

TM-0826

Guide to the Japanese Demonstrations
at France-Japan Artificial Intelligence
and Computer Science Symposium '89

by
Y. Yoshioka

November, 1989

©1989, ICOT

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

Guide to the Japanese Demonstrations

France-Japan Artificial Intelligence
and
Computer Science Symposium 89

November 15-17, 1989

Laforet Shuzenji Club, Izu

Institute for New Generation
Computer Technology

4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan
phone: +81-3-456-2511 fax: +81-3-456-1618

Program

Wednesday, 15 November

14:30–16:30 Demonstrations

◦ Demo Session A (on PSI-II)

14:30–15:00 CAL

15:00–15:30 CAP-LA

15:30–16:00 VISTA

16:00–16:30 Kappa

◦ Demo Session B (on PSI-II)

14:30–15:00 APRICOT/0

15:00–15:30 MECHANICOT

15:30–16:00 LTB

Thursday, 16 November

13:30–15:30 Demonstrations

◦ Demo Session A (on PSI-II)

13:30–14:00 CAL

14:00–14:30 CAP-LA

14:30–15:00 VISTA

15:00–15:30 Kappa

◦ Demo Session B (on PSI-II)

13:30–14:00 APRICOT/0

14:00–14:30 MECHANICOT

14:30–15:00 LTB

A List of Demonstrations

- *Demo Session A (on PSI-II)*

1. *Constraint Logic Programming Experimental System* **CAL**

2. *Computer Aided Proof* **CAP-LA**

3. *Visualization and Transformation Apprentice*

for Concurrent Logic Programming **VISTA**

4. *Knowledge Application Oriented Advanced DBMS* **Kappa**

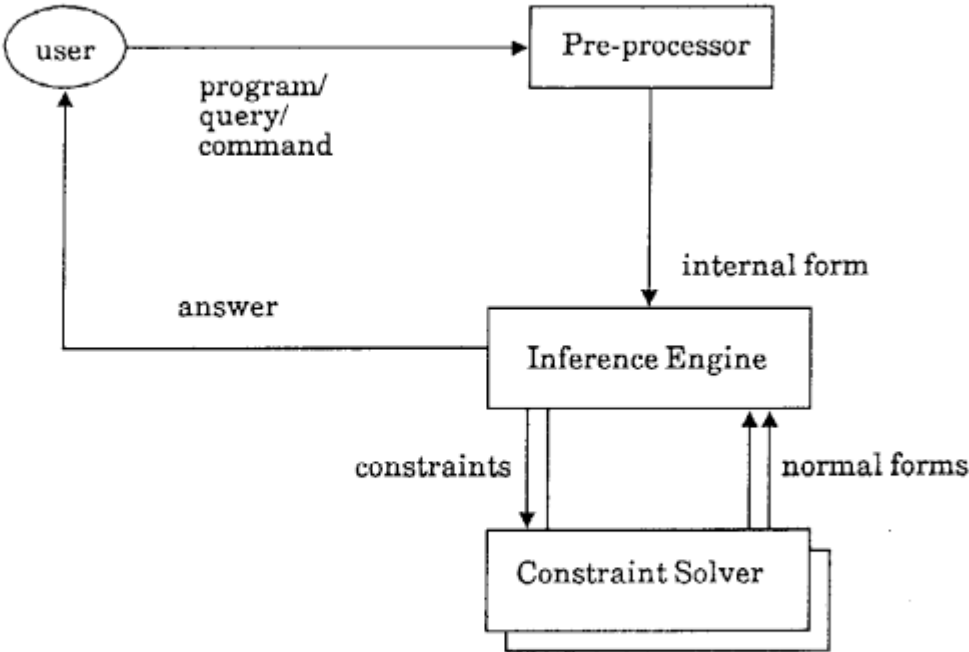
- *Demo Session B (on PSI-II)*

5. *Assumption-Based Problem Solver* **APRICOT/0**

6. *Constraint-Based Knowledge Compiler*

-Design Expert System Building Tool- **MECHANICOT**

7. *A Software Library for Japanese Language Processing* **LTB**

Title	Constraint Logic Programming Experimental System CAL
Purpose	1) Programs easier to write and read 2) Highly abstract programming 3) Research on efficient problem solving techniques
Outline & Features	1) Amalgamation of logic programming and constraint solving 2) Multiple Constraint Solvers 3) Solution of non-linear algebraic equations and Boolean equations 4) Natural extension of Prolog
System Configuration	 <pre> graph TD User((user)) -- "program/ query/ command" --> PreProcessor[Pre-processor] PreProcessor -- "internal form" --> InferenceEngine[Inference Engine] InferenceEngine -- "constraints" --> ConstraintSolver[Constraint Solver] ConstraintSolver -- "normal forms" --> InferenceEngine InferenceEngine -- "answer" --> User </pre> The diagram illustrates the system configuration. A user (represented by an oval) sends a "program/query/command" to a "Pre-processor" (rectangle). The "Pre-processor" sends an "internal form" to the "Inference Engine" (rectangle). The "Inference Engine" sends "constraints" to the "Constraint Solver" (rectangle). The "Constraint Solver" sends "normal forms" back to the "Inference Engine". Finally, the "Inference Engine" sends an "answer" back to the user.

★ Examples of CAL : Heron's Formula(non-linear)

Program & Query&Answer

```

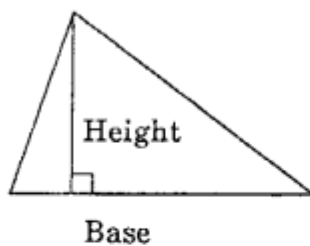
:- public triangle/4, triangle1/4.
surface(Height, Base, Area) :- Base * Height = 2*Area.
pythagoras(A, B, Hypotenuse) :- A^2 + B^2 = Hypotenuse^2.
triangle(A, B, C, S) :-
    C = CA + CB:alg,
    pythagoras(CA, H, A),
    pythagoras(CB, H, B),
    surface(H, C, S).
triangle1(A, B, C, S) :- precedence(S :> 0), triangle(A, B, C, S).

?- heron:triangle1(a, b, c, s).

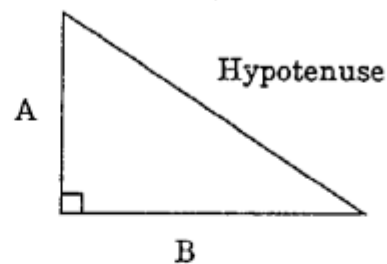
```

$$s^2 = -1/16*b^4 + 1/8*a^2*b^2 - 1/16*a^4 + 1/8*c^2*b^2 + 1/8*c^2*a^2 - 1/16*c^4$$

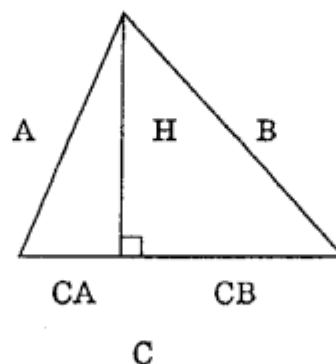
surface:



pythagoras:



triangle:



★ Examples of CAL : Cone Volume using Lagrange(dynamic constraints)

