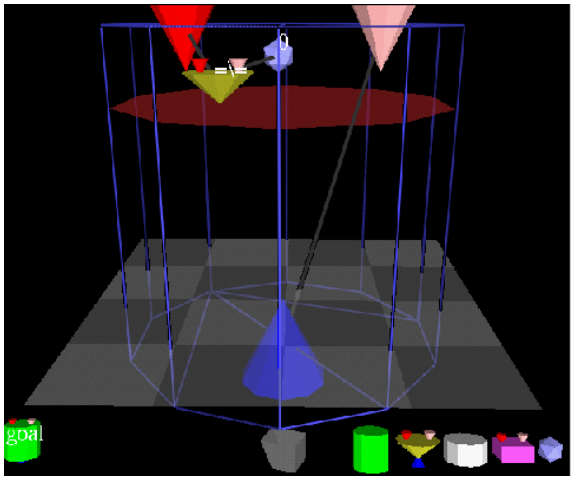


図 4.13: 拡大表示状態の節のオブジェクト

4.4 ファンクタ、コンス、アトムの編集

生成された直後のファンクタは引数が一つも設定されていない状態であるが、ゴールと同様の操作で引数の数を増減することができる。しかし、ファンクタの引数は何らかの値を返す目的で使用されることはないため、引数の入出力モードを考慮する必要はない。従って、入出力モードの変更操作は存在しない。ファンクタの引数オブジェクトに他のオブジェクトを結線し、ユニフィケーションを表すことでファンクタに引数として与えることができる。

コンスは引数の数が 2 で固定であり、ファンクタと同様に、引数は入出力モードを持っていない。よって、引数の変更を行うことはない。コンスは、ファンクタと同様に、引数オブジェクトに他のオブジェクトを結線し、引数として与えることができる。

アトムは引数を持たないため、引数に関しての操作は存在しない。アトムに対して行える編集には、アトムの値変更とユニフィケーションの 2 つの編集が考えられる。アトムの値はラベルで表示しているため、値の変更はラベルを変更することによって行う。ユニフィケーションは他と同様に、エッジ結線をすることによって行う。

4.5 オペレータの編集

生成された直後のオペレータは、“+” のラベルを持っており、整数演算の加算を表している。演算の種別の変更は 4.1.5節で解説したラベル変更によって行う。ラベルは扱う演算の種類と同じく、整数演算には “+”、 “-”、 “*”、 “/”、 “mod”、比較演算には “<”、 “>”、 “=<”、 “>=”、 “=:=”、 “=\=” のいずれかのみとなる。

オペレータは、演算の種別が、変更操作により整数演算から比較演算へ、またはその逆に変化した場合、出力引数オブジェクトの数が変化する。出力引数オブジェクトの数が減少したときに、消滅した引数オブジェクトにエッジが結線されていたならば、全て切断される。

4.6 操作一覧

表 4.1は、本システムにおける編集操作の一覧である。

本システムの操作は、左、右ボタンが、オブジェクトの移動、大きさの変更を伴う操作である、オブジェクトの移動、拡大表示、拡大表示解除に割り当てられている。また、中ボタンがオブジェクトの移動、大きさの変更を伴わない操作である、エッジ結線、ラベル変更に割り当てられている。

対象	操作	処理
オブジェクト	左ドラッグ & ドロップ	地面に平行に移動
	右ドラッグ & ドロップ	画面に平行に移動
	中ドラッグ & ドロップ	エッジ結線
	中ダブルクリック	ラベル変更
	ゴミ箱アイコンにドロップ	オブジェクト削除
ゴール	左ダブルクリック	拡大表示
	引数の面の上で左ダブルクリック	引数オブジェクトの生成
	左クリック	拡大表示解除
	右ダブルクリック	節の生成
節	左ダブルクリック	拡大表示
ファンクタ	引数の面の上で左ダブルクリック	引数オブジェクトの生成
	左クリック	拡大表示解除
引数オブジェクト	付加されているゴールにドロップ	入出力モード変更
アイコン	左クリック	オブジェクト生成
なし	左ドラッグ	視点変更 (回転)
	右ドラッグ	パン・ズーム

