輪講資料
KL1
並列処理言語

重田 大助

平成9年5月8日 平成9年5月29日

1 はじめに

まず、KL1を理解するために、論理型プログラミング言語について説明し、その後で、KL1という言語についての説明をする。それから、KL1の処理系KLICの説明をする。

2 論理型プログラミング言語

2.1 論理プログラミング

論理プログラミングでは、プログラムはある論理体型に従った論理式の集合で表される。問題と解の性質を用いて、その問題に内在する関係や条件などを論理式として仕様化し、論理プログラムとして宣言的に記述しておけば、論理型言語を処理するシステムが自動的にその問題の解き方の制御を行ってくれる。

2.2 論理型プログラミング言語の特徴

1. 論理プログラムは述語 (predicate) とよばれる論理式によって構成される。述語は、

   $$P(t_1, t_2, \cdots, t_n) \qquad (n \geq 0) \tag{1}$$

   のように、述語名を表す記号と項 (term) とよばれる0個以上の引数とからなる。述語はその引数として、論理値 (true/false) をもつ。項としては、(論理) 変数/定数/構造体 (functor; ファンクタ) の3つが許されている。構造体は、

   $$F(t_1, t_2, \cdots, t_n) \qquad (m \geq 0) \tag{2}$$

   のように、構造体名を表す記号と0個以上の項からなる引数リストから構成される。

2. 一階述語論理の場合、論理式の集まりは節 (clause) とよばれるつぎのいずれかの形式にしたがって述語を列挙したものになる。

   $$P_0 \tag{3}$$

   $$P_0 \leftarrow P_1 P_2 \cdots P_n \qquad (n \geq 1) \tag{4}$$

   $$\leftarrow P_1 P_2 \cdots P_n \qquad (n \geq 1) \tag{5}$$

   すなわち、←の左側にある節の頭部 (head) はたかだか1つの項から構成される。一方、←の右側にある節の本体 (body) には、いくつの項があってもよい。また、節本体を構成する複数個の述語はゴール (goal) ともよばれる。

3. 論理変数の値には、項以外に「未定義 (undefined)」というものが許されている。「未定義」の意味は「変数の値がまだ確定していない」ことを示している。項のうち論理変数だけがつぎに述べるユニフィケーションによって操作可能である。

4. プログラムの実行はユニフィケーション (unification; 単一化) とよばれる記号パターンのマッチング (照合) を繰り返すことである。ユニフィケーションとは、2つの項を代入によって同一化/統一化することである。ここで、XとYを2つの項とすると、

   (a) XとYとが同一定数あるいは同一変数ならばなにもしない。ユニフィケーションが成功するだけである。

$$\begin{array}{lllllll}
 & & & \text{A} & \text{N} & \text{D 並} & \text{列 性} \\
 & & & \downarrow & \downarrow & & \downarrow \\
 & \rightarrow P_1 & :- & Q_1, & Q_2, & \cdots, & Q_n \\
 & \rightarrow P_2 & :- & \sim & & & \\
\text{OR 並列性} & \rightarrow P_3 & :- & \sim & & & \\
 & & & & & & \\
 & \rightarrow P_m & :- & \sim & & &
\end{array}$$

図 1: 論理型プログラミング言語の並列性

(b) X が未束縛変数 (値が未定義の変数) ならば、Y の (定数でも変数でもその) 値を X の値とする (具体化)。ユニフィケーションは成功する。Y も未束縛変数ならば、X と Y とは同一の値をもつ変数とする。すなわち、それらのいずれかに値が束縛された場合には片一方の変数にも同じ値が束縛される。

(c) X も Y も構造体ならば、その要素 (項) すべてについてユニフィケーションを行う。要素すべてのユニフィケーションが成功した場合だけが X と Y とのユニフィケーションの成功である。