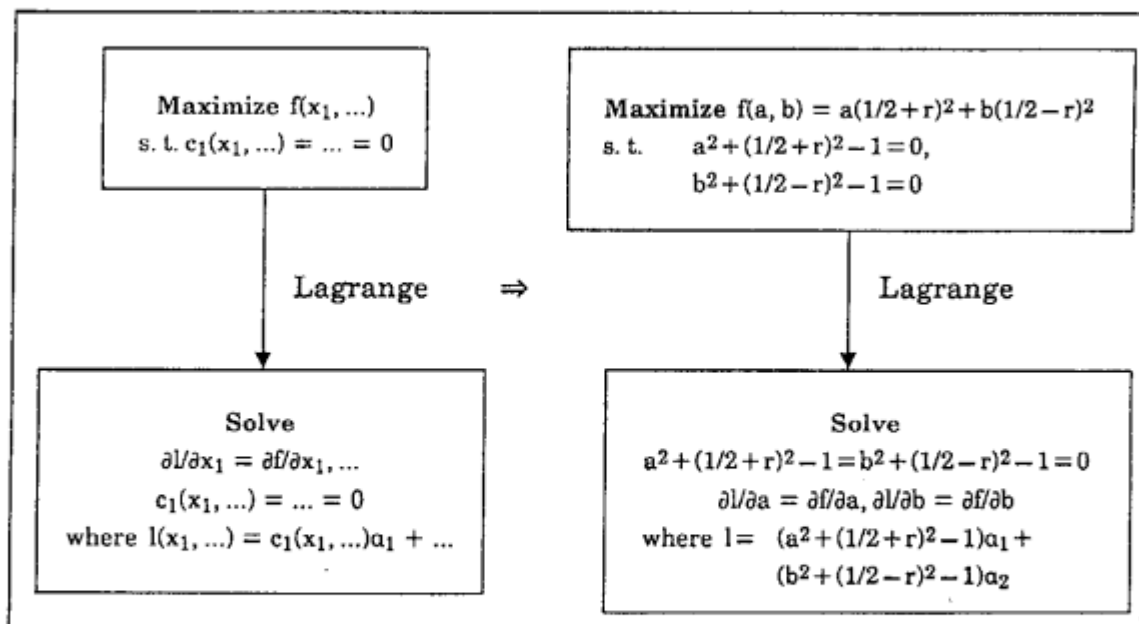
Program & Query&Answer

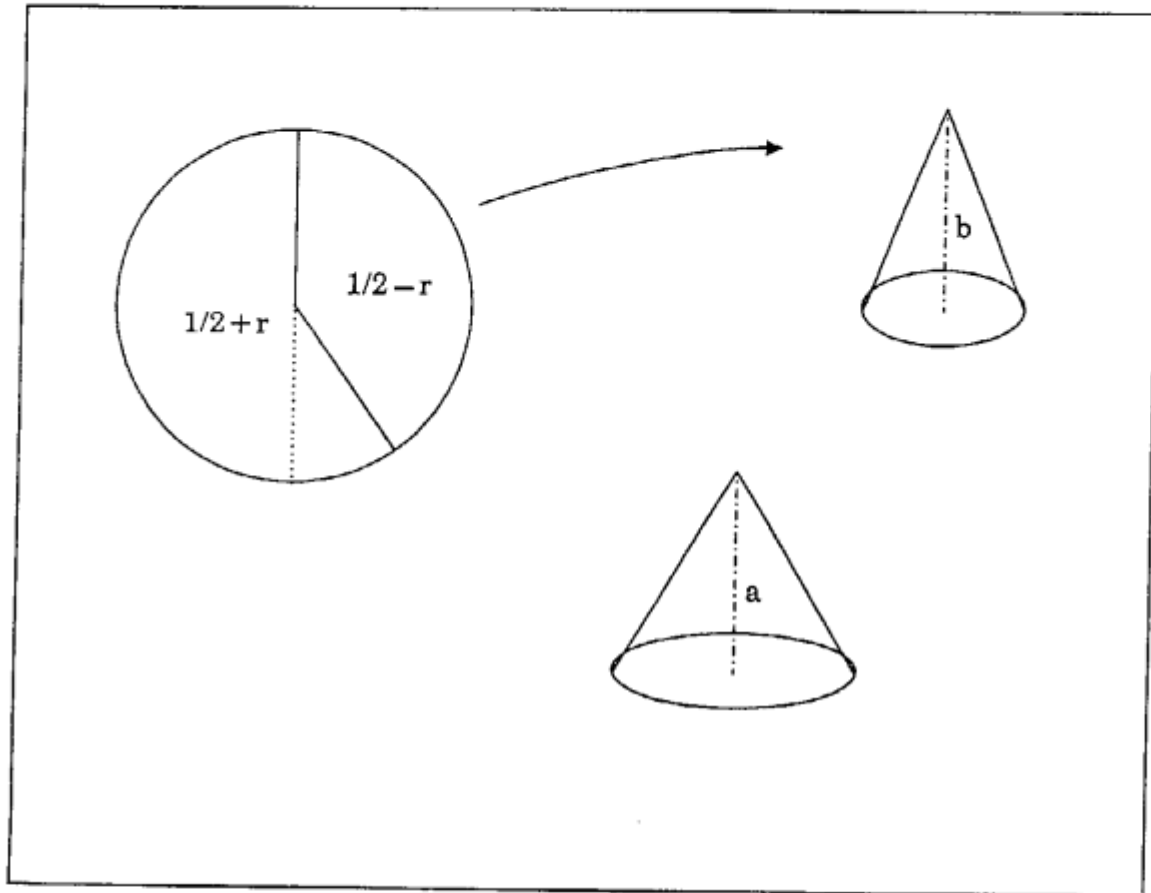
```

:- public lagrange/3.
lagrange(F, Constraints, Vars):-
    construct_l(Constraints, L),
    partials(Vars, F, L).
partials([], __, __):-!.
partials([Var|Vars], F, L):-
    dif(F, Var) = dif(L, Var):alg,
    partials(Vars, F, L).
construct_l([], 0):-!.
construct_l([C|Cs], C*Alpha + L):-
    C = 0:alg,
    construct_l(Cs, L), !.

?- lagrange((1/2+r)^2*a+(1/2-r)^2*b,
    [a^2+(1/2+r)^2 = 1, b^2+(1/2-r)^2 = 1],
    [a, b, r]).

r^7=(29/12)*r^5+(-17/48)*r^3+(5/576)*r
    
```





Query

?-lagrange($(1/2 + r)^2 \cdot a + (1/2 - r)^2 \cdot b$,
 $[a^2 + (1/2 + r)^2 = 1, b^2 + (1/2 - r)^2 = 1]$,
 $[a, b, r]$).

★ Examples of Boolean CAL : Counter Circuit

Program&Query&Answer

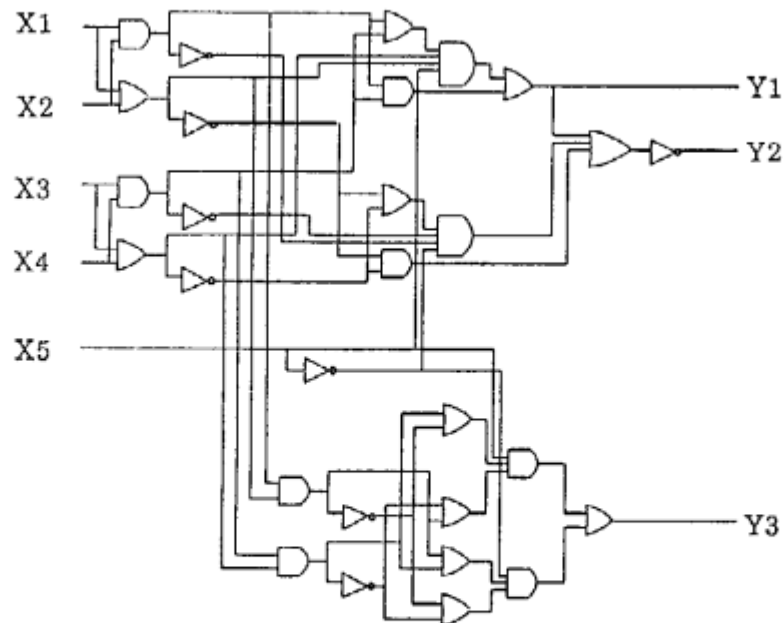
:- public circuit/8.

circuit(X1, X2, X3, X4, X5, Y1, Y2, Y3) :-

I1 = X1&X2:bool,	I2 = X1 ∨ X2:bool,	I3 = X3&X4:bool,
I4 = X3 ∨ X4:bool,	I5 = I1:bool,	I6 = I2:bool,
I7 = I3:bool,	I8 = I4:bool,	I9 = I1 ∨ I3:bool,
I10 = I1&I3:bool,	I11 = I6 ∨ I8:bool,	I12 = I6&I8:bool,
I13 = X5:bool,	I14 = I5&I2:bool,	I15 = I7&I4:bool,
I16 = I14:bool,	I17 = I15:bool,	I18 = I15 ∨ I16:bool,
I19 = I14 ∨ I17:bool,	I20 = I14 ∨ I15:bool,	I21 = I16 ∨ I17:bool,
I22 = I9&I4&I2&X5:bool,	I23 = I11&I7&I5&I13:bool,	
I24 = X5&I18&I19:bool,	I25 = I13&I20&I21:bool,	
I26 = I22 ∨ I10:bool,	I27 = I26 ∨ I23 ∨ I2:bool,	
Y1 = I26:bool,	Y2 = I27:bool.	Y3 = I24 ∨ I25:bool.