表 4.1: 操作一覧

4.7 プログラム編集例

本節では、本システムにおけるプログラム編集の例を解説する。編集するプログラムは、 3.5節で挙げた、最大公約数 (GCD) を求めるゴール “gcd” である。

ゴール “gcd” は３つの引数をもっており、４つの対応する節が存在する。ここでは、ゴール “gcd” の編集と、ゴール “gcd” の節の一つである、

```
gcd(X,Y,Out) :- Y =\= 0, X >= Y | gcd(Y,Y1,Out), Y1 := X mod Y.
```

を編集する課程を解説する。

ゴール “gcd” の生成

始めに、画面右下のゴールと同じ形をしたアイコンを左クリックし、ゴールのオブジェクトを生成する (図 4.14)。また、中ボタンダブルクリックして、ラベル変更ウインドウを呼び出し、ゴールのラベルを “gcd” に変更する。

引数を持たないゴール “gcd” が作成された。

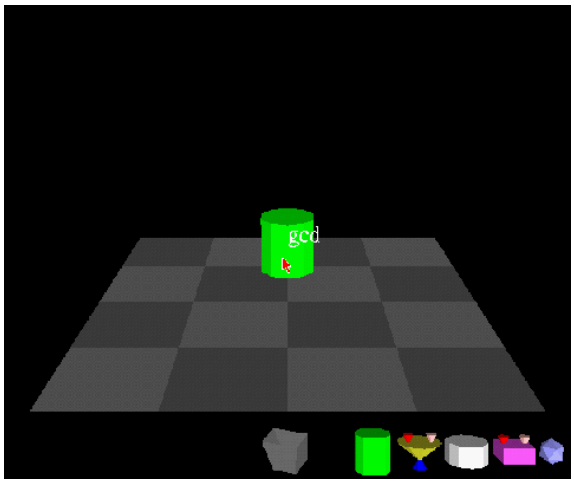


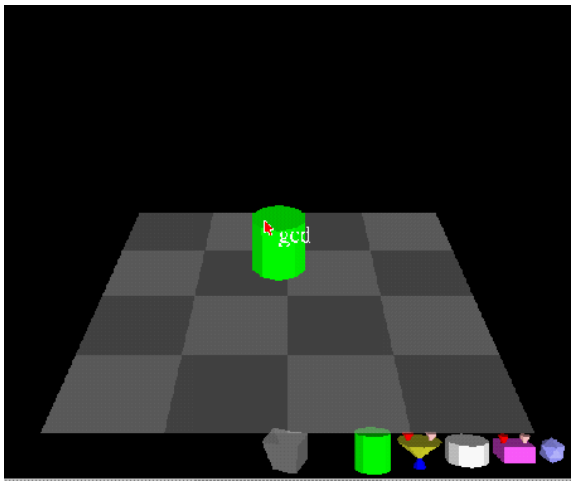
図 4.14: ゴール “gcd” の生成

引数の設定

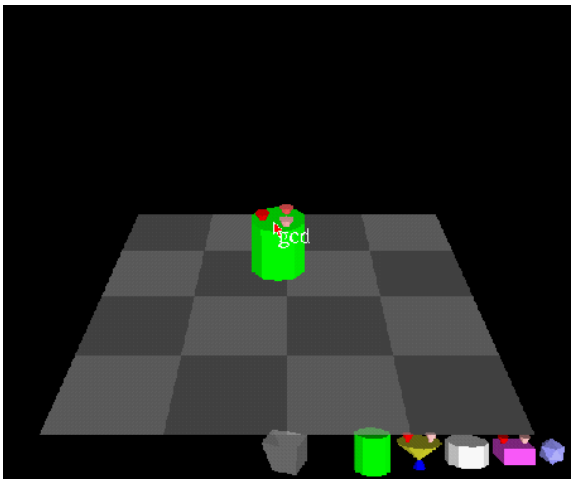
次に、引数の数、入出力モードを設定する。ゴール “gcd” は “X”、 “Y”、 “Out” の３つの引数をもっており、 “X”、 “Y” は入力、 “Out” は出力として用いられる。

