1999 年度 卒業論文

# KL1による分散KL1言語処理系の実装

早稲田大学理工学部情報学科
G96P073-6 高木祐介

指導： 上田和紀 教授

提出日： 2000年2月4日

# 目 次


**第 1 章　研究の背景と目的** **1**

1.1　KL1 の概略 . . . . . 1

1.2　既存の分散 KL1 言語処理系 . . . . . 2

1.3　本論文の構成 . . . . . 2

**第 2 章　KL1 の概略** **3**

2.1　言語のクラス階層 . . . . . 3

2.2　字句構造 . . . . . 3

2.3　文法 . . . . . 5

2.4　KL1 プログラムの動作 . . . . . 6

2.5　並列実行プラグマ . . . . . 7

2.6　分散実行プラグマ . . . . . 8

**第 3 章　dKLIC の設計** **9**

3.1　実装の階層 . . . . . 9

3.2　遠隔実行の仕組み . . . . . 10

3.3　遠隔呼び出しの利点 . . . . . 10

3.4　実装の方針 . . . . . 11

**第 4 章　予備実験** **12**

4.1　実験の環境 . . . . . 12

4.2　直列化処理の必要性 . . . . . 13

4.3　遠隔呼び出し . . . . . 13

4.4　未定義変数の識別 . . . . . 14

4.5　未定義変数の wrap/unwrap . . . . . 15

**第 5 章　実装の詳細** **17**

5.1　直列化 . . . . . 17

5.2　変数表 . . . . . 18

5.2.1　変数表の設計 . . . . . 19

5.2.2　変数表の実装 . . . . . 21

5.3　KL1 項の送受信 . . . . . 22

**第 6 章　dKLIC の評価** **23**
6.1　遠隔実行の例 . . . . . 23
6.2　機械翻訳アルゴリズム . . . . . 25
6.3　実装の方針 . . . . . 26
6.4　動的変数名生成 . . . . . 29
6.5　変数名と変数表のデータ型 . . . . . 29

**第 7 章　今後の課題** **31**
7.1　まとめ . . . . . 31
7.2　機械翻訳 . . . . . 31
7.3　未定義変数表の整列 . . . . . 32
7.4　変数表の並列化 . . . . . 32
7.5　その他の遠い目標 . . . . . 33

**謝辞** **34**

**参考文献** **35**

**図目次** **37**

**付 録 A　実行時ライブラリ** **38**
A.1 type . . . . . 38
A.2 vartab . . . . . 49
A.3 vsock . . . . . 55

**付 録 B　プログラム例** **71**
B.1 appcli . . . . . 71
B.2 appsrv . . . . . 73

**付 録 C　実験** **75**
C.1　変数の識別 . . . . . 75
C.1.1　実行結果 . . . . . 75
C.2　メタ KL1 項 . . . . . 76
C.2.1　実行例 . . . . . 76
C.3　クライアント-サーバ . . . . . 77
C.3.1　クライアント . . . . . 77
C.3.2　サーバ . . . . . 77
C.4　遠隔ゴール実行 . . . . . 78
C.4.1　実行結果 . . . . . 78

## 概 要

ネットワーク環境の普及につれ、分散プログラミングに対する需要が高まっている。しかし、逐次手続き型言語による分散処理の記述は非常に煩雑である。

並行論理型言語は、実行の各単位が並行に実行され、プロセス間通信を宣言的に記述できる。また、分散処理は並行処理の一例なので、並行論理型言語で分散処理を記述するのはごく自然なことであろう。

KL1 は、並行論理型言語の中で最も単純な GHC を基にした、実用的な並列論理型言語である。既存の処理系 KLIC の定める並列 KL1 では、並列のための限られた分散処理と、Unix レベルのソケット通信を記述することができるが、一般の分散処理を言語レベルで記述することはできない。

本研究では、KL1 を分散化するために、分散 KL1 の極小部分集合 dKL1 を定め、KLIC を利用して、dKL1 の言語処理系 dKLIC を実装した。

dKL1 に近い KLIC/KL1 を実装言語に選んだために、dKLIC 実装の本質部分に集中でき、実装の手間は少なかった。また、dKL1 の定める十分広いクラスのプログラムに対して、機械翻訳を行うアルゴリズムを示すことができた。

# 第1章　研究の背景と目的

分散処理系を記述する汎用プログラミング言語としては、Java が現在、最も現実的な選択肢だと言えるだろう。実際、Java にはオブジェクト指向、マルチスレッド、ネットワーキング API といった分散処理向きの特徴が備わっているが、それにもかかわらず、分散処理系の記述、検証は難しい。これは、Java がプリミティブには逐次処理の手続き型言語であることに因るものと思われる。

並行論理型言語は、実行の各単位が並行に実行され、プロセス間通信を宣言的に記述できる。また、分散処理は並行処理の一例なので、並行論理型言語で分散処理を記述するのはごく自然なことであろう。

KL1 は、並行論理型言語の中で最も単純な GHC を基にした、実用的な並列論理型言語である。既存の処理系 KLIC の定める並列 KL1 では、並列のための限られた分散処理と、Unix レベルのソケット通信を記述することができるが、一般の分散処理を言語レベルで記述することはできない。

本論文では、KL1 の分散化について論ずる。KL1 のデファクト・スタンダードは、前述の KLIC 処理系によって定義されているため、断り無しに並列 KL1、あるいは単に KL1 と言う場合、KLIC/KL1 を指す。また、本研究で実装しようとしている処理系を dKLIC、dKLIC によって定義される分散 KL1 言語を dKL1 と名付ける。d は distributed の頭文字である。

## 1.1　KL1 の概略

KL1 プログラムは述語の集合 ( および、モジュールと初期ゴール ) によって定義され、述語は節の集合によって定義される。節は以下のようなものである。

```
Head :- Guard | Body.
```

ここで、Head は原子論理式であり、Guard と Body は原子論理式のマルチ集合である。Guard と Body 中の原子論理式はゴール、すなわち述語呼び出しであり、Head は定義される述語 ( 節 ) を呼び出すゴールにマッチするプロトタイプである。

ゴールと述語呼び出しとの違いは、オブジェクト指向の「メッセージ」と「メソッド呼び出し」との違いに似ている。すなわち、ゴールは述語呼び出しとして実装される概念であるが、混同しても実用上の問題はなく、しばしば同一視されているため、本論文でも特に区別しない。

## 1.2 既存の分散 KL1 言語処理系

分散 KL1 言語処理系としては既に、並列推論マシン (PIM) 上の処理系、Unix/C 上の処理系 (KLIC)、Java 上の処理系 (KLIJava、KL1j) などがある。しかし、本研究の目的である汎用分散論理型言語としての KL1 を提唱しているのは KL1j [12] のみである。

KL1j は実行時ライブラリが実装済みだが、KL1 から遠い Java を実装言語に選んだため、考えなければならないことが多く、機械翻訳の方法論が確立していない。

本研究では、KL1j のコンパイラ実装の基礎技術を固めるため、KL1j の提唱する分散 KL1 の極小部分集合を定め、実動できる処理系を構築することを目指す。

## 1.3 本論文の構成

第 1 章では研究の背景と目的を述べた。
第 2 章では研究に関する基礎知識を述べる。
第 3 章では dKLIC 実装の方針を述べる。
第 4 章では実装方法に関する予備実験を示す。
第 5 章では実装の詳細を述べる。
第 6 章では実装に関する考察をする。
第 7 章では本論文についてまとめ、今後の予定を述べる。

# 第2章　KL1の概略

本章では、dKLIC 実装のために学んだ、KL1 言語と周辺知識について述べる。

第 2.1 節で近傍言語との関係を述べ、第 2.2 節で字句構造、第 2.3 節で文法、第 2.4 節で KL1 プログラムの意味、第 2.5 節と第 2.6 節で意味を変えないプラグマを示す。

## 2.1　言語のクラス階層

Guarded Horn Clauses (GHC) [1] は並行論理型言語の一つである。

GHC は、プログラムに対する制約が少ないという意味で単純かつ強力な並行論理型言語であった。このことは、GHC フルセットの実現が容易でないことを示し、実現可能なサブセットが必要とされた。ガードゴールに対する制約を強めた Flat GHC [2] という GHC のサブセットは、比較的簡単に効率の良い実装ができる。

KL1 は、Flat GHC に様々なデータ型、モジュール、実行制御機能を付け加えた実用言語である。

既存の KL1 言語処理系としては、並列推論マシン (PIM) 上の処理系、Unix/C 上の KLIC、Java 上の KLIJava [11]、KL1j [12] などがある。

現在、KLIC で定義された KL1 が事実上の標準となっている。以下に述べる KL1 の概略は、KLIC/KL1 に基づいている。

## 2.2　字句構造

KL1 の字句構造は、おおよそ伝統的な Prolog [13] の字句構造を受け継いでいる。

KL1 の字句構造を、正則文法によって示す。角括弧 ( < > ) で囲まれたものはメタ変数 ( 字句の集合 ) である。? は 0 回または 1 回の出現 ( 省略できること ) を表す。* と + は、それぞれ 0 回以上、 1 回以上の出現 ( 繰り返し ) を表す。- は集合の引き算 ( 差集合 ) を表す。括弧 ( ( ) ) は優先を表す。引用符 ( ‘ ’ ) で囲まれた文字列は、KL1 ソースプログラム中に現れる文字列である。( 以下はマニュアル [9] を参考にしたものだが、十進以外の整数記法、PIM 互換の整数記法、文字列内の文字コード 記法を省略した。また、関数記号を中置・省略したり、括

弧を省略したりするファンクタの記法は、マニュアル [9] に明記されておらず、ここでも省略した。)

```
<Variable> ::= <Upper> <DigitOrLetter>*.
<DigitOrLetter> ::= <Digit> | <Upper> | <Lower>.

<Digit> ::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’.
<Upper> ::= ‘A’ | ‘B’ | ‘C’ | ‘D’ | ‘E’ | ‘F’ | ‘G’ | ‘H’ | ‘I’ |
            ‘J’ | ‘K’ | ‘L’ | ‘M’ | ‘N’ | ‘O’ | ‘P’ | ‘Q’ | ‘R’ |
            ‘S’ | ‘T’ | ‘U’ | ‘V’ | ‘W’ | ‘X’ | ‘Y’ | ‘Z’ | ‘_’.
<Lower> ::= ‘a’ | ‘b’ | ‘c’ | ‘d’ | ‘e’ | ‘f’ | ‘g’ | ‘h’ | ‘i’ |
            ‘j’ | ‘k’ | ‘l’ | ‘m’ | ‘n’ | ‘o’ | ’p’ | ‘q’ | ‘r’ |
            ‘s’ | ‘t’ | ‘u’ | ‘v’ | ‘w’ | ‘x’ | ‘y’ | ‘z’.
```

KL1 では、変数を束縛する値は型を持つが、変数自体は型を持たない。単一代入であり、束縛済みの変数が別の値に束縛されることはない。また、論理変数と呼ばれ、束縛前は未定義という状態を持つ。変数を値に束縛することを具体化と言う。

_ ( 下線 1 文字 ) という変数は無名変数であり、同じ節の中に複数出現しても、互いに違う変数を意味する。

```
<Atom> ::= <Symbol> | <Quoted> | <Special> | <OneCharacter> | <Nil>.
<Symbol> ::= <Lower> <DigitOrLetter>*.
<Quoted> ::= ‘’’ <QuotedCharacter>+ ‘’’.
<Special> ::= <SpecialCharacter>+.
<QuotedCharacter> ::= <Character>-‘’’ | <QuotedQuote>.

<Nil> ::= ‘[’ ‘]’.
<QuotedQuote> ::= ‘’’’ | ‘\’’.
<OneCharacter> ::= ‘!’ | ‘|’ | ‘;’.
<SpecialCharacter> ::= ‘~’ | ‘+’ | ‘-’ | ‘*’ | ‘/’ | ‘\’ |
                       ‘^’ | ‘<’ | ‘>’ | ‘=’ | ‘‘’ | ‘:’ |
                       ‘.’ | ‘?’ | ‘@’ | ‘#’ | ‘$’ | ‘&’.

<Integer> ::= <Decimal> | <CharacterCode>.
<Decimal> ::= ‘-’? <Digit>+.
<CharacterCode> ::= ‘-’? ‘0’’ <Character>.

<Float> ::= <Sign>? <Digit>+ ‘.’ <Digit>+ (‘e’ <Sign>? <Digit>+)?.
<Sign> ::= ‘+’ | ‘-’.
```

```
<Functor> ::= <Atom> ‘(’ <Term> (‘,’ <Term>)* ‘)’.
<Term> ::= <Variable> | <Atom> | <Integer> | <Float> |
           <Functor> | <List> | <Vector> | <String>.
```

ファンクタは、名前と、引数の個数 (Arity) によって区別される。これを主ファンクタと称し、次の表記法に従う。

```
<PrincipalFunctor> ::= <Atom> ‘/’ <Digit>+
```

例えば、次に示すリストとは、./2 という特別なファンクタ ( コンス・セルである ) を組み合わせたデータ構造の略記法である。

```
<List> ::= ‘[’ <Term> (‘,’ <Term>)* (‘|’ <Term>)? ‘]’.

<Vector> ::= ‘{’ (<Term> (‘,’ <Term>)*)? ‘}’.

<String> ::= ‘"’ <Element>* ‘"’.
<Element> ::= <Character>-‘"’ | <Escaped>.
<Escaped> ::= ‘\a’ | ‘\b’ | ‘\f’ | ‘\n’ | ‘\r’ | ‘\t’ | ‘\v’
            | ‘\’’ | ‘\"’ | ‘\?’ | ‘\\’ | ‘\’ <NewLineCharacter>.
```

## 2.3 文法

KL1 の文法は、Prolog に似ている。大きな違いは、節のガードとボディがコミット演算子 ( | ) によって分けられていることである。

KL1 の文法を、BNF によって示す。

```
<Program> ::= <Module>+

<Module> ::= <ModuleDeclaration> <Predicate>+.
<ModuleDeclaration> ::= ‘:-’ ‘module’ <Atom> ‘.’.
```

KL1 のモジュールは、述語の名前空間である。全ての述語は、いずれかのモジュールに属している。

```
<Predicate> ::= <Clause>+ (‘otherwise’ ‘.’ <Clause>+)*.
<Clause> ::= <Functor> (‘:-’ (<Functor>+ ‘|’)? <Functor>+)? ‘.’.
```

節のトップレベルに現れるファンクタは、構文上は区別されないが、構造データを表すファンクタと異なる、原子論理式である。構造データは関数記号に引数を与えたものであり、原子論理式は述語記号に引数を与えたものである。両者の名前空間は別であり、同形の構造データと原子論理式は異なるものを表す。

節の最初に現れる原子論理式は、ヘッダと呼ばれ、定義される述語を呼び出すゴールのプロトタイプである。コミット演算子の前後に現れる原子論理式 ( のマルチ集合 ) は、それぞれガードゴール、ボディゴールと呼ばれる。ガードゴール

は、ヘッダに加えて、その節が選択されるための条件を表し、ボディゴールは、その節が選択されたときに生成されるサブゴールである。

otherwise. より後の節は、otherwise. より前の節が選択されないと分かったときに限り、選択される可能性がある。全ての節の間に otherwise. を置けば、節の選択に関して Prolog と同じルールになる。すなわち、デフォルトでは節の選択は、前から順になされるとは限らない。ある述語の節のガードが互いに素でない ( 積集合が空で無い ) とき、それらの節の選択は非決定的であり、その述語の意味は非決定的である。

otherwise. の代わりに alternatively. を置くこともできる。直感的には、alternatively. は、節の選択優先度を変えるプラグマである。( otherwise. はプラグマではない。)otherwise. と違い、alternatively. 後の節は、alternatively. 前の節のガードが失敗しなくても選択される可能性がある。変数の具体化を待って中断しているゴールは、成功も失敗もしていない。このプラグマによって、プログラムの停止性が変わる場合がある。

述語は、ファンクタと同様に名前と引数個数で区別され、主ファンクタと同様に表記される。また、述語の属するモジュールを加えて、次の表記法に従う。

```
<PredicateID> ::= <Atom> ‘:’ <PrincipalFunctor>
```

## 2.4 KL1プログラムの動作

KL1 プログラムは、与えられた初期ゴール ( KLIC/KL1 では main:main/0 ) を、そのゴールと同形のヘッダを持つ節に従ってリダクションを繰り返す。すなわち、節はリダクションルールを表している。

Prolog との違いは 2 つあり、並行論理型言語の最大の特徴である並行性をもたらしている。

1 つは、待ちゴールの実行順序が定義されていないことである。Prolog では、初期ゴールを根とし、リダクションされたサブゴールに対して枝を伸ばした多分木を描くと、深さ優先でゴールが実行された。このため、述語の節の順序を入れ替えるとプログラムの動作が変わることがあった。

もう 1 つの違いはバックトラックしないことである。このため、非決定的なプログラムを書くことができ、自動定理証明系として見たとき、Prolog よりも完全性で劣る。しかし、実用的に広いクラスの Prolog プログラムを GHC プログラムにコンパイルできることが知られている。文献 [5, 第 5 章 GHC による全解探索] を参照せよ。

バックトラックがないため、誤った節を選択しないように、ガードの条件がチェックされる。ガードでは、未定義の変数が束縛されることはない。ガードで値がチェックされるべき変数が未定義だった場合、そのゴールは中断され、変数が束縛されるのを待つことになる。

この仕組みを利用してプロセス通信を宣言的に記述できる。すなわち、変数を通信路と見なし、変数に値を束縛( 変数を具体化すると言う )することによって値を送信し、変数の値をガードでチェックすることで受信待ちをすることができる。

また、未定義変数を要素に持つファンクタやベクタを未完成データ構造と言う。特に、徐々に要素が具体化されるリストはストリームと呼ばれ、クライアント-サーバ型のプロセス通信に多用される。

ストリーム通信で流されるデータをメッセージと呼ぶ。ストリームの cdr 連鎖を引数に持つ再帰ゴール列は、純関数型言語の continuation に相当するプログラム技法である。人の視点で、特定の仕事を行う再帰ゴール列をプロセスと呼ぶ。

リストと、そのリストに繋がる cdr 連鎖を持つリストの組を、差分リストと呼び、リストの要素を任意の場所から具体化するのに用いられる。差分リストを表す 2 つのリストをそれぞれ頭、尾と呼ぶ。

## 2.5 並列実行プラグマ

論理的並行性は言語によって表され、プロセスの同期 ( 逐次性 ) は変数による通信を利用して表される。

並行なプロセスは、物理的に同時に実行されるとは限らない。例えば、Unix の各プロセスは、時分割によって並行に動く。これに対して、物理的に同時に複数の並行プロセスが実行されることを、並列実行と言う。

並行プロセスを自動的に並列実行する手法は確立していない。例えば、MPU のパイプライン処理が挙げられるが、一般に最適なコードを生成する手法は知られていない。

そこで、KLIC/KL1 ではプリミティブな並列実行方法をプログラマに記述させるという選択肢が採られた。

KLIC の並列実行プラグマは、以下の 2 通りである。

**Goal@priority(P)** ゴールの実行優先度を指定する。

**Goal@node(N)** ゴールを実行するノード (Processor Element) 番号を指定する。機械語プログラムを各ノードで共有するため、各ノードは同じアーキテクチャの計算機で、共有メモリや Parallel Virtural Machine (PVM) によって通信できなければならない。

優先度指定は、Prolog で言えば節やゴールの順序を入れ替えることに相当する。

ノード指定は、並列実行のための分散指定と言える。分散実行については後述するが、並列効果を上げるためには、ノード間の通信コストが安くなければならない。既に述べたように、KLIC では各ノードが同じアーキテクチャでなければならない。要するに、マルチプロセッサを持つ「1 台の並列計算機」が想定されている。

## 2.6 分散実行プラグマ

プロセスの並列実行では、「狭い範囲」に分散されたプロセスが協調して特定の仕事を分担し、結果として短時間で仕事を終えることが目的であった。

分散実行には、分散資源を活用するのが目的である場合( 並列を含む )の他に、遠隔ノードでプロセスが実行されることが目的である場合がある。例えば、インターネットのチャットプログラムは、当事者の端末間における通信こそが目的である。

遠隔実行が目的である場合、前述の並列実行プラグマでは不足である。例えば、インターネットのノードを番号で指定するには、全てのノードが順序付けられていなければならない。さらに、遠隔実行を依頼するローカルノードが、その全順序を知っていなければならない。

そこで、KL1j [12] は、以下のような分散実行プラグマを提唱している。

**Goal@node(URL)** 各ノードは、同じアーキテクチャでなくても良い。URL は以下のような文字列である。

```
"kl1://Host:Port"
```

ここで、Host はホスト名ないし IP アドレス、Port はポート番号である。

前提とされている分散環境は、直感的にはアーキテクチャの混在した広域コンピュータネットワーク、具体的にはインターネットである、ということが URL を指定することから読み取れるであろう。

前述の分散実行プラグマは、一対一の分散環境を仮定している。すなわち、遠隔実行を依頼するローカルノードと、実行を依頼される遠隔ノードである。

分散処理系で多用されるクライアント-サーバモデルは、サービスを依頼するクライアントと、サービスを提供するサーバを区別する。このようなクライアント-サーバモデルを分散 KL1 で記述する場合、クライアントがサーバに遠隔実行を依頼する形が自然だろう。なぜなら、一般にクライアント-サーバモデルにおいて、サーバはクライアントを予知せず、クライアントは望みのサービスを提供するサーバの識別名を知っているからである。

# 第3章　dKLICの設計

本章では、dKLIC の実装方針について述べる。

分散 KL1 言語処理系の先駆けとして、KL1j [12] がある。KL1j は実行時ライブラリが実装済みだが、KL1 から遠い Java を実装言語に選んだため、考えなければならないことが多く、機械翻訳の方法論が確立していない。

KL1j のコンパイラ実装の基礎技術を固めるため、KL1j の提唱する分散 KL1 の極小部分集合として dKL1 言語を定め、実動できる dKLIC 処理系を構築することを目指す。実現性を重視するため、実現の容易さを追求する。

並行論理型言語における処理の単位として、遠隔ノードにある述語を呼び出すことを分散 KL1 の最低条件とした。

手続き型言語や関数型言語の場合、遠隔呼び出しは手続きあるいは関数呼び出しであり、呼び出す手続きないし関数の識別名と、引数、そして関数なら結果の値を通信すれば済む。しかし、並行論理型言語の場合、未完成データ構造を利用したプロセスの時系列が存在するので、途中まで実行の終わった状態を通信できる必要がある。具体的には、未定義変数および、その具体化を通信できなければならない。

## 3.1　実装の階層

まず、分散言語としての極小部分集合を実装する。その後、分散言語に必要な機能を徐々に付け加えていきたい。実現の容易さのため、全てをフラットに実装するのではなく、インクリメンタルな実装方式を採る。

分散言語としての必要最低条件は、遠隔での処理を行えることであろう。並行論理型言語での処理の単位は、述語とゴールであるから、まず実現すべきは遠隔ゴール実行である。

遠隔計算機にプログラムを送信することは考えない。遠隔実行されるモジュールと述語は、遠隔計算機上にあるものと仮定する。分散処理系に広く用いられているクライアント-サーバモデルでは一般にプログラムの通信は不要であり、この仮定は妥当なものである。Java アプレットではバイトコード ( プログラム ) の通信が行われるが、これはクライアントがバイトコード インタプリタを内蔵しているためである。KL1 において述語の送信を行うためには、受信した述語を実行できる分散 KL1 言語処理系 ( Java のバイトコード インタプリタに相当する ) が

必要だが、そのような分散 KL1 言語処理系は、dKL1 を用いて記述することができる。

## 3.2 遠隔実行の仕組み

遠隔呼び出しのためのモデルとして、分散環境に対する透過性を提供する、スタブ-スケルトンモデルがある。遠隔呼び出しを行うゴールと、遠隔実行される述語は分散環境を認識せず、分散のための通信はスタブとスケルトンの中にカプセル化されるのである。図 3.1 を見よ。

KLIC/KL1 は、ゴールの分散制御を実現していないが、ゴールのメタ実行と、Unix レベルのソケット通信を提供しているので、これを利用して分散 KL1 言語処理系を記述する。

Java で記述された KL1j に比較して、KLIC で分散 KL1 言語処理系を記述することの利点は、手間がかからないことである。KL1 のオブジェクトを Java のオブジェクトにマッピングする必要がなく、実装しなければならないのは、ゴールの送受信と、受信したゴールの実行のみである。

## 3.3 遠隔呼び出しの利点

KLIC のソケット通信を直接利用した場合、送信側と受信側の変数値の同一性は OS レベルで定義されている。

遠隔呼び出しの記述では、送信側と受信側の変数値の同一性はゴールの実引数を述語の仮引数に代入することにより、言語レベルで定義される。すなわち、形式言語論の俎上に乗ったことになる。

形式言語論に乗りやすいことは、この場合、人の目にも直感的であることを意味する。スタブ-スケルトンによるカプセル化と、形式言語論の単純さは、直感的に分かりやすいことの表裏である。

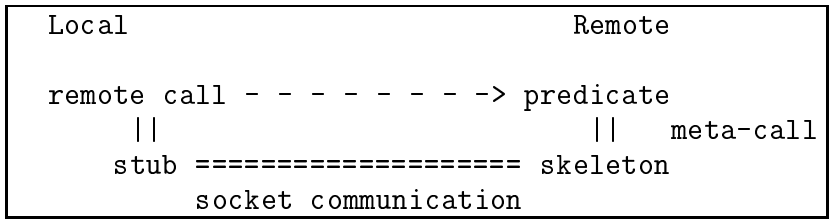


図 3.1: Stub-Skeleton Framework for Remote Predicate Call

## 3.4 実装の方針

汎用性と移植性を高くするため、機種や OS、ヴァージョンに依存する KLIC 機能はできるだけ使わない。

実現の容易さを念頭に置いているため、単純な処理系を目指す。

資源使用のために自動的に分散実行を行う機能は提供しない。KL1j でも、プログラマが明示的に分散指定を行う機能のみを提唱している。同様に、既存の並列 KL1 において、実行速度向上のために自動的に並列実行を行う機能は提供されていない。自動並列、自動分散は、一般の方針が確立していないので、言語機能として提供するのではなく、プログラマが明示指定を用いて実験することが望まれている。

単純化のため、通信は eager に行う。( lazy には行わない。) 遠隔実行ゴールの引数は、具体化されている限り一度に通信してしまう。通信に用いられた変数が具体化されたら、変数と値との単一化ゴールを通信する。したがって、通信されるものは、遠隔指定されたゴールと未定義変数の具体化ゴールのみである。

# 第4章　予備実験

dKLIC を実装するのに使用する KLIC の機能を調べるため、前準備として、あるいは dKLIC の実装中に、いくつかの予備実験を行った。本章では、それらの予備実験について述べる。

第 4.1 節で実験を行った環境を示し、第 4.2 節で直列化処理の必要性を明らかにした実験、第 4.3 節でゴール通信およびメタ実行、第 4.4 節と第 4.5 節で変数を管理識別する必要性を明らかにした実験について述べる。

## 4.1　実験の環境

実験は、以下の環境で行った。

**記述言語** KL1 on KLIC-3.003
: KLIC に添付されているマニュアル [9] は、以前にリリースされた本体に対応している。つまり、本体の改変に対応しきれていない。第 3 章で述べたように、ヴァージョン独立を目指すが、対応の古いマニュアルに既に記されている機能ならば、使っても問題ないと判断した。

**OS/Machine** Linux on IBM-PC 互換機、または Sun OS on Workstation
: 第 3 章で述べたように、環境独立を目指すが、PC 互換機とワークステーションは、KLIC の動作する環境のうちで十分広い差があるものと思われる。従って、この両方で動作が確認できれば、目標とする汎用性、移植性は達成していると言えよう。

逆に、実験中に利用できない機能を発見したとき、実験環境に関わらず、その機能を使用することは避け、他の環境で利用できるか否か、特に比較実験を行うことはしなかった。従って、本章で「ある機能が利用できない」と言うのは、特定の環境下に限られたことかもしれない。

上記 2 つの環境にインストールされた KLIC-3.003 を記述言語に用いた。KLIC 付属のインストーラによるデフォルト・インストールを行ったが、PC 環境では、libm と libnsl だけがシステムにあり、libsocket は存在しなかった。

## 4.2 直列化処理の必要性

付録 C.3 は、クライアントがサーバに接続し、"Hello." という文字列を送って接続を切るプログラムである。

文字列や、文字コードによる入出力は、低レベル入出力と呼ばれる。この他に KLIC では、KL1 項による高レベル入出力機能が用意されているが、付録 C.3 のプログラムを変形して実験したところ、この機能をソケット通信では利用できないことが明らかになった。従って、ゴールを送受信する dKLIC は、KL1 項の直列化・復元を行う必要がある。

## 4.3 遠隔呼び出し

付録 C.4 は、クライアントがゴールを文字列化したと想定して送信し、サーバが受信した文字列からゴールを復元して実行するプログラムである。送受信されたゴールはファンクタの形を取っているが、第 2 章で述べたようにゴールは構文上ファンクタによって表されるので、これは妥当である。

ファンクタによって表されたゴールをメタ実行する機能が、KLIC の拡張機能 `generic:call/N`, `generic:apply/2` である。拡張機能を使うことは汎用性の目標とは相反するかもしれないが、ファンクタをゴールとして実行するには、この種の高階機能が必要である。KLIC における高階機能は拡張としてのみ用意されているので、この機能を使うことは必要であった。KLIC の拡張機能の使用をこれのみに留めることで、できるかぎり汎用性を維持することができる。

プログラムでは、 `call`, `apply` の両方について、述語定数、述語定数に具体化した変数、動的に生成した述語オブジェクトの 3 通りを与えて実験した。

```
1  "apply dynamic"
2  "call dynamic"
3  "apply variable"
4  "call variable"
5  "apply constant"
6  "call constant"
```

$2 \times 3 = 6$ 通りの全ての組み合わせが成功したことを、前述 2 つの実験環境 ( PC と WS ) で確認した。しかし、マニュアル [9] によると、述語オブジェクトの動的生成は環境によっては利用できないとのことなので、避けるべきだろう。

実際、述語オブジェクトの動的生成によってのみ呼び出されるモジュールを持つプログラムを書いて実験したところ、「モジュールが存在しない」という実行時エラーが起きた。これは、KLIC の下で利用されている GCC が最適化を行い、静的にどこからも呼ばれない C 関数 ( KL1 のモジュール ) を実行プログラムから取り除いてしまうためだと思われる。その証拠に、そのモジュールの静的呼び出しを付け加えた結果、述語の動的生成も成功した。GCC の最適化オプションは、KLIC のデフォルトインストールのままであり、以下の通りである。

```
-O2 -g -fomit-frame-pointer
```

以上の実験によって、ファンクタを送受信できれば遠隔ゴール実行が可能であることが分かった。既に述べたように、KLIC のソケットにはバイト列しか流せないので、ファンクタを送受信するには KL1 項を直列化しなければならない。

KL1 項を直列化するモジュールを実装し、実験を行ったところ、基底項 ( 変数を含まない項 ) に関しては問題なく直列化・復元が可能であることが分かった。付録 A.1 を参照せよ。

## 4.4 未定義変数の識別

未定義変数同士を識別できるかどうかの実験である。

未定義変数変数 X と、 X, Y, Z について、まず、それぞれ同定できるか否かチェックし、次に、それぞれ区別できるか否かチェックする。述語 gu(X,Y) は、引数 X,Y を同定 ( 単一化 ) できれば =(X,Y) と表示し、同定できなければ ~=(X,Y) と表示する。gu は guard unification の略である。逆に述語 gnu(X,Y) は、引数 X,Y を区別できれば \=(X,Y) と表示し、区別できなければ ~\=(X,Y) と表示する。詳細は、付録 C.1 を見よ。

結果は以下のようであった。

```
1  xyz(_B,_B,_7)
2  ~=(_B,_B)
3  ~=(_B,_B)
4  ~=(_B,_7)
5  ~\=(_B,_B)
6  ~\=(_B,_B)
7  ~\=(_B,_7)
```

_B, _7 というのは、変数 X,Y,Z の実行時の名前である。X=Y の単一化によって X,Y が同一の変数 _B となっているが、それにも関わらず、 X=X, X=Y という 2 つの単一化はいずれも成功しない。すなわち、KLIC/KL1 では同一の未定義変数を同定することができない。逆に、 X\=X, X\=Y, X\=Z という、「 単一化が失敗するという判断 」も成功しない。すなわち、「 異なる 」未定義変数同士を KL1 レベルで区別することもできない。

異なる変数を区別できないことは、適切な言語定義に従った仕様である。なぜなら、「 異なる 」変数は将来、単一化される同一の変数かもしれないからである。

しかし、同一の変数を同定できないことは、不適切であると思われる。実際、組込み述語 builtin:print/1 で、X,Y のいずれも同名 _B で表示され、同定されている。alternatively. を otherwise. に変えて実験しても、永久中断が発生するだけで、同一変数の同定はできなかった。

このことは、KLIC にかなり重大な機能欠落をもたらしており、目標とする処理系にもその影響が及ぶことになる。詳細については、第 4.5 節を参照せよ。

## 4.5 未定義変数の wrap/unwrap

第 4.3 節の実験によって、基底項で表せるゴールは遠隔実行可能であることが分かった。しかし、変数を含むゴールを遠隔実行できなければ、実用に耐えるとは言えない。

クライアント-サーバプロセスを KL1 で書くには、変数生成と束縛に時間差があるストリームの使用が必須である。未定義変数は実行途中の状態を表し、具体化された変数は処理の終了を表す。dKLIC では変数の具体化を通信するため、未定義変数を識別する必要がある。

以下の実験によって、KLIC では動的変数を識別できないことが明らかになった。dKLIC では動的変数の識別のため、変数表を実装しなければならなかった。

KLIC には、KL1 項を動的にメタ操作する `variable:wrap/2`, `variable:unwrap/2` という述語があり、この機能を使って変数を基底項で表すことができれば、変数を間接的に直列化して、変数を含むゴールを遠隔実行することができることになる。

付録 C.2 のプログラムは、標準入力から得た KL1 項を `wrap` して表示する。実行例を以下に示す。

```
1  Wrapped compile-time variable is variable("_")
2  Enter end_of_file. to exit!
3  Var.
4  Wrapped Var is variable("Var")
5  end_of_file.
6  Wrapped end_of_file is atom(end_of_file)
7  bye!
```

1 行目に表示されているのは、静的変数 `X` を `wrap` した結果である。静的名 `X` が消えて無名変数 `variable("_")` になってしまった。マニュアル [9] によると、`wrap` した変数名が消えるのはバグである。一方、3 行目で入力した変数 `Var` を `wrap` した結果は、4 行目の `variable("Var")` である。

もっと詳しく説明すると、高レベル入力ストリーム `klicio:klicio(Stream)` に対するメッセージ `getwt/1` は入力された KL1 項を `wrap` して返す。`variable:wrap/2` は全ての変数名を消してしまうが、`getwt/1` は変数名を文字列として読み込むので変数名は失われない。`variable:wrap/2` で変数名が失われてしまうのは、翻訳時に静的名が失われるからである。もし静的名を保っていたとしても、複数回呼び出される述語の中で `variable:wrap/2` が行われていた場合、同じ静的名を持つ複数の動的変数を識別することはできない。例えば、リストの要素を `wrap` して返す以下の述語を考えよう。

```
wrap_list(L, WL) :- L = [] |
  WL = [].

wrap_list(L, WL) :- L = [X|L1] |
  variable:wrap(X, WX),
  wrap_list(L1, WL1),
  WL = [WX | WL1].
```

この述語にリスト [X,Y] を与えると、以下の再帰ゴール列が生成される。

```
wrap_list([X,Y],WL), wrap_list([Y],WL1), wrap_list([],[])
```

variable:wrap(X,WX) が静的名 X を保っても、生成される

```
WL = [variable(X),variable(X)]
```

は間違っている。 X\=Y であるかもしれないからである。望みの処理は、動的識別名を与えることである。

さらに細かな実験をいくつか行なった。未定義変数を wrap すると無名変数になる。variable(String) を unwrap すると変数ができるが、これは

```
variable:unwrap(variable(String), Variable)
```

というゴールの第 2 引数 Variable そのままである。

getwt/1 メッセージで Variable と入力すると、基底項 variable("Variable") を得る。この基底項を putwtq/1 メッセージで出力すると、Variable と表示される。入出力で操作される「変数」は、入出力プログラムにとっては変数ではなく単なる文字列である。

KL1 項を入力するメッセージ gett/1 で変数 Variable を入力すると変数を得るが、これは gett(X) というメッセージの引数 X そのままである。puttq/1 メッセージで変数を出力しようとすると、中断して変数の具体化を待つ。

これらの結果から、動的な未定義変数は識別できないことが明らかになった。第 4.4 節で述べたように、KLIC では同一変数の同定ができない。unbound/2 組込み述語は、未定義変数に対してそのアドレスを返すが、ごみ集め言語機能によって変数のアドレスは変化するため、完全とは言えない。unbound/2 のアドレスによる判断が KLIC における変数識別の唯一の手がかりだが、目標処理系の汎用性を劣化させると判断し、今回は使用を見送った。

# 第5章　実装の詳細

本章では、dKLIC 実装の詳細について述べる。

ゴールを通信する dKLIC が直列化処理を行う必要があることは、第 4.2 節の実験で分かった。また、未定義変数とその具体化を通信するために、dKLIC が通信された未定義変数を管理し、識別名を与える必要があることは、第 4.4 節と第 4.5 節の実験で分かった。

実装の階層を図 5.1 に示す。

第 5.1 節で、最下層の直列化を実装し、第 5.2 節で、その上の変数表、第 5.3 節で高レベルソケット通信を実装する。

下 3 層の実行時ライブラリは実現できたが、最上層の遠隔実行は試作と、機械翻訳のアルゴリズムを示すにとどまった。試作については、第 6.1 節、機械翻訳のアルゴリズムは、第 6.2 節で示す。

## 5.1　直列化

第 4.2 節で述べたように、KLIC のソケットには文字列か文字コードしか流すことができない。KL1 項を流すには、KL1 項をバイト列に変換する直列化の処理が必要である。モジュール `type` は、任意の KL1 の型を直列化する処理を行う。付録 A.1 を参照せよ。

直列化と言っても、結果は暗号のようなバイト列ではなく、文字の列である。テストとデバッグの容易さのため、KL1 項をプログラムテキスト上の見た目と同じ文字の列に変換したかった。

最初の仕様では、このモジュールは任意の型と文字列型との変換機能を提供するものであった。モジュール名 `type` は、当モジュールが一般の型変換機能を提供することを示す名残である。

```
Layer 3:  Remote Predicate Call
Layer 2:  High Level Socket Communication
Layer 1:  Variable Table
Layer 0:  Marshalling
```