?- count:circuit(x1,x2,x3,x4,x5,1,0,1).

?- count:circuit(1,x2,x3,x4,x5,y1,y2,y3), x2 = x3 : bool.x5 = 1.



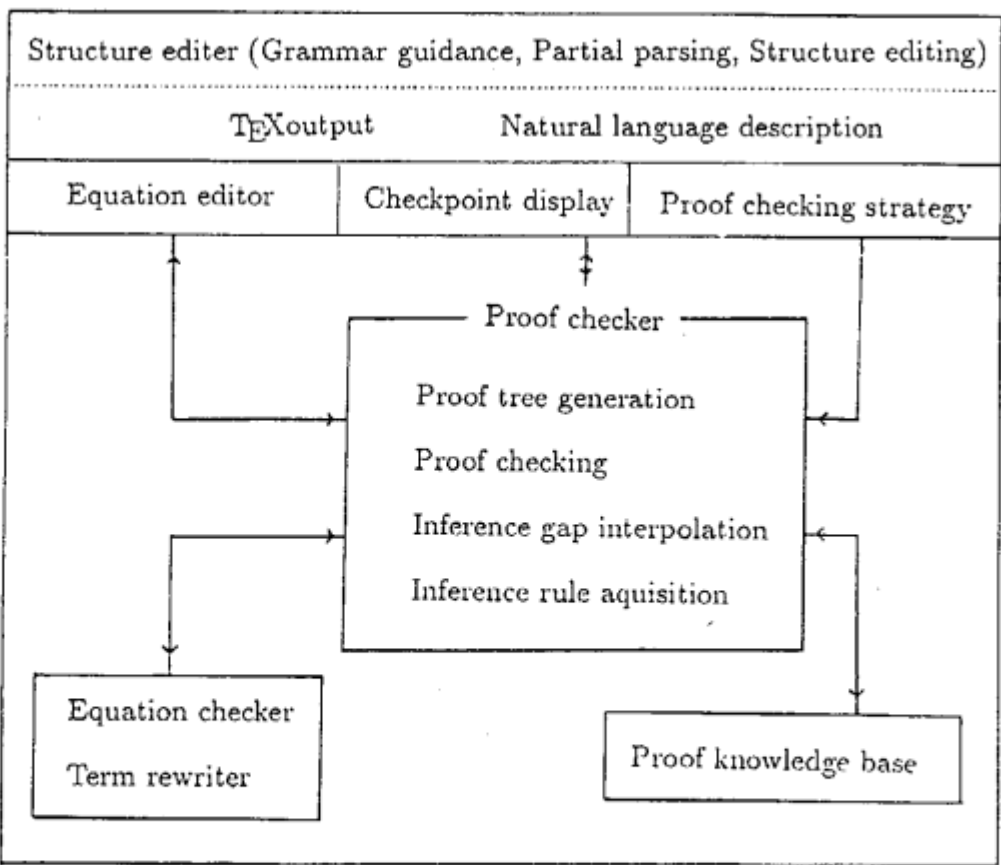
★ Conclusions

- *Constraint Logic Programming*

- 1) Constraint Logic Programming extends unification to constraint solving, allowing symbolic answers to queries
- 2) Powerful semantically clean, language in which to generate and solve constraints.
- 3) More powerful pruning of search space.

- *CAL*

- 1) Solves linear and non-linear equalities over complex numbers using Gröbner Base method.
- 2) Solves Boolean constraints using Boolean Gröbner Base method developed at ICOT.

Title	Computer Aided Proof (CAP-LA)
Purpose	(1) Man-machine cooperation in mathematical problem solving (2) Assistance in proof checking and formula manipulation
Outline & Features	outline (1) Checking theorems and proofs in linear algebra (2) Assistance in writing theorems and proofs features (1) Checking proofs with inference gaps (2) Interactive checking and debugging (3) Proof-structure-oriented writing and editing
System Configu- ration	 <pre> graph TD SE[Structure editor Grammar guidance, Partial parsing, Structure editing] subgraph SE_Box [] direction LR TE[TeXoutput] NL[Natural language description] end EE[Equation editor] CD[Checkpoint display] PCS[Proof checking strategy] PC[Proof checker] subgraph PC_Box [] PTG[Proof tree generation] PCH[Proof checking] IG[Inference gap interpolation] IRA[Inference rule acquisition] end EC[Equation checker] TR[Term rewriter] PKB[Proof knowledge base] SE --> EE SE --> CD SE --> PCS EE --> PC CD --> PC PCS --> PC PC --> EE PC --> CD PC --> PCS PC --> EC PC --> TR PC --> PKB EC --> EE TR --> EE PKB --> PC </pre> The diagram illustrates the system configuration of CAP-LA. At the top is the 'Structure editor (Grammar guidance, Partial parsing, Structure editing)'. Below it is a horizontal bar divided into 'TeXoutput' and 'Natural language description'. Underneath this bar are three components: 'Equation editor', 'Checkpoint display', and 'Proof checking strategy'. These three components have arrows pointing to a central 'Proof checker' box. The 'Proof checker' box contains four sub-components: 'Proof tree generation', 'Proof checking', 'Inference gap interpolation', and 'Inference rule acquisition'. Arrows point from the 'Proof checker' box to the 'Equation editor', 'Checkpoint display', and 'Proof checking strategy'. Below the 'Proof checker' box are two more boxes: 'Equation checker' and 'Term rewriter' (grouped together), and 'Proof knowledge base'. Arrows point from the 'Proof checker' box to these two boxes, and from the 'Equation checker' and 'Term rewriter' box back to the 'Equation editor'. An arrow also points from the 'Proof knowledge base' box to the 'Proof checker' box.

Proof written in PDL (Proof
Description Language)

```

theorem
  det_trans:
  all A:squ_mat.
    det(A)=det(trans(A))
  proof
    let A:squ_mat be arbitrary:
    row(A)=col(trans(A));
    col(A)=row(trans(A));
    det(A)=sigma p:perm<col(A)>.
      (sgn(p)*pi i:1..col(A).
        A[p(i), i])
    =sigma p:perm<col(A)>.
      (sgn(inv(p))*pi i:1..col(A).
        A[(inv(p))(i), i])
    using sigma_pi
    =sigma p:perm<col(A)>.
      (sgn(p)*pi i:1..col(A).
        A[(inv(p))(i), p(i)])
    using sigma_pi
    =sigma p:perm<col(A)>.
      (sgn(p)*pi i:1..col(A).
        A[(inv(p))(p(i)), p(i)])
    using sigma_pi
    =sigma p:perm<col(A)>.
      (sgn(p)*pi i:1..col(A).
        A[i, p(i)])
    using sigma_pi
    =sigma p:perm<col(A)>.
      (sgn(p)*pi i:1..col(A).
        (trans(A))[p(i), i])
    using sigma_pi
    =det(trans(A))
  end_proof
end_theorem

```

TEX output

THEOREM: *det.trans:*

For all A Esquare matrix,

$$\det(A) = \det({}^tA)$$

PROOF:

Now let A Esquare matrix be arbitrary.

$$\text{row}(A) = \text{col}({}^tA)$$

$$\text{col}(A) = \text{row}({}^tA)$$

$$\begin{aligned}
 \det(A) &= \sum_{p \in S_{\text{col}(A)}} \text{sgn}(p) \prod_{i=1}^{\text{col}(A)} A_{p(i), i} \\
 &= \sum_{p \in S_{\text{col}(A)}} \text{sgn}(p^{-1}) \prod_{i=1}^{\text{col}(A)} A_{p^{-1}(i), i} \\
 &\quad \text{using sigma_pi} \\
 &= \sum_{p \in S_{\text{col}(A)}} \text{sgn}(p) \prod_{i=1}^{\text{col}(A)} A_{p^{-1}(i), p(i)} \\
 &\quad \text{using sigma_pi} \\
 &= \sum_{p \in S_{\text{col}(A)}} \text{sgn}(p) \prod_{i=1}^{\text{col}(A)} A_{p^{-1}(p(i)), p(i)} \\
 &\quad \text{using sigma_pi}
 \end{aligned}$$