引数オブジェクトを配置する面で左ダブルクリックを行い、引数オブジェクトを生成する。今回は、出力引数オブジェクトを配置する面が、隠れてしまっているため (図 4.14)、入力引数オブジェクトを３つ生成し、そのうちの１つの入出力モードを変更する方法をとる (図 4.15)。３つの引数オブジェクトのうちの１つをゴール “gcd” のオブジェクトにドラッグ & ドロップし、入出力モードを変更する。

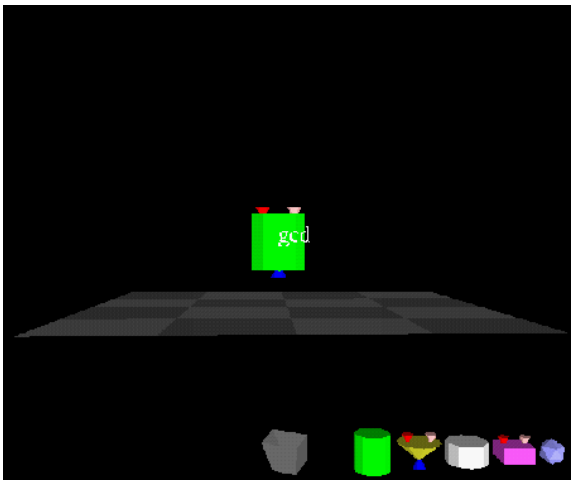
ゴール “gcd” に入力モードの引数が２つ、出力モードの引数が１つ設定された。引数オブジェクトは、最も濃い赤色のものが “X”、薄い赤色のものが “Y”、濃い青色のものが “Out” をそれぞれ表している。



始めの状態



入力引数を３つ生成した状態



入力引数の一つを出力モードに変更した後

図 4.15: ゴール “gcd” の引数の生成

節の生成

次に、節のオブジェクトを生成する。

ゴール “gcd” のオブジェクトを右ボタンダブルクリックし、節のオブジェクトを内部に生成する。この操作を計４回行い、４つの節のオブジェクトを生成する。

ゴール “gcd” は、４つの、何も定義されていない節を持つゴールとなった (図 4.16)。

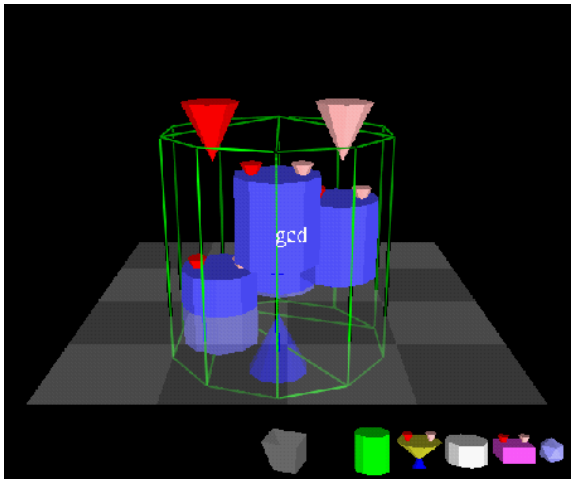


図 4.16: 生成された４つの節

節の拡大表示

次に、ゴール “gcd” のオブジェクトを左ボタンダブルクリックし、ゴールを拡大表示状態にし、節の編集が行える状態にする (図 4.16)。その後、４つの節のオブジェクトのうち１つを選び、それを左ダブルクリック操作で拡大表示状態にし、節の編集を開始する。

ガードの定義

まず、ガードの定義を行う。この節のガードの定義に必要なオブジェクトは、オペレータ “=\=”、オペレータ “>=”、アトム “0” である。これらを画面右下のアイコンを用いて生成し、ラベル変更によってオペレータの演算の種別、アトムのもつ値を変更する。

その後、定義に合うようにエッジ結線を行う。オペレータ “=\=” は、“Y =\= 0” を表すように、第１引数をゴール “gcd” の引数 “Y”、第２引数をアトム “0” と結線する。

