

|                |   |
|----------------|---|
| 題名             | 実験的リフレクティブ・プログラミング・システム 「ExReps」  |
| 目的             | 核言語（GHC）による、メタインタプリタの段階的拡張及びそれによる並列計算機の記述から、仮想並列ハードウェア上の環境を提供し、GHCのリフレクト機能拡張の研究開発を支援する。   |
| 概要<br>及び<br>特徴 | <p>(概要)</p> <ul style="list-style-type: none"> <li>(1) メタインタプリタの段階的拡張           <ul style="list-style-type: none"> <li>・4行メタインタプリタ</li> <li>・キューとリダクション・カウントを持つメタインタプリタ</li> <li>・変数管理を行うメタインタプリタ</li> </ul> </li> <li>(2) 抽象マシンとプラグマ           <ul style="list-style-type: none"> <li>・様々な抽象マシンの構築とプラグマによる指示</li> </ul> </li> <li>(3) プログラミング・システム上でのプログラム実行           <ul style="list-style-type: none"> <li>・4クイーン・オッドラーニング・グッドパス</li> </ul> </li> <li>(4) リフレクション機能の実現           <ul style="list-style-type: none"> <li>・プロセス・キューの動的表示</li> <li>・リダクション・カウントの動的コントロール</li> <li>・動的ロード・バランス</li> </ul> </li> </ul> <p>(特徴)</p> <ul style="list-style-type: none"> <li>(1) 全てGHCにより記述されている</li> <li>(2) 分散インプリメントに対応</li> <li>(3) インタプリタベースの柔らかなシステム</li> </ul> |
| 構成             | <pre> graph TD     A[ExReps] --- B[ ]     B --- C[Application]     B --- D[Programming System]     B --- E[Abstract Machine]     B --- F[GHC System]     B --- G[PSI-II]   </pre> <p>例えは、4クイーン、グッド・パス、オッド・ラーニングなど</p> <p>シェル、データベース・サーバなどユーザ・インターフェース機能</p> <p>様々なPEの結合トポロジーをもつ仮想分散計算環境を実現</p> <p>PSI上のFGHC処理系に多少の組み込み述語を追加</p>   |

## デモ内 容 (1/3)

## 1 GHCメタインタプリタの段階的拡張

GHCの最も簡単なメタインタプリタは4行で記述できるが、さらにこれを段階的に拡張することで、さまざまな機能を獲得することができる。以下に、各種メタインタプリタを示す。

```

①(行メタインタプリタ
exec(true):- true! true,
exec((P,Q)):- true ! exec(P),exec(Q),
exec(Goal):- nonsys(Goal) !
    reduce(Goal,ReducedGs),
    exec(ReducedGs),
exec(Goal):- sys(Goal) !
    sys_exec(Goal).

②スケジューリング・キューとリダクション
カウンタを管理するメタインタプリタ
exec(G,Res):- true!
    exec([GIT],T,Res,0),
exec([Goal|IH],T,Res,Rc):-
    nonsys(Goal),Goal\=true,Goal\=fail !
    reduce(Goal,H,T,NH,NT,Rc,NRc),
    exec(NH,NT,Res,NRc),
exec([Goal|IH],T,Res,Rc):-
    sys(Goal) !
    sys_exec(Goal,T,NT,Rc,NRc),
    exec(H,NT,Res,NRc),
exec([true|IH],T,Res,Rc):- true !
    exec(H,T,Res,Rc),
exec(T,T,Res,Rc):- true!
    Res=success(Rc),
exec([fail|IH],T,Res,Rc):- true !
    Res=fail(Rc).