Equation Editor

(1) Initialization

```

CAP-LA_48 ==> Structure Mode...
Theorem
  det_trans:
  all A:squ_mat.
    det(A)=det(trans(A))
  since
    let A:squ_mat be arbitrary:
      det(A)=det(trans(A))
    end_since
  end_theorem

SEMACS(esp)(95.15] demol sys>user>:
Read: icps:180::>sys>user>semacs>te

EQUALITY EDITOR ==> RIGHT HAND SIDE
det(A)

det(trans(A))

SEMACS(esp)(57.6] #48/2# --top-- #

```

(2) Formula manipulation

```

EQUALITY EDITOR ==> LEFT HAND SIDE
sigma p:perm(col(A)).
  ((sign(p)*pi i:1..col(A).
    (A[i. (p) (i)])))

det(trans(A))

SEMACS(esp)(57.6] #48/1# --top-- #
this rule {y/n/a(bort)}?y
replace(i_1): p
replace(i_2): i

RULE WINDOW
Rule Tag: def func det
Condition: (A:squ_mat)

```

(3) Final result

```

CAP-LA_48 ==> Structure Mode...
all A:squ_mat.
  det(A)=det(trans(A))
  since
    let A:squ_mat be arbitrary:
      det(A)=sigma p:perm(col(A)).
        ((sign(p)*pi i:1..col(A).
          (A[i. (p) (i)])))
      *sigma p:perm(col(trans(A)).
        ((sign(p)*pi i:1..col(trans(A)).
          (A[p(i). i])))
      *sigma p:perm(col(trans(A)).
        ((sign(p)*pi i:1..col(trans(A)).
          ((trans(A))[i. (p) (i)])))
      *det(trans(A))
    end_since
  end_theorem

SEMACS(esp)(95.15] demol sys>user>semacs>text>det_trans..1 --4X-- #

```

Structure-oriented proof development

(This example illustrates "universal quantifier elimination")

Before elimination

```

CAP-LA_48 ==> Structure Mode...
theorem
  det_trans:
    all A:squ_mat.
      det(A)=det(trans(A))
end_theorem

```

menu

<rule_menu>

univ_elim

contradiction

After elimination

```

CAP-LA_48 ==> Structure Mode...
theorem
  det_trans:
    all A:squ_mat.
      det(A)=det(trans(A))
    since
      let A:squ_mat be arbitrary;
      det(A)=det(trans(A))
    end_since
end_theorem

```

Proof checking (This example illustrates an error in indexing)

```

CAP-LA_55 ==> Structure Mode...
theorem
  trans_trans:
    all m, n:pos. A:matrix(m, n).
      (trans(trans(A)))=A
    since
      let m, n:pos, A:matrix(m, n) be arbitrary;
      trans(trans(A))=A
      since using theorem mat_E0:
        col(trans(trans(A)))=row(trans(A))
        =col(A);
        row(trans(trans(A)))=col(trans(A))
        =row(A);
        all i:1..col(trans(trans(A))), j:1..row(A).
          (trans(trans(A)))[i, j]=(A[i, j])
      since
        let i:1..col(trans(trans(A))), j:1..row(A) be arbitrary;
        (trans(trans(A)))[i, j]=(trans(A))[i, j]
        =A[i, j]
      end_since
    end_since
end_theorem

```

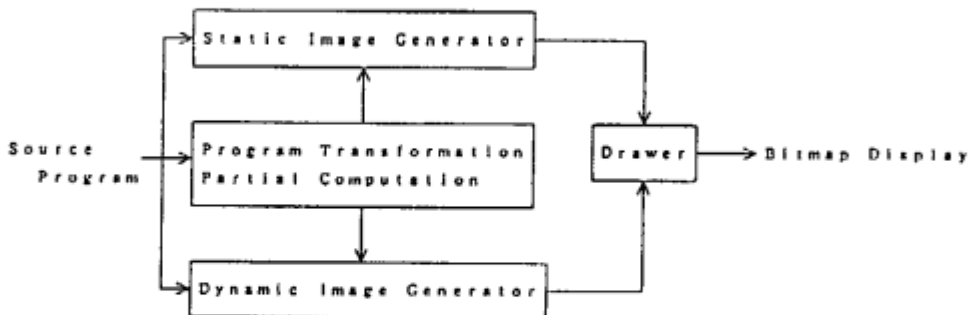
SEMACS(esp)(96.19) sys>user>semacs>text>TRIR.88.1 --29%--

Grammar name? >pd1

Top Category name(theory)? >

Buffer (TRIR): ^G

CAP CHECKER	TRS RULES
UNIFY MONITOR goal: (trans(trans(A)))[i, j] = (trans(A))[i, j] rule: (trans(A))[_B, _C] = _A[_C, _B] found difference of... i and _B, j and _C, i and _C, j and _B	EQUALITY >>col(trans(A)) = n >>row(trans(trans(A))) = n >>col(trans(trans(A))) = m >>row(trans(A)) = m >>row(A) = n >>col(A) = m INEQUALITY >>n >= 1 >>n >= j >>j >= 1 >>m >= 1 >>m >= i >>i >= 1
RULE POOL MONITOR restore function : trans ... O.K.	
EQUALITY TRS MONITOR (6.2) Goal>(trans(trans(A)))[i, j] = (trans(A))[i, j] : (by equality) press any key	

Title	VISTA : Visualization and Transformation Apprentice for Concurrent Logic Programming
Purpose	The system helps programmers develop concurrent logic programs easily by providing visualization of program structures which may be useful to understand programs.
Outline & Features	◇ VISTA has two visualization modes - Static mode : a structure of the program is visualized by unfolding a given top level goal. - Dynamic mode : an execution of the program is visualized by replacing a parent process with child processes and by redrawing the stream lines between processes. ◇ VISTA can be used to examine concurrent programming techniques - Layered-stream program - Knuth-Bendix completion program ◇ VISTA can be used to compare different versions of programs - Two versions of unification program - Partially evaluated program and its original
System Configu- ration	 <pre> graph LR SP[Source Program] --> SIG[Static Image Generator] SP --> PTPC[Program Transformation Partial Computation] SP --> DIG[Dynamic Image Generator] PTPC --> SIG PTPC --> DIG SIG --> D[Drawer] DIG --> D D --> BDD[Bitmap Display] </pre> The diagram illustrates the system configuration. A 'Source Program' input splits into three parallel paths: one to the 'Static Image Generator', one to the 'Program Transformation Partial Computation' block, and one to the 'Dynamic Image Generator'. The 'Program Transformation Partial Computation' block has bidirectional arrows connecting it to both the 'Static Image Generator' and the 'Dynamic Image Generator'. Both the 'Static Image Generator' and the 'Dynamic Image Generator' output to a central 'Drawer' block. The 'Drawer' block then outputs to a 'Bitmap Display'.

◇ Layered-stream program

Layered stream is a type of data structure, which is designed for efficient search programming in GHC. Through a layered stream, information is propagated to consumer processes as soon as possible, even if the information is incomplete. Thus it provides very high parallelism. In our demonstration, a 4-Queens program using a Layered Stream is visualized.