オペレータ “>=” は “X >= Y” を表すように、第１引数をゴール “gcd” の引数 “X”、第２引数を引数 “Y” と結線する。

ガード

```
Y =\= 0, X >= Y
```

の定義が完成した (図 4.17)。

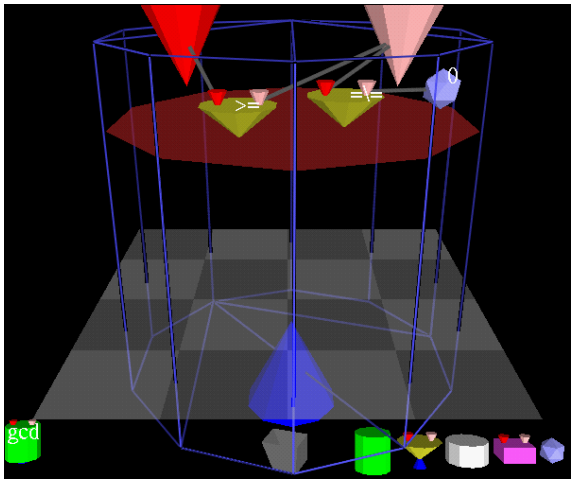


図 4.17: 編集後のガード部

ボディの定義

最後に、ボディの定義を行う。この節のボディに必要なオブジェクトは、ゴール”gcd”の再起呼び出しを表すための、ゴール”gcd” のコピーと、オペレータ “mod” である。

ゴール”gcd” のコピーは、画面左下のゴール”gcd” と同じ形をしたアイコンを左クリックして生成する。また、オペレータのオブジェクトを生成し、演算の種別を “mod” にする。

オペレータ “mod” は “Y mod X” を表すように、第 1 引数に “Y”、第 2 引数に “X” を結線する。

ゴール”gcd” の再起呼び出しは、テキストでは、 “gcd(Y,Y1,Out)” の部分である。”Y” は、元のゴール “gcd” の第 2 引数 “Y” とのユニフィケーションを表しているので、再起呼び出しのゴール “gcd” の第 1 引数と元のゴール “gcd” の “Y” を結線する。

“Y1” は、 “Y1 := Y mod X” より、オペレータ “mod” の計算結果である。よって、再起呼び出しのゴール “gcd” の第 2 引数と、オペレータ “mod” の出力引数を結線する。

“Out” は元のゴール “gcd” の出力引数 “Out” とのユニフィケーションである。よって、再起呼び出しのゴール “gcd” の出力引数と元のゴール “gcd” の出力引数 “Out” を結線する。

以上で、この節のボディの定義が完成し、ゴール “gcd” の 1 つの節の定義が完成した (図 4.18)。

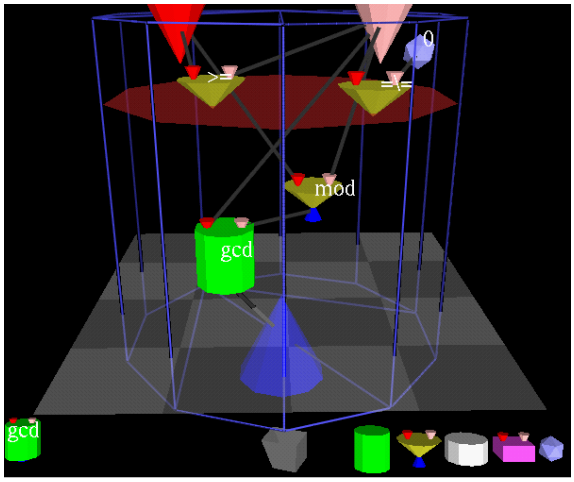


図 4.18: 完成した、節の定義

第 5 章

実装

5.1 環境と使用言語

本システムは IRIX6.3[8] 上で実装を行い、プログラミング言語は C++[9] を使用した。また、３次元図形表示およびマウス、キー操作処理には、 OpenGL[10] と GLU、GLUT[11] ライブラリを使用した。

本システムは、オブジェクトの表示、操作方法については、 Claymore[12] , 従来の 3D-PP を参考にしているが、従来のシステムを元にクラス設計を見直し、独自の実装を行っている。

5.2 クラス階層

本システムのクラス階層を図 5.1に示す。なお、本システムの Step 数は約 15,000 である。

本システムは KL1 プログラムを表すクラス、３次元オブジェクトのクラス、マウスイベント処理のクラスから構成されている。

KL1 プログラムを表すクラスは Program クラスを基底のクラスとしている。 Program クラスからはプログラムの処理、文法にあたる Syntax クラス、データにあたる Data クラスが派生する。 Syntax にはゴール、節、オペレータを表すクラスである Goal, Clause, Operator クラスがあり、 Data にはファンクタ、コンス、アトムを表すクラスである Functa, Conscell, Atom クラスがある。また、 Program は Graph クラスと Edge クラスを用いて無向グラフを作成することができる。

３次元オブジェクトのクラスは GraphicalObject クラスを基底のクラスとしている。GraphicalObject からはプログラム部と対応する、 GraphicalGoal、 GraphicalClause、GraphicalOperator、 GraphicalAtom,GraphicalFuncta,GraphicalConscell の各クラスが派生する。以下、これらのクラスを Graphical~ と呼ぶ。

Graphical~ は対応するプログラム部のクラスと相互参照を持っており、 Graphical~ の変更はプログラム部へも影響する。 Graphical~ はそのオブジェクト固有のマウスイベント処理のメソッドを持っている。また、 Edge にも対応する３次元オブジェクトのクラス GraphicalEdge が存在する。

上記のプログラム部に対応する Graphical~ クラス以外の３次元オブジェクトのク

ラスとしては、 Icon クラスが存在する。 Icon クラスからは用途に応じて、オブジェクト作成アイコンのクラス CreateIcon, ゴミ箱アイコンのクラス TrashIcon クラスが派生する。

マウスイベント処理のクラスには MouseEvent, MouseEventDispatcher がある。3D-PP 実行時に起きたマウスイベントの情報は MouseEvent に格納され、 MouseEvent-Dispatcher でイベントに対応したマウスイベント処理メソッドが呼び出される。ここで呼び出されるマウスイベント処理メソッドはマウスポインタで注目しているオブジェクトの持つ固有のマウスイベント処理メソッドである。

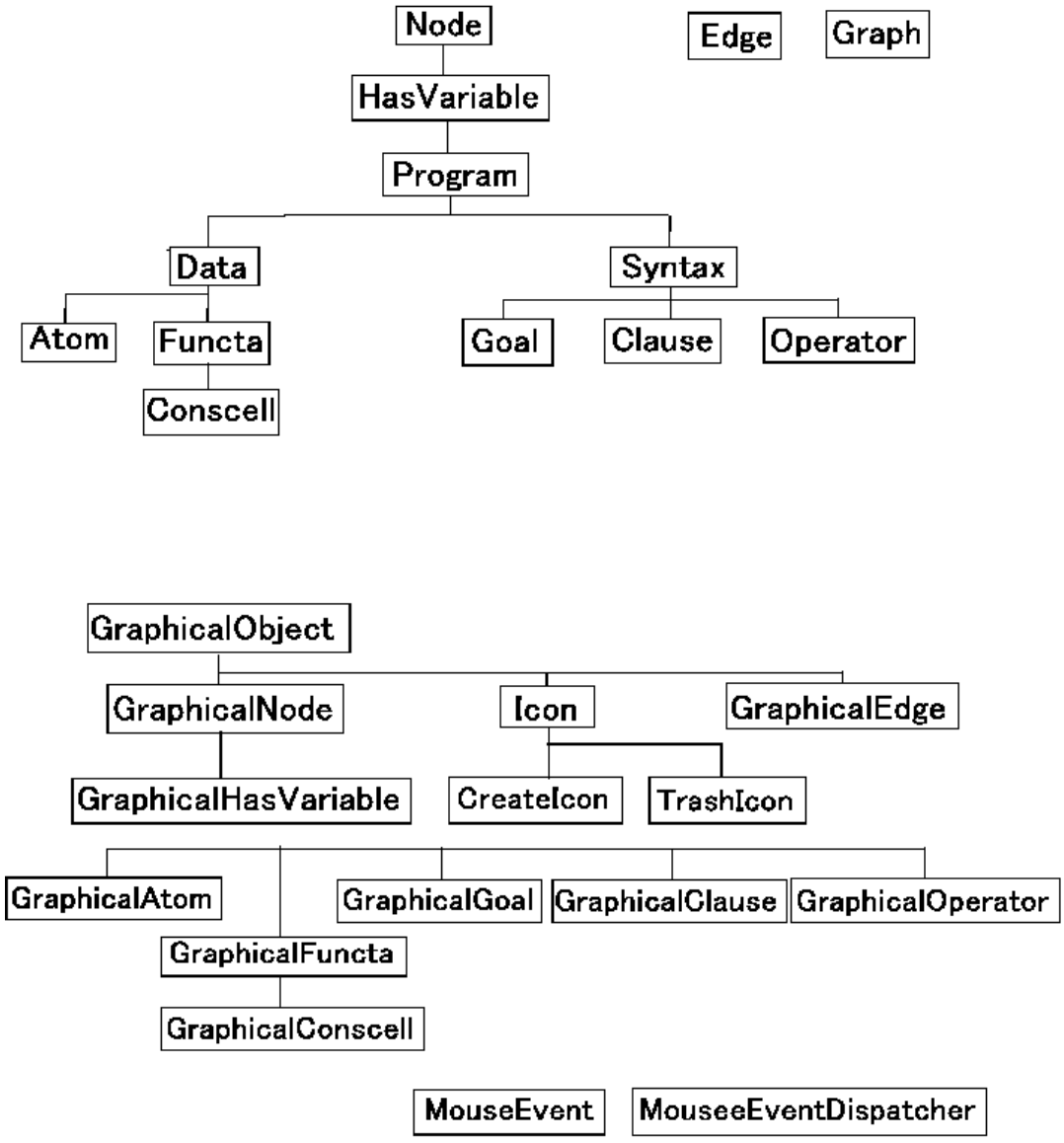


図 5.1: クラス階層図

第 6 章

関連研究

6.1 ToonTalk

ToonTalk[4] は、子供向けの教育用ビジュアルプログラミングシステムであり、人物や動物等のキャラクタや、操作に用いるアイテム等を、親しみやすい表示の３次元図形で表している。

ユーザは画面上の３次元キャラクタ、３次元図形を操作することで、プログラムを作成し、実行をアニメーションで見ることができる。３次元キャラクタ、３次元図形はマウスを用いて、掴む、移動する等の直接操作をすることができる。また、図形等の操作は全て２次元的なドラッグ & ドロップで行い、３次元空間の様な奥行きはない。

本研究との関連としては、プログラム表現の違いを挙げることができる。 ToonTalk では、人から教わることなく、子供でもプログラミングができるように、プログラミング言語を意識させない表現を用いているが、本研究の手法では、 KL1 プログラムの基本要素をそれぞれ３次元オブジェクトで表し、 KL1 プログラムを余すところなく、詳細に表現することを目的としている。

6.2 Pictorial Janus

Pictorial Janus[5, 6] は、並列論理型言語 Janus の実行を２次元アニメーションで視覚化するシステムである。 Pictorial Janus では、プログラムの基本となる要素を、閉じた輪郭線で表している。輪郭線はどのような形、サイズ、色でも良く、好きな形でプログラムを表現することができ、プログラム表現は図形の回転、大きさの変化によって変わることはないように設計されている。

本研究との関連としては、 Pictorial Janus には引数の存在を表す port という図形表現があり、それらは規則性をもって閉じた輪郭線上に配置される、という点が挙げられる。これは提案手法の引数オブジェクトの付加と類似している。

相違点としては、引数オブジェクトは数、モードによって決まった色、形、配置位置、配置順番を用いて引数の区別を行っているが、 port の場合は、並び順は決まっていても、配置位置は、閉じた輪郭線上のどの位置にでもなりうるという点が挙げられる。

第 7 章

まとめ

本研究では、従来の 3D-PP におけるプログラム表現の問題点として、ゴールが複数の引数を持っている場合、エッジの結線方法に差が無いため引数の対応関係の区別ができない、という問題と、ゴールと節の対応関係を表す手段がないため、対応関係の判断がプログラマにとって負担となる問題点と、節とその定義が表示空間を圧迫するという問題点を挙げ、視覚化されたプログラムを理解することが困難になることを指摘した。

我々は、これらの問題点は、従来のプログラム表現が、引数の表現について曖昧な点をもっており、また、対応関係を表すための表現が不足していたことに起因すると考えた。そして、これらの問題点は、引数の対応関係と入出力モードを明確にする引数オブジェクトの導入と、節とゴールの対応関係を表す入れ子表示の導入で解決することを示した。また、引数オブジェクトと入れ子表示を 3D-PP に実装し、これらに適合する編集方法を 3D-PP の編集操作として実装した。

引数オブジェクトの付加と、ゴールと節の入れ子表示の導入によって、従来の 3D-PP のプログラム表現と比較して、 KL1 プログラムの、より詳細で、より実用的なプログラム表現が可能となり、実装によって、実用的なプログラム編集が可能となった。

謝辞

本研究を進めるにあたって、指導教官である田中二郎教授、および助手である志築文太郎先生、三浦元喜先生からは終始丁寧なご指導を頂きました。心より感謝致します。また、 IPLAB の皆さんからも、貴重なご意見、ご指導を頂きました。特に、同じ VS グループのメンバーである小川徹さん、飯塚和久さん、劉学軍さん、山田英仁さん、神田正和さんからは、研究の進め方に関して大変有益なご意見を頂き、システム実装に関してもご協力を頂きました。ここに感謝の意を表します。

参考文献

[1] 田中 二郎, 太田 祐紀子: GHC プログラムの視覚的入力システム:FE'92, 電気情報通信学会 研究報告 COMP 90-67, pp.53-62, 1990.

[2] 宮下 貴史: 3 次元ビジュアルプログラム編集環境の構築, 平成 11 年度 筑波大学大学院修士課程理工学研究科修士論文,2000.

[3] KLIC 講習会テキスト -KL1 言語編-, 財団法人 新世代コンピュータ技術開発機構 作成, 財団法人 日本情報処理開発協会開発室 改訂,1995.

[4] Ken Kahn: ToonTalk – An Animated Programming Environment for Children, Journal of Visual Languages and Computing, pp.197-217, 1996.

[5] Ken Kahn: Concurrent Constraint Programs to Parse and Animate Pictures of Concurrent Constraint Programs, Technical Report SSL91-16/P91-00143, XEROX PARC, Palo Alto, CA ,1991.

[6] Kenneth M. Kahn and Vijay A. Saraswat: Complete Visualization of Concurrent Programs and Their Executions, Workshop on Logic Programming Environments (LPE 1990), pp.30-34, 1990.

[7] 淵 一博監修, 古川康一・溝口文雄共編: 並列論理型言語 GHC とその応用, 共立出版, 1987.

[8] . IRIX Programmer's Reference Manual, Silicon Graphics, Inc., Mountain View, CA.

[9] Margaret A. Ellis and B. Stroustrup: The Annotated C++ Reference Manual, Addison-Wesley Publising Company, Inc., 1992.

[10] OpenGL Architecture Review Boad, Manson Woo,Jackie Neider and Tom Davis: OpenGL プログラミングガイド第 2 版, ピアソン・エデュケーション, 1997.

[11] Mark J. Kilgard: The OpenGL Utility Toolkit (GLUT) Programming Interface: API Version 3, Silicon Graphics Inc., 1996.

[12] 大芝 崇, 田中 二郎: 3 次元モデリングツール "Claymore": 付加情報によって強化された直接操作, 日本ソフトウェア科学会第 15 会大会論文集, pp.161-164, 1998.

[13] 神谷 誠: 3 次元ビジュアルプログラミングシステムにおけるドラッグ＆ドロップ手法の拡張, 平成 10 年度 筑波大学第三学群工学システム学類卒業論文, 1999.