2.3 論理型プログラミング言語の並列性

1. OR 並列性

   図 1に示すように、候補節 (頭部の述語記号と引数の個数が同じであるような節の集合) 間には論理的な OR 関係がある。**OR 並列性** (OR parallelism) とは、ある述語呼び出し (ゴールの評価) に対し、OR 関係にある各候補節の頭部とのユニフィケーションおよびそれに引き続く本体ゴール列の実行を並列に行うことができることを指す。

2. AND 並列性

   図 1に示すように、節本体のゴール列間には論理的な AND 関係がある。**AND 並列性** (AND parallelism) とは、節本体の各ゴールの実行 (評価) を並列に行うことができることを示す。

3. 引数間並列性

   引数間並列性とは、複数の引数をもつ述語 (節) や構造体のユニフィケーションを行う際に各引数のユニフィケーションを並列に行うことができることを指す。

4. その他の並列性

   ユニフィケーションにおいて、事実 (データベース) を探索する場合などには、一般的なデータベース操作と同様にサーチ並列性が存在する。サーチ並列性とは、事実の探索時に線形探索ではなく連想メモリなどを利用した並列探索が可能であることを指す。

2.4 論理型並列プログラミング言語

2.3で述べた並列性を本質的なセマンティクスとして言語機能に含んでしまうと、つぎのような点でプログラミング言語としての実用性に欠ける。

1. 非決定性 (並列性) をシステムやプログラマが制御できない。
2. 論理的な並列性の爆発に対応できる物理的ファシリティを装備することは困難である。

したがって実用的には、並列性を野放しにするのではなく、それをなんらかの形で制御できるセマンティクスが必要である。

KL1 は、AND 並列性を制限せずに OR 並列性を制限するセマンティクスをもつ AND 並列言語である。AND 並列言語はコミット選択 (commited choice) 型プログラミング言語ともよばれる。コミット選択言語では、OR 並列性をゴールと節頭部のユニフィケーション時に利用するが、それに引き続く本体の実行をたかだか 1 個のゴール列に絞ってその OR 並列性を制限するセマンティクス (ガード (guard) 機構とかトラスト (trust) 機構とよぶ) を採用している。

3 KL1

3.1 構文と実行機能の基本

3.1.1 プログラムの形式と基本実行機能

KL1 では、2.3で述べた OR 並列性を制限するために、節の形式を、一般的につぎの形で表す。

述語名 (引数, ⋯) : −ガード | ボディ.

引数 : KL1では変数名の有効範囲はひとつの節の中だけで、同じ名前でも別の節にあれば全く別のものとみなされる。同じ述語の再帰的な呼び出しについても、そのたびごとに同じ名前の変数が別の変数を意味する。

ガード : 節の適用条件を指定する部分。カンマで区切ってゴールをいくつでも書け、すべてを満足した場合にだけ適用条件を満たしたものとする。ガードには、ユニフィケーションの他に、大小比較など、言語で定義するある決まった種類の述語の呼び出しだけが書ける。

ボディ : 節を選んだときに実行すべき部分。やはりカンマで区切ってゴールをいくつでも書け、その節が選ばれたらすべてを実行する。ボディにはユニフィケーションの他に、他の述語、あるいは自分自身を呼び出すゴールが書ける。

ユニフィケーション : KL1でいうユニフィケーションは、等号(“=”)の両辺に値を書いた形のゴールのことをもいう。

ガード·ユニフィケーション : 節の選択条件になるガードでのユニフィケーションは、両辺が等しいことが条件の一部であることを示すものである。これは単に等しいかどうかを試すだけなので受動的なユニフィケーション(passive unification)とも呼ばれる。

ボディ·ユニフィケーション : 実行を指定するボディでのユニフィケーションは、両辺を等しいものにするという積極的な操作で、能動的なユニフィケーション(active unification)とも呼ばれる。

KL1で呼び出せる述語には、あらかじめ言語で定義する組込述語(built-in predicates)と、プログラム中に定義するユーザ定義述語(user-defined predicates)の2種類がある。

組込み述語の多くは種々のデータについての基本操作や基本的な値の検査のためのもので、それらがどのように動作するかはKL1言語の仕様の一部として定められている。特殊な組込み述語として、引数のない述語”`true`” がある。これはガードに現われれば常に真(つまり無条件)、ボディに現われれば何もしないこと(no operation)を意味する。

```
:- module main.

main :- not(1,X), not(X,Y),
        io:outstream([print(Y),nl]).

not(In,Out) :- In=0 | Out=1.
not(In,Out) :- In=1 | Out=0.
```

図 2: KL1のプログラム例

ガードがOR並列性を制限している部分で、ゴールを実行する際に、その述語を定義する節がいくつかあるとすると、まず各節の適用条件であるガードの真偽を試す。次にガードが真であるとわかった節をひとつだけ選び、そのボディのゴール(ユニフィケーションを含む)をすべて今後実行すべきゴールとする。これを繰り返すのがKL1プログラムの実行過程である。

3.1.2 並列実行、通信、同期の機構

節はある条件(ガード)を満たすゴールを、ボディのゴールに書き換える書換ルールであると見ることもできる。書き換えを繰り返すことによって最終的にはすべてのゴールが何もしないゴール”`true`” にまで書き換えられると実行は終了する。この過程は、最初に与えられたゴールを”`true`” にまで書き換えていく簡約化(reduction)であるとも考えられる。

この書換えはかならずしも逐次的にひとつずつ行わなくとも良い。複数のゴールについて同時に行なえば、実行は並列になる。これが2.3のAND並列性の実現である。

KL1では複数のゴールが同じ変数を共有(2.3の共有変数)することがある。あるゴールが変数の値を決めれば、それを共有する他のゴールは、その値を使ってどの節を使って簡約化するかを決めることもできる。これがKL1の通信機構である。つまり、通信は共有変数を用いて行なう。

KL1では、複数のゴールがあるときにはどの順

番に実行するかは特に決めておらず、処理系の都合で決めていいことになっている。仮に、それらのゴールの実行順序のために、あるゴールが未具体化の変数をガードで参照したとすると、どの節を使って良いのかわからないので、このような場合、当面の間このゴールの実行は見合わせる。これを実行の中断 (suspension) と呼ぶ。

このゴールが実行できないとなると、他のゴールに実行が移る。ここで、先程参照した未具体化変数の具体化が行なわれると、実行を中断していたゴールを中断し続ける理由はなくなった。そこで、先程のゴールの実行を再開する。これが KL1 の同期の機構である。つまり、同期はガードで必要な値を検査すると自動的に行なわれる。

3.1.3　論理変数

KL1 などの論理型言語が C などの手続き型言語ともっとも大きく異なるのは、変数の扱いである。手続き型言語の変数とはっきり区別するために論理変数などと呼ぶこともある。

- 変数はメモリ番地ではない
  普通の手続き型言語の変数は、特定の番地から始まるある大きさのメモリ領域につけた名前である。計算機にメモリを表すものだから、中身を読み出したり書き込んだりすることができる。KL1 の変数はメモリのような値の格納場所を表すものではなく、値につけた名前である。だから、その値を参照することはできても、値を変更するなどということはできない。

- 変数に型はない
  Kl1 では変数についてどんな型のデータが入るかは宣言しない。ソース ・プログラム中の同じ節の同じ変数が、ある呼ばれ方をしたときはひとつの型のデータ、別のときには別の型のデータを値とすることがある。

3.1.4　構造を持ったデータ

構造を持ったデータとは、要素となるデータを集めてできているデータのことである。KL1 で扱える構造を持つデータの代表的なものには、以下のものがある。

- ファンクタ　　2.2参照

- コンス
  コンスは任意の型のふたつの要素を持つ構造である。ふたつの要素は car,cdr と呼ぶ。表記にはカギ括弧対 (“`[`” と”`]`”) の間に car,cdr の要素を縦棒 (“`|`”) で区切って書き並べる。たとえば”`[a|b]`” は car が`a`、cdr が`b` であるようなコンスを意味する。

  コンスそれ自体はふたつの要素だけを持つ構造データだが、これを組み合わせて様々なデータ構造を作れる。その代表がリスト構造である。コンスの car がひとつの要素、cdr がリストの残り、要素がひとつもないリストは”`[]`” で表すものと取り決めれば、ふたつの要素 a,b を持つリストは”`[a|[b|[]]]`” と、ネストしたふたつのコンスを用いて表すことができる。このままではリストが長くなると多数の括弧が必要になるので、これを”`[a,b]`” のように書き表す。

- 未完成データ構造

  ファンクタやコンスのように任意の型の要素を持てる構造データの場合、要素の値がまだ決まっていないことがあっても良い。このような構造を 未完成データ構造 (incomplete data structure) と呼ぶ。

  未完成データ構造の値の決まっていない要素は、変数として書き表す。たとえば、”`[a|X]`” は car が`a`、cdr が`X` という名前を持つ変数であるようなコンスを表す。

3.2　プロセスとストリーム通信

これまでに KL1 の基本的な言語仕様を解説した。ここでは、Kl1 で良く使われるプロセス (process) とプロセス間をつなぐストリーム (stream) を用いたプログラミング ・スタイルについて解説する。

3.2.1　プロセス

前述の通り、KL1 の基本的な実行機構は簡約化の (並列な) 繰り返しである。しかし、単純な簡約化の繰り返しというだけでは、複雑な計算をわかりやすく記述するには役不足で、数多くの簡約化

```
count_down(0).
count_down(N) :- N>0 |
                M:=N-1, count_down(M).
```

図 3: プロセスの例

操作をうまくまとめあげるような概念が有用である。そのような役割を果たすのがプロセスという考え方である。

ボディに自身と同じ述語を呼び出すゴールをひとつ含むような節は、(引数は変わるが) 同じことを繰り返し行なうループと見ることができる。基本的な実行機構である一つ一つに簡約化は細切れの操作なのだが、繰り返し全体を見ると、ある程度大きさを持った連続性のある計算過程と考えることができる。このような計算過程をプロセスと呼ぶ。

3.2.2 ストリーム通信

- プロセスの複合

  ひとつのプロセスの計算結果を用いて、別のプロセスが計算をするように、プロセスを組み合わせるプログラムを組むことができる。節の各ゴール間には実行の並列性があるので、計算が最後まで終わっていなくても、途中までの答を返すことができる。

  たとえば、出力側のプロセスはコンス構造の car は値が確定していて、cdr は変数のままであるような未完成データ構造を共有変数として出力することができる。しかし、計算結果が全体としてコンスであることが決まっているだけで受信側のプロセスのガードの判定ができるならば、出力側のプロセスの実行が少し進んだだけで、受信側のプロセスもすぐに実行できる。

  このように、要素がまだ決まっていない構造体を結果として返し、後からその要素を確定していくことができる、という仕組みは、KL1 の大きな特徴のひとつである。要素がすべて決まるまで結果を返せないのだとしたら、出力側の実行がすべて終わるまで入力側の実行を始めることができない。それだけ並列度が低くなってしまうわけである。

```
sum([],PSum,Sum) :- Sum=PSum.
sum([One|Rest],PSum,Sum) :-
      NewPSum:=PSum+One,
      sum(Rest,NewPSum,Sum).

sum(List,Sum) :- sum(List,0,Sum).

naturals(N,M,List) :- N>=M | List=[].
naturals(N,M,List) :- N<M |
      List=[N|Rest], N1:=N+1,
      naturals(N1,M,Rest).

naturals(M,List) :- naturals(0,M,List).

sum_up_to(N,Sum) :- naturals(N,List),
                    sum(List,Sum).
```

図 4: プロセスの複合

- ストリーム通信

  要素が段々決まっていくような未完成データ構造を、データが次々に流れていくような通信路だと考えることもできる。出力側のプロセスはこの通信路に次々にデータを流していき、受信側のプロセスがその通信路の出口からデータを読み取っては、計算を進める。データがまだ到着していなければ、読み取り側のプロセスは待たされる。

  通信路を実現しているのはデータ構造コンスである。その car にはストリームを流れるデータが入り、cdr はストリームの続きを表す。空のリスト [] は、必要な通信が終りまで来てしまい、もうこれ以上メッセージがないことに対応する。このように通信路がデータ構造になっているので、データの流れる順序はどのようなデータ構造をつくるかで完全に決まってしまうので、計算の実行順序や通信遅延の違いが、そのままデータの順序に影響することはない。このような通信路を ストリーム (stream) という。また、ストリームを流れる

データはプロセス間で受け渡される メッセージ (message) であるとも考えられる。

4 KLIC

4.1 KLIC の基本処理

Kl1 の処理系 KLIC は KL1 プログラムをいったん C プログラムに変換し、ターゲットマシンのオブジェクトコード生成は、その計算機用の C コンパイラに行なわせる方式を取っている。

KLIC 分散処理系の計算モデルは、共有メモリを持たない複数のノードが、互いにメッセージ通信をしながら処理を行なうというものである。並列実行は`@node` プラグマを用いて明示的に記述し、$goal(X_1, \cdots, X_n)$`@node`(N_c) という形のボディゴールがノード (個別のプロセサ、または内部では自動負荷分散するようなプロセサの集まり)N_p上に現われると、このゴールはノード N_cに送られ、そこで実行される。このとき、未具体化の変数 X_iについては、N_p上の変数を指す外部参照ポインタが N_cに送られる。

N_c上で X_iの値が参照されると、N_cから N_pに、この外部参照ポインタが指す値を要求する read メッセージが送られる。N_pは answer メッセージにより、要求された値を送信する。また、N_c上で X_iを具体化した場合は、N_pにこの具体値との同一化を要求する unify メッセージを送る。 answer を待っている間や、unify が来る前に N_pで値を参照しようとした場合、それらのゴールは外部参照ポインタや X_iにフックし (未定義変数が具体化待ち変数になりこのゴールを指す)、他のゴールの実行に切り替わる。answer メッセージをノード N_cが受信すると、中断ゴールの実行が再開され、ノード間ガーベジコレクションのためにノード N_pに対して release メッセージを送り、X_iが参照されなくなったことを知らせる。

4.2 スケジューリングと通信延期による効率化

- スケジューリングによる効率化

  KL1 は一つ一つのゴールが並行実行の単位であるため、その細粒度の並行性制御に伴うオーバヘッドが速度低下を引き起こしている。そこで、実行時のオーバヘッドを軽減するために並行実行の粒度を大きくする。

  そのために、プログラムのデータ依存解析を行なうことで、デッドロックを起こさないようにプログラムの一部の実行順序を決定する。そして、逐次化されたゴールを並列実行の単位とすれば、並行性制御に伴うオーバヘッドを軽減することができる。

- 通信延期による効率化

  KLIC では、一回のデータのやりとりにおいて、read、answer、release、という 3 種類の物理通信が必要となる。このため、細粒度通信が頻発すると大きなオーバヘッドになる。そこで、一度のメッセージ通信でまとめてデータを送信すると、通信の粒度を上げて実行速度を向上させることができる。通信延期は、返信を適度に送らせることで一度のメッセージ通信で送信するデータを多くして、実行を速くする方法である。

  ただし、遅延して次のデータができるのを待っているときに、遅延によってためているデータが次のデータを作るための条件となっているときには、遅延によっても新しいデータができず、遅延自体に意味がない。そのためこうした無意味な遅延を引き起こすような場合を静的解析によってあらかじめ検出し、遅延を行なわないようにする。

5 つぎに

並列処理言語の特質について触れ、その後いくつかの例についてみてみる。

6 並列処理言語

並列処理言語の特質は、並列動作を記述できること、非決定性、並列処理の基本操作 (排他/同期/通信) が安全かつ簡便に書けることである。以下、それぞれについて説明する。

6.1　並列動作の記述

並列性を明示する方法は、プロセスを 1 つのモジュールとして宣言する方法と、1 つのモジュール内で並列実行可能な部分を指定する方法との 2 つにわけられる。

プロセスを 1 つのモジュールとして宣言すると、変数などの管理が容易であり、論理構造に基づいたモジュール分割が可能である反面、細かな並列動作の記述に不便である。

1 つのモジュール内で並列実行可能な部分を指定すると、効率の良いコード生成ができ、単純であるのに対し、変数のアクセス制御が困難で、モジュール化できないという問題がある。

6.2　非決定性

非決定性とは実行可能な命令系列が複数存在することである。これには二つの解釈が可能であり、一つは、個々の命令系列が出す結果の中で与えられた条件を満たすものを全体の答とする場合であり、もう一つは、個々の命令系列から生じる結果の中でどれを全体の答としても良い場合である。

6.3　排他

二つ以上のプロセスで逐次的資源 (プロセス間で共用される資源 (共有資源) のうち同時にはただ一つのプロセスしか使用できない資源) を使用する場合には、プロセス間で同期をとり、たかだか一つのプロセスがその資源を占有するようにしなければならない。すなわち、一つのプロセスが資源を使用している場合には、その資源が解放されるまで他のプロセスを待たせなければならない。これは、相互排除 (mutual exclusion) あるいは排他制御 (exclusive control) と呼ばれる。各プロセスにおいて、このような逐次的資源を使用する部分は、臨界領域 (critical region) あるいはきわどい部分と呼ばれている。

6.4　同期

並列に動作するプロセスが協力して処理を進めるためには、プロセス間で必要な情報を交換する必要があるとともに、プロセスの実行を調整して正しい結果となる実行順序を保証しなければならない。

制御情報の交信、すなわち同期は、正しい結果になるように実行の順序を保証する。同期の使われ方には、条件同期と相互排除がある。条件同期とは、あるプロセスが他のプロセスによってある条件を満たされるまで待つことである。変数を共有するタイプのプロセス間の通信では、同時にアクセスされる共有変数が存在するため、条件同期と相互排除をともに解決するための機能が必要になる。一方メッセージ転送タイプのプロセス間通信では共有する変数が存在せず、直接プロセス間でデータを移動させるので相互排除の問題は生じない。

6.5　通信

通信の実現方法は、共有変数を用いる、モニタを用いる、メッセージ通信、遠隔手続き呼び出しの 4 つにわけられる。以下にそれぞれの説明を行なう。

6.5.1　共有変数

プロセス間通信を実現する最も簡単な方法で、共有変数に情報を書き込んだり、そこから情報を読み出し対する方法。

6.5.2　モニタ

モニタとは資源の定義とその資源への操作を 1 箇所に集めたものである。モニタは資源の状態を保持するための状態変数 (局所変数) と、その資源に対する操作を実現しているいくつかのグローバル手続き、状態変数を初期化するための文列等で構成される。

モニタの入れ子は禁止されており、モニタはすべてのモジュールからアクセス可能である。

モニタは複数のモジュールから同時にアクセスすることができるが、モニタ内の手続きの実行は完全に排他制御されるのが普通である。

6.5.3　メッセージ通信

知らせたい内容を直接やり取りするものであり、送信側と受信側がそれぞれ送信命令と受信命令を

実行して始めて、データのやり取りがされる。送信側が待たされるかどうかで、同期式メッセージ通信と非同期式メッセージ通信に区別される。

6.5.4 遠隔手続き呼び出し

双方向通信を基本にしたものである。send と receive が組になって現われるのが双方向通信であり、これを一度に行なえるようにしたのが遠隔手続き呼び出しである。

7 並列処理言語の例

ここでは例として、Concurrent Pascal と CSP と ABCL/1 を挙げる。

7.1 Concurrent Pascal

この言語は従来の Pascal に基づいており、並列処理用に process、monitor を用意し、抽象データ型として class を用意している。

7.1.1 process

process は非同期に実行されるプログラム要素であり、内部では逐次処理が行なわれる。命令間の並列性は記述せず、モジュール間の並列性のみを記述する。

process の起動は、init 文により、動的に行なわれる。

7.1.2 monitor

monitor は process 間の情報交換や資源共有を制御するプログラム要素である。

7.1.3 class

class は抽象データ型を定義し、それに対する操作手続きを提供する。class と monitor の違いは同一の class が複数の process から同時に参照されることはないので、class 内の手続きを排他制御する必要はない。

7.2 CSP

7.2.1 プロセス

モジュールという概念が存在しないため、プロセス内外の間で変数のスコープに壁ができず、変数管理が難しい。また生成されるプロセスの数は固定であるので、プロセスの数が動的に変化するような記述は不可能である。

7.2.2 情報交換

プロセス間の相互作用を記述する手段としてメッセージ交換を用意し、非決定的なメッセージ交換を可能にしている。よってデータの一時的やり取りには適しているが、プロセス間でオブジェクトを共有するような場合の記述が困難である。

相互作用は入出力命令を用いて記述するが、この際相手プロセス名を指定する。よって不特定プロセスを相手にするサーバプロセスを記述することができない。

7.2.3 非決定性

ガードを用いて利用できる。具体的には guard 部に論理式に加えて入力文を書くことができる。入力文が guard に含まれている場合は、論理式が真でかつ入力文の相手プロセスの準備ができていれば guard は真とみなされる。これにより必ずしも相手の応答を待つ必要がなく、可能なメッセージ交換から実行できる。

7.3 ABCL/1

オブジェクト計算モデルでは、記憶と計算の能力を持ったオブジェクトと呼ばれる実体が、互いにメッセージを送受信しあいながら計算を進めていく。

ABCL/1 の目的はいろいろな問題が自然に記述できるような通信操作を提供することにある。この目的を達成するために、ABCL/1 では、past、now、future 型の 3 種のメッセージ送信を用意しており、それぞれに ordinary と express という 2 種類の緊急度のものがある。express モードのメッセージは ordinary モードのメッセージの処理中に割り込むことができる。あるメッセージの処理中

に、他のメッセージを待ち合わせる機能 (select) がある。

7.3.1　メッセージ送信

1. past 型

   送りっぱなしのメッセージ

2. now 型

   返信があるまで送信側が待たされるもの

3. future 型

   返信内容はいずれ指定の場所にいれておいてもらうことにして、自分は直ちに次の仕事を始めるというもの

ABCL/1 ではメッセージは送信順に到着する。

7.3.2　select

select が持っているメッセージは優先処理される。つまり、同じ緊急度のメッセージの間でも、追い越しが起きうる。

7.3.3　並列性

ABCL/1 の並列性は

1. 個々のオブジェクトの持つ並列性

2. past 型、future 型のメッセージ通信による並列性

3. parallel construct、multicasting による並列性

- parallel construct

  メッセージの記述の仕方により複数のメッセージを同時に送ることができる。now 型のとき特に効果大。

- multicasting

  メッセージを送るオブジェクトのリストに同時にメッセージを送る方法。

参考文献

[1] 柴山潔. “並列記号処理”. コロナ社, 1991.

[2] KLIC 講習会テキスト -KL1 言語編-, 1993.

[3] 大野和彦, 伊川雅彦, 森眞一郎, 中島浩, 富田眞治. “静的解析による並列論理型言語 KL1 の通信最適化”. In *JSPP’95*, 1995.

[4] 伊川雅彦. “並列論理型言語 KL1 におけるゴール・スケジューリング最適化手法”. Master’s thesis, 京都大学大学院工学研究科情報工学専攻, 1997.

[5] 黒田圭. “KL1 言語処理系 KLIC の動的粒度制御による通信高速化”. 特別研究報告書, 1997.

[6] 大久保英嗣萩原宏. “現代オペレーティングシステムの基礎”. オーム社, 1988.

[7] 計算機システム特論第二プリント.

[8] 伊川雅彦. 輪講資料, 1994.

[9] 津邑公暁. 輪講資料, 1995.

[10] 黒田圭. 輪講資料, 1996.