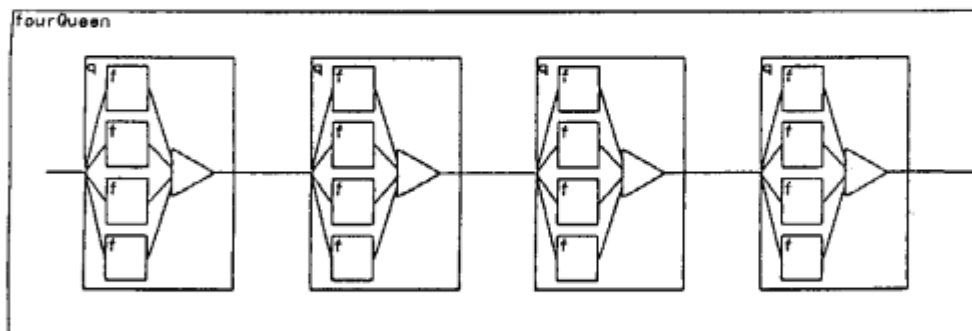
Source program:

```
fourQueens(Q4) :- true |
    q(begin,Q1), q(Q1,Q2), q(Q2,Q3), q(Q3,Q4).

q(In,Out) :- true |
    filter(In,1,1,Out1), filter(In,2,1,Out2),
    filter(In,3,1,Out3), filter(In,4,1,Out4),
    Out = [1*Out1,2*Out2,3*Out3,4*Out4].

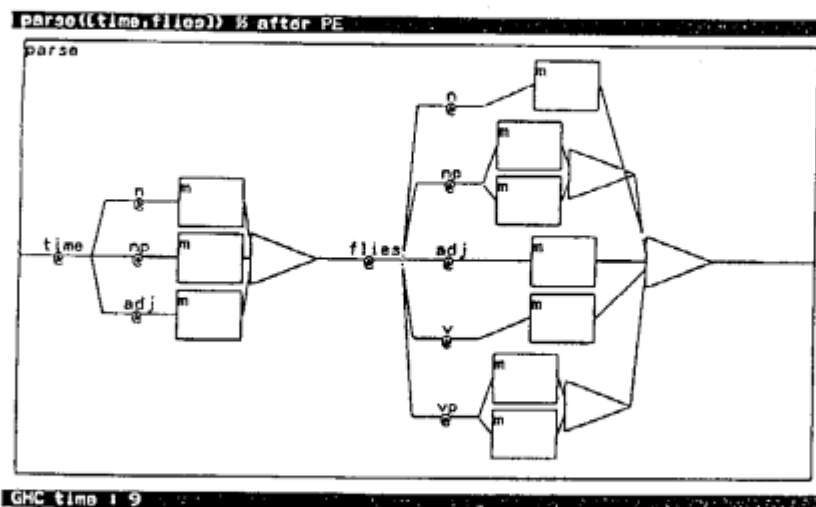
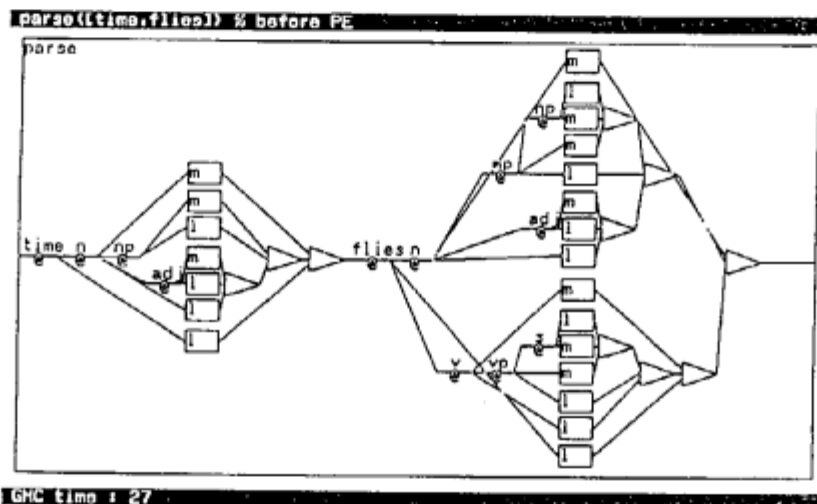
filter(begin,_,_,Out) :- true | Out = begin.
filter([],_,_,Out) :- true | Out = [].
filter([I*_|Ins],I,D,Out) :- true | filter(Ins,I,D,Out).
filter([J*_|Ins],I,D,Out) :- D == I-J | filter(Ins,I,D,Out).
filter([J*_|Ins],I,D,Out) :- D == J-I | filter(Ins,I,D,Out).
filter([J*In1|Ins],I,D,Out) :- J \= I, D \= I-J, D \= J-I |
    D1 := D+1, filter(In1,I,D1,Out1),
    filter(Ins,I,D,Outs),
    Out = [J*Out1|Outs].
```

fourQueen



◇ Parsing program

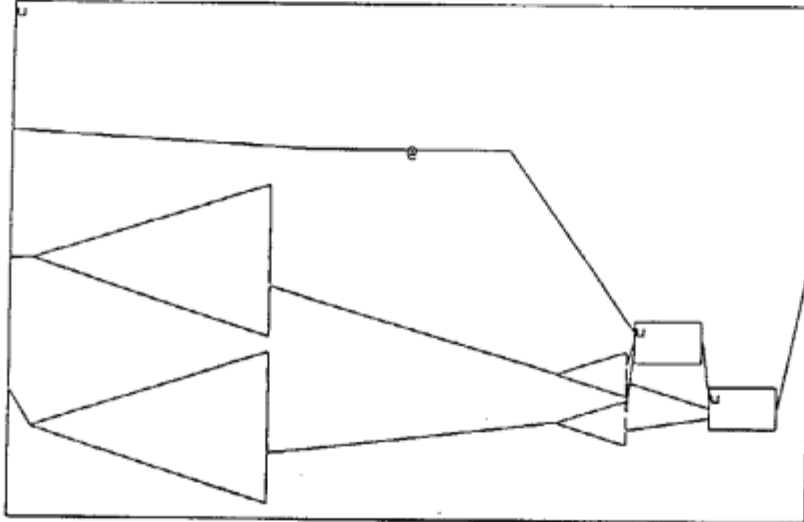
A parsing program called Meta-PAX is presented. Meta-PAX, given a set of grammar rules and a sentence, checks whether the sentence is acceptable. A sentence varies each time Meta-PAX is used whereas a set of grammar rules is fixed for some time while Meta-PAX is used, for parsing many sentences. Using partial evaluation, a version of the Meta-PAX program specialized for the given set of grammar rules is obtained which is more efficient than the original program.



◇ Unification program

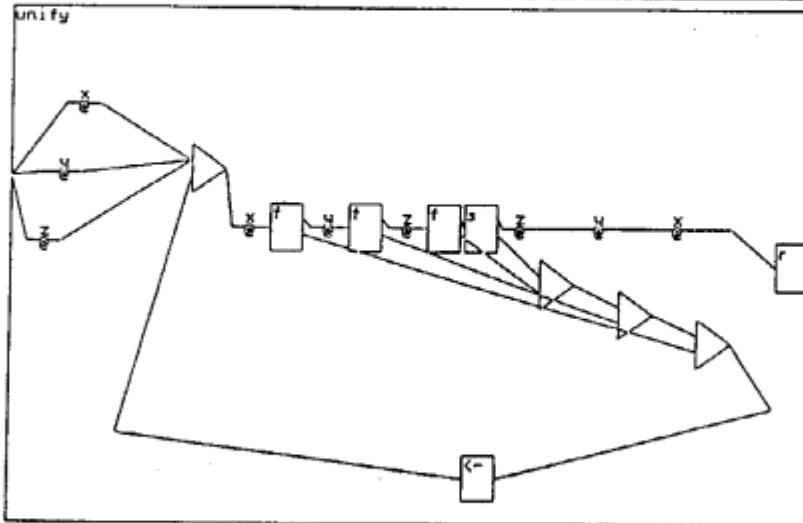
Two programs for unifying two terms are presented. The difference between the programs can be easily seen by visualizing them. The first one tries to unify two compound terms one branch at a time, updating the intermediate result of the unifying substitution. The second one tries to generate a unifying substitution at each branch of the given compound terms in parallel and tries to check consistency between separately generated substitutions.

sequential unification



GHC time 1.28

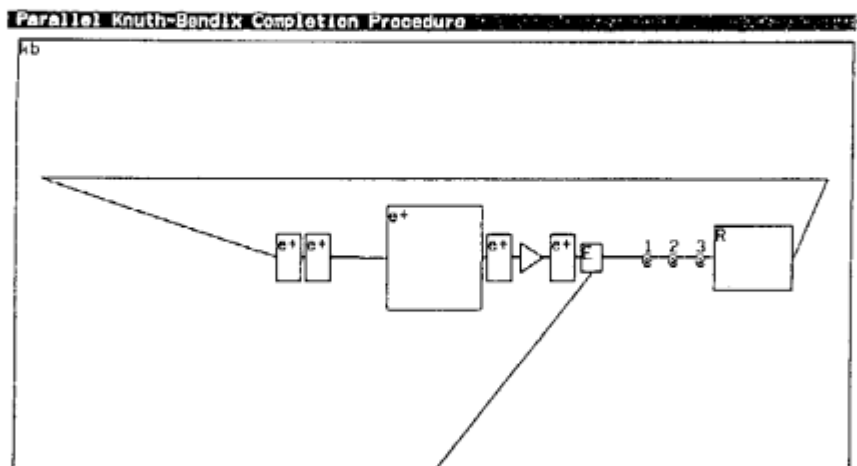
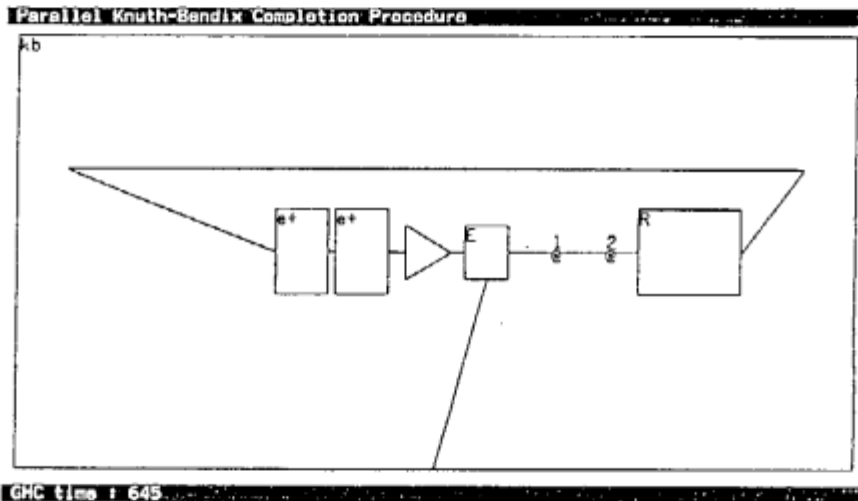
parallel unification

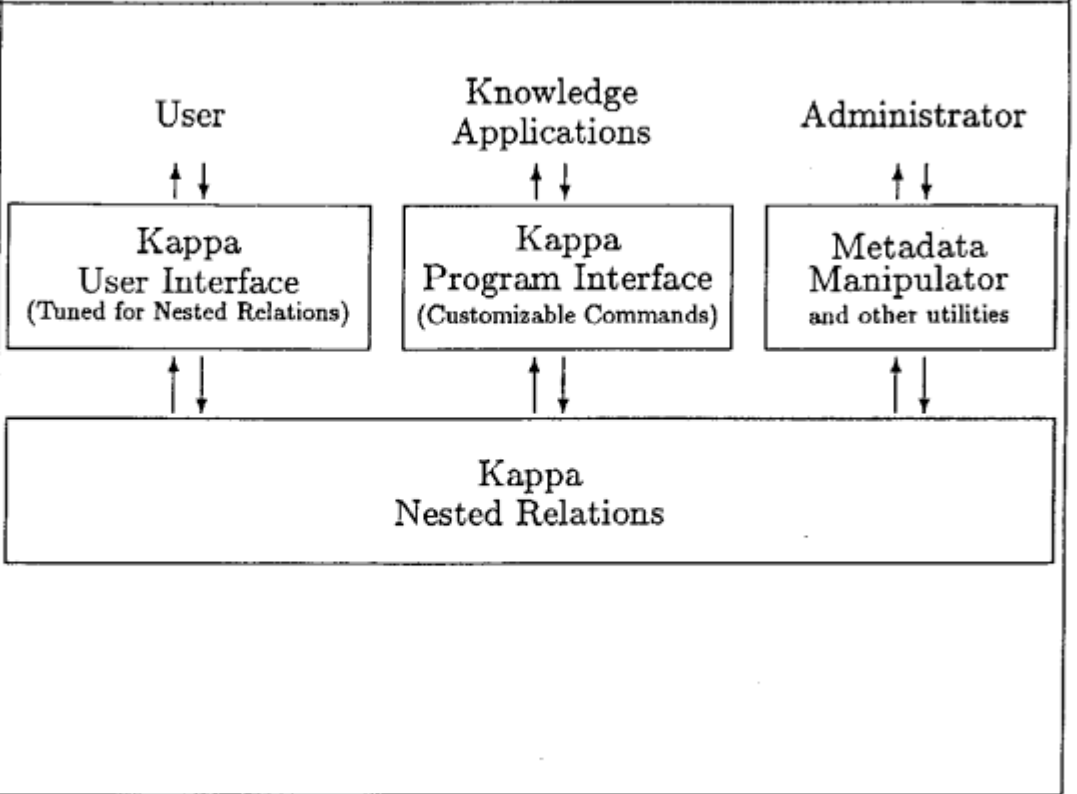


GHC time 1.18

◇ Knuth-Bendix completion program

The Knuth-Bendix completion program is a kind of a compiler that, given a set of equations, produces a set of term rewriting rules which has a nice property called canonical. There are two major processes running in parallel in the program: one for converting an equation to a rewrite rule, the other for generating a new equation (called a critical pair) from a pair of rewrite rules. Critical pair generation can be done in parallel for every pair of rewriting rules, whereas conversion of an equation has to be serialized so that the simpler rewriting rule is generated first. The serialization of the conversion process is realized by using a unique programming technique that makes the most of stream parallelism.



Title	Knowledge Application Oriented Advanced DBMS : Kappa
Purpose	Kappa, a DBMS based on the nested relational model, is implemented on PSI-II in order to study management of very large and/or complex structured databases in the logic programming environment and to provide a platform for implementing various knowledge applications (including deductive DBs and semantic networks).
Outline & Features	Kappa is a DBMS with the following features: 1. Nested relational model is adopted. 2. Terms stored as one data type are retrieved by unification. 3. Large amounts of data, such as electronic dictionaries, mathematical knowledge and genetic information, are effectively accessed. 4. An User interface suitable for nested relations and program interface customizable for various applications are provided. 5. It's written in object-oriented logic programming language ESP, and will be rewritten in parallel language KL1.
System Configuration	 <pre> graph TD User[User] <--> UI[Kappa User Interface (Tuned for Nested Relations)] KA[Knowledge Applications] <--> KPI[Kappa Program Interface (Customizable Commands)] Admin[Administrator] <--> MM[Metadata Manipulator and other utilities] UI <--> KNR[Kappa Nested Relations] KPI <--> KNR MM <--> KNR </pre> The diagram illustrates the system configuration of Kappa. At the top, three entities are listed: User, Knowledge Applications, and Administrator. Below each entity is a box representing its interface: 'Kappa User Interface (Tuned for Nested Relations)', 'Kappa Program Interface (Customizable Commands)', and 'Metadata Manipulator and other utilities'. Bidirectional arrows connect each entity to its respective interface box. At the bottom, a large box labeled 'Kappa Nested Relations' serves as the central data layer. Bidirectional arrows connect each of the three interface boxes to this central layer.

GenBank Sequence DB on Kappa

1 Guidance

1.1 Features of Nested Relational Model

The definition of the nested relational model (according to the standpoint of Kappa) is:

$$NR \subseteq E_1 \times \dots \times E_n$$

$$E_i ::= D \mid 2^{NR}$$

while that of the relational model is:

$$R \subseteq D_1 \times \dots \times D_n$$

where D_i is a domain, R is a relation and NR is a nested relation.

In Kappa (the nested relational model) :

1. We can represent tree-structure naturally.
 - It's suitable for complex structured data.
 - It's more user-friendly than the relational model.
2. We can utilize features of the relational model.
 - Extended relational algebra is available.
 - Entity-relationship concept are effective at the design and the management phases.
 - Deductive database system is implemented on Kappa.

1.2 Schema of GenBank/Kappa

Schema based on nested relational model for GenBank data is shown in Fig. 1 in detail.

gene : main table which has locus name, definition, accession, keywords, identifiers to the other tables, and so on.

reference : table which has its authors, title, journal where it has appeared, and so on.

feature : consists of a region of the sequence and its feature.

seqdata : sequence data represented in string form.

1.3 Stored Data

Data we stored for this demonstration is sequences of invertebrate, virus, bacteria and phage. The total amount is 7285 entries, 10 mega bases, and 22 mega characters in the original data, which is stored into 60 mega bytes (30 mega characters) including 10 indexes and about 16 mega bytes (8 mega characters) of 'textdata' in Kappa.

GenBank sequence DB (89.6.15) has 26323 entries, 32 Mega bases. So it will cost about 240 mega bytes in Kappa.

2 Demonstration

2.1 Show Schema : Metadata Manipulator

We can see the schema of the database.

We show the schema of all four tables of GenBank/Kappa (gene, seqdata, reference and feature, Fig. 1) through metadata manipulator.

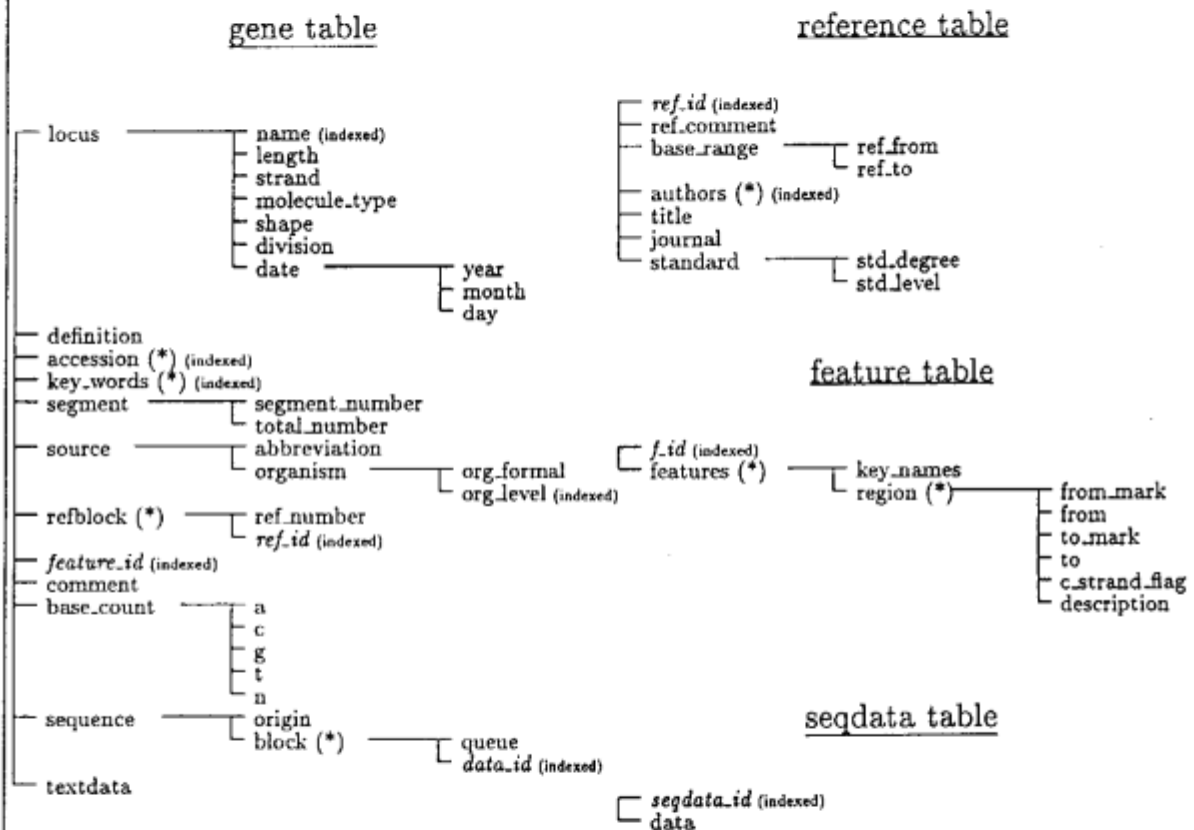


Fig. 1 Schema of GenBank/Kappa ('(*)' means repeating)

2.2 Display Tables : Kappa User Interface

We can see how the data is stored.

We show all four tables by projecting into smaller tables (Fig 2.1), and the contents of their attributes by clicking each cell. Kappa can have attributes with variable length directly in each record. Now in the attribute *textdata* we can see the excerpt of each flat gene data as it is in GenBank original (Fig 2.2).

Kappa can also have attributes with multi-values. So it is necessary to scroll values in the user interface.

VF2 TABLE USER I/F V1

Make Save CAN CEL Kapp a V1

reference/table/4 (set0) ■

ref_1	ref_comment	authors	title	journal
inv1	bases 1 to 1571	Heiligen, W.	Actin genes and actin messenger RNA in Acanthamoeba castellanii	J. Mol. Biol. 159, 1-18 (1982)

seqdata/table/1

seqdata	seqdata
inv1	ggagagagat
inv2	gtcgacagcc
inv3	gccttcgccc
inv4	ggcgagagcc
inv5	ctcgacagcc
inv6	aagcttacc
inv7	aagcccgaga
inv8	ggagagagcc
inv9	agccagagcc

feature/table/3 (set0) ■

f_id	features	key_name	region	from	to	description
inv1	psst	137	451	actin exon 1		
	pre-mag	581				
	gene	1				
	locus					
	name					
	long					
	defn					
	accession					
	source					
	ref1					
	featu					
	comment					
	seq					
	text					

gene/table/1 (set0) ■

name	long	defn	accession	source	ref1	featu	comment	seq	text
ACAACT1	1571	Acanthamoeba (A. castellanii) actin gene-1.	J01016	Acanthamoeba castellanii (amoeba)	inv1	inv1	A potential Goldberg-Hogness box (5'-a-a-a-a-a-3') lies 21 or		LOCUS
ACAACT2	1571	A. castellanii non-muscle myosin heavy chain gene, partial	V00524	Acanthamoeba (amoeba)	inv2	inv2			LOCUS
ACAACT3	1571	A. castellanii non-muscle myosin heavy chain gene, partial	M12702	NA	inv3	inv3			LOCUS
ACAACT4	1571	A. castellanii non-muscle myosin heavy chain gene, partial	M12703	NA	inv4	inv4			LOCUS
ACAACT5	1571	A. castellanii non-muscle myosin heavy chain gene, partial	J02874	Acanthamoeba (amoeba)	inv5	inv5	clean copy of sequence [1] kindly pr		LOCUS
ACAACT6	1571	A. castellanii non-muscle myosin heavy chain gene, partial	M13564	NA	inv6	inv6			LOCUS
ACAACT7	1571	A. castellanii non-muscle myosin heavy chain gene, partial	M13565	NA	inv7	inv7			LOCUS
ACAACT8	1571	A. castellanii non-muscle myosin heavy chain gene, partial	M13565	NA	inv8	inv8			LOCUS

数字形式リストを作成しています... 終了しました。
シートを作成します。マウスで位置を決めて下さい... 終了しました。
シートを読み込んでいます... 終了しました。

Fig. 2.1 Kappa User Interface

I/O WINDOW

LOCUS ACAACT1 1571 bp DNA INV 04-SEP-1984

DEFINITION Acanthamoeba (A. castellanii) actin gene-1.

ACCESSION J01016

KEYWORDS actin.

SOURCE Acanthamoeba castellanii (amoeba) dna.

ORGANISM Acanthamoeba castellanii

REFERENCE 1 (bases 1 to 1571)

AUTHORS Heiligen, W. and Gallwitz, D.

TITLE Actin genes and actin messenger RNA in Acanthamoeba castellanii: nucleotide sequence of the split actin gene 1

JOURNAL J. Mol. Biol. 159, 1-18 (1982)

STANDARD full start/review

COMMENT A potential Goldberg-Hogness box (5'-a-a-a-a-a-3') lies 21 or

FEATURES from tospan description

psst 137 451 actin exon 1

pre-mag 581 1383 actin exon 2

pre-mag 581 1383 actin gene-1 mRNA (5' and +/- 1bp)

INV 452 580 actin gene-1 intron 1

recount 1 1 numbered -136 in [1]; zero not used

BASE COUNT 313 a 535 c 389 g 334 t

ORIGIN Nine-III site about 136 bp upstream from the actin cda start

1 ggaagagcgt gcaagcaata accaagagca agagcaact ctaggagcc agcccccac

61 taaagagcgt gcttcagcga cgtatctt tagttatca cactcttacc acacttacc

121 caactatata atcaaatagg gagaagaggt taggagcttg gttatcagca accgttcagg

Fig. 2.2 GenBank Original Textdata

2.3 Retrieve Data

We can retrieve data by various conditions.

Example: make a table of genes whose reference Mr. Smith,C. writes.

$$\pi_{\text{ref_id}}(\sigma_{\text{authors}='Smith,C.'}(\text{reference})) \bowtie \text{gene}$$

Tables and attributes are shown in Fig. 3.

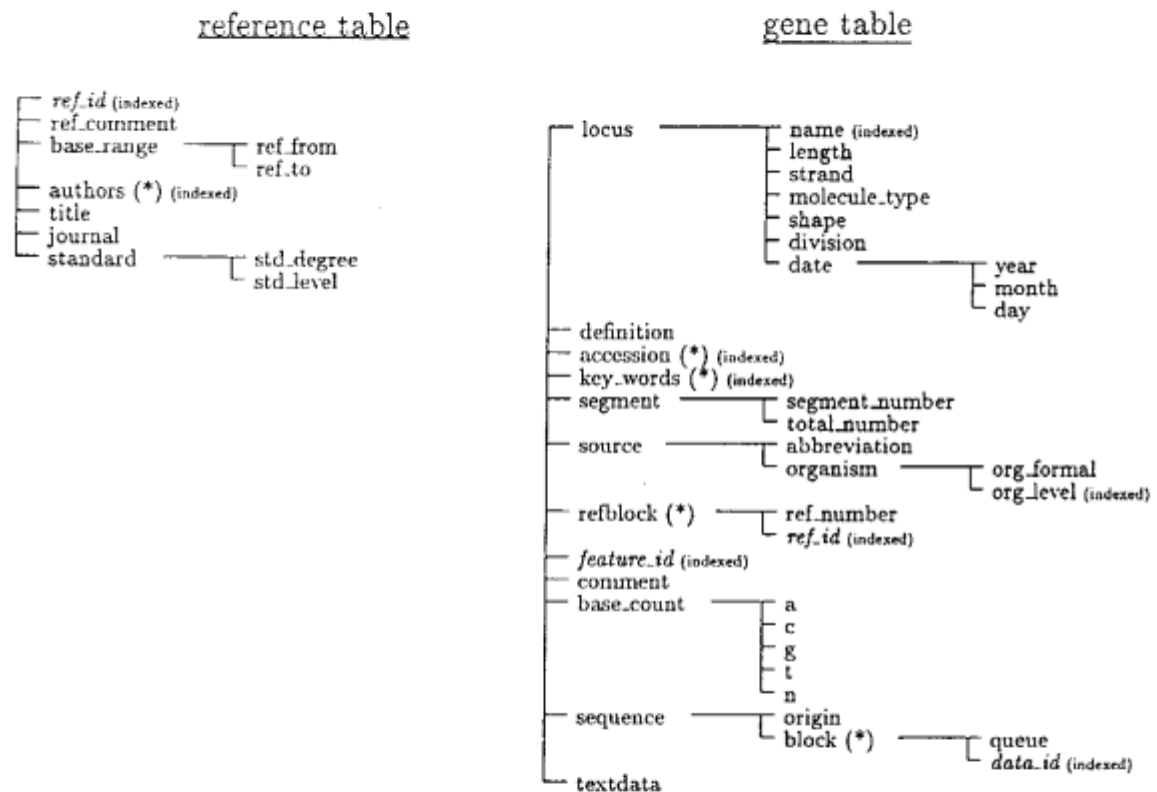


Fig. 3 Tables and Attributes('(*)' means repeating)

2.4 Translate into Protein & DP matching

We can translate DNA code into amino acid sequence and execute DP matching.

Flowchart is shown in Fig. 4.1.

Translation We select a DNA sequence in the feature table to translate into an amino acid sequence. The sequence is translated according to the table shown in Fig. 4.2.

DP matching We compare the 'translated' sequence with the object sequence determined in advance. The sequences are compared according to the table selected by the user shown in Fig. 4.3 for example.

(GenBank DNA)

atggttgactgggcaatggacgtacactaa

Translation Phase

atg $\Rightarrow$ start(m)

$\vdots$

gac $\Rightarrow$ d

gta $\Rightarrow$ v

$\vdots$

tag,taa,tga $\Rightarrow$ end.

mvdwamdqv

(Amino Acid)

(Object Amino Acid)

mvwpldhq

DP matching Phase

mv-wpld-hq

||.||||..|

mvdwamdqv-q

Fig. 4.1 Translation and DP matching

Fig. 4.2 DNA-Amino Acid Table

1st (5'end)	2nd				3rd (3'end)
	T	C	A	G	
T	f	s	y	c	T
			*	*	C
	l		*	w	A
C					G
	l	p	h	r	T
			q		C
A					A
	i	t	n	s	G
			k	r	T
G	m				C
	v	a	d	g	A
			e		G

(* terminate codon)

Fig. 4.3 Amino Acid Difference Table

Mutability	c	s	t	p	a	g	n	d	e
c: cysteine	6								
s: serine	4	6							
t: threonine	2	5	6						
p: proline	2	4	4	6					
a: alanine	2	5	5	5	6				
g: glycine	3	5	2	3	5	6			
n: asparagine	2	5	4	2	3	3	6		
d: aspartic acid	1	3	2	2	4	4	5	6	
e: glutamic acid	0	3	3	3	4	4	3	5	6
q: glutamine	1	3	3	3	3	2	3	4	4
h: histidine	2	3	2	3	2	1	4	3	2
r: arginine	2	3	3	3	2	3	2	2	2
k: lysine	0	3	4	2	3	2	4	3	4
m: methionine	2	1	3	2	2	1	1	0	1
i: isoleucine	2	2	3	2	2	2	2	1	1
l: leucine	2	2	2	3	2	2	1	1	1
v: valine	2	2	3	3	5	4	2	3	4
f: phenylalanine	3	3	1	2	2	1	1	1	0
y: tyrosine	3	3	2	2	2	2	3	2	1
w: tryptophan	3	2	1	2	2	3	0	0	1

Title	Hypothetical Reasoning System: APRICOT/0
Purpose	The purpose of this demonstration is to show that APRICOT/0 is an efficient system in both rule compiling and hypothetical reasoning. As an example on APRICOT/0, a logic design problem is taken up.
Outline & Features	The APRICOT/0 system for hypothetical reasoning consists of an ATMS and a rule-based inference engine. The ATMS maintains consistency based on environments which are sets of assumptions. The rule-based inference engine is a forward-chaining system based on the Rete algorithm extended to handle multiple contexts. APRICOT/0 is implemented in ESP. It has the following features: 1. Generating hypotheses dynamically. 2. Avoiding redundant execution of problem-solving tasks which are shared among multiple contexts in the ATMS. 3. Allowing faster hypothetical reasoning since the two-input nodes in the Rete network keep the internal labels which are sets of environments. 4. Recompiling knowledge base faster due to incremental construction method of the Rete network.
System Configu- ration	<pre> graph TD User[User] -- "Addition and Removal of rules" --> IC[Incremental compiler] IC -- "Rete network Premises Hypotheses" --> IE[Inference engine] IE -- "Beliefs" --> ATMS[ATMS] ATMS -- "Generation of hypotheses Justifications" --> IE IC -- "State of a network" --> User ATMS -- "State of beliefs" --> User </pre>

In design problems such as logic design, many choice alternatives often appear in different levels of a hierarchy. APRICOT/0 can deal with those alternatives as hypotheses generated dynamically. As an example of logic design, the circuit calculating the greatest common divisor (G. C. D.) between two integers by using the Euclidean algorithm is taken up.

The rule set contains several kinds of knowledge: datapath design (Fig. 1), component design (Fig. 3), CMOS standard cell library (Fig. 5), and area and time constraints.

The input to APRICOT/0 is the rule set (Fig. 2, Fig. 4, Fig. 6) and output is circuits which are described by CMOS standard cells, that satisfy all given constraints.

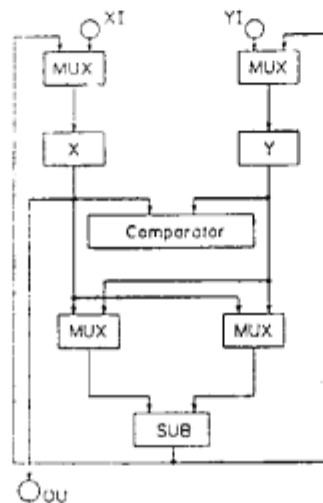


Fig. 1: An example of knowledge about block diagrams of datapaