ICOT Technical Memorandum: TM-0384

TM-0384

Knowledge Base Machine Based on Parallel Kernel Language (Oral Manuscript)

by
H. Itoh

August, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Knowledge Base Machine Based on Parallel Kernel Language

Hidenori ITOH

ICOT Research Center
Institute for New Generation Computer Technology
Mita Kokusai Bldg., 21*F*, 1-4-28, *Mita*, *Minato-Ku*, *Tokyo* 108, *Japan*

[S1. Title]

Thank you, Dr. Chairman. I very much appreciate this opportunity to report on our research and development of a parallel inference machine and knowledge base machine based on parallel logic programming. As I belong to the knowledge base machine research and development group, I'll explain the knowledge base machine in more detail than the parallel inference machine.

[S2. Section 1. Introduction and ICOT philosophy]

Logic programming plays an important role in computer science and is a bridge between knowledge information processing and parallel computer

architecture.

[S3. Fifth Generation Computer System (FGCS)]

A parallel kernel language corresponding to a conventional machine language was developed from a logic programming language. The parallel kernel language (PKL) is the nucleus of hardware and software systems for the FGCS project. An inference mechanism and knowledge base mechanism will be developed as hardware systems. These mechanisms will be integrated into the FGCS.

Basic and application software will be developed as software systems. A parallel operating system (POS) for the FGCS and a knowledge base management system (KBMS) for the KBM will also be developed.

[S4. Development Project and Its Progress]

Development is being pursued in three stages. The first was from 1982 to 1984. We are now in the middle of the second stage, which is to end in 1988.

In the initial stage, a sequential kernel language was developed with an object-oriented programming feature added to Prolog. The personal sequential inference machine, PSI, was developed as an inference mechanism based on the sequential kernel language. The relational data base experimental machine, DELTA, was developed as a relational data base storage

and retrieval mechanism. DELTA was connected with the PSI physically by a LAN and logically by relational algebraic type commands.

In the second stage, Guarded Horn Clauses (GHC) was developed as the PKL by Ueda. From this GHC, the parallel inference experimental mechanisms and their operating system are being developed. The knowledge base experimental mechanisms are also being developed.

In the final stage, these experimental mechanisms and software systems, based on the PKL, will be integrated into the prototype of the FGCS.

[S5. Section 2. Parallel Kernel Language/Guarded Horn Clauses — Syntax]

Let me explain two basic concepts before going into details.

(1) *Guarded Horn Clauses* for the PKL, and

(2) the new idea of *retrieval-by-unification* (RBU).

First, I'll explain the GHC. The definition of the GHC syntax is quite simple.

Each clause written in GHC is in this form.

Here, the connectives :- and , are common to ordinary Horn clauses. | is called commit operator. The part of a clause before | is called the guard, and the part after it is called the body. The calculation results of the guard are only effective in the guard. A guarded clause with no head is a goal clause, as in Prolog.

[S6. Guarded Horn Clauses — Semantics]

The semantics of GHC are also quite simple.

Imposition of guards and restriction of nondeterminism where the clause which has passed the guard first is used for subsequent computation. The execution of a GHC program proceeds by reducing a given goal clause to the empty clause under a few rules.

Here, these clauses have same head, p(X).

A p(X) is selected nondeterministically from these predicates, p(X)s. If the guard part of the p(X) clause is committed, then the body part of its clause is calculated in the AND-parallel way and the other p(X) clauses are erased. This nondeterministic is called the *don't – care nondeterministic* feature.

[S7. Processing Mechanism of GHC]

This is a simple example of the processing mechanism of GHC. Here, a goal ?- p(X), q(X) is given in the parallel goal pool and p(X) :- G | X=[a|Y],p(Y). q(X) :- X=[a|Y] | q(Y). are given in the candidate clause pool. The goal manager sends an instruction to the scheduler to process the goal. The scheduler selects the candidate clauses and suspends them in parallel. If their guards are committed, then their body parts are executed.

Here, p(X) and q(X) are processes. p(X) generates and sends the data stream. q(X) receives and consumes it. Therefore, p(X) is called a stream

generator and q(X) is called a stream consumer. The variable, X, is a channel of the stream. A process defined in GHC can be both a generator and a consumer.

[S8. Section 3. Retrieval By Unification]

Here, we premise that a term is the representation language of knowledge. A term is logical structured data that includes variables. Therefore, our knowledge base is called by another term, that is, term base. We have defined the *Cartesian Products* of terms as the knowledge base model. This knowledge base model is also called a relational term base model. Since this model is the basic model for the machine level, the higher model needs to be defined for the application program.

I have already introduced the new idea of retrieval by unification on this model (in reference No.13). RBU is the retrieval by equality check between a query and data enhanced to the retrieval by unifiability check between goals as the query and terms.

From this idea, *unification-restriction* and *unification-join* are defined. The following two examples explain these operations to retrieve the term relation.

[S9. Unification-restriction]

S is a set of terms. σ is a *unification-restriction* operation. O is

$\sigma_{f(a,X)} \diamond S$. That is to say, O is the set of terms in S that are unified by $f(a, X)$.

[S10. Unification-join]

The next example is *unification-join*. S_1 and S_2 are sets of terms. S_3 is a set of terms in which the second attribute of S_1 and the first attribute of S_2 are unified. Here, it is important to sort terms for processing efficiency. I'll explain how to sort terms later.

[S11. Section 4. Interface between GHC and RBU]

To connect RBU with GHC, we should consider the characteristics of GHC and RBU. GHC is an *AND-parallel language*, and committed choice is performed don't care nondeterministically. RBU is based on *OR-parallel* processing, and searches the whole set of objects. The set contains items which do not satisfy the retrieval condition. In other words, RBU must collect all the solutions. These characteristics are contradictory.

To solve this problem, GHC has been enhanced to have built-in predicates so that GHC can use RBU.

Let me explain it using a simple example.

Here, this "rbu" is a built-in predicate. "rbu" is also the generator of the set of the results, and the consumer of the command stream from "loop". "loop" is the generator of the command stream and is also the consumer

of the result stream from "rbu".

[S12. Example of an Interpreter Written in GHC]

This is an example of an interpreter written in GHC. This "rbu" is the built-in predicate. The first "loop" is for termination of this program. The second "loop" is for collecting results and stacking them into R. The last "loop" is for iterative calculation of RBU commands. C_1 is the incomplete command message including the variable 'S'. C_2 is the subsequent command stream. S is a channel to ensure that the retrieved results flow in a stream. "solve" lives while "loop" and "rbu" retrieve and collect the solutions and is erased after all solutions are finished to collect by "loop" and "rbu".

[S13. Example of a Term Relation]

This is a term relation of the parent and ancestor. Here, "an(A,B)" indicates that B is an ancestor of A, and "pa(A,B)" indicates that B is a parent of A.

Using the RBU commands, a set of a's ancestors is retrieved from this relation.

[S14. Execution Example of RBU Commands]

Let me explain the descendant retrieval algorithm by "rbu" in detail.

(1) First, 'Result' is empty.

(2) KB is a given term relation about the parent and ancestor. K_0 is introduced from the operation, $\sigma_{goalohead}\ KB$.

(3) K_{i+1} is also introduced from the operation, $K_i\ _{unification-join}\ KB$. Attributes 1, 2 and 3 are projected into the head and body.

(4) R is the collection of heads of K_i, which has a null body.

(5) If K_i is empty, then this program stops.

[S15. Execution Example of RBU Commands in GHC]

Here, 1, 2 and 3 are the attribute numbers for the projection. Each S_i is calculated by each unification-restriction called from the "solve" process. Since S_2 and S_5 contain empty bodies in the resolvents, "an(a,b)" and "an(a,c)" are the answers to the query. "an(a,b)" and "an(a,c)" flow into 'Result' in the "solve" process. Since S_8 and S_9 are the only empty clauses, then iteration is terminated at S_8 and S_9.

For simplicity, only unification-restriction is used in this example. Because unification-join and unification-restriction are dependent on each other, unification-join is defined by iteration of unification-restriction and ordinary-join.

[S16. Section 5. Parallel Control Methods of Retrieval Processes in PKL]

(1) Stream Division

When the length of the stream is very large and there are many retrieval processes currently available, the stream should be divided among these processes for processing efficiency.

These processes are controlled in parallel, applying the features of GHC. Evaluating the relationship between the stream length, number of retrieval processes and command types, the best retrieval process assignment method in GHC has been introduced.

(2) Meta level Control

The meta-level controller controls three data-division methods according to the current load of processors. To do this, it controls by meta-knowledge showing which method is currently the best. The meta-level controller program is also written in GHC.

[S17. Parallel Control Methods of Retrieval Processes in GHC]

This is an example of the part of the parallel control program with the status management for REs. Since this program always inspect the RE's status, it knows which REs are free. When this program receives a retrieval command, it divides the object to be retrieved by its command into the number of REs that are currently free. Each object and the command

are assigned to each RE that is free.

[S18. Meta-level Control Program in GHC]

This is an example of the meta-level control program. The RE assignment is determined by the command type, the number of the REs that are free, the size of the object to be manipulated, and the load of RE assignment methods. This program can be replaced by the clause 'REscheduler' which was shown in the last program. Therefore, the new program which combines the last program and this meta-level control program can determine the best assignment method.

[S19. Section 6. KBM Model/Parallel Inference Mechanism]

Next, let me explain the FGCS model. The FGCS is composed of the parallel inference machine and knowledge base mechanism. The parallel inference machine (PIM) is composed of a mesh-type network and a number of clusters (CLs) of inference processing elements (PEs). Here, n is about 100 and each CL has about 10 PEs. Each CL has a local shared memory (LSM) for its PEs.

[S20. Knowledge Base Mechanism]

The knowledge base mechanism (KBM) is composed of a mesh-type

network, a number of retrieval elements (REs), and a global shared memory unit (GSM). The RE receives the retrieval commands from CLs through the network. In the network, the destination REs of retrieval commands should be dynamically determined. For this purpose, the RE parallel control method in GHC needs to be integrated into the network. Here, m is about one tenth of the number of CLs.

The GSM is composed of a number of memory banks. Each bank has ports for reading and writing knowledge stored in logical page size. A logical page lies across all the banks. Each port connected to the banks can access the same page. The GSM mechanism permits REs to be accessed in the same logical domain at the same time.

The LSM and GSM are the components of the hierarchical shared memory mechanism of the FGCS.

[S21. RE]

The RE is the dedicated hardware element for processing the built-in predicate "rbu". Since "rbu" is composed of simple and iterative operations, the dedicated hardware RE operates efficiently when there are many objects to be searched, because it reduces the PE load.

The RE is composed of two sorters (ST), a pair generator (PG) and a unifier (UR).

When the RE receives a retrieve command, it starts to access the GSM.

The sets of retrieved terms stream into the RE. While the pairs of sets of terms flow into the RE, the ST sorts the terms in the generality defined in unifiability for the efficiency of the "rbu" operation. The PG detects and erases any pairs that are not unifiable. The unifiable sets of pairs are unified in a pipelined way in the UR and are sent to CLs as the response of retrieval commands. The stream flowing into the UR has an affinity with the definition of the data stream in GHC.

[S22. POS and KBMS]

The parallel operating system (POS) has a layered structure. The lowest layer is called the *PKL base* and is an abstract machine instruction set for the *PKL core* and built-in predicates. The middle layer has two components. One is called the *PKL core*. It contains *flat*-GHC and meta-call predicates. Another is called the *PKL pragma*. It controls allocation of CLs and REs and their priority scheduling. The highest layer is called the *PKL user* and is open to the user. Because the *PKL user* has modular programming features, KBMS is written in it.

[S23. Ideal Model of FGCS]

The ideal hardware model of the FGCS is composed of a PIM and a KBM. Physically, the upper and lower networks are the same. They are controlled by a POS. The KBMS on the POS manages the knowledge base.

Here, the REs are controlled in parallel as the KBM class defined by the PKL pragma in the POS.

[S24. Section 7. Conclusion]

In conclusion, our knowledge base machine model is based on a parallel kernel language.

GHC has been introduced as the PKL and a term relation and its operations, named RBU, have been introduced as the knowledge base model. We have combined RBU and GHC with built-in predicates so that all solution collecting features required in RBU are realized in GHC, which has only a don't care nondeterministic feature.

Parallel control in GHC is effectively used for the retrieval elements, and the data stream manipulation method that has an affinity with the definition of the data stream in GHC is introduced into the retrieval element.

The parallel inference mechanism and knowledge base mechanism will be integrated into a prototype of the FGCS.

The basic functions introduced in this FGCS have also been evaluated by developing some applications on it.

Thank you.

♠ ♠ ♠

◇ ◇ ◇

Knowledge Base Machine

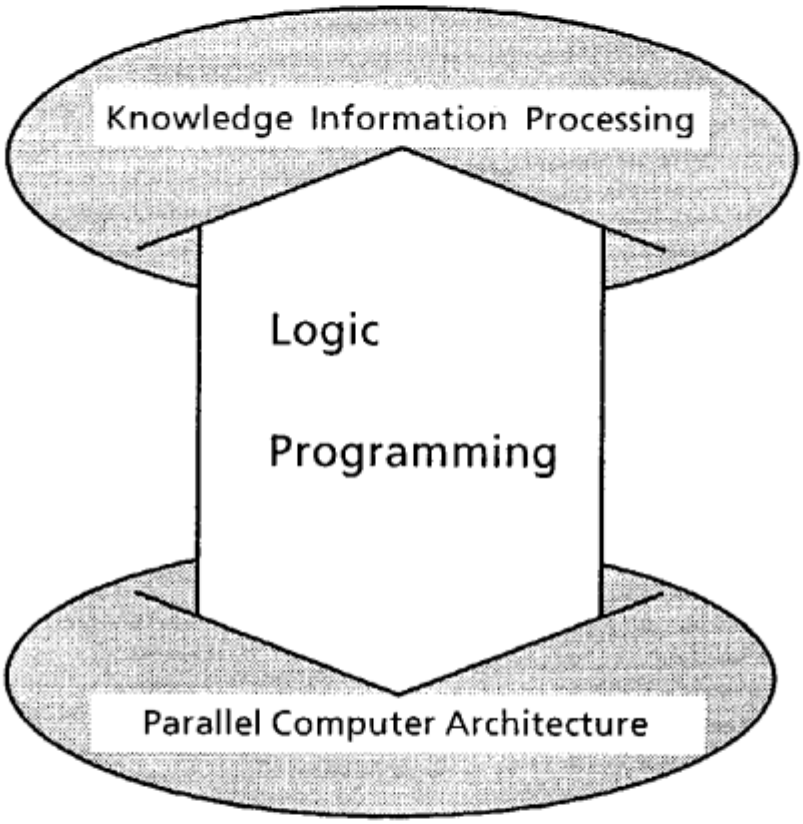
Based on Parallel Kernel Language

H . ITOH [†]
T . TAKEWAKI [††]
H . YOKOTA [†††]

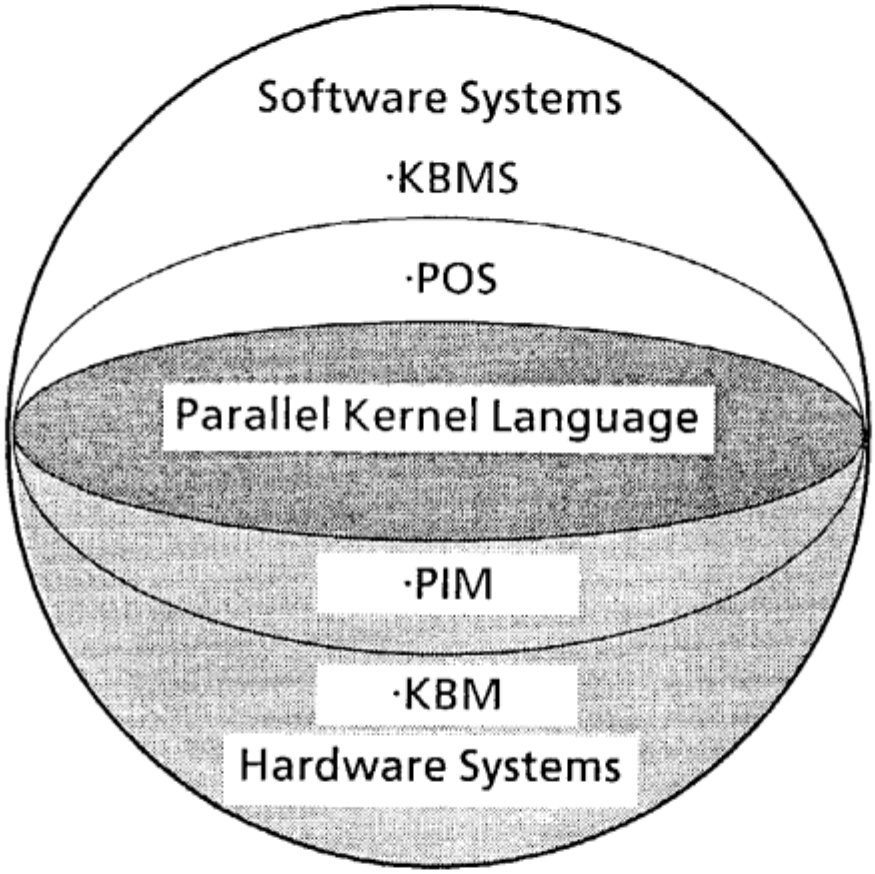
[†] ICOT
[††] Toshiba
[†††] Fujitsu

Oct. 1987

1. INTRODUCTION

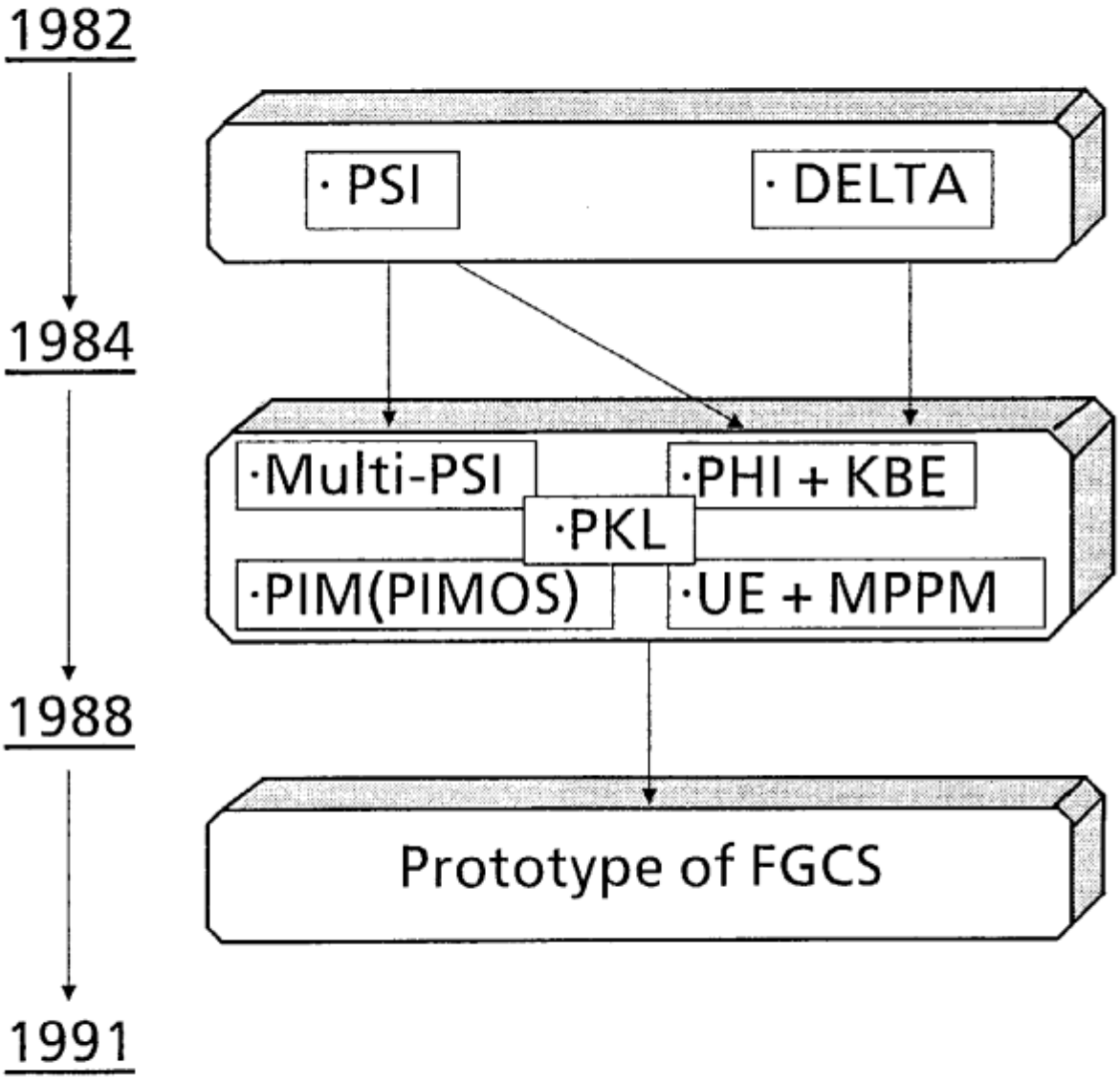


Fifth Generation Computer System



PIM : Parallel Inference Mechanism
POS : Parallel Operating System
KBM : Knowledge Base Mechanism
KBMS : Knowledge Base Management System

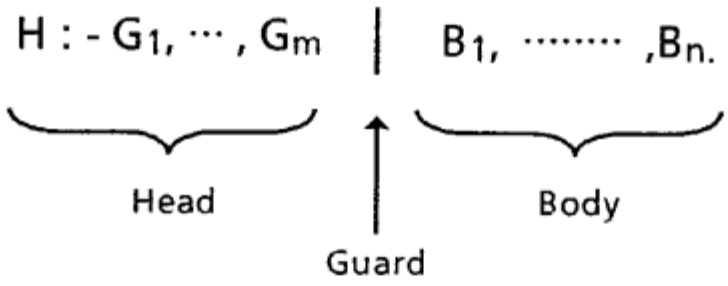
Development Stages



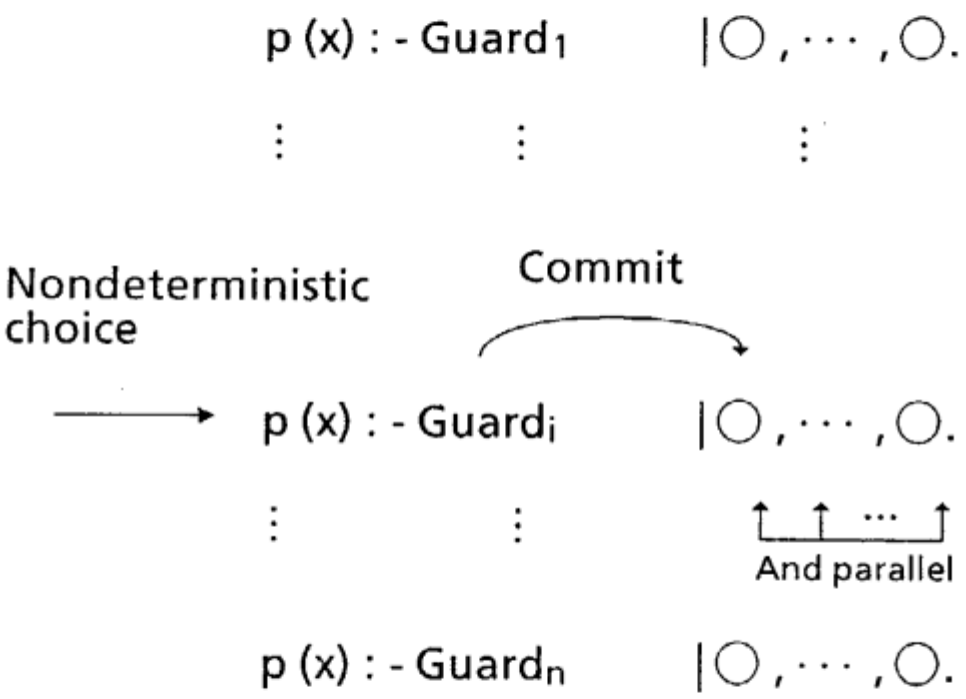
2. PARALLEL KERNEL LANGUAGE (PKL)

GHC : Guarded Horn Clauses [Ueda 85]

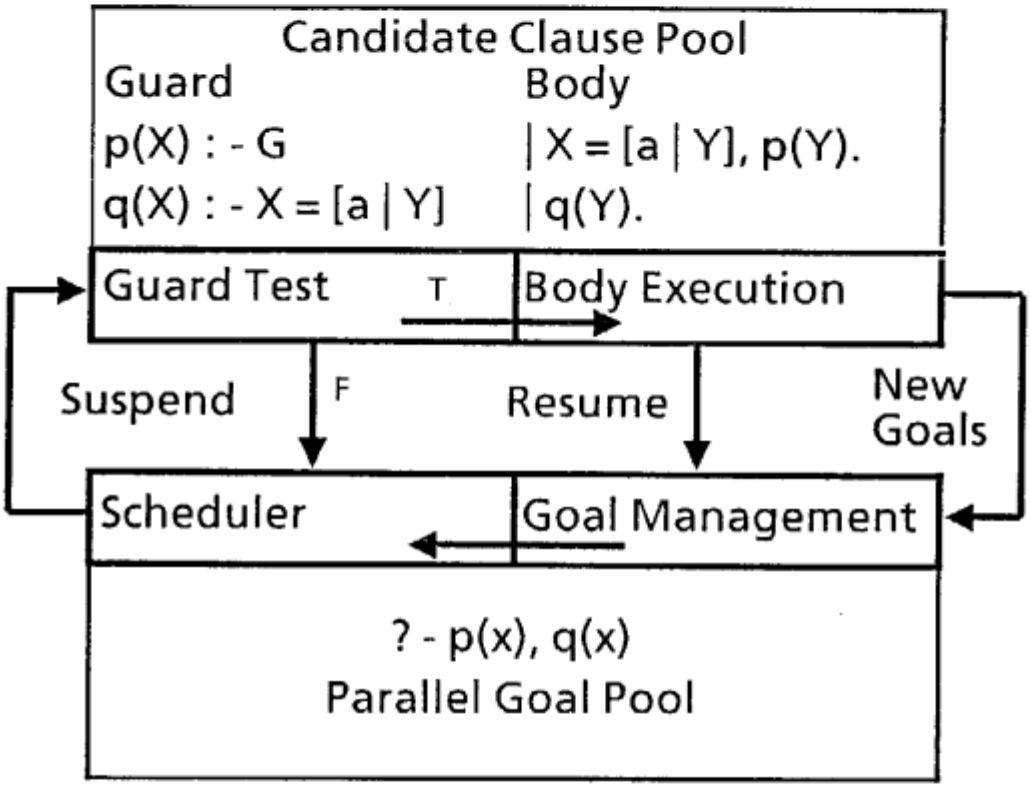
Syntax

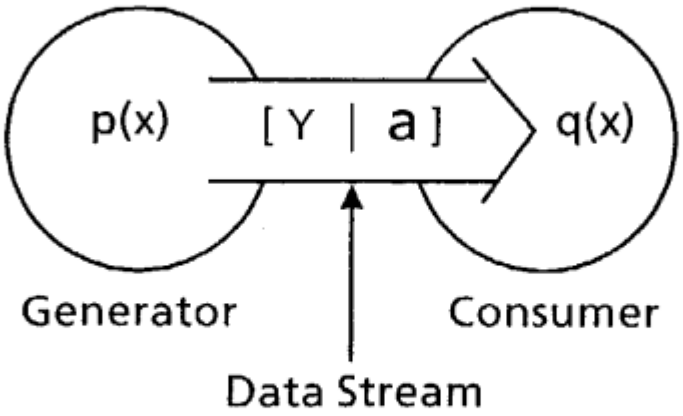


GHC : Semantics



Processing Mechanism of GHC





3. KB : Knowledge Base

Term ← Data

Cartesian Products of Terms

(term, term, term, ·····, term)

term relation

RBU : Retrieval-by-Unification

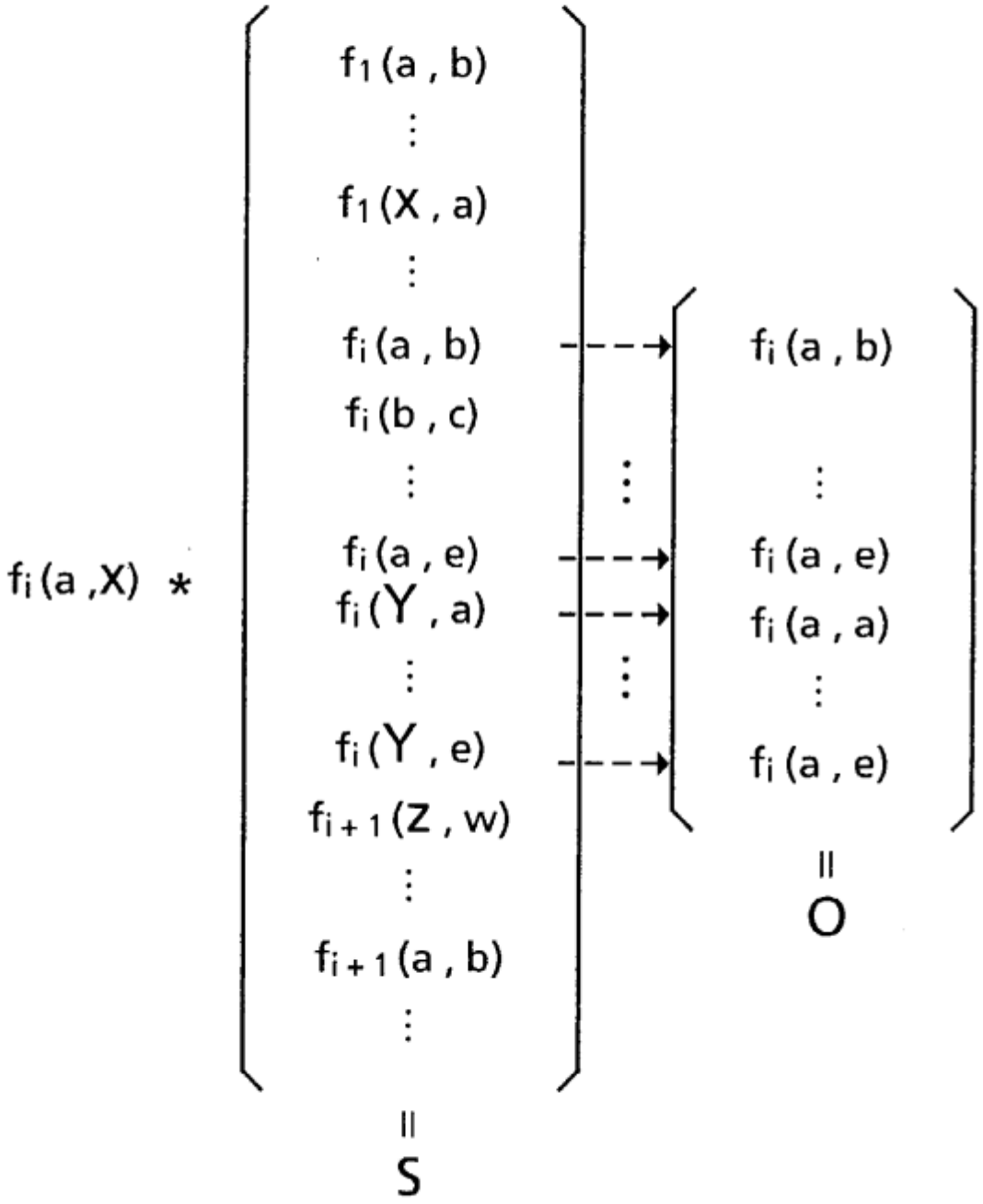
Unifiability check ← Equality check

Unification-restriction

Unification-join

RBU : Unification - Restriction *

Example $f_i(a,x) * S = O$



RBU : Unification - Join : ⋈

Example $S_1 \bowtie S_2 = S_3$

$S_1 = (\ t_1\ ,\ t_2\ )$

①	f(a, X)
②	f(b, b)
③	f(a, Y)

$\bowtie_{t_2 \cup t_1'}$

$S_2 = (\ t_1'\ ,\ t_2'\ )$

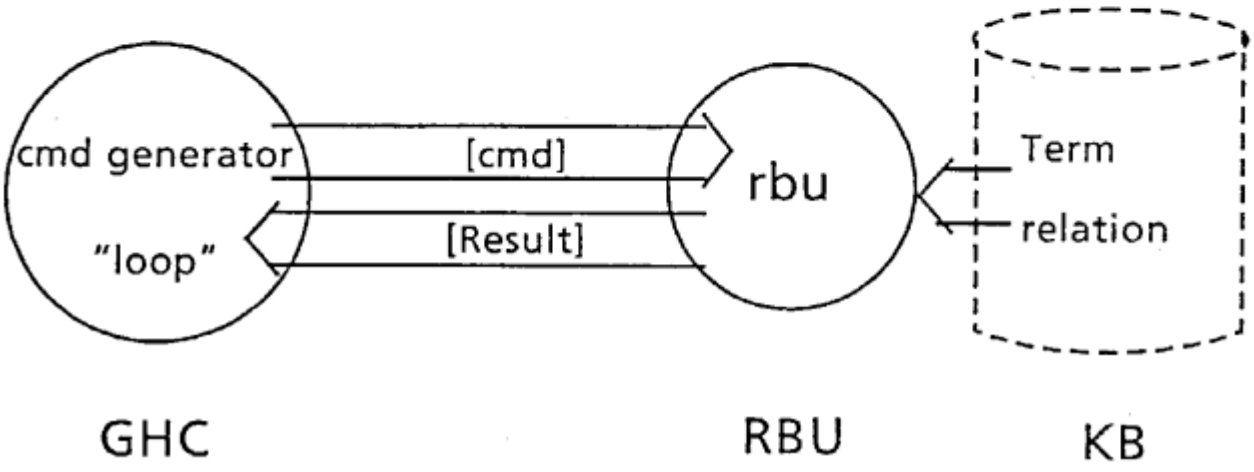
f(a, a)	[1]
f(Z, b)	[2]
f(w, c)	[3]

$S_3 = (\ t_1''\ ,\ t_2''\ ,\ t_3''\ )$

$=$

①	f(a, a)	[1]
③	f(a, a)	[1]
①	f(a, b)	[2]
②	f(b, b)	[2]
③	f(a, b)	[2]
①	f(a, c)	[3]
③	f(a, c)	[3]

4. Interface between GHC and RBU



First order logic ⟷ First order logic

Don't care nondeterministic ⟷ Built-in predicate ⟷ Collection of all solutions

Data stream ⟷ Data stream

Horn Logic Interpreter Written in GHC

```
solve(KB,Goal,Result) :- true |
    loop(KB,cmd(C1,C2),[[[Goal],[Goal]]],Result),
    rbu(cmd(C1,C2)).

loop(KB,cmd(C1,C2) ,[ ]     ,X) :- true  | X = [ ].
loop(KB,CMD     ,[[G,R]|L],X) :- R = [ ] | X = [G|Y],
    loop(KB,CMD,L,Y).
loop(KB,cmd(C1,C3),[[G,R]|L],X) :- R \= [ ] |
  C1 = [unification_restriction(KB,[1 = G,2 = R],[1,3],S)|C2],
    merge(L,S,N),
    loop(KB,cmd(C2,C3),N,X).
```

Example of a Term Relation

kb1

G	[an(A,B)|S]	[pa(A,B)|S]
G	[an(A,B)|S]	[pa(A,C),an(C,B)|S]
G	[pa(a,b)|S]	S
G	[pa(b,c)|S]	S

Resolution Using RBU

$R \leftarrow \varnothing$

$K_0 \leftarrow \sigma_{head} \blacklozenge goal(KB)$

$i = 0$

while $K_i \neq \varnothing$ do

begin

$R \leftarrow \sigma_{body = [\,]}(K_i) \cup R$

$K_{i+1} \leftarrow \pi_{Ki.head,KB.body}(K_{i\ body} \bowtie \blacklozenge_{head} KB)$

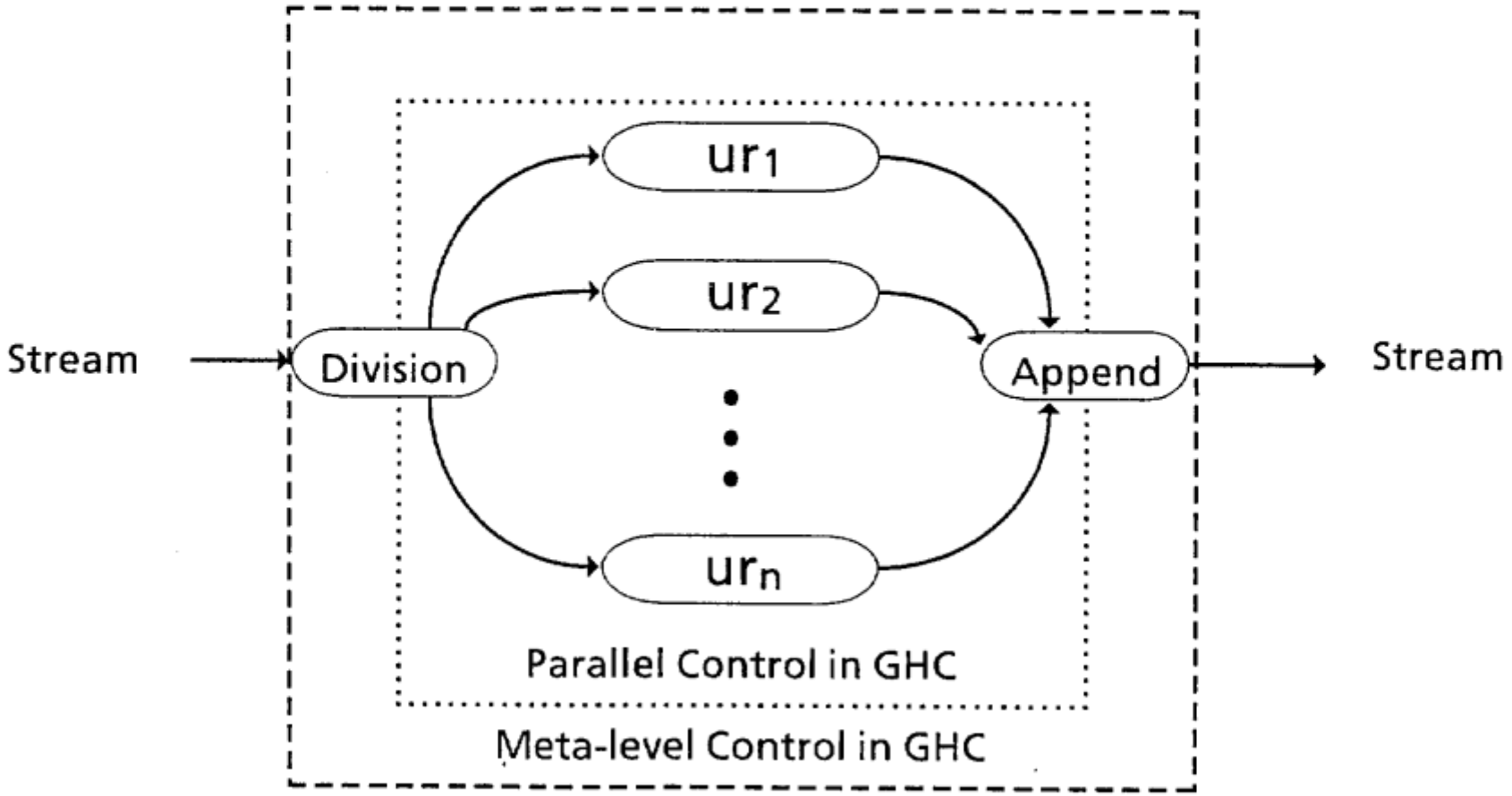
$i \leftarrow i + 1$

end

Execution Examples of RBU Commands

```
unification_restriction(kb1,[1 = [an(a,X)],2 = [an(a,X)]],[1,3],S1)
 S1 = [[[an(a,X)],[pa(a,X)]],[[an(a,X)],[pa(a,C),an(C,X)]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [pa(a,X)]],[1,3],S2)
 S2 = [[[an(a,b)],[]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [pa(a,C),an(C,X)]],[1,3],S3)
 S3 = [[[an(a,X)],[an(b,X)]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [an(b,X)]],[1,3],S4)
 S4 = [[[an(a,X)],[pa(b,X)]],[[an(a,X)],[pa(b,C),an(C,X)]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [pa(b,X)]],[1,3],S5)
 S5 = [[[an(a,c)],[]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [pa(b,C),an(C,X)]],[1,3],S6)
 S6 = [[[an(a,X)],[an(c,X)]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [an(c,X)]],[1,3],S7)
 S7 = [[[an(a,X)],[pa(c,X)]],[[an(a,X)],[pa(c,C),an(C,X)]]]
unification_restriction(kb1,[1 = [an(a,X)],2 = [pa(c,X)]],[1,3],S8)
 S8 = []
unification_restriction(kb1,[1 = [an(a,X)],2 = [pa(c,C),an(C,X)]],[1,3],S9)
 S9 = []
```

Data Division Process of Unification Restriction Operation



Parallel Controller with Status Management

```
% Top level
'REscheduler'([C|T],Stream) :- true | availableREs(Stream, NewStream,Free),
                        divide(Free,[C|T],NewStream).
'REscheduler'([ ]  ,Stream) :- true | closeStream(Stream).
availableREs(St, NS,Free) :- true | inspect(St,NS,Ins), checking__free(Ins,Free).
% Inspection of REs status
inspect([ ]            ,New,Ins) :- true | New = [ ], Ins = [ ].
inspect([stream(N,St)|Rest],New,Ins) :- true | New = [stream(N,SR)|NR],
    Ins = [(N,State)|IR], St = [ins(State)|SR], inspect(Rest,NR,IR).
divide([ ] ,C   ,St) :- true    | 'REscheduler'(C,St).     % All REs are busy.
divide(REs,[C|T],St) :- List\ = [ ] | 'REscheduler'(T,NS),
   data__division(C, REs, SubC), send__out(REs,SubC,St,NS).  % send out C to free REs
% 'RE' manages status of REs, SendToRE sends C to Nth RE.
'RE'(N,[term |Command]) :- true | Command = [ ].          % termination
'RE'(N,[ins(C)|Command]) :- true | C = free, 'RE'(N,Command). % inspection of status
'RE'(N,[cmd(C)|Command]) :- true | sendToRE(N,C,Response),  % retrieval command
   response(Response,Command,Next), 'RE'(N,Next).
response(end,Command    ,N) :- true | Command = N.        % Process ends.
response(R  ,[ins(C)|Cmd],N) :- true | C = busy,response(R,Cmd,N).% inspection of status.
```

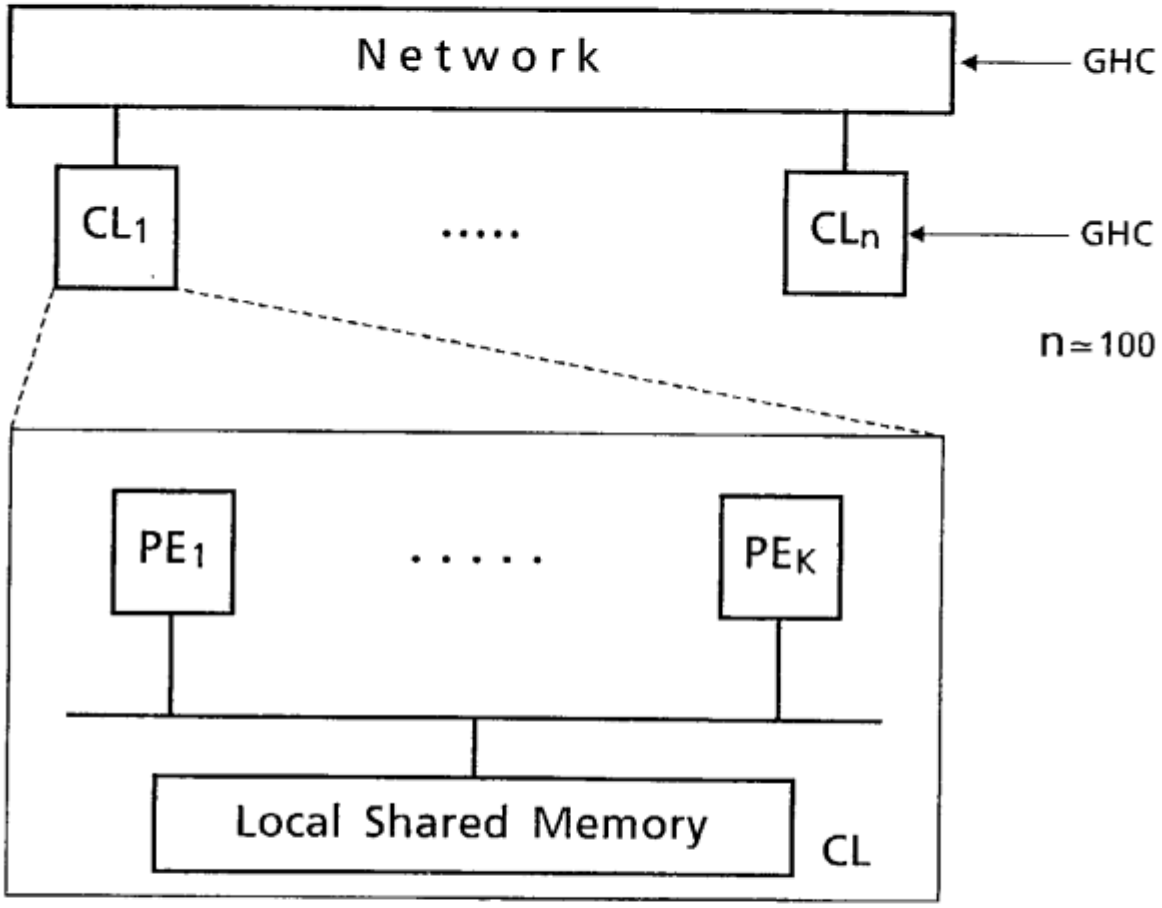
Part of Program for Meta-level Control

```
meta__control([ ]              , ST, REs) :- true | colseStream(ST).
meta__control([cmd(C,Type,Size)|Rest], ST, REs) :- true |
        availableREs(ST, IntST, Free),
        strategy(C,Type,Size,Free, Method),
        solve(Method, REs, Free, cmd(C), IntST, NewST),
        meta__control(REs, Rest,NewST).
```



```
solve(method1, REs, Free, Cmd, ST, NewST) :- true |
        selectRE(Free, RE), send__out([Cmd],[RE], ST, NewST).
solve(method2, REs, Free, Cmd, ST, NewST) :- true |
        data__division(Cmd, Free, SubCmd), send__out(SubCmd,Free, ST, NewST).
solve(method3, REs, Free, Cmd, ST, NewST) :- true |
        data__division(Cmd, REs, SubCmd), send__out(SubCmd, Free, ST, NewST).
```

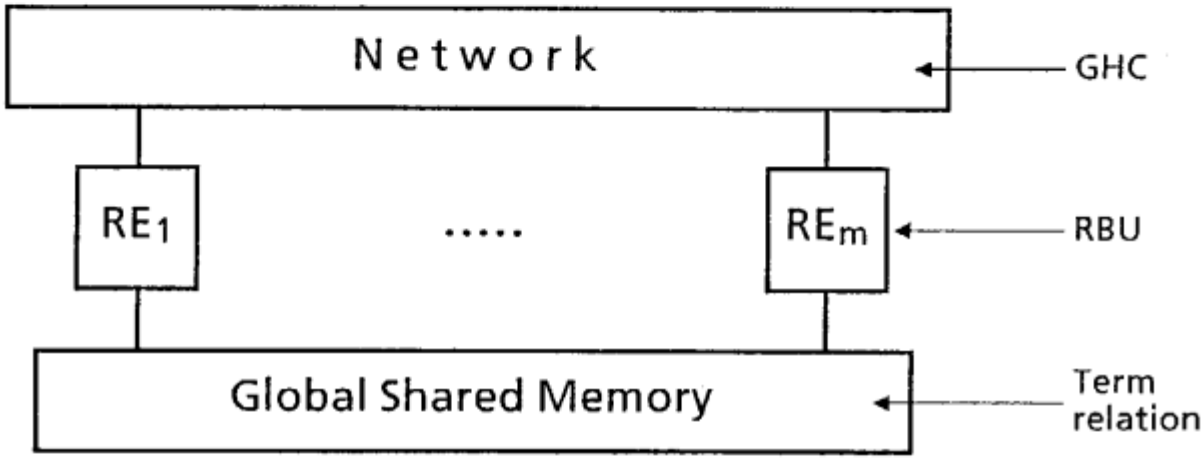
5. Parallel Inference Mechanism (PIM)



Ideal Model of Cluster

CL : Cluster
PE : Processing Element
k≃10, k × n≃1000

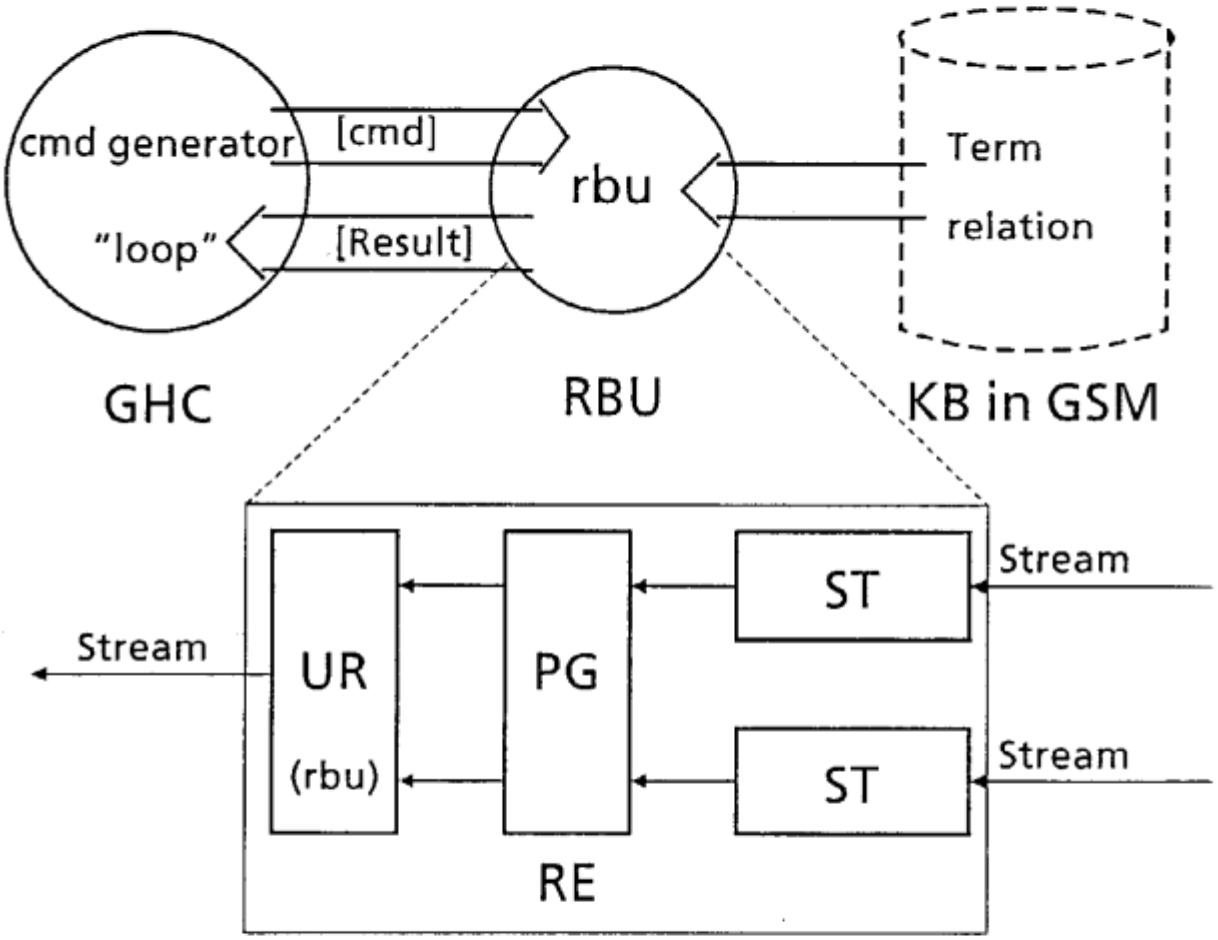
Knowledge Base Mechanism (KBM)



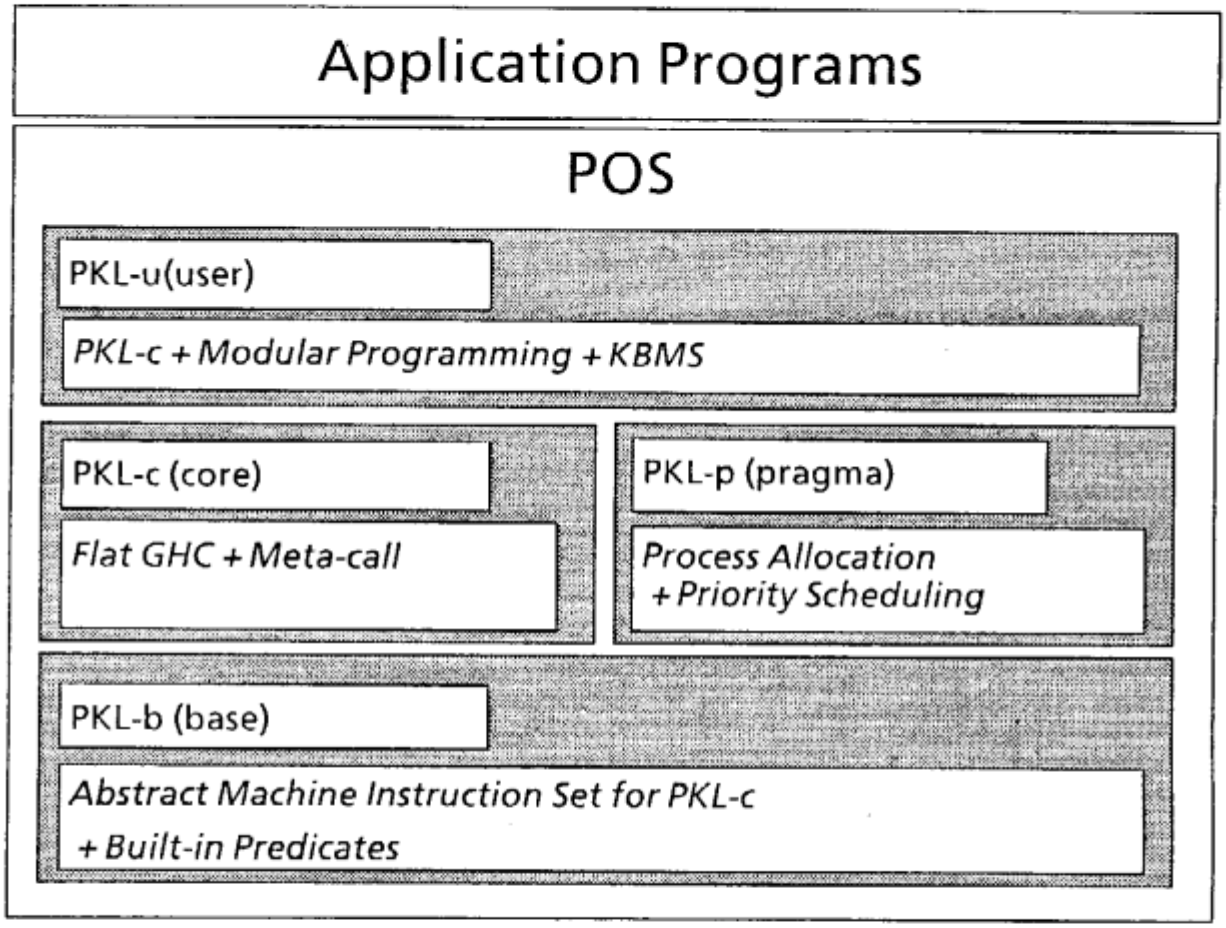
m≃n / 10

RE : Retrieval Element

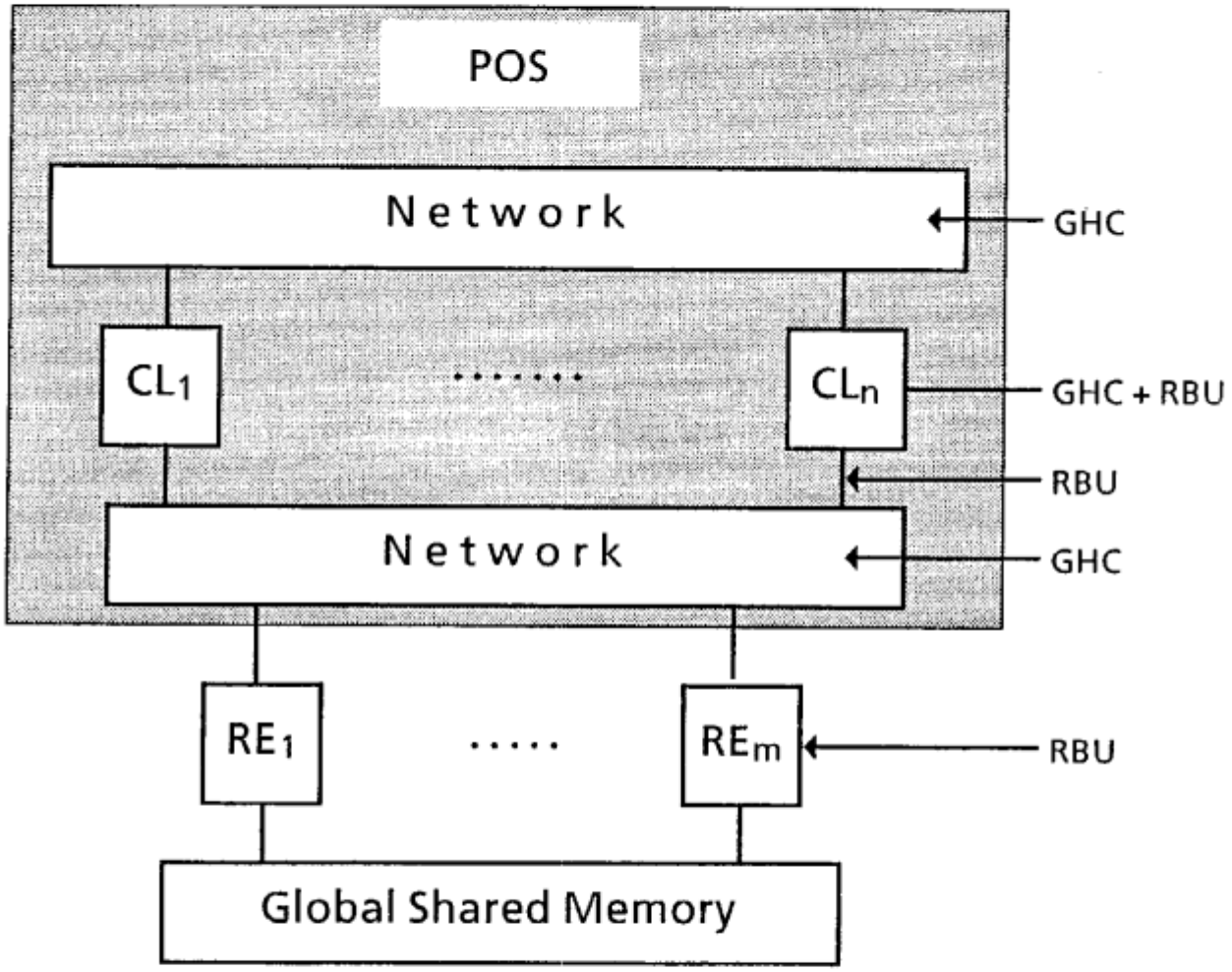
Dedicated Hardware Unit for rbu
RE: Retrieval Engine



ST : Sorter
PG: Pair Generator
UR : Unifier


Application Programs
POS
PKL-u(user)
PKL-c + Modular Programming + KBMS
PKL-c (core)
Flat GHC + Meta-call
PKL-p (pragma)
Process Allocation
+ Priority Scheduling
PKL-b (base)
Abstract Machine Instruction Set for PKL-c
+ Built-in Predicates

Prototype of FGCS



$m \simeq n / 10, n \simeq 100$

7. CONCLUSION

- First order logic ⟷ First order logic
- Collection of all solutions ⟷ Built-in predicate ⟷ Don't care nondeterministic
- Data stream ⟷ Data stream
- Parallel ⟷ Parallel