図 5.1: Layers in dKLIC Implementation

文字列型の値は、プログラムテキスト上で二重引用符で囲まれて表記される。一方、ソケットに流されたバイト列を全て二重引用符で囲みたくはなかった。プログラムがバイト列を文字列として受け取ったときに、初めてバイト列は文字列値として表記される。同様に、KL1 プログラムが 1 つの長い文字列として表記されないことを思い起こすと良い。

当初の仕様に沿ったモジュールを試してみて、文字列が二重引用符で囲まれないことに気づいた。文字列を直列化したバイト列は二重引用符に囲まれていなければならないが、文字列以外の型を直列化したバイト列に二重引用符は不要なので、文字列型と直列化の結果のバイト列とを区別することにした。

現在の仕様では、モジュール `type` は、任意の型と、文字の差分リストとの変換機能を提供する。

文字列でなければ文字であるのは、KLIC のソケットの制限である。

リストであるのは、KL1 の構造型のうち最も自然だからである。文字列は既に消去されたし、ベクタは要素数が固定である。ファンクタは、その特別な場合で便利な型がリストなのであるから選択肢に入らなかった。改めて理由を述べるまでも無く、KL1 で要素数不定の構造型が必要なときは普通、何らかの形でリストが用いられる。

差分リストを選択した理由は、文字コードを直接流すとその文字が出力されるという入出力ストリームの構文糖が利用できるからである。( `putc/1` の代わりに、引数の文字を直接流せる。) 普通のリストだと、直列化した KL1 項を 1 つ流したとき、リストの終わりに到達して入出力ストリーム全体が終了してしまう。一方、差分リストを用いると、前の項の尾に次の項の頭を繋げて任意個数の KL1 項を流すことができる。

## 5.2 変数表

変数表とは、KLIC の変数を識別するために、識別名と KLIC 変数とを対応づける辞書である。

第 4.4 節、および第 4.5 節の実験から、KLIC では変数の識別ができないことが分かった。しかし、目標処理系では並行論理型言語最大の特徴である論理変数を扱う必要がある。値をとりあえず未定義変数としておき、後で具体化することが

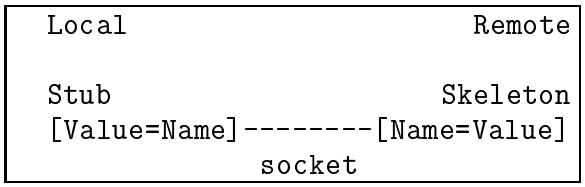


図 5.2: Variable Tables

```
append(X,Y,Z) :- X = [] |
  Z = Y.
append(X,Y,Z) :- X = [E|X1] |
  append(X1, Y, Z1),
  Z = [E | Z1].
```

図 5.3: Predicate append/3

できなければ、クライアント-サーバ型通信の依頼 ( 変数 )–返事 ( 具体値 ) というパターンを踏むことができない。

このため、遠隔ゴールだけではなく、変数の具体化ゴールを通信する必要がある。変数の具体化には、「どの変数」を「どんな値」に具体化するという情報が必要である。これは、 Variable=Value という具体化ゴールの形から明らかである。相手から受け取った変数を覚えておき、後に具体化される変数を識別するため、変数表を実装した。付録 A.2 を参照せよ。

変数表は、ソケットの両端に貼り付ける形で使用される。両端のノードで使用される内部変数は互いに異なる実体だが、外部変数 ( 名前 ) と内部変数 ( 値 ) の対応づけを変数表に登録するのである。図 5.2 を見よ。

ソケットには変数の識別名が流れる。高レベルソケット通信で変数を流そうとすると、変数表を参照して識別名を流し、識別名を受け取ると対応する KLIC 変数を扱う。

### 5.2.1 変数表の設計

このモジュールで扱う変数は単一代入ではなく手続き型言語の変数であった。すなわち、変数の値を「変える」機能が提供されていた。その理由は、通信に用いられるストリームの cdr 連鎖を同名で ( 1 つの変数として ) 管理しようと考えたためである。この方式は、cdr 連鎖を引数に持つ再帰ゴール列に適合し、変数名の動的生成が不要であるという利点がある。

例として、append/3 サーバを挙げる。図 5.3 を見よ。

このサーバに、 X=[a,b] を与えると、以下のような再帰ゴール列が生成される。

```
append([a,b],Y,Z), append([b],Y,Z1), append([],Y,_Z1)
```

第 3 引数 Z, Z1, _Z1 はそれぞれ異なる変数であるが、1 つの通信路であると見なして静的名 Z, Z1 で統一することができる。

クライアントが X=[a,b] を徐々に具体化するものとしよう。通信されるべきゴールは以下のようである。

```
Local                    Remote

append(X,Y,Z) ->
X = [a|"X1"]  ->
              <- Z = [a|"Z1"]
X = [b|"X1"]  ->
              <- Z = [b|"Z1"]
X = []        ->
              <- Z = Y
```

"X1", "Z1" は「次の」X, Z を表している。まず、append(X,Y,Z) が送信され、以降、X の要素が具体化、送信されるたびに Z の要素が具体化されて返ってくる。

既に述べたように、実用的な分散処理の多くがストリーム通信の形で書ける。しかし、この方式を完全に利用できるためにはプログラムの解析が必要である。前掲の例でも、再帰と cdr 連鎖を発見できなければならないし、さらに複雑な場合は、相互再帰述語の極小集合や変数の別名付けを発見し、互いに異なる変数名を与えなければならない。静的にではあるが、いずれにせよ変数名の生成は必要なのである。

実現の容易さを優先して、今回の実装では静的名を完全に無視し、動的に変数名を生成することにした。KLIC の variable:wrap/2 述語は全ての変数に対して動的無名 _ を生成するが、これでは同一変数の同定ができない。当モジュールでは、builtin:print/1 述語に倣って、下線に整数が続く名前を生成することにした。すなわち、append に対して図 5.4 のようである。

動的に変数名を生成することを選んだため、変数名の使い回しは起こらず、変数の値を変える機能は不要となった。従って、当モジュールで変数表に登録される変数は単一代入で、KL1 と同じ論理変数である。

新しい変数名を生成するために用いる数字の増分は、変数名生成のときだけでは足りない。なぜなら、変数表はソケットの両端にあり、相手側で生成された変数名との重複が起こってしまうからである。そこで、相手から新しい変数を受け取るときにも数字を増やすことにした。append の例では、append(_0,_1,_2), _0=[a|_3] を受け取った遠隔ノードは、新しい変数名として _4 を生成しなければならない。

```
append(_0,_1,_2) ->
_0 = [a|_3]      ->
                 <- _2 = [a|_4]
_3 = [b|_5]      ->
                 <- _4 = [b|_6]
_5 = []          ->
                 <- _6 = _1
```

図 5.4: Communication for append/3

動的に変数名を生成することの利点として、静的解析が不要であることがあった。これは、処理系が扱うプログラムの変形は最小限であることを意味する。プログラム変形を最小限に留めるため、スタブ-スケルトンモデルが採用された。すなわち、遠隔呼び出しをスタブ呼び出しに翻訳して、スタブが遠隔のスケルトンと通信し、スケルトンが目的の述語を呼び出すのである。呼び出される述語の変形は不要である。

### 5.2.2 変数表の実装

実装した変数表を詳説する。

変数表は、ファンクタ（`Variable=Value`）を変数名で逆順に整列したリストを引数に持つ再帰ゴール列として実装した。

再帰ゴール列であるのは、変数表が時間によって変化するデータ構造であるためである。第 5.2.1 項で述べたように、変数は単一代入で不変なのだが、変数表自体は動的に新しい変数が登録される可変データ構造である。論理型言語に限らず、純関数型言語など単一代入変数を持つ言語で、時間変化するデータを扱うには、変化した値の時系列を引数に持つ再帰呼び出し列が使われる。

リストであるのは、登録される変数の個数が不定であるためである。整列されているのは、目的の変数を見つける手間を軽くするためである。整列によって平均 2 倍の探索速度が見込まれる。さらに、整列された他の表との比較、合併、共通部分の発見は、整列されていない場合の $O(N^2)$ に対して $O(N)$ になる。当モジュールでは手を抜いて、未定義変数表を整列しないが、これを整列することによって大幅に性能が上がる可能性がある。しかし実際には次に述べる理由から、この可能性に期待せず、手を抜くという選択をした。

逆順に整列するのは、第 5.2.1 項で述べたように変数名が整数であるためである。リストに要素を追加するとき、尾に追加するにはリストを総なめして $O(N)$ の手間がかかるのに対して、頭に追加するには新しいコンス・セルを作る $O(1)$ の手間で済む。整数の増分を使って生成された新しい変数名は、それ以前に生成された変数名より順番が後である。この新しい変数を変数表に登録するために、正順に整列されていれば $O(N)$、逆順ならば $O(1)$ の手間がかかる。このことから、変数の登録は実際には $O(1)$ であると期待されるため、前述の未定義変数表は整列しないという手抜きが選択された。

同一変数の同定ができないため、実装された変数表は巨大になり易い。ある変数を送信するたびに、新しい名前を登録するためである。

## 5.3 KL1 項の送受信

ここで、ソケットに被せて任意の KL1 項を流せるようにするモジュール vsock を実装する。付録 A.3 を参照せよ。

理想的には、直列化は KLIC が提供すべき機能である。実際、標準入出力で高レベルの KL1 項を扱う機能が提供されている。高レベル入出力が、当モジュールのように低レベル入出力に被せる形で提供されるか、あるいは、高レベル入出力の代わりに完全な直列化機能が提供されるだけでも、直列化モジュールおよび当モジュールによる高レベルソケット通信の実装は不要であった。

以上の理由と、実現の容易さの目標から、モジュール type の直列化および当モジュール vsock による復元は、完全な KL1 の文法に従っていない。第 2.2 節でも省略された、十進以外の整数や特殊なファンクタの記法を受け付けない。ソケット通信は人目に触れる必要がないので、type で直列化して vsock で復元するのには問題ない。

直列化の結果が、プログラムテキスト上に現れるものと同じ文字の列であったから、復元には、KL1 プログラムを読解する技法が必要である。実現の容易さのため、当モジュールでは $LL(1)$ が選択された。$LR(k)$ や $LL(k), (k > 1)$ は、実現の容易さに反する。$LL(0)$ では、アトムとファンクタを区別することができないし、少なくとも 1 文字先読みしなければアトムや整数の終わりも判断できない。

直列化と復元を分けた理由は、やはり実現の容易さである。直列化は、言わば型変換であり、外部との通信が不要な独立の処理である。一方、復元は $LL(1)$ のために外部から同期読み込みを行わなければならない。

当モジュールは、高レベル入出力メッセージを受け取ると、低レベル入出力メッセージに翻訳して、被さった入出力ストリームに流す。低レベル入出力メッセージは、そのまま入出力ストリームに流す。変数表を貼り付けてあるため、変数表に対するメッセージは変数表に渡す。

当モジュールを標準入出力に被せてテストしたところ、標準入力でバッファリングが行われていないようだったので、先読みした文字を ungetc/1 で戻すのではなく、内部状態として覚えておく処理を付け加えた。ソケットでバッファリングが行われているか否か調べてはいないが、それに関わらず当モジュールを使うことができる。

変数表が時間変化する逐次プロセスであるため、変数表を参照する当モジュールの提供するソケット通信も非同期に行えない。このため、KLIC のソケット入出力が非同期に行えないという問題は、解決する必要がなくなった。変数表の並列化は今後の課題である。

# 第6章　dKLICの評価

本章では、実装したdKLICの評価、分析と考察を行う。

dKLICの構成について、図6.1を見よ。

第6.1節で実行時ライブラリを用いた遠隔実行の例、第6.2節で機械翻訳のアルゴリズムを示す。

第6.3節で実装方針の評価、第6.4節と第6.5節で実装した変数表の評価を行う。

## 6.1　遠隔実行の例

実装を進めてきた実行時ライブラリを用いて、append/3 の遠隔実行を行うプログラム例を挙げる。

ここに挙げる例は、dKLICのコンパイラが出力すべき翻訳結果である。コンパイラ自体は時間の制約から、まだ実装されていない。本節では、dKLICコンパイラが図6.2のdKL1プログラムを読んで、付録BのKLIC/KL1プログラムを出力したものと想定しているが、付録BのKLIC/KL1プログラムはかなり恣意的にハンドコンパイルされており、機械翻訳の結果とは言い難い。機械翻訳のアルゴリズムについては、第6.2節を見よ。

当初の構想では、通信の必要がある遠隔ゴールと具体化ゴールのいずれも、直列化によってファンクタとして送受信するつもりだった。しかしプログラム例によって、遠隔ゴールの後に具体化ゴールが続くというパターンが明らかになり、具体化ゴールを理解するのは変数表だけで十分であることが分かった。

そこで、変数表に以下の機能が追加された。未定義変数表は、登録時に未定義だった変数を覚えておくデータ構造である。bound/1 メッセージは、未定義変数表のうち具体化された部分表を返す。bind/1 メッセージで具体化された部分表を与えると、変数表をそれに従って具体化する。

ローカルで bound された具体化変数表を、遠隔に通信して bind すればよい。このため、具体化ゴールの通信を具体化表の通信に置き換えた。これを実装したものが、付録Bである。

ここで用いた通信プロトコルは、以下の通りである。まず、クライアントが append(_0,_1,_2) を送信し、サーバから具体化表を受信する。続いて、クライアントが _0,_1 の要素を徐々に具体化するたびに、サーバとの間で具体化表をやり取りすることを繰り返す。図6.3を見よ。

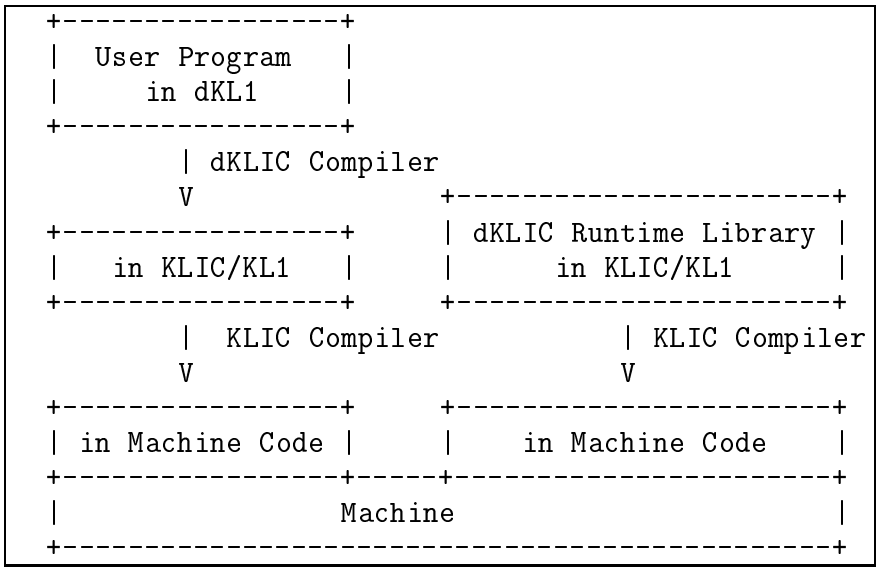


図 6.1: dKLIC Configuration

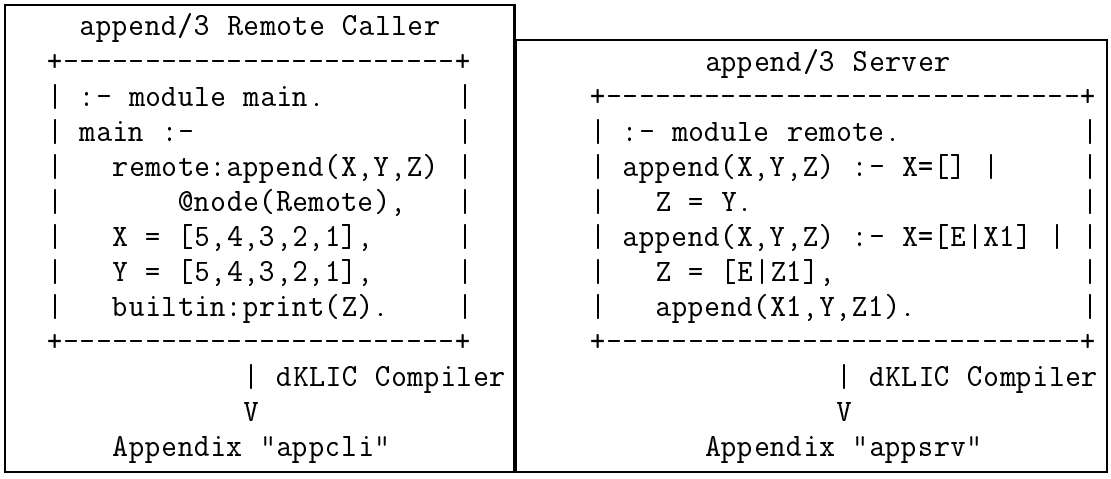


図 6.2: Remote append/3 Call in dKL1

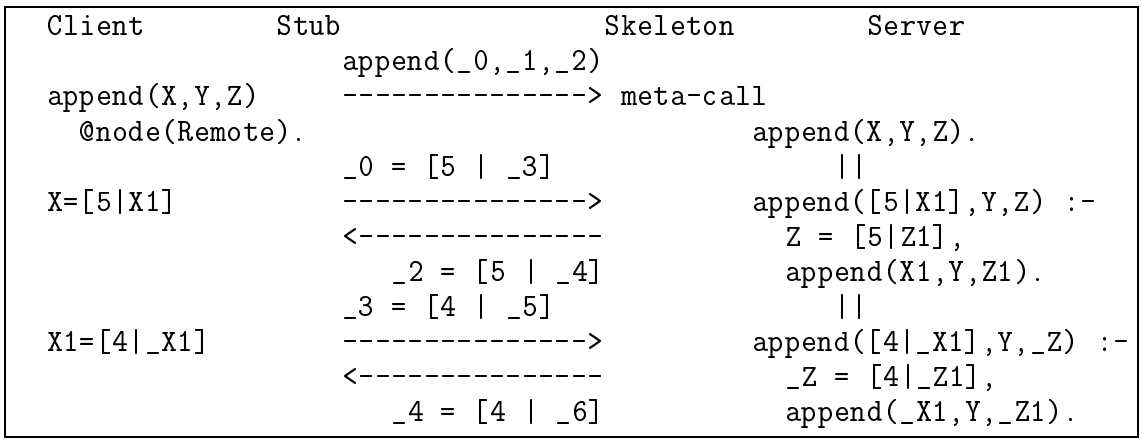


図 6.3: Communication and Reduction for append/3

この例では人手による翻訳で、_0,_1 の要素の具体化と具体化表の送信を同期させているのだが、一般の遠隔述語呼び出しで同期を行うには静的解析が必要になる。実現の容易さを保つために静的解析を導入しないという実装方針を採ったのであるから、機械翻訳にはもっと単純な通信プロトコルが必要である。第 6.2 節で機械翻訳のアルゴリズムと通信プロトコルを示す。

## 6.2 機械翻訳アルゴリズム

本節で、一般の遠隔述語呼び出しに対する機械翻訳アルゴリズムを示す。翻訳アルゴリズムを決める前に、通信プロトコルを決めなければならない。

第 6.1 節で、変数の具体化と具体化表の送信を同期させていたのは、具体化表を通信しようとしたら既に通信路が閉じていたといった事態を防ぐためであるから、実際には通信の終わりを検出できるだけで良い。append の例で明らかなように、通信の終わりには全ての変数が具体化されている。よって、未定義変数がなくなったことを通信の終了条件とすることができる。

通信の途中で未定義変数が無くなることはない理由を述べる。クライアント-サーバ間のメッセージストリームは、通信が続く限り閉じることはない。append/3 の第 3 引数が cdr 連鎖を持つことを思い起こすと良い。全てのストリームが閉じたら、通信が終わったことを示す。全ての変数が具体化されたら、それ以降の通信には新しい未定義変数が必要である。従って、通信の途中で未定義変数が無くなることはない。

通信が終われば未定義変数は無くなることが望ましい理由を述べる。決して具体化されない変数の具体化を待つゴールは永久中断され、実行時エラーを起こす。具体化を待つゴールがある変数は、いずれ具体化されるべきである。同様に、決して具体化されない変数を通信すべきではないし、通信された変数が具体化され

ないのは、おそらくバグである。従って、通信が終われば未定義変数は無くなるべきである。

通信プロトコルが決まったので、このプロトコルに従う翻訳アルゴリズムを示す。

図 6.4 のような遠隔呼び出しを行う dKL1 クライアントを考える。

このような遠隔呼び出しは、図 6.5 のスタブを呼び出すゴールに置き換えれば良い。

このスタブは、最初に遠隔ゴールを送信し、それ以降は通信終了まで具体化表の送受信を繰り返すものである。`END_COMM` は全ての変数が具体化済みで、通信が終了することを意味する。`Bindings` は具体化表で、通信が続くことを意味する。

対応するサーバ側のスケルトンは、図 6.6 のようになる。このスケルトンは、最初に遠隔ゴールを受信してメタ実行し、それ以降は通信終了まで具体化表の送受信を繰り返す。呼び出される述語を変形する必要はなく、そのままで良い。

機械翻訳のアルゴリズムを示すことができたが、コンパイラを実装するには、通信の終了条件を検出できるように実行時ライブラリを改造しなければならない。

## 6.3 実装の方針

最低目標であった遠隔ゴール実行が実現したところで、本処理系の実装の方針から詳細までを評価してみる。

実現不可能な良い言語よりも実現可能な言語の方が役に立つという観点から、実現の容易さの目標は妥当であったと考える。GHC が並行論理型言語族の内でも最も単純な言語であったこと、継ぐ KL1 が比較的、低レベルの言語であること（Prolog が GHC に翻訳できることを思い起こすと良い）、KLIC が低レベルの並列制御を提供していることなど、この目標が KL1 言語の流れにおいて自然であることを示す傍証をいくつか挙げられる。

実装の階層を設定し、分散 KL1 の極小部分集合として遠隔ゴール実行を選んだのは良い選択であった。真の分散言語と言えるためには、他に少なくとも述語の送受信が必要であろうが、これは明らかに遠隔ゴール実行を利用して実装可能である。いま思い付く最も簡単な実装方法は、プログラムテキストを送受信して、遠隔でファイルに落とし動的にコンパイル・実行するというものである。

もちろん、遠隔実行層が述語通信層の下にある必要はないのだし、すべてをフラットに実装してもかまわないのだが、フラットな実装は実現の容易さの目標に反する。複雑な問題は独立の部分問題に分けて考えるというのが問題解決の原則

```
modu:pred(X,..) @node(Remote).
```

図 6.4: General Remote Call

```
stub(X,..) :-
  get_host_and_port(Remote, Host, Port),
  unix:unix([connect(inet(Host, Port), normal(S))]),
  socket:socket(Socket, S),
  Socket = [puttq(modu:pred(X,..)), 0'., nl | Socket1]
  receive(Socket1).

receive(Socket) :-
  Socket = [fflush(0), gett(AtRemote) | Socket1],
  receive(AtRemote, Socket1).

receive(AtRemote, Socket) :- AtRemote = END_COMM |
  Socket = [].
receive(AtRemote, Socket) :- AtRemote = Bindings |
  Socket = [bind(AtRemote), bound(AtLocal) | Socket1],
  send(AtLocal, Socket1).

send(AtLocal, Socket) :- AtLocal = END_COMM |
  Socket = [puttq(AtLocal), 0'., nl, fflush(0)].
send(AtLocal, Socket) :- AtLocal = Bindings |
  Socket = [puttq(AtLocal), 0'., nl | Socket1],
  receive(Socket1).
```

図 6.5: General Stub

```
:- module main.

main :-
  Port = 10000,
  unix:unix([bind(inet(Port), normal(SS)),
  SS = [accept(normal(S))],
  socket:socket(Socket, S),
  Socket = [gett(Call) | Socket1],
  skeleton(Call, Socket1).

skeleton(Call, Socket) :- Call = (modu:pred(X,..)) |
  generic:call(predicate#(modu:pred/N), X,..),
  Socket = [bound(Here) | Socket1],
  send(Here, Socket1).

send(Here, Socket) :- Here = END_COMM |
  Socket = [puttq(Here), 0'., nl, fflush(0)].
send(Here, Socket) :- Here = Bindings |
  Socket = [puttq(Here), 0'., nl, fflush(0),
            gett(There) | Socket1],
  receive(There, Socket1).

receive(There, Socket) :- There = END_COMM |
  Socket = [].
receive(There, Socket) :- There = Bindings |
  Socket = [bind(There), bound(Here) | Socket1],
  send(Here, Socket1).
```

図 6.6: General Skeleton

である。

遠隔実行層が述語通信層より上にあるべきでない理由は、述語通信だけ実装された場合を考えれば明らかである。遠隔実行が実装されていない言語は分散言語とは呼びづらいからである。遠隔実行と述語通信が同じ階層で独立に並んでいてもかまわないが、少なくとも遠隔実行を実装することが分散言語と呼べる必要条件であった。

述語通信は、遠隔にない述語を遠隔実行するための高級機能なのである。遠隔実行、述語通信ともに実装されている場合は、述語通信を利用して任意の述語の遠隔実行を行う高級遠隔実行層も考えられる。

## 6.4 動的変数名生成

動的に変数名を生成すること、および生成された変数名については、KLIC の builtin:print 述語や、プログラムトレースで表示される変数名から妥当なものであると考えるが、いま述べた 2 つの例はともに、デバッグ用の機能であると思われる。動的な変数の識別 ( 従って動的変数名 ) の必要性は、主にデバッグのためなのであろう。

特定のクラスのプログラムは、静的解析による最適化を施すことができる。第 5.2 節で挙げた append/3 述語の例では、末尾再帰と、入出力モード解析による変数の最適化が可能であった。

静的解析による最適化の導入は今後の課題である。静的解析を導入した場合、変数表の意味が大きく変わる可能性がある。デバッグのための必要性がなければ動的変数名と変数表は不要になるかもしれない。

静的解析によって、プログラムテキスト上の静的変数と、動的に生成される変数との意味が大きく異なるものになる可能性がある。KLIC がユーザプログラムに動的未定義変数の識別を許さないことは、静的解析による最適化の観点からは妥当なものであったと言えよう。本処理系のように、動的未定義変数を識別したいシステムプログラムは、KLIC の舞台裏に潜り込む必要がある。

## 6.5 変数名と変数表のデータ型

実装における変数名の生成と変数表のデータ型は全く不適切であった。最大の原因は、新しく生成すべき変数名は整数型であるのに、変数表に登録された変数名は文字列型であるという、データ型の不統一である。

変数表に登録された変数名も整数型であれば、整列された変数表における変数名の比較がもっと安価になった。

また、実行時ライブラリを分散 KL1 言語処理系とは独立に変数を理解する標準入出力として用いる場合、現在の方式では _N 以外の変数名に対しても無差別に

整数を増分してしまうので大きな数の名前ができやすいが、整数型の変数名を扱えば、もっと適切に変数名を生成できるかもしれない。整数型の大きな利点は、新しい整数を生成する方法が豊富に知られていることである。

逆に、文字列型で統一したとしても、利点が考えられる。文字列型は、整数型よりも汎用性が高い。文字列で整数を表すことはできるが、整数で任意の文字列を表すことはできない。整数型は最大個数が決まっているが、文字列型の変数名は無限に生成できるかもしれない。

# 第7章　今後の課題

本章では、論文についてまとめ、今後の課題を述べる。

## 7.1　まとめ

本研究では、dKLIC の最低条件を遠隔ゴール実行と定め、KLIC/KL1 によって実装を行った。

分散 KL1 言語処理系の先駆けである KL1j が Java で実装されているのに比較して、KL1 によって実装を行ったことで Java の不具合に悩まされずに済んだ。代わりに KLIC の不具合に悩まされることになったが、この経験は KL1 を深く理解するために非常に役立ったと考えている。

分散 KL1 の極小部分集合を遠隔ゴール実行と定めたことの妥当性は、第 6 章で論じた。実装の階層を適切に定めたことによって、短期間で遠隔実行を実現することができた。また、遠隔実行層の上に単純な述語通信層を設けることができることも述べた。

機械翻訳のアルゴリズムは、第 6.2 節で示した。コンパイラは、本論文には間に合わなかったが、近いうちに実現可能である。Java よりも分散 KL1 に近い KLIC を実装言語に選んだために、コンパイラと言うよりもトランスレータと言うべき処理系で十分であろう。

## 7.2　機械翻訳

遠隔呼び出しの構文は第 2.6 節で示した。また、機械翻訳のアルゴリズムについては第 6.2 節で述べた。

しかし、呼び出される述語は、どのように書けば良いのだろうか？実現の容易さを追求した結果、呼び出される述語に手を加える必要はなくなった。しかし、遠隔呼び出しされる述語が、遠隔呼び出しされない述語と全く同じ形をしているならば、どこでその区別をすれば良いのか？スケルトンを生成すべき述語はどれか？

現在の予定では、モジュール宣言に似た構文を用いて、遠隔呼び出しされる述語をプログラマに宣言させるのが良いと考えている。例えば、 append サーバは以下のようになろう。

```
:- module server.
:- remote( (append/3, append/4) ).

append(X,Y,Z) :- ...

append(A,B,C,D) :- ...
```

もっと単純な実装としては、全ての述語に対してスケルトンを生成するという案がある。しかし、セキュリティの観点から、任意の遠隔述語を呼び出せることは望ましくないと考える。また、このことは実行プログラムを不必要に巨大化させてしまう。

遠隔呼び出しされる述語を特定できる場合、generic:call/N によるメタゴール実行は不要である。メタゴールを受信したときに述語を呼出すゴールを、静的に書けばよい。スケルトンは以下のようになる。

```
skeleton(Call, Socket) :- Call = (modu:pred(X,..)) |
  modu:pred(X,..),  %%% <-- static call
  Socket = [bound(Here) | Socket1],
  send(Here, Socket1).
```

## 7.3 未定義変数表の整列

未定義変数表の実装では手を抜いて整列を行わないことにしたが、やはり整列によって大きく性能が上がる可能性がある。実装方式の違いによる比較定量実験は今後の課題である。

## 7.4 変数表の並列化

変数表の並列操作は今後の課題である。変数表を並列操作できなければソケット通信の非同期化は難しい。現行の処理系では並列化の必要はないのだが、並列操作できない言語機能は、並列言語としての KL1 の利点を殺してしまう。

本処理系が採った動的変数名生成によって、変数表は変数の束縛が決定的 (deterministic) であるという望ましい単一代入性を持ち、並列化のための解析に向いていると思われる。しかし、静的解析を導入することによって、変数表が不要になるかもしれないし、逆に単一代入性を失ってしまうかもしれない。

変数表のデータ型をリストからベクタに変えるだけで、並列操作の可能性が見えるかもしれない。

## 7.5 その他の遠い目標

要求に応じた具体化ゴールの通信ができると良い。dKLIC では変数の具体化を eager に通信しているが、これではノードの透明性という、並行言語としての KL1 の利点を殺してしまう。lazy にオンデマンドで変数の具体化を通信できれば、通信量が減り、処理が速くなる可能性がある。

遠隔プロセスの識別ができると良い。現在の dKL1 では、厳密にはサーバを書くことができず、サーバとして働く述語を書くことができるだけである。ftpd や telnetd のような常時待機するサーバを書くためには、クライアントがサーバプロセスを識別し通信できなければならない。KL1j [12] では、ストリームの名前付け機能を提唱している。これによって、多対多の分散処理を記述することができるようになる。

以上で本論を終える。

# 謝辞

論文の書き方を丁寧に指導して下さいました上田和紀先生、面白い研究の種を下さいました五十嵐宏氏に感謝致します。

分っているつもりだが意識していなかった変数の生成について、的確な助言を頂きました加藤紀夫氏に感謝致します。

最後に、KLIC 作成グループの方々に感謝致します。KLIC で変数の識別が可能であれば、dKLIC の実現は間単すぎて論文にならなかったでしょう。

# 参考文献


[1] Ueda, K., Guarded Horn Clauses. ICOT Tech. Report TR-103, ICOT, Tokyo, 1985. Also in *Logic Programming '85*, Wada, E. (ed.), LNCS 221, Springer-Verlag, Berlin, 1986, pp. 168–179.

[2] Ueda, K. and Chikayama, T. Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, Vol. 33, No. 6 (1990), pp. 494–500.

[3] Chikayama, T., Fujise, T. and Sekita, D., A Portable and Efficient Implementation of KL1. In *Proc. 6th Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'94)*, LNCS 844, Springer-Verlag, Berlin, 1994, pp. 25–39.

[4] Kazunori Ueda, Concurrent Logic/Constraint Programming: The Next 10 Years. In The Logic Programming Paradigm: A 25-Year Perspective, K. R. Apt, V. W. Marek, M. Truszczynski, and D. S. Warren (eds.), Springer-Verlag, 1999, pp.53-71.

[5] 淵一博 監修, 古川康一 溝口文雄 共編, 古川康一 竹内彰一 上田和紀 他 共著: "並列論理型言語 GHC とその応用", 共立出版, 1987, ISBN4-320-02266-1.

[6] KLIC 講習会テキスト "KLIC システム編", "KL1 言語編", 新世代コンピュータ技術開発機構 ( 平成 5 年 ), 日本情報処理開発協会開発研究室 ( 平成 7 年 ).

[7] ICOT: URL "http://www.icot.or.jp/".

[8] KLIC 作成グループ: KLIC-3.003, 1999, URL "http://www.klic.org/".

[9] KLIC 作成グループ: KLIC 添付のマニュアル "KLIC.info".

[10] 関田大吾: "Inside KLIC Version 1.0", 1998.

[11] 倉持聡: 早稲田大学理工学研究科情報学修士論文 "KLIJava: KL1 から Java へのコンパイラおよび実行環境", 1999,
URL "http://www.ueda.info.waseda.ac.jp/~satoshi/klijava/".

[12] 五十嵐宏: 早稲田大学理工学部情報学科卒業論文 “分散 KL1 言語処理系の設計と実装”, 1999,
URL "http://www.ueda.info.waseda.ac.jp/~igarashi/".

[13] P. Deransart, A. Ed-Dbali, L. Cervoni: “Prolog: The Standard” Reference Manual, Springer-Verlag, 1996, ISBN 3-540-59304-7.

[14] Andrew W. Appel with Maia Ginsburg: “Modern Compiler Implementation in C”, Cambridge University Press, 1998.

# 図 目 次

3.1 Stub-Skeleton Framework for Remote Predicate Call . . . . . 10

5.1 Layers in dKLIC Implementation . . . . . 17
5.2 Variable Tables . . . . . 18
5.3 Predicate append/3 . . . . . 19
5.4 Communication for append/3 . . . . . 20

6.1 dKLIC Configuration . . . . . 24
6.2 Remote append/3 Call in dKL1 . . . . . 24
6.3 Communication and Reduction for append/3 . . . . . 25
6.4 General Remote Call . . . . . 26
6.5 General Stub . . . . . 27
6.6 General Skeleton . . . . . 28

# 付 録A　実行時ライブラリ

本章では、dKLIC 処理系のうち実装が進んだ実行時ライブラリを示す。詳細については、第 5 章を参照せよ。

## A.1　type

```
/*************************************************************
* Type Conversions
*
* by Yusuke TAKAGI (mailto:takagi@ueda.info.waseda.ac.jp)
* 1999/12/25
*
* Types:
*  any       Any Terms
*  variable  Unbound Variables (< any)
*  bound     Bound Values      (< any)
*  atomic    atom + int        (< bound)
*  atom      Symbolic Atoms    (< functor)
*  int       Integers          (< bound)
*  float     Floating Points   (< bound)
*  cons      Cons Cells        (< functor)
*  functor   Functors          (< bound)
*  vector    Vectors           (< bound)
*  string    Strings           (< bound)
*  list      Lists             (< any)
*  diff      Difference Lists  (< functor)
*  chars     Character Lists   (< diff)
*
* Conversions:
*  bound     -> chars
*  atom      -> int, string
*  int       -> float
*  float     -> int
*  functor   -> vector, string, list, diff
*  vector    -> string, list, diff
*  string    -> atom, vector, list, diff
*  list      -> functor, vector, string, diff
*  diff      -> functor, vector, string, list
*
*-------------------------------------------------------------
*
* Naming Variables:
*  A : atom
*  B : bound
*  C : char
*  D : diff       (H, T)
```

```
41  *  F : functor    A(any*)
42  *  H : list       (Head of Difference List)
43  *  L : cons, list [any | any]
44  *  R : float      (Real Number)
45  *  S : string     "char"
46  *  T : list       (Tail of Difference List)
47  *  U : any        (Unknown)
48  *  V : vector     {any*}
49  *  X : variable
50  *  Z : int        (Zahl)
51  *  _s: list       (List of _-s)
52  *
53  * Type Tree:
54  *  any
55  *   * bound
56  *      + int, float, vector, string
57  *      + functor
58  *         - atom, cons
59  *         - diff
60  *            . chars
61  *   * variable
62  *   * list (= variable + {[]:atom} + {[U|L] | U:any, L:list})
63  ************************************************************/
64
65  :- module type.
66
67  :- with((CSize = 8)).  % Size of Character
68
69
70  /************************************************************
71  * -> atomic +B -A
72  ************************************************************/
73  atomic(B, A) :- atomic(B) |
74    A = B.
75  atomic(B, A) :- float(B) |
76    float_int(B, A).
77  atomic(B, A) :- string(B, _, CSize) |
78    string_atom(B, A).
79
80  /************************************************************
81  * -> atom +B -A
82  ************************************************************/
83  atom(B, A) :- atom(B) |
84    A = B.
85  atom(B, A) :- string(B, _, CSize) |
86    string_atom(B, A).
87
88  /************************************************************
89  * -> int +B -Z
90  ************************************************************/
91  int(B, Z) :- atom(B) |
92    atom_int(B, Z).
93  int(B, Z) :- integer(B) |
94    Z = B.
95  int(B, Z) :- float(B) |
96    float_int(B, Z).
97
98  /************************************************************
```

```
 99  * -> functor +B -F
100  ************************************************************/
101  functor(B, F) :- atom(B) |
102    F = B.
103  functor(B, F) :- list(B) |
104    list_functor(B, F).
105  functor(B, F) :- functor(B,_,_) |
106    F = B.
107  functor(B, F) :- vector(B,_) |
108    vector_functor(B, F).
109
110  /************************************************************
111  * -> vector +B -V
112  ************************************************************/
113  vector(B, V) :- atomic(B) |
114    atomic_vector(B, V).
115  vector(B, V) :- float(B) |
116    float_vector(B, V).
117  vector(B, V) :- list(B) |
118    list_vector(B, V).
119  vector(B, V) :- functor(B,_,_) |
120    functor_vector(B, V).
121
122  /************************************************************
123  * -> string +B -S
124  ************************************************************/
125  string(B, S) :- atom(B) |
126    atom_string(B, S).
127  string(B, S) :- integer(B) |
128    int_string(B, S).
129  string(B, S) :- float(B) |
130    float_string(B, S).
131  string(B, S) :- list(B) |
132    list_string(B, S).
133  string(B, S) :- vector(B,_) |
134    vector_string(B, S).
135  string(B, S) :- string(B, _, CSize) |
136    S = B.
137
138  /************************************************************
139  * -> list +B -L
140  ************************************************************/
141  list(B, L) :- atomic(B) |
142    atomic_list(B, L).
143  list(B, L) :- float(B) |
144    float_list(B, L).
145  list(B, L) :- list(B) |
146    L = B.
147  list(B, L) :- functor(B,_,_) |
148    functor_list(B, L).
149  list(B, L) :- vector(B,_) |
150    vector_list(B, L).
151  list(B, L) :- string(B, _, CSize) |
152    string_list(B, L).
153
154  /************************************************************
155  * -> diff +B -H ?T
156  ************************************************************/
```

```
157 diff(B, H, T) :- atomic(B) |
158   atomic_diff(B, H, T).
159 diff(B, H, T) :- float(B) |
160   float_diff(B, H, T).
161 diff(B, H, T) :- list(B) |
162   list_diff(B, H, T).
163 diff(B, H, T) :- functor(B,_,_) |
164   functor_diff(B, H, T).
165 diff(B, H, T) :- vector(B,_) |
166   vector_diff(B, H, T).
167 diff(B, H, T) :- string(B, _, CSize) |
168   string_diff(B, H, T).
169
170 /*****************************************************************
171 * -> chars +B -H ?T
172 *****************************************************************/
173 chars(B, H, T) :- atom(B) |
174   atom_chars(B, H, T).
175 chars(B, H, T) :- integer(B) |
176   int_chars(B, H, T).
177 chars(B, H, T) :- float(B) |
178   float_chars(B, H, T).
179 chars(B, H, T) :- list(B) |
180   cons_chars(B, H, T).
181 chars(B, H, T) :- functor(B,_,_) |
182   functor_chars(B, H, T).
183 chars(B, H, T) :- vector(B,_) |
184   vector_chars(B, H, T).
185 chars(B, H, T) :- string(B, _, CSize) |
186   string_chars(B, H, T).
187
188
189 /*****************************************************************
190 * atomic
191 *  -> vector, list, diff
192 *****************************************************************/
193 atomic_vector(A, V) :- atomic(A) |
194   V = {A}.
195
196 atomic_list(A, L) :- atomic(A) |
197   L = [A].
198
199 atomic_diff(A, H, T) :- atomic(A) |
200   H = [A | T].
201
202
203 /*****************************************************************
204 * atom
205 *  -> int    = Internal Number for Atom
206 *  -> string = Name for Atom
207 *  -> chars
208 *****************************************************************/
209 atom_int(A, Z) :- atom(A) |
210   atom_table:atom_number(A, Z).
211
212 %----------------------------------------------------------------
213 atom_string(A, S) :- atom(A) |
214   atom_table:get_atom_string(A, S).
```

```
215
216 %-----------------------------------------------------------------
217 /**
218 * atom_chars +A -H ?T
219 *
220 * 'atom' -> [0'a t o m' | T]
221 */
222 atom_chars(A, H, T) :- atom(A) |
223   atom_string(A, S),
224   string_diff(S, H, T).
225
226
227 /*****************************************************************
228 * int
229 *  -> float
230 *  -> chars
231 *****************************************************************/
232 int_float(Z, R) :- integer(Z) |
233   R $:= float(Z).
234
235 %-----------------------------------------------------------------
236 int_string(Z, S) :- integer(Z) |
237   int_chars(Z, L, []),
238   list_string(L, S).
239
240 %-----------------------------------------------------------------
241 /**
242 * int_chars +Z -H ?T
243 *
244 * -13 -> [0'- 1 3' | T]
245 */
246 int_chars(Z, H, T) :- Z=:=0 |  % one-digit integer zero
247   H = [0'0 | T].
248 int_chars(Z, H, T) :- Z > 0 |
249   positive_chars(Z, H, T).
250 int_chars(Z, H, T) :- Z < 0 |
251   positive_chars(~(-Z), H1, T),
252   H = [0'- | H1].
253
254 /**
255 * PRIVATE positive_chars +Z -H ?T
256 *
257 * 13 -> [0'1 3' | T]
258 */
259 positive_chars(Z, H, T) :- Z=:=0 |  % 0 -> []
260   H = T.
261 positive_chars(Z, H, T) :- Z > 0 |
262   positive_chars(~(Z/10), H, [~(0'0 + (Z mod 10)) | T]).
263
264
265 /*****************************************************************
266 * float
267 *  -> int
268 *  -> chars
269 *****************************************************************/
270 float_int(R, Z) :- float(R) |
271   Z := int(R).
272
```

```
273 %--------------------------------------------------------------
274 float_vector(R, V) :- float(R) |
275   V = {R}.
276
277 %--------------------------------------------------------------
278 float_string(R, S) :- float(R) |
279   float_chars(R, L, []),
280   list_string(L, S).
281
282 %--------------------------------------------------------------
283 float_list(R, L) :- float(R) |
284   L = [R].
285
286 float_diff(R, H, T) :- float(R) |
287   H = [R | T].
288
289 %--------------------------------------------------------------
290 /**
291 * float_chars +R -H ?T
292 *
293 * -13.0 -> [0'- 1 3 . 0' | T]
294 */
295 float_chars(R, H, T) :- float(R) |
296   float_int(R, Z),
297   int_chars(Z, H, [0'., 0'0 | T]).
298
299
300 /*************************************************************
301 * cons
302 *  -> chars
303 *************************************************************/
304 %--------------------------------------------------------------
305 /**
306 * cons_chars +L -H ?T
307 *
308 * [x|1] -> [ 0'[ x | 1 ]' | T]
309 */
310 cons_chars(L, H, T) :- L=[Car|Cdr] |
311   chars(Cdr, H2, [0'] | T]),
312   chars(Car, H1, [0'| | H2]),
313   H = [0'[ | H1].
314
315
316 /*************************************************************
317 * functor
318 *  -> vector, string, list, diff
319 *  -> chars
320 *************************************************************/
321 functor_vector(F, V) :- functor(F,_,_) |
322   functor_list(F, L),
323   list_vector(L, V).
324
325 arg_vector(F, V) :- functor(F,_,_) |
326   arg_list(F, Args),
327   list_vector(Args, V).
328
329 %--------------------------------------------------------------
330 functor_string(F, S) :- functor(F,_,_) |
```

```
331   functor_list(F, L),
332   list_string(L, S).
333
334 %-----------------------------------------------------------------
335 functor_list(F, L) :- functor(F,_,_) |
336   functor_diff(F, L, []).
337
338 arg_list(F, Args) :- functor(F,_,_) |
339   arg_diff(F, Args, []).
340
341 %-----------------------------------------------------------------
342 /**
343 * functor_diff +F -H ?T
344 *
345 * f(a1,a2) -> [f, a1, a2 | T]
346 */
347 functor_diff(F, H, T) :- functor(F, Name, _) |
348   arg_diff(F, Args, T),
349   H = [Name | Args].
350
351 arg_diff(F, H, T) :- functor(F, _, Arity) |
352   arg_diff(F, Arity, H, T).
353
354 arg_diff(F, Arity, H, T) :- Arity < 1 |
355   H = T.
356 arg_diff(F, Arity, H, T) :- Arity=:=1 |
357   arg(Arity, F, Arg),
358   H = [Arg | T].
359 arg_diff(F, Arity, H, T) :- Arity > 1 |
360   arg(Arity, F, Arg),
361   arg_diff(F, ~(Arity-1), H, [Arg | T]).
362
363 %-----------------------------------------------------------------
364 /**
365 * functor_chars +F -H ?T
366 *
367 * f(a,1) -> [ 0'f ( a , 1 )' | T]
368 */
369 functor_chars(F, H, T) :- functor(F, Name, Arity), Arity=:=0 |
370   atom_chars(Name, H, T).
371 functor_chars(F, H, T) :- functor(F, Name, Arity), Arity > 0 |
372   arg_chars(F, Args, [0') | T]),
373   atom_chars(Name, H, [0'( | Args]).
374
375 arg_chars(F, H, T) :- functor(F, _, Arity) |
376   arg_chars(F, Arity, H, T).
377
378 arg_chars(F, Arity, H, T) :- Arity < 1 |
379   H = T.
380 arg_chars(F, Arity, H, T) :- Arity=:=1 |
381   arg(Arity, F, Arg),
382   chars(Arg, H, T).
383 arg_chars(F, Arity, H, T) :- Arity > 1 |
384   arg(Arity, F, Arg),
385   chars(Arg, HArg, T),
386   arg_chars(F, ~(Arity-1), H, [0', | HArg]).
387
388
```

```
389 /*************************************************************
390 * vector
391 *  -> string, list, diff
392 *  -> chars
393 *************************************************************/
394 vector_functor(V, F) :- vector(V,_) |
395   vector_list(V, L),
396   list_functor(L, F).
397
398 %-------------------------------------------------------------
399 vector_string(V, S) :- vector(V,_) |
400   vector_list(V, L),
401   list_string(L, S).
402
403 %-------------------------------------------------------------
404 vector_list(V, L) :- vector(V,_) |
405   vector_diff(V, L, []).
406
407 %-------------------------------------------------------------
408 /**
409 * vector_diff +V -H ?T
410 *
411 * {e0,e1} -> [e0, e1 | T]
412 */
413 vector_diff(V, H, T) :- vector(V, Size) |
414   vector_diff(V, Size, H, T).
415
416 vector_diff(V, Size, H, T) :- Size < 1 |
417   H = T.
418 vector_diff(V, Size, H, T) :- Size=:=1 |
419   vector_element(V, ~(Size-1), Elem),
420   H = [Elem | T].
421 vector_diff(V, Size, H, T) :- Size > 1 |
422   vector_element(V, ~(Size-1), Elem),
423   vector_diff(V, ~(Size-1), H, [Elem | T]).
424
425 %-------------------------------------------------------------
426 /**
427 * vector_chars +V -H ?T
428 *
429 * {a,1} -> [ 0'{ a , 1 }' | T]
430 */
431 vector_chars(V, H, T) :- vector(V, Size) |
432   vector_chars(V, Size, Elems, [0'} | T]),
433   H = [0'{ | Elems].
434
435 vector_chars(V, Size, H, T) :- Size < 1 |
436   H = T.
437 vector_chars(V, Size, H, T) :- Size=:=1 |
438   vector_element(V, ~(Size-1), Elem),
439   chars(Elem, H, T).
440 vector_chars(V, Size, H, T) :- Size > 1 |
441   vector_element(V, ~(Size-1), Elem),
442   chars(Elem, HElem, T),
443   vector_chars(V, ~(Size-1), H, [0', | HElem]).
444
445
446 /*************************************************************
```

```
447 * string
448 *  -> atom, vector, list, diff
449 *  -> chars
450 ****************************************************************/
451 string_atom(S, A) :- string(S, _, CSize) |
452   atom_table:make_atom(S, A).
453
454 %----------------------------------------------------------------
455 string_vector(S, V) :- string(S, _, CSize) |
456   string_list(S, L),
457   list_vector(L, V).
458
459 %----------------------------------------------------------------
460 string_list(S, L) :- string(S, _, CSize) |
461   string_diff(S, L, []).
462
463 %----------------------------------------------------------------
464 /**
465 * string_diff +S -H ?T
466 *
467 * "string" -> [0's t r i n g' | T]
468 */
469 string_diff(S, H, T) :- string(S, Size, CSize) |
470   string_diff(S, Size, H, T).
471
472 string_diff(S, Size, H, T) :- Size < 1 |
473   H = T.
474 string_diff(S, Size, H, T) :- Size >= 1 |
475   string_element(S, ~(Size-1), C),
476   string_diff(S, ~(Size-1), H, [C|T]).
477
478 %----------------------------------------------------------------
479 /**
480 * string_chars +S -H ?T
481 *
482 * "string" -> [0'" s t r i n g "' | T]
483 */
484 string_chars(S, H, T) :- string(S, Size, CSize) |
485   string_diff(S, Size, Elems, [0'" | T]),
486   H = [0'" | Elems].
487
488
489 /****************************************************************
490 * list
491 *  -> functor, vector, string, diff
492 ****************************************************************/
493 list_functor(L, F) :-
494   functor_table:(F =.. L).
495
496 %----------------------------------------------------------------
497 list_vector(L, V) :-
498   new_vector(V, L).
499
500 %----------------------------------------------------------------
501 list_string(L, S) :-
502   new_string(S, L, CSize).
503
504 %----------------------------------------------------------------
```

```
505  /**
506  * list_diff +L -H ?T
507  *
508  * [a,1] -> [a,1 | T]
509  */
510  list_diff(L, H, T) :- L = [] |
511    H = T.
512  list_diff(L, H, T) :- L = [E|L1] |
513    H = [E | H1],
514    list_diff(L1, H1, T).
515
516
517  /*****************************************************************
518  * diff
519  *  -> functor, vector, string, list
520  *****************************************************************/
521  diff_functor(H, T, F) :-
522    diff_list(H, T, L),
523    list_functor(L, F).
524
525  %-----------------------------------------------------------------
526  diff_vector(H, T, V) :-
527    diff_list(H, T, L),
528    list_vector(L, V).
529
530  %-----------------------------------------------------------------
531  diff_string(H, T, S) :-
532    diff_list(H, T, L),
533    list_string(L, S).
534
535  %-----------------------------------------------------------------
536  /**
537  * diff_list +H ?T -L
538  *
539  * [a,1 | T] -> [a,1]
540  */
541  diff_list(H, T, L) :- H = T |
542    L = [].
543  otherwise.
544  diff_list(H, T, L) :- H = [E|H1] |
545    diff_list(H1, T, L1),
546    L = [E | L1].
547
548  %-----------------------------------------------------------------
549  /**
550  * PRIVATE diff_search +H ?T +E -Where
551  *
552  * [a,1,b,1 | T], 1  ->  [1,b,1 | T]
553  */
554  diff_search(H, T, E, Where) :- H = T |  % Not Found!
555    Where = [].
556  otherwise.
557  diff_search(H, T, E, Where) :- H = [E|H1] |  % Found!
558    Where = H.
559  otherwise.
560  diff_search(H, T, E, Where) :- H = [_|H1] |
561    diff_search(H1, T, E, Where).
562
```

```
563
564  % end type.
```

## A.2 vartab

```
/*************************************************************
* Variable Table
*
* by Yusuke TAKAGI (mailto:takagi@ueda.info.waseda.ac.jp)
* 1999/12/26  deleted 'set_value'.
*             added 'bound'.
* 1999/12/28  added 'bind'.
* 1999/12/29  ordered new names in wrap/unwrap correctly.
*
* Predicate:
*  variable_table +Stream
*
* Messages on Stream:
*  new_name  -Name             makes a new name Name.
*  new       -Name ?Value      makes a new variable Name = Value.
*  variable  +Name -Var
*  value     +Name ?Value      gets the value of variable Name.
*  wrap      ?Term -Wrapped
*  unwrap    +Wrapped ?Term
*  variable_table -Table       gets the whole table.
*  bound          -Table       gets the newly bound variables.
*  bind           +Table       binds variables.
*
* Result of 'Var':
*  normal(Value):  variable has Value (bound or unbound).
*  abnormal     :  variable has no entries into the table.
*
* Types:
*  name : string
*  value: term
*  table: list of (Name = Value)
*
* Precondition:
*  ent(S): variable S has an entry   into the table.
*  non(S): variable S has no entries into the table.
*
* Postcondition:
*  new(S): variable S has a new entry.
*  old(S): variable S keeps the old entry.
*  non(S): variable S has no entries into the table.
*
* new_name :  non(S) => non(S)
* new      :  non(S) => new(S)
* variable :  non(S) => non(S)  &  ent(S) => old(S)
* value    :  non(S) => new(S)  &  ent(S) => old(S)
*
*-----------------------------------------------------------
*
* + The table Vs is a sorted list of (Name = Value)
*     in reverse lexicographical order.
* + Us is a list of unbound variables. (not sorted)
*************************************************************/

:- module variable_table.

variable_table(Stream) :-
```

```
57    variable_table(Stream, 0, [], []).
58
59  variable_table([], _, _, _).
60  variable_table(Stream, N, Vs, Us) :- Stream = [new_name(S) | Stream1] |
61    new_name(S, N, NN),
62    variable_table(Stream1, NN, Vs, Us).
63  variable_table(Stream, N, Vs, Us) :- Stream = [new(S,X) | Stream1] |
64    new(S, X, N, NN, Vs, VsN, Us, UsN),
65    variable_table(Stream1, NN, VsN, UsN).
66  variable_table(Stream, N, Vs, Us) :- Stream = [variable(S,Var) | Stream1] |
67    variable(S, Var, Vs),
68    variable_table(Stream1, N, Vs, Us).
69  variable_table(Stream, N, Vs, Us) :- Stream = [value(S,X) | Stream1] |
70    value(S, X, N, NN, Vs, VsN, Us, UsN),
71    variable_table(Stream1, NN, VsN, UsN).
72  variable_table(Stream, N, Vs, Us) :- Stream = [wrap(X,W) | Stream1] |
73    wrap(X, W, N, NN, Vs, VsN, Us, UsN),
74    variable_table(Stream1, NN, VsN, UsN).
75  variable_table(Stream, N, Vs, Us) :- Stream = [unwrap(W,X) | Stream1] |
76    unwrap(W, X, N, NN, Vs, VsN, Us, UsN),
77    variable_table(Stream1, NN, VsN, UsN).
78  variable_table(Stream, N, Vs, Us) :- Stream = [variable_table(Tab) | Stream1] |
79    Tab = Vs,
80    variable_table(Stream1, N, Vs, Us).
81  variable_table(Stream, N, Vs, Us) :- Stream = [bound(Tab) | Stream1] |
82    bound(Tab, Us, UsN) @lower_priority,
83    variable_table(Stream1, N, Vs, UsN).
84  variable_table(Stream, N, Vs, Us) :- Stream = [bind(Tab) | Stream1] |
85    bind(Tab, N, NN, Vs, VsN, Us, UsN),
86    variable_table(Stream1, NN, VsN, UsN).
87
88  %-----------------------------------------------------------------
89  bind(Tab, N, NN, Vs, VsN, Us, UsN) :- Tab = [] |
90    NN = N,
91    VsN = Vs,
92    UsN = Us.
93  bind(Tab, N, NN, Vs, VsN, Us, UsN) :- Tab = [S=V | Tab1] |
94    value(S, V, N, N1, Vs, Vs1, Us, Us1),
95    bind(Tab1, N1, NN, Vs1, VsN, Us1, UsN).
96
97  %-----------------------------------------------------------------
98  type(X, Type) :- atom(X) |
99    Type = atom(X).
100 type(X, Type) :- integer(X) |
101   Type = integer(X).
102 type(X, Type) :- float(X) |
103   Type = floating_point(X).
104 type(X, Type) :- list(X) |
105   Type = list(X).
106 type(X, Type) :- functor(X,_,_) |
107   Type = functor(X).
108 type(X, Type) :- vector(X,_) |
109   Type = vector(X).
110 type(X, Type) :- string(X,_,_) |
111   Type = string(X).
112 alternatively.
113 type(X, Type) :- % unbound(X,{_,_,X}) |
114   Type = variable.
```

```
115
116 %-----------------------------------------------------------------
117 bound(Tab, Us, UsN) :- Us = [] |  % NO UNBOUND VARIABLE
118   Tab = [],
119   UsN = [].
120 bound(Tab, Us, UsN) :- Us = [S=V | _] |
121   type(V, Type),
122   bound(Type, Tab, Us, UsN).
123
124 bound(Type, Tab, [S=V | Us], UsN) :- Type = variable |
125   UsN = [S=V | UsN1],
126   bound(Tab, Us, UsN1).
127 bound(Type, Tab, [S=V | Us], UsN) :- Type \= variable |
128   Tab = [S=V | Tab1],
129   bound(Tab1, Us, UsN).
130
131 %-----------------------------------------------------------------
132 new(S, X, N, NN, Vs, VsN, Us, UsN) :-
133   new_name(S, N, NN),
134   value(S, X, N, NN, Vs, VsN, Us, UsN).
135
136 new_name(S, N, NN) :-
137   new_name(N, S),
138   NN := N + 1.
139
140 new_name(N, S) :-
141   type:int_chars(N, L, []),
142   type:list_string([0'_ | L], S).
143
144 %-----------------------------------------------------------------
145 variable(S, Var, Vs) :- Vs = [] |
146   Var = abnormal.
147 variable(S, Var, Vs) :- Vs = [S=V | _] |
148   Var = normal(V).
149 variable(S, Var, Vs) :- Vs = [S1=_ | _], string_less_than(S1, S) |
150   Var = abnormal.
151 variable(S, Var, Vs) :- Vs = [S1=_ | Vs1], string_less_than(S, S1) |
152   variable(S, Var, Vs1).
153
154 %-----------------------------------------------------------------
155 value(S, X, N, NN, Vs, VsN, Us, UsN) :- Vs = [] |
156   VsN = [S = X],
157   unbound(S=X, N, NN, Us, UsN).
158 value(S, X, N, NN, Vs, VsN, Us, UsN) :- Vs = [S=V | _] |
159   X = V,
160   NN = N,
161   VsN = Vs,
162   UsN = Us.  % Already added even if S=V is unbound.
163 value(S, X, N, NN, Vs, VsN, Us, UsN) :- Vs = [S1=_|_], string_less_than(S1,S) |
164   VsN = [S = X | Vs],
165   unbound(S=X, N, NN, Us, UsN).
166 value(S, X, N, NN, Vs, VsN, Us, UsN) :- Vs=[S1=V|Vs1], string_less_than(S,S1) |
167   value(S, X, N, NN, Vs1, VsN1, Us, UsN),
168   VsN = [S1=V | VsN1].
169
170 unbound(S=X, N, NN, Us, UsN) :-
171   new_name(_, N, NN),
172   type(X, Type),
```

```
173   unbound(Type, S=X, Us, UsN).
174
175 unbound(Type, S=X, Us, UsN) :- Type = variable |
176   UsN = [S=X | Us].
177 unbound(Type, S=X, Us, UsN) :- Type \= variable |
178   UsN = Us.
179
180 %-----------------------------------------------------------------
181 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- atomic(X) |
182   wrap_bound(X, W, N, NN, Vs, VsN, Us, UsN).
183 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- float(X) |
184   wrap_bound(X, W, N, NN, Vs, VsN, Us, UsN).
185 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- list(X) |
186   wrap_list(X, W, N, NN, Vs, VsN, Us, UsN).
187 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- functor(X, Sym, Arity) |
188   wrap_functor(X, Sym, Arity, W, N, NN, Vs, VsN, Us, UsN).
189 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- vector(X, Size) |
190   wrap_vector(X, Size, W, N, NN, Vs, VsN, Us, UsN).
191 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- string(X,_,_) |
192   wrap_bound(X, W, N, NN, Vs, VsN, Us, UsN).
193 alternatively.
194 wrap(X, W, N, NN, Vs, VsN, Us, UsN) :- % unbound(X,{_,_,X}) |
195   wrap_variable(X, W, N, NN, Vs, VsN, Us, UsN).
196
197 wrap_bound(X, W, N, NN, Vs, VsN, Us, UsN) :-
198   variable:wrap(X, W),
199   NN = N,
200   VsN = Vs,
201   UsN = Us.
202
203 wrap_variable(X, W, N, NN, Vs, VsN, Us, UsN) :-
204   % ALWAYS MAKE A NEW VARIABLE.
205   new(S, X, N, NN, Vs, VsN, Us, UsN),
206   W = variable(S).
207
208 wrap_list([Car|Cdr], W, N, NN, Vs, VsN, Us, UsN) :-
209   wrap(Car, W1, N , N1, Vs , Vs1, Us , Us1),
210   wrap(Cdr, W2, N1, NN, Vs1, VsN, Us1, UsN),
211   W = list([W1 | W2]).
212
213 wrap_functor(F, Sym, Arity, W, N, NN, Vs, VsN, Us, UsN) :-
214   wrap_args(F, Arity, Args, [], N, NN, Vs, VsN, Us, UsN),
215   functor_table:(WF =.. [Sym | Args]),
216   W = functor(WF).
217
218 wrap_args(F, Arity, H, T, N, NN, Vs, VsN, Us, UsN) :- Arity=:=0 |
219   H = T,
220   NN = N,
221   VsN = Vs,
222   UsN = Us.
223 wrap_args(F, Arity, H, T, N, NN, Vs, VsN, Us, UsN) :- Arity > 0 |
224   arg(Arity, F, Arg),
225   wrap(Arg, W, N1, NN, Vs1, VsN, Us1, UsN),
226   wrap_args(F, ~(Arity-1), H, [W|T], N, N1, Vs, Vs1, Us, Us1).
227
228 wrap_vector(V, Size, W, N, NN, Vs, VsN, Us, UsN) :- Size=:=0 |
229   W = vector(V),
230   NN = N,
```

```
231     VsN = Vs,
232     UsN = Us.
233   wrap_vector(V, Size, W, N, NN, Vs, VsN, Us, UsN) :- Size > 0 |
234     Index := Size - 1,
235     set_vector_element(V, Index, Elem, WElem, V1),
236     wrap(Elem, WElem, N1, NN, Vs1, VsN, Us1, UsN),
237     wrap_vector(V1, Index, W, N, N1, Vs, Vs1, Us, Us1).
238
239   %-----------------------------------------------------------------
240   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = variable(S) |
241     value(S, X, N, NN, Vs, VsN, Us, UsN).
242   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = atom(_) |
243     unwrap_bound(W, X, N, NN, Vs, VsN, Us, UsN).
244   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = integer(_) |
245     unwrap_bound(W, X, N, NN, Vs, VsN, Us, UsN).
246   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = floating_point(_) |
247     unwrap_bound(W, X, N, NN, Vs, VsN, Us, UsN).
248   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = list(L) |
249     unwrap_list(L, X, N, NN, Vs, VsN, Us, UsN).
250   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = functor(F) |
251     unwrap_functor(F, X, N, NN, Vs, VsN, Us, UsN).
252   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = vector(V) |
253     unwrap_vector(V, X, N, NN, Vs, VsN, Us, UsN).
254   unwrap(W, X, N, NN, Vs, VsN, Us, UsN) :- W = string(_) |
255     unwrap_bound(W, X, N, NN, Vs, VsN, Us, UsN).
256
257   unwrap_bound(W, X, N, NN, Vs, VsN, Us, UsN) :-
258     variable:unwrap(W, X),
259     NN = N,
260     VsN = Vs,
261     UsN = Us.
262
263   unwrap_list([Car|Cdr], X, N, NN, Vs, VsN, Us, UsN) :-
264     unwrap(Car, X1, N , N1, Vs , Vs1, Us , Us1),
265     unwrap(Cdr, X2, N1, NN, Vs1, VsN, Us1, UsN),
266     X = [X1 | X2].
267
268   unwrap_functor(F, X, N, NN, Vs, VsN, Us, UsN) :- functor(F, Sym, Arity) |
269     unwrap_args(F, Arity, Args, [], N, NN, Vs, VsN, Us, UsN),
270     functor_table:(X =.. [Sym | Args]).
271
272   unwrap_args(F, Arity, H, T, N, NN, Vs, VsN, Us, UsN) :- Arity=:=0 |
273     H = T,
274     NN = N,
275     VsN = Vs,
276     UsN = Us.
277   unwrap_args(F, Arity, H, T, N, NN, Vs, VsN, Us, UsN) :- Arity > 0 |
278     arg(Arity, F, W),
279     unwrap(W, X, N1, NN, Vs1, VsN, Us1, UsN),
280     unwrap_args(F, ~(Arity-1), H, [X|T], N, N1, Vs, Vs1, Us, Us1).
281
282   unwrap_vector(V, X, N, NN, Vs, VsN, Us, UsN) :- vector(V, Size) |
283     unwrap_vector(V, Size, X, N, NN, Vs, VsN, Us, UsN).
284
285   unwrap_vector(V, Size, X, N, NN, Vs, VsN, Us, UsN) :- Size=:=0 |
286     X = V,
287     NN = N,
288     VsN = Vs,
```

```
289     UsN = Us.
290   unwrap_vector(V, Size, X, N, NN, Vs, VsN, Us, UsN) :- Size > 0 |
291     Index := Size - 1,
292     set_vector_element(V, Index, W, Elem, V1),
293     unwrap(W, Elem, N1, NN, Vs1, VsN, Us1, UsN),
294     unwrap_vector(V1, Index, X, N, N1, Vs, Vs1, Us, Us1).
295
296
297   % end variable_table
```

## A.3 vsock

```
/*************************************************************
* Socket I/O with Variable handling
*
* by Yusuke TAKAGI (mailto:takagi@ueda.info.waseda.ac.jp)
* 1999/12/27  Variables can get 'puttq'ed.
* 1999/12/29  Only Bounds can be 'puttq'ed,
*             but they may have unbound components.
*
* Predicate:
*  socket +Sock -Cio
*
* Messages on Sock:
*  gett   -Term
*  getwt  -normal(WrappedTerm)
*  puttq  +Term
*  putwtq +WrappedTerm
*  nl
*  ungetc +C
*  (& messages on Cio)
*
* Socket streams usually eats only C-like I/O messages.
*  -> unix:unix([connect(inet(Host,Port), normal(Socket))]),
*     socket:socket(I_can_touch_terms, Socket),
*     I_can_touch_terms = [gett(T), puttq(T), nl].
*
* Standard input stream does not usually eat ungetc.
*  -> unix:unix([stdin(normal(Stdin))]),
*     socket:socket(I_can_eat_ungetc, Stdin).
*     I_can_eat_ungetc = [getc(C), ungetc(C)].
*
*-------------------------------------------------------------
*
* A : atom
* C : character
* F : functor
* H : head
* L : list
* LA: look ahead (character)
* N : number
* R : real number
* S : string
* T : tail
* U : unknown
* V : vector
* W : wrapped
* X : variable
* Z : zahl (integer)
*
* gett -Term -LA -?Cio -?Vars
* geta -A -LA -?Cio -?Vars
* getz -Z -LA -?Cio -?Vars
* getn -N -LA -?Cio -?Vars
* getl -L -LA -?Cio -?Vars
* getf -F -LA -?Cio -?Vars
* getv -V -LA -?Cio -?Vars
* gets -S -LA -?Cio -?Vars
```

```
57  *
58  * getwt -normal(W) -LA -?Cio -?Vars
59  * getwx -WX -LA -?Cio -?Vars
60  * getwa -WA -LA -?Cio -?Vars
61  * getwi -WZ -LA -?Cio -?Vars
62  * getwn -WN -LA -?Cio -?Vars
63  * getwl -WL -LA -?Cio -?Vars
64  * getwf -WF -LA -?Cio -?Vars
65  * getwv -WV -LA -?Cio -?Vars
66  * getws -WS -LA -?Cio -?Vars
67  *
68  * puttq +Term -?Cio -?Vars
69  * putaq +A -?Cio -?Vars
70  * putz  +Z -?Cio -?Vars
71  * putn  +R -?Cio -?Vars
72  * putlq +L -?Cio -?Vars
73  * putfq +F -?Cio -?Vars
74  * putvq +V -?Cio -?Vars
75  * puts  +S -?Cio -?Vars
76  *
77  * putwtq +W -?Cio -?Vars
78  * putwx  +WX -?Cio -?Vars
79  * putwaq +WA -?Cio -?Vars
80  * putwi  +WZ -?Cio -?Vars
81  * putwn  +WR -?Cio -?Vars
82  * putwlq +WL -?Cio -?Vars
83  * putwfq +WF -?Cio -?Vars
84  * putwvq +WV -?Cio -?Vars
85  * putws  +WS -?Cio -?Vars
86  * putwuq +WU -?Cio -?Vars
87  ****************************************************************/
88
89  :- module socket.
90
91  :- with((
92    CSIZE = 8,  % size of character
93    EOF = -1,  % end of file
94
95  CTYPE =
96  "ccccccccc_____ccccccccccccccccc_1\"ss%s'()ss,sss0000000000s1ssss\
97  sAAAAAAAAAAAAAAAAAAAAAAAAA[s]sAsaaaaaaaaaaaaaaaaaaaaaaaaa{1}sc"
98   )).
99
100 % 0'c  control character
101 % 0'_  space
102 % 0'0  digit
103 % 0'A  UPPER CASE
104 % 0'a  lower case
105 % 0's  special character
106 % 0'1  one-character atom
107
108 ctype(C, Type) :- C = EOF |
109   Type = EOF.
110 ctype(C, Type) :- 0 =< C, C =< 127 |
111   string_element(CTYPE, C, Type).
112
113
114 /****************************************************************
```

```
115 * socket +Sock -Cio
116 ************************************************************/
117
118 socket(Sock, Cio) :-
119   variable_table:variable_table(Vars),
120   socket(Sock, Cio, Vars).
121
122 socket([], Cio, Vars) :-
123   Cio = [],
124   Vars = [].
125 socket([Mes | Sock], Cio, Vars) :- Mes = new_name(S) |
126   Vars = [Mes | Vars1],
127   socket(Sock, Cio, Vars1).
128 socket([Mes | Sock], Cio, Vars) :- Mes = new(S,X) |
129   Vars = [Mes | Vars1],
130   socket(Sock, Cio, Vars1).
131 socket([Mes | Sock], Cio, Vars) :- Mes = variable(S,Var) |
132   Vars = [Mes | Vars1],
133   socket(Sock, Cio, Vars1).
134 socket([Mes | Sock], Cio, Vars) :- Mes = value(S,X) |
135   Vars = [Mes | Vars1],
136   socket(Sock, Cio, Vars1).
137 socket([Mes | Sock], Cio, Vars) :- Mes = wrap(X,W) |
138   Vars = [Mes | Vars1],
139   socket(Sock, Cio, Vars1).
140 socket([Mes | Sock], Cio, Vars) :- Mes = unwrap(W,X) |
141   Vars = [Mes | Vars1],
142   socket(Sock, Cio, Vars1).
143 socket([Mes | Sock], Cio, Vars) :- Mes = variable_table(Tab) |
144   Vars = [Mes | Vars1],
145   socket(Sock, Cio, Vars1).
146 socket([Mes | Sock], Cio, Vars) :- Mes = bound(Tab) |
147   Vars = [Mes | Vars1],
148   socket(Sock, Cio, Vars1).
149 socket([Mes | Sock], Cio, Vars) :- Mes = bind(Tab) |
150   Vars = [Mes | Vars1],
151   socket(Sock, Cio, Vars1).
152 socket([ungetc(C)     | Sock], Cio, Vars) :-
153   socket(Sock, C, Cio, Vars).
154 socket([gett(Term)    | Sock], Cio, Vars) :-
155   gett(Term, LA, Cio, Cio1, Vars, Vars1),
156   socket(Sock, LA, Cio1, Vars1).
157 socket([getwt(Result) | Sock], Cio, Vars) :-
158   getwt(Result, LA, Cio, Cio1, Vars, Vars1),
159   socket(Sock, LA, Cio1, Vars1).
160 socket([puttq(Term)   | Sock], Cio, Vars) :-
161   puttq(Term, Cio, Cio1, Vars, Vars1),
162   socket(Sock, Cio1, Vars1).
163 socket([putwtq(W)     | Sock], Cio, Vars) :-
164   putwtq(W, Cio, Cio1, Vars, Vars1),
165   socket(Sock, Cio1, Vars1).
166 socket([nl            | Sock], Cio, Vars) :-
167   Cio = [fwrite("\n") | Cio1],
168   socket(Sock, Cio1, Vars).
169 otherwise.
170 socket([Mes | Sock], Cio, Vars) :-
171   Cio = [Mes | Cio1],
172   socket(Sock, Cio1, Vars).
```

```
173
174 socket([], C, Cio, Vars) :-
175   Cio = [],
176   Vars = [].
177 socket([Mes | Sock], C, Cio, Vars) :- Mes = new_name(S) |
178   Vars = [Mes | Vars1],
179   socket(Sock, C, Cio, Vars1).
180 socket([Mes | Sock], C, Cio, Vars) :- Mes = new(S,X) |
181   Vars = [Mes | Vars1],
182   socket(Sock, C, Cio, Vars1).
183 socket([Mes | Sock], C, Cio, Vars) :- Mes = variable(S,Var) |
184   Vars = [Mes | Vars1],
185   socket(Sock, C, Cio, Vars1).
186 socket([Mes | Sock], C, Cio, Vars) :- Mes = value(S,X) |
187   Vars = [Mes | Vars1],
188   socket(Sock, C, Cio, Vars1).
189 socket([Mes | Sock], C, Cio, Vars) :- Mes = wrap(X,W) |
190   Vars = [Mes | Vars1],
191   socket(Sock, C, Cio, Vars1).
192 socket([Mes | Sock], C, Cio, Vars) :- Mes = unwrap(W,X) |
193   Vars = [Mes | Vars1],
194   socket(Sock, C, Cio, Vars1).
195 socket([Mes | Sock], C, Cio, Vars) :- Mes = variable_table(Tab) |
196   Vars = [Mes | Vars1],
197   socket(Sock, C, Cio, Vars1).
198 socket([Mes | Sock], C, Cio, Vars) :- Mes = bound(Tab) |
199   Vars = [Mes | Vars1],
200   socket(Sock, C, Cio, Vars1).
201 socket([Mes | Sock], C, Cio, Vars) :- Mes = bind(Tab) |
202   Vars = [Mes | Vars1],
203   socket(Sock, C, Cio, Vars1).
204 socket([getc(Char)    | Sock], C, Cio, Vars) :-
205   Char = C,
206   socket(Sock, Cio, Vars).
207 socket([gett(Term)    | Sock], C, Cio, Vars) :-
208   gett(Term, LA, C, Cio, Cio1, Vars, Vars1),
209   socket(Sock, LA, Cio1, Vars1).
210 socket([getwt(Result) | Sock], C, Cio, Vars) :-
211   getwt(Result, LA, C, Cio, Cio1, Vars, Vars1),
212   socket(Sock, LA, Cio1, Vars1).
213 socket([puttq(Term)   | Sock], C, Cio, Vars) :-
214   puttq(Term, Cio, Cio1, Vars, Vars1),
215   socket(Sock, C, Cio1, Vars1).
216 socket([putwtq(W)     | Sock], C, Cio, Vars) :-
217   putwtq(W, Cio, Cio1, Vars, Vars1),
218   socket(Sock, C, Cio1, Vars1).
219 socket([nl            | Sock], C, Cio, Vars) :-
220   Cio = [fwrite("\n") | Cio1],
221   socket(Sock, C, Cio1, Vars).
222 otherwise.
223 socket([Mes | Sock], C, Cio, Vars) :-
224   Cio = [Mes | Cio1],
225   socket(Sock, C, Cio1, Vars).
226
227
228 /*****************************************************************
229 * puttq  +Term -H ?T -VH ?VT
230 * putwtq +W    -H ?T -VH ?VT
```

```
231 ************************************************************/
232
233 puttq(Term, H, T, VH, VT) :-
234   puttq(0, Term, H, T, VH, VT) @lower_priority.
235
236 puttq(_, Term, H, T, VH, VT) :- atom(Term) |
237   putaq(Term, H, T, VH, VT).
238 puttq(_, Term, H, T, VH, VT) :- integer(Term) |
239   putz(Term, H, T, VH, VT).
240 puttq(_, Term, H, T, VH, VT) :- float(Term) |
241   putn(Term, H, T, VH, VT).
242 puttq(_, Term, H, T, VH, VT) :- list(Term) |
243   putlq(Term, H, T, VH, VT).
244 puttq(_, Term, H, T, VH, VT) :- functor(Term,_,_) |
245   putfq(Term, H, T, VH, VT).
246 puttq(_, Term, H, T, VH, VT) :- vector(Term,_) |
247   putvq(Term, H, T, VH, VT).
248 puttq(_, Term, H, T, VH, VT) :- string(Term,_,CSIZE) |
249   puts(Term, H, T, VH, VT).
250 alternatively.
251 puttq(1, Term, H, T, VH, VT) :- % unbound(Term, {_,_,Term}) |
252   VH = [wrap(Term,W) | VH1],
253   putwtq(W, H, T, VH1, VT).
254
255 %-------------------------------------------------------------
256
257 putwtq(W, H, T, VH, VT) :- W = variable(S) |
258   putwxq(W, H, T, VH, VT).
259 putwtq(W, H, T, VH, VT) :- W = atom(A) |
260   putwaq(W, H, T, VH, VT).
261 putwtq(W, H, T, VH, VT) :- W = integer(Z) |
262   putwz(W, H, T, VH, VT).
263 putwtq(W, H, T, VH, VT) :- W = floating_point(R) |
264   putwn(W, H, T, VH, VT).
265 putwtq(W, H, T, VH, VT) :- W = list(L) |
266   putwlq(W, H, T, VH, VT).
267 putwtq(W, H, T, VH, VT) :- W = functor(F) |
268   putwfq(W, H, T, VH, VT).
269 putwtq(W, H, T, VH, VT) :- W = vector(V) |
270   putwvq(W, H, T, VH, VT).
271 putwtq(W, H, T, VH, VT) :- W = string(S) |
272   putws(W, H, T, VH, VT).
273 putwtq(W, H, T, VH, VT) :- W = unknown(U) |
274   puttq(U, H, T, VH, VT).
275
276
277 %-------------------------------------------------------------
278 % putwxq +WV -H ?T -VH ?VT
279 %-------------------------------------------------------------
280 putwxq(variable(S), H, T, VH, VT) :-
281   VH = [value(S,X) | VH1],
282   putwxq(S, X, H, T, VH1, VT).
283
284 putwxq(S, X, H, T, VH, VT) :- atom(X) |
285   putaq(X, H, T, VH, VT).
286 putwxq(S, X, H, T, VH, VT) :- integer(X) |
287   putz(X, H, T, VH, VT).
288 putwxq(S, X, H, T, VH, VT) :- float(X) |
```

```
289   putn(X, H, T, VH, VT).
290 putwxq(S, X, H, T, VH, VT) :- list(X) |
291   putlq(X, H, T, VH, VT).
292 putwxq(S, X, H, T, VH, VT) :- functor(X,_,_) |
293   putfq(X, H, T, VH, VT).
294 putwxq(S, X, H, T, VH, VT) :- vector(X,_) |
295   putvq(X, H, T, VH, VT).
296 putwxq(S, X, H, T, VH, VT) :- string(X,_,CSIZE) |
297   puts(X, H, T, VH, VT).
298 alternatively.
299 putwxq(S, X, H, T, VH, VT) :- % unbound(X, {_,_,X}) |
300   H = [fwrite(S) | T],
301   VH = VT.
302
303
304 %-----------------------------------------------------------------
305 % putaq  +A  -H ?T -VH ?VT
306 % putwaq +WA -H ?T -VH ?VT
307 %-----------------------------------------------------------------
308
309 putaq(A, H, T, VH, VT) :- atom(A) |
310   atom_table:get_atom_string(A, S),
311   H = [fwrite(S) | T],
312   VH = VT.
313
314 putwaq(atom(A), H, T, VH, VT) :-
315   putaq(A, H, T, VH, VT).
316
317
318 %-----------------------------------------------------------------
319 % putz  +Z  -H ?T -VH ?VT
320 % putwz +WZ -H ?T -VH ?VT
321 %-----------------------------------------------------------------
322
323 putz(Z, H, T, VH, VT) :-
324   type:int_chars(Z, H, T),
325   VH = VT.
326
327 %-----------------------------------------------------------------
328
329 putwz(integer(Z), H, T, VH, VT) :-
330   putz(Z, H, T, VH, VT).
331
332
333 %-----------------------------------------------------------------
334 % putn  +R  -H ?T -VH ?VT
335 % putwn +WR -H ?T -VH ?VT
336 %-----------------------------------------------------------------
337
338 putn(R, H, T, VH, VT) :- float(R) |
339   type:float_chars(R, H, T),
340   VH = VT.
341
342 putwn(floating_point(R), H, T, VH, VT) :-
343   putn(R, H, T, VH, VT).
344
345
346 %-----------------------------------------------------------------
```

```
% putlq  +L  -H ?T -VH ?VT
% putwlq +WL -H ?T -VH ?VT
%-----------------------------------------------------------------

putlq(L, H, T, VH, VT) :- L = [Car|Cdr] |
  H = [0'[ | H1],
  puttq(1, Car, H1, [0'| | H2], VH, VH1) @lower_priority,
  puttq(1, Cdr, H2, [0'] | T], VH1, VT) @lower_priority.

putwlq(list(L), H, T, VH, VT) :- L = [Car|Cdr] |
  H = [0'[ | H1],
  putwtq(Car, H1, [0'| | H2], VH, VH1),
  putwtq(Cdr, H2, [0'] | T], VH1, VT).


%-----------------------------------------------------------------
% putfq  +F  -H ?T -VH ?VT
% putwfq +WF -H ?T -VH ?VT
%-----------------------------------------------------------------

putfq(F, H, T, VH, VT) :- functor(F, Name, Arity) |
  puttq(Name, H, H1, VH, VH1),
  putfq_args(F, Arity, H1, T, VH1, VT).

%--- putfq_args +F +Arity -H ?T -VH ?VT

putfq_args(F, Arity, H, T, VH, VT) :- Arity=:=0 |
  H = T,
  VH = VT.
putfq_args(F, Arity, H, T, VH, VT) :- Arity > 0 |
  H = [0'( | H1],
  putfq_args1(F, Arity, H1, [0') | T], VH, VT).

putfq_args1(F, Pos, H, T, VH, VT) :- Pos=:=1 |
  arg(Pos, F, Arg),
  puttq(1, Arg, H, T, VH, VT) @lower_priority.
putfq_args1(F, Pos, H, T, VH, VT) :- Pos > 1 |
  putfq_args1(F, ~(Pos-1), H, [0', | H1], VH, VH1),
  arg(Pos, F, Arg),
  puttq(1, Arg, H1, T, VH1, VT) @lower_priority.

%-----------------------------------------------------------------

putwfq(functor(F), H, T, VH, VT) :- functor(F, Name, Arity) |
  puttq(Name, H, H1, VH, VH1),
  putwfq_args(F, Arity, H1, T, VH1, VT).

%--- putwfq_args +F +Arity -H ?T -VH ?VT

putwfq_args(F, Arity, H, T, VH, VT) :- Arity=:=0 |
  H = T,
  VH = VT.
putwfq_args(F, Arity, H, T, VH, VT) :- Arity > 0 |
  H = [0'( | H1],
  putwfq_args1(F, Arity, H1, [0') | T], VH, VT).

putwfq_args1(F, Pos, H, T, VH, VT) :- Pos=:=1 |
  arg(Pos, F, Arg),
```

```
405     putwtq(Arg, H, T, VH, VT).
406   putwfq_args1(F, Pos, H, T, VH, VT) :- Pos > 1 |
407     putwfq_args1(F, ~(Pos-1), H, [0', | H1], VH, VH1),
408     arg(Pos, F, Arg),
409     putwtq(Arg, H1, T, VH1, VT).
410
411
412   %-----------------------------------------------------------------
413   % putvq  +V  -H ?T -VH ?VT
414   % putwvq +WV -H ?T -VH ?VT
415   %-----------------------------------------------------------------
416
417   putvq(V, H, T, VH, VT) :- vector(V, Size) |
418     H = [0'{ | H1],
419     putvq_elems(V, ~(Size-1), H1, [0'} | T], VH, VT).
420
421   %--- putvq_elems +V +Index -H ?T -VH ?VT
422
423   putvq_elems(V, Index, H, T, VH, VT) :- Index < 0 |
424     H = T,
425     VH = VT.
426   putvq_elems(V, Index, H, T, VH, VT) :- Index=:=0 |
427     vector_element(V, Index, Elem),
428     puttq(1, Elem, H, T, VH, VT) @lower_priority.
429   putvq_elems(V, Index, H, T, VH, VT) :- Index > 0 |
430     putvq_elems(V, ~(Index-1), H, [0', | H1], VH, VH1),
431     vector_element(V, Index, Elem),
432     puttq(1, Elem, H1, T, VH1, VT) @lower_priority.
433
434   %-----------------------------------------------------------------
435
436   putwvq(vector(V), H, T, VH, VT) :- vector(V, Size) |
437     H = [0'{ | H1],
438     putwvq_elems(V, ~(Size-1), H1, [0'} | T], VH, VT).
439
440   %--- putwvq_elems +V +Index -H ?T -VH ?VT
441
442   putwvq_elems(V, Index, H, T, VH, VT) :- Index < 0 |
443     H = T,
444     VH = VT.
445   putwvq_elems(V, Index, H, T, VH, VT) :- Index=:=0 |
446     vector_element(V, Index, Elem),
447     putwtq(Elem, H, T, VH, VT).
448   putwvq_elems(V, Index, H, T, VH, VT) :- Index > 0 |
449     putwvq_elems(V, ~(Index-1), H, [0', | H1], VH, VH1),
450     vector_element(V, Index, Elem),
451     putwtq(Elem, H1, T, VH1, VT).
452
453
454   %-----------------------------------------------------------------
455   % puts  +S  -H ?T -VH ?VT
456   % putws +WS -H ?T -VH ?VT
457   %-----------------------------------------------------------------
458
459   puts(S, H, T, VH, VT) :- string(S, Size, CSIZE) |
460     H = [0'", fwrite(S, Size), 0'" | T],
461     VH = VT.
462
```

```
463  putws(string(S), H, T, VH, VT) :-
464    puts(S, H, T, VH, VT).
465
466
467  /*************************************************************
468  * gett  -Term   -LA -H ?T -VH ?VT
469  * getwt -Result -LA -H ?T -VH ?VT
470  *************************************************************/
471
472  gett(Term, LA, H, T, VH, VT) :-
473    H = [getc(C) | H1],
474    gett(Term, LA, C, H1, T, VH, VT).
475
476  gett(Term, LA, C, H, T, VH, VT) :-
477    getwt(normal(W), LA, C, H, T, VH, VH1),
478    gett(W, Term, VH1, VT).
479
480  gett(W, Term, VH, VT) :- W = end_of_file |
481    Term = end_of_file,
482    VH = VT.
483  gett(W, Term, VH, VT) :- functor(W, _, 1) |
484    VH = [unwrap(W, Term) | VT].
485
486  %-------------------------------------------------------------
487
488  getwt(Result, LA, H, T, VH, VT) :-
489    H = [getc(C) | H1],
490    getwt(Result, LA, C, H1, T, VH, VT).
491
492  getwt(Result, LA, C, H, T, VH, VT) :-
493    getwt1(W, C1, C, H, H1, VH, VT),
494    get_end(LA, C1, H1, T),
495    Result = normal(W).
496
497  getwt1(W, LA, H, T, VH, VT) :-
498    H = [getc(C) | H1],
499    getwt1(W, LA, C, H1, T, VH, VT).
500
501  getwt1(W, LA, C, H, T, VH, VT) :-
502    get_skip(C1, C, H, H1),
503    ctype(C1, Type),
504    getwt1(W, LA, Type, C1, H1, T, VH, VT).
505
506  getwt1(W, LA, Type, C, H, T, VH, VT) :- Type = EOF |
507    W = end_of_file,
508    LA = EOF,
509    H = T,
510    VH = VT.
511  getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'0 |
512    getwn(W, LA, C, H, T, VH, VT).
513  getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'A |
514    getwx(W, LA, C, H, T, VH, VT).
515  getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'a |
516    getwf(W, LA, C, H, T, VH, VT).
517  getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0's |
518    getwf(W, LA, C, H, T, VH, VT).
519  getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'1 |
520    getwf(W, LA, C, H, T, VH, VT).
```

```
getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'' |
  getwf(W, LA, C, H, T, VH, VT).
getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'[ |
  getwl(W, LA, C, H, T, VH, VT).
getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'{ |
  getwv(W, LA, C, H, T, VH, VT).
getwt1(W, LA, Type, C, H, T, VH, VT) :- Type=:=0'" |
  getws(W, LA, C, H, T, VH, VT).

%-----------------------------------------------------------------

get_skip(LA, H, T) :-
  H = [getc(C) | H1],
  get_skip(LA, C, H1, T).

get_skip(LA, C, H, T) :-
  ctype(C, Type),
  get_skip(LA, Type, C, H, T).

get_skip(LA, Type, C, H, T) :- Type =:= 0'_ |
  get_skip(LA, H, T).
get_skip(LA, Type, C, H, T) :- Type =\= 0'_ |
  LA = C,
  H = T.

%-----------------------------------------------------------------

get_end(LA, C, H, T) :-
  get_skip(C1, C, H, H1),
  get_end1(LA, C1, H1, T).

get_end1(LA, C, H, T) :- C = EOF |
  LA = C,
  H = T.
get_end1(LA, C, H, T) :- C=:=0'. |
  H = [getc(LA) | T].


%-----------------------------------------------------------------
% getwx -WX -LA +C -H ?T -VH ?VT
%-----------------------------------------------------------------

getwx(WX, LA, C, H, T, VH, VT) :-
  ctype(C, 0'A),
  get_symbol(L, LA, H, T),
  new_string(S, [C|L], CSIZE),
  WX = variable(S),
  VH = [value(S,_) | VT].

%--- get_symbol -L -LA -H ?T

get_symbol(L, LA, H, T) :-
  H = [getc(C) | H1],
  ctype(C, Type),
  get_symbol(L, LA, Type, C, H1, T).

get_symbol(L, LA, Type, C, H, T) :- Type =:= 0'0 |
  L = [C | L1],
```

```
579     get_symbol(L1, LA, H, T).
580   get_symbol(L, LA, Type, C, H, T) :- Type =:= 0'A |
581     L = [C | L1],
582     get_symbol(L1, LA, H, T).
583   get_symbol(L, LA, Type, C, H, T) :- Type =:= 0'a |
584     L = [C | L1],
585     get_symbol(L1, LA, H, T).
586   otherwise.
587   get_symbol(L, LA, Type, C, H, T) :-
588     L = [],
589     LA = C,
590     H = T.
591
592
593   %-------------------------------------------------------------
594   % geta  -A  -LA +C -H ?T -VH ?VT
595   % getwa -WA -LA +C -H ?T -VH ?VT
596   %-------------------------------------------------------------
597
598   geta(A, LA, C, H, T, VH, VT) :-
599     ctype(C, Type),
600     geta(L, LA, Type, C, H, T),
601     new_string(S, L, CSIZE),
602     atom_table:make_atom(S, A),
603     VH = VT.
604
605   geta(L, LA, Type, C, H, T) :- Type =:= 0'a |
606     L = [C | L1],
607     get_symbol(L1, LA, H, T).
608   geta(L, LA, Type, C, H, T) :- Type =:= 0's |
609     L = [C | L1],
610     get_special(L1, LA, H, T).
611   geta(L, LA, Type, C, H, T) :- Type =:= 0'' |
612     L = [C | L1],
613     get_quote(L1, LA, H, T).
614   geta(L, LA, Type, C, H, T) :- Type =:= 0'1 |
615     L = [C],
616     H = [getc(LA) | T].
617
618   %--- get_special -L -LA -H ?T
619
620   get_special(L, LA, H, T) :-
621     H = [getc(C) | H1],
622     ctype(C, Type),
623     get_special(L, LA, Type, C, H1, T).
624
625   get_special(L, LA, Type, C, H, T) :- Type =:= 0's |
626     L = [C | L1],
627     get_special(L1, LA, H, T).
628   get_special(L, LA, Type, C, H, T) :- Type =\= 0's |
629     L = [],
630     LA = C,
631     H = T.
632
633   %--- get_quote -L -LA -H ?T
634
635   get_quote(L, LA, H, T) :-
636     H = [getc(C) | H1],
```

```
637   get_quote(L, LA, C, H1, T).
638
639 get_quote(L, LA, C, H, T) :- C =:= 0'\ |
640   get_quote1(L, LA, C, H, T).
641 get_quote(L, LA, C, H, T) :- C =:= 0'' |
642   get_quote1(L, LA, C, H, T).
643 get_quote(L, LA, C, H, T) :- C =\= 0'\, C =\= 0'' |
644   L = [C | L1],
645   get_quote(L1, LA, H, T).
646
647 get_quote1(L, LA, C, H, T) :-
648   H = [getc(C1) | H1],
649   get_quote1(L, LA, C, C1, H1, T).
650
651 get_quote1(L, LA, C, C1, H, T) :- C =:= 0'\, C1 =:= 0'' |
652   L = [C1 | L1],
653   get_quote(L1, LA, H, T).
654 get_quote1(L, LA, C, C1, H, T) :- C =:= 0'', C1 =:= 0'' |
655   L = [C1 | L1],
656   get_quote(L1, LA, H, T).
657 get_quote1(L, LA, C, C1, H, T) :- C =:= 0'\, C1 =\= 0'' |
658   L = [C | L1],
659   get_quote(L1, LA, C1, H, T).
660 get_quote1(L, LA, C, C1, H, T) :- C =:= 0'', C1 =\= 0'' |
661   L = [C],
662   LA = C1,
663   H = T.
664
665 %-----------------------------------------------------------------
666
667 getwa(WA, LA, C, H, T, VH, VT) :-
668   geta(A, LA, C, H, T, VH, VT),
669   WA = atom(A).
670
671
672 %-----------------------------------------------------------------
673 % getz  -Z  -LA +C -H ?T -VH ?VT
674 % getwz -WZ -LA +C -H ?T -VH ?VT
675 %-----------------------------------------------------------------
676
677 getz(Z, LA, C, H, T, VH, VT) :-
678   ctype(C, 0'0),
679   getz1(10, 0, Z, LA, C, H, T),
680   VH = VT.
681
682 getz1(Base, Acc, Z, LA, C, H, T) :-
683   ctype(C, Type),
684   getz1(Base, Acc, Z, LA, Type, C, H, T).
685
686 getz1(Base, Acc, Z, LA, 0'0, C, H, T) :- D :=    C-0'0, D < Base |
687   getz1(Base, ~(Acc*Base + D), Z, LA, H, T).
688 getz1(Base, Acc, Z, LA, 0'A, C, H, T) :- D := 10+C-0'A, D < Base |
689   getz1(Base, ~(Acc*Base + D), Z, LA, H, T).
690 getz1(Base, Acc, Z, LA, 0'a, C, H, T) :- D := 10+C-0'a, D < Base |
691   getz1(Base, ~(Acc*Base + D), Z, LA, H, T).
692 otherwise.
693 getz1(Base, Acc, Z, LA, Type, C, H, T) :-
694   Z = Acc,
```

```
695     LA = C,
696     H = T.
697
698   getz1(Base, Acc, Z, LA, H, T) :-
699     H = [getc(C) | H1],
700     getz1(Base, Acc, Z, LA, C, H1, T).
701
702   %-----------------------------------------------------------------
703
704   getwz(WZ, LA, C, H, T, VH, VT) :-
705     getz(Z, LA, C, H, T, VH, VT),
706     WZ = integer(Z).
707
708
709   %-----------------------------------------------------------------
710   % getn  -N  -LA +C -H ?T -VH ?VT
711   % getwn -WN -LA +C -H ?T -VH ?VT
712   %-----------------------------------------------------------------
713
714   getn(N, LA, C, H, T, VH, VT) :-
715     getz(N, LA, C, H, T, VH, VT).
716
717   getwn(WN, LA, C, H, T, VH, VT) :-
718     getwz(WN, LA, C, H, T, VH, VT).
719
720
721   %-----------------------------------------------------------------
722   % getl  -L  -LA +C -H ?T -VH ?VT
723   % getwl -WL -LA +C -H ?T -VH ?VT
724   %-----------------------------------------------------------------
725
726   % getl(L, LA, 0'[, H, T, VH, VT) :-
727
728   %-----------------------------------------------------------------
729   getwl(WL, LA, 0'[, H, T, VH, VT) :-
730     H = [getc(C) | H1],
731     getwl1(L, LA, C, H1, T, VH, VT),
732     getwl(L, WL).
733
734   getwl(L, WL) :- L = [] |
735     WL = atom(L).
736   getwl(L, WL) :- list(L) |
737     WL = list(L).
738
739   getwl1(L, LA, C, H, T, VH, VT) :- C =:= 0'] |
740     L = [],
741     H = [getc(LA) | T],
742     VH = VT.
743   getwl1(L, LA, C, H, T, VH, VT) :- C =\= 0'] |
744     getwt1(Car, 0'|, C, H, H1, VH, VH1),
745     getwt1(Cdr, 0'],   H1, H2, VH1, VT),
746     L = [Car | Cdr],
747     H2 = [getc(LA) | T].
748
749
750   %-----------------------------------------------------------------
751   % getf  -F  -LA +C -H ?T -VH ?VT
752   % getwf -WF -LA +C -H ?T -VH ?VT
```

```
753 %-------------------------------------------------------------
754
755 % getf(F, LA, C, H, T, VH, VT) :-
756 %   geta(Name, C1, C, H, H1, VH, VH1),
757 %   getf_args(Args, LA, C1, H1, T, VH1, VT),
758 %   functor_table:(F =.. [Name|Args]).
759
760 %-------------------------------------------------------------
761
762 getwf(WF, LA, C, H, T, VH, VT) :-
763   geta(Name, C1, C, H, H1, VH, VH1),
764   getwf_args(Args, LA, C1, H1, T, VH1, VT),
765   functor_table:(F =.. [Name|Args]),
766   getwf(F, WF).
767
768 getwf(F, WF) :- atom(F) |
769   WF = atom(F).
770 getwf(F, WF) :- functor(F, _, Arity), Arity > 0 |
771   WF = functor(F).
772
773 getwf_args(Args, LA, C, H, T, VH, VT) :- C =:= 0'( |
774   getwf_args1(Args, LA, H, T, VH, VT).
775 getwf_args(Args, LA, C, H, T, VH, VT) :- C =\= 0'( |
776   Args = [],
777   LA = C,
778   H = T,
779   VH = VT.
780
781 getwf_args1(Args, LA, H, T, VH, VT) :-
782   getwt1(Arg, C1, H, H1, VH, VH1),
783   Args = [Arg | Args1],
784   getwf_args1(Args1, LA, C1, H1, T, VH1, VT).
785
786 getwf_args1(Args, LA, C, H, T, VH, VT) :- C =:= 0', |
787   getwf_args1(Args, LA, H, T, VH, VT).
788 getwf_args1(Args, LA, C, H, T, VH, VT) :- C =:= 0') |
789   Args = [],
790   H = [getc(LA) | T],
791   VH = VT.
792
793
794 %-------------------------------------------------------------
795 % getv  -V  -LA +C -H ?T -VH ?VT
796 % getwv -WV -LA +C -H ?T -VH ?VT
797 %-------------------------------------------------------------
798
799 % getv(V, LA, 0'{, H, T, VH, VT) :-
800
801 %-------------------------------------------------------------
802
803 getwv(WV, LA, 0'{, H, T, VH, VT) :-
804   H = [getc(C) | H1],
805   getwv_elems(Elems, LA, C, H1, T, VH, VT),
806   new_vector(V, Elems),
807   WV = vector(V).
808
809 getwv_elems(Elems, LA, C, H, T, VH, VT) :- C =:= 0'} |
810   Elems = [],
```

```
811     H = [getc(LA) | T],
812     VH = VT.
813   getwv_elems(Elems, LA, C, H, T, VH, VT) :- C =\= 0'} |
814     getwt1(E, C1, C, H, H1, VH, VH1),
815     Elems = [E | Elems1],
816     getwv_elems1(Elems1, LA, C1, H1, T, VH1, VT).
817
818   getwv_elems1(Elems, LA, C, H, T, VH, VT) :- C =:= 0'} |
819     Elems = [],
820     H = [getc(LA) | T],
821     VH = VT.
822   getwv_elems1(Elems, LA, C, H, T, VH, VT) :- C =:= 0', |
823     getwt1(E, C1, H, H1, VH, VH1),
824     Elems = [E | Elems1],
825     getwv_elems1(Elems1, LA, C1, H1, T, VH1, VT).
826
827
828   %-----------------------------------------------------------------
829   % gets  -S  -LA +C -H ?T -VH ?VT
830   % getws -WS -LA +C -H ?T -VH ?VT
831   %-----------------------------------------------------------------
832
833   gets(S, LA, 0'", H, T, VH, VT) :-
834     gets1(L, LA, H, T),
835     new_string(S, L, CSIZE),
836     VH = VT.
837
838   gets1(L, LA, H, T) :-
839     H = [getc(C) | H1],
840     gets1(L, LA, C, H1, T).
841
842   gets1(L, LA, C, H, T) :- C =:= 0'" |
843     L = [],
844     H = [getc(LA) | T].
845   gets1(L, LA, C, H, T) :- C =:= 0'\ |
846     H = [getc(C1) | H1],
847     gets_escape(L, LA, C1, H1, T).
848   gets1(L, LA, C, H, T) :- C =\= 0'", C =\= 0'\ |
849     L = [C | L1],
850     gets1(L1, LA, H, T).
851
852   gets_escape(L, LA, C, H, T) :- C =:= 0'a |  % Alert
853     L = [7 | L1],
854     gets1(L1, LA, H, T).
855   gets_escape(L, LA, C, H, T) :- C =:= 0'b |  % Backspace
856     L = [8 | L1],
857     gets1(L1, LA, H, T).
858   gets_escape(L, LA, C, H, T) :- C =:= 0't |  % Tab
859     L = [9 | L1],
860     gets1(L1, LA, H, T).
861   gets_escape(L, LA, C, H, T) :- C =:= 0'n |  % Newline
862     L = [10 | L1],
863     gets1(L1, LA, H, T).
864   gets_escape(L, LA, C, H, T) :- C =:= 0'v |  % Vertical Tab
865     L = [11 | L1],
866     gets1(L1, LA, H, T).
867   gets_escape(L, LA, C, H, T) :- C =:= 0'f |  % Formfeed
868     L = [12 | L1],
```

```
869     gets1(L1, LA, H, T).
870   gets_escape(L, LA, C, H, T) :- C =:= 0'r |  % Carriage Return
871     L = [13 | L1],
872     gets1(L1, LA, H, T).
873   gets_escape(L, LA, C, H, T) :- C =:= 10 |
874     %--- Escaped Newline is ignored.
875     gets1(L, LA, H, T).
876   otherwise.
877   gets_escape(L, LA, C, H, T) :-
878     L = [0'\, C | L1],
879     gets1(L1, LA, H, T).
880
881   %-----------------------------------------------------------------
882
883   getws(WS, LA, C, H, T, VH, VT) :-
884     gets(S, LA, C, H, T, VH, VT),
885     WS = string(S).
886
887
888   % end socket.
```

# 付 録B　プログラム例

本章では、dKLIC の出力する KLIC/KL1 プログラムの例を示す。詳細については、第 6.1 節を参照せよ。

## B.1　appcli

```
1  /*************************************************************
2  * Append Client using vsock.kl1
3  *
4  * Yusuke TAKAGI (mailto:takagi@ueda.info.waseda.ac.jp)
5  * 1999/12/29  syncronized socket I/O
6  *************************************************************/
7
8  :- module main.
9
10 main :-
11   Port = 10000,
12   unix:argc(ArgC),
13   args(ArgC, Host),
14   klicio:klicio([
15     connect(inet(Host, Port), normal(S)),
16     stdout(normal([puttq(remote:append/3), nl,
17       fflush(0) | Stdout ])) ]),
18   socket:socket([puttq(remote:append(X,Y,Z)), 0'., nl,
19     fflush(0), gett(Bound), bind(Bound) | Sock ], S ),
20   %% builtin:print(Bound) @lower_priority,
21   send(5, X, Y, Z, Stdout, Sock).
22
23 send(N, X, Y, Z, Stdout, Sock) :- N=:=0 |
24   X = [],
25   Y = [],
26   Sock = [bound(XY) | Sock1],
27   end(XY, Z, Stdout, Sock1) @lower_priority.
28 send(N, X, Y, Z, Stdout, Sock) :- N > 0 |
29   X = [N | X1],
30   Y = [N | Y1],
31   Sock = [bound(XY) | Sock1],
32   send(XY, ~(N-1), X1, Y1, Z, Stdout, Sock1) @lower_priority.
33
34 send(Bound, N, X, Y, Z, Stdout, Sock) :- wait(Bound) |
35   Sock = [puttq(Bound), 0'., nl,
36     fflush(0), gett(Res), bind(Res) | Sock1 ],
37   builtin:print(Bound),
38   send(N, X, Y, Z, Stdout, Sock1) @lower_priority.
39
40 end(Bound, Z, Stdout, Sock) :- wait(Bound) |
```

```
41    Sock = [puttq(Bound), 0'., nl,
42      fflush(0), gett(Res), bind(Res) ],
43    Stdout = [fwrite("append: "), puttq(Z), nl, fflush(0)].
44
45  args(C, Host) :- C=:=0 |
46    Host = "localhost".
47  args(C, Host) :- C > 0 |
48    unix:argv([Host | _]).
49
50
51  % end Append Client
```

## B.2 appsrv

```
1  /*************************************************************
2  * Append Server using vsock.kl1
3  *
4  * Yusuke TAKAGI (mailto:takagi@ueda.info.waseda.ac.jp)
5  * 1999/12/29  syncronized socket I/O
6  *************************************************************/
7
8  :- module main.
9
10 main :-
11   Port = 10000,
12   klicio:klicio([
13     stdout(normal(Stdout)),
14     bind(inet(Port), normal(ServerSocket)) ]),
15   ServerSocket = [accept(normal(S))],
16   socket:socket(Sock, S),
17   serve(Stdout, Sock).
18
19 serve(Stdout, Sock) :-
20   Sock = [gett(T) | Sock1],
21   serve(T, Stdout, Sock1).
22
23 serve(T, Stdout, Sock) :- T = end_of_file |
24   Stdout = [fwrite("Socket ended.\n"), fflush(0)],
25   Sock = [variable_table(Tab) | Sock1],
26   end(Tab, Sock1).
27 serve(T, Stdout, Sock) :- T = (remote:append(X,Y,Z)) |
28   generic:call(predicate#(remote:append/3), X, Y, Z),
29   Stdout = [fwrite("remote:append/3\n"), fflush(0) | Stdout1],
30   res(Stdout1, Sock).
31 otherwise.
32 serve(T, Stdout, Sock) :-
33   Sock = [bind(T) | Sock1],
34   %% builtin:print(T),
35   res(Stdout, Sock1) @lower_priority.
36
37 res(Stdout, Sock) :-
38   Sock = [bound(Bound) | Sock1],
39   res(Bound, Stdout, Sock1) @lower_priority.
40
41 res(Bound, Stdout, Sock) :- wait(Bound) |
42   Sock = [puttq(Bound), 0'., nl, fflush(0) | Sock1],
43   builtin:print(Bound),
44   serve(Stdout, Sock1).
45
46 end(Tab, Sock) :- wait(Tab) |
47   builtin:print(Tab),
48   Sock = [/*puttq(Tab), 0'., nl, fflush(0)*/].
49
50
51 :- module remote.
52
53 append(X, Y, Z) :- X = [] |
54   Z = Y.
55 append(X, Y, Z) :- X = [A | X1] |
56   append(X1, Y, Z1),
```

```
57    Z = [A | Z1].
58
59
60  % end Append Server
```

# 付 録C　実験

本章では、dKLIC 実装のために行われた予備実験のプログラムと、その実行結果を示す。実験の詳細については、第 4 章を参照せよ。

## C.1　変数の識別

```
1  % Guard Unification of Two Unbound Variables
2
3  :- module main.
4
5  main :-
6    builtin:print(xyz(X,Y,Z)),
7    X=Y,
8    gu(X,X),
9    gu(X,Y),
10   gu(X,Z),
11   gnu(X,X),
12   gnu(X,Y),
13   gnu(X,Z).
14
15 gu(X, Y) :- X = Y | builtin:print(=(X,Y)).
16 alternatively.
17 gu(X, Y) :-         builtin:print(~=(X,Y)).
18
19 gnu(X, Y) :- X \= Y | builtin:print(\=(X,Y)).
20 alternatively.
21 gnu(X, Y) :-          builtin:print(~\=(X,Y)).
22
23 % end GUnification of 2 Variables
```

### C.1.1　実行結果

```
1  xyz(_B,_B,_7)
2  ~=(_B,_B)
3  ~=(_B,_B)
4  ~=(_B,_7)
5  ~\=(_B,_B)
6  ~\=(_B,_B)
7  ~\=(_B,_7)
```

## C.2 メタKL1項

```
1  /*************************************************************
2  * Test Wrapped Terms
3  *
4  * by Yusuke TAKAGI (mailto:takagi@ueda.info.waseda.ac.jp)
5  * 1999/11/18
6  *************************************************************/
7
8  :- module main.
9
10 main :-
11   klicio:klicio([
12     stdin(normal(Stdin)),
13     stdout(normal(Stdout)) ]),
14   variable:wrap(X, W),
15   Stdout = [
16     fwrite("Wrapped compile-time variable is "), putt(W), nl,
17     fwrite("Enter end_of_file. to exit!"), nl,
18     fflush(0) | Stdout2],
19   main(Stdin, Stdout2).
20
21 main(Stdin, Stdout) :-
22   Stdin = [getwt(normal(WTerm)) | Stdin2],
23   main(WTerm, Stdin2, Stdout).
24
25 main(WTerm, Stdin, Stdout) :-
26   WTerm = end_of_file |
27   Stdin = [],
28   put(WTerm, Stdout, [fwrite("bye!"), nl, fflush(0)]).
29 main(WTerm, Stdin, Stdout) :-
30   WTerm = atom(end_of_file) |
31   Stdin = [],
32   put(WTerm, Stdout, [fwrite("bye!"), nl, fflush(0)]).
33 otherwise.
34 main(WTerm, Stdin, Stdout) :-
35   put(WTerm, Stdout, Stdout2),
36   main(Stdin, Stdout2).
37
38 put(WTerm, Stdout, Stdout2) :-
39   Stdout = [fwrite("Wrapped "), putwt(WTerm),
40     fwrite(" is "), putt(WTerm), nl, fflush(0) | Stdout2].
41
42 % end wterm.kl1
```

### C.2.1 実行例

```
1  Wrapped compile-time variable is variable("_")
2  Enter end_of_file. to exit!
3  Var.
4  Wrapped Var is variable("Var")
5  end_of_file.
6  Wrapped end_of_file is atom(end_of_file)
7  bye!
```

## C.3 クライアント-サーバ

### C.3.1 クライアント

```
1  /*************************************************************
2  * Client test
3  *
4  * Yusuke Takagi (mailto:takagi@ueda.info.waseda.ac.jp)
5  * 1999/11/3
6  *************************************************************/
7
8  :- module main.
9
10 main :-
11   Port = 10000,
12   unix:argc(ArgC),
13   args(ArgC, Host, Message),
14   unix:unix([connect(inet(Host, Port), normal(Socket))]),
15   Socket = [fwrite(Message), fflush(0)],
16   io:outstream([
17     print('Client said '),
18     print(Message),
19     print(' to Server.'), nl ]).
20
21 args(0, Host, Message) :-
22   Host = "localhost",
23   Message = "Hello.".
24 args(1, Host, Message) :-
25   unix:argv([Host]),
26   Message = "Hello.".
27 args(2, Host, Message) :-
28   unix:argv([Host, Message]).
```

### C.3.2 サーバ

```
1  /*************************************************************
2  * Server test
3  *
4  * Yusuke Takagi (mailto:takagi@ueda.info.waseda.ac.jp)
5  * 1999/11/3
6  *************************************************************/
7
8  :- module main.
9
10 main :-
11   Port = 10000,
12   unix:unix([bind(inet(Port), normal(ServerSocket))]),
13   ServerSocket = [accept(normal(Socket))],
14   Socket = [fread(1000,String)],
15   io:outstream([
16     print('Server heard '),
17     print(String),
18     print(' from Client.'), nl ]).
```

## C.4 遠隔ゴール実行

```
1  /*************************************************************
2  * Test of Remote Predicate Call
3  *
4  * Yusuke Takagi (mailto:takagi@ueda.info.waseda.ac.jp)
5  * 1999/11/18
6  *************************************************************/
7
8  :- module main.
9
10 main :-
11   unix:unix([
12     bind(inet(8000), normal(ServerSocket)),
13     connect(inet("localhost", 8000), normal(Client)) ]),
14   ServerSocket = [accept(normal(Server))],
15   client:client(Client),
16   Server = [fread(4096,String)],
17   main(String).
18
19 main("server:server(Server)") :-
20   generic:call(predicate#(server:server/1), "call constant"),
21   generic:apply(predicate#(server:server/1), {"apply constant"}),
22   %-----------------------------------------------------------
23   % Predicate = predicate#(server:server/1),
24   Predicate = predicate#(:(server, /(server, 1))),
25   generic:call(Predicate, "call variable"),
26   generic:apply(Predicate, {"apply variable"}),
27   %-----------------------------------------------------------
28   generic:new(module, M, server),
29   generic:new(predicate, Dynamic, M, server, 1),
30   generic:call(Dynamic, "call dynamic"),
31   generic:apply(Dynamic, {"apply dynamic"}).
32
33
34 :- module client.
35 client(Client) :-
36   Client = [fwrite("server:server(Server)")].
37
38
39 :- module server.
40 server(Server) :-
41   builtin:print(Server).
```

### C.4.1 実行結果

```
1 "apply dynamic"
2 "call dynamic"
3 "apply variable"
4 "call variable"
5 "apply constant"
6 "call constant"
```