```

```

③変数管理をするメタインタプリタ

m_shc(FGoal,Out):- true !
    transfer(FGoal,NGoal,1,Id,Env),
    exec([NGoal|T],T,Id,Env,NEnv,Res),
    print_result(Res,NGoal,Out,NEnv).

exec([GIH],T,Id,Env,NEnv,Res):-
    nonsys(G), G#\=true, G#\=fail!
    reduce((G,T,NT,HN,NT,Id,NId,Env1,Env1),
    exec(HN,NT,Id,Env1,NEnv,Res),
exec([GIH],T,Id,Env,NEnv,Res):-
    sys(G) !
    sys_exec(G,T,NT,Env,Env1),
    exec(H,NT,Id,Env1,NEnv,Res),
exec(T,T,Id,Env,NEnv,Res):- true!
    Res=success,NEnv=Env,
exec([trueIH],T,Id,Env,NEnv,Res):- true !
    exec(H,T,Id,Env,NEnv,Res),
exec([failIH],T,Id,Env,NEnv,Res):- true !
    Res=fail,NEnv=Env.

```

また、図1にこれらメタインタプリタ上での例題実行の一部を示す。

```

GHC> exec(qsort([2,1,3],S),Res,[show_q(10),show_q]
(200)]).

$$\text{re}('18') \Rightarrow [\text{part}(1,[1,A,B]),\text{qsort}(A,C,[1|C]),\text{qsort}(B,D,[2|E])],\text{qsort}([3|E,C])\text{!F}$$


$$\text{re}('20') \Rightarrow [\text{unity}(A,[1]),\text{qsort}([1],B,[3|C]),\text{qsort}(A,C,[1])\text{!D}$$


lsec. 863 msec.
Res => success(25)
S => [1,2,3]
yes

GHC> m_ghc(qsort([2,1,3],S),MOut).
lsec. 498 msec.
MOut => qsort([2,1,3],[1,2,3])
S => 'E1'
yes

GHC> m_ghc(gc_test(S),MOut).

$$++++env\_before\_gc++++$$

[('E2',undf), ('#21',gc_test("E1")), ('#26',undf), ('#25',undf), ('#24',undf), ... ]

$$++++initial\_goal\_before\_gc++++$$

gc_test("E1")

$$++++env\_after\_gc++++$$

[('E1',undf)]

$$++++initial\_goal\_after\_gc++++$$

gc_test([1,2,3] '#18')
gc_test([1,2,3] '#18')

982 msec.
MOut => gc_test([1,2,3] '#18')
S => 'E1'
yes
GHC "

```

図1 メタインタプリタ上の例題実行

## 2 抽象GHCマシンとプログラマ

抽象GHCマシンは、複数のPEがつながれたネットワークから成る。この上で実行されるゴールは、プログラマを付けることによりネットワークを通ってPE間を渡り歩くことができる。図2に、本システムで実現したネットワークの形を示す。

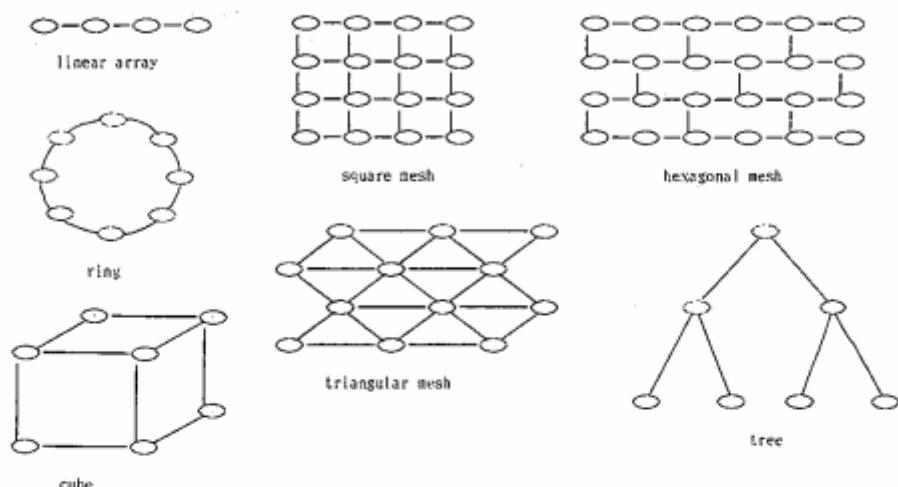
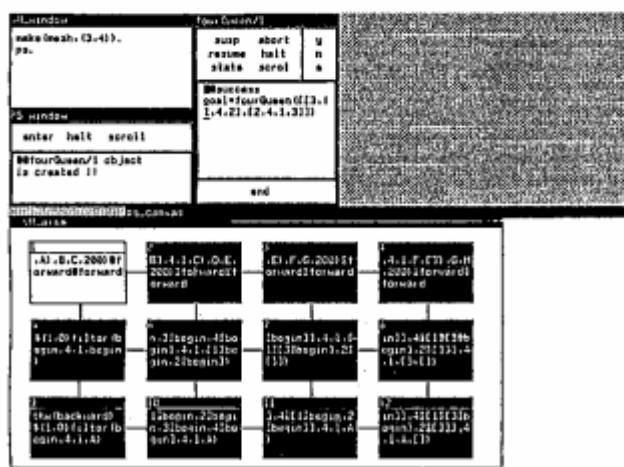


図2 本システムで実現した抽象GHCマシンのネットワーク

## 3. プログラミング・システム上でのプログラム実行

構築した抽象GHCマシンの上にプログラミング・システムを立ち上げることができる。アプリケーション・プログラムはこのプログラミング・システム上で実行される。

図3では、4クイーンのプログラムを、抽象GHCマシン上で分散して実行した。  
図4では、グッドパスのプログラムの並列実行を視覚的に表示した。



#### 4 リフレクト機能

リフレクト機能とは、プログラムを実行しながら、自分の現在の実行状態を取り出したり、変更したりする機能である。

リフレクト機能は、その言語による自己記述つまりメタインタプリタにより実現可能。

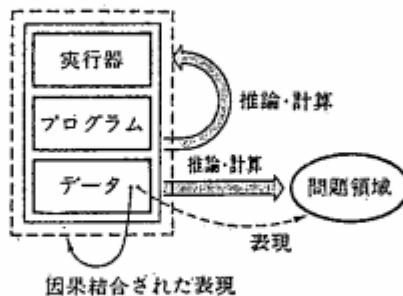


図5 リフレクティブ・システム

本システムでは、拡張したメタインタプリタを用い、次に実行するリフレクト機能を実現した。

図6に、ユーザ・プロセスのキューを表示するプログラムをユーザが記述して実行した例を示す。図7に、ユーザ・プロセスに許されたリダクション数を動的に変更できるプログラムを実行した例を示す。

```
reflect-test/0
suspend abort y
resume halt n
state scroll e
show queue ?? yes
@@unify(A,[2:B])
@@gen(A,7,B)
@@ad(A,2,1)
@@shift([2:A],B)
show queue ??
```

図6 プロセス・キューの動的表示

```
reflect-test2/0
suspend abort y
resume halt n
state scroll e
change_rc?? no
rc_rest=154
change_rc?? yes
add_rc>>80,
rc_rest=177
change_rc??
```

図7 リダクション・カウントの動的コントロール

図8に、マシン上で直接実行することによって、マシンの負荷分散が行えるプログラムを実行した例を示す。

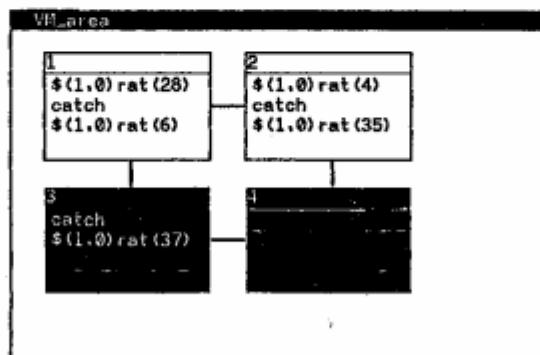


図8 動的ロード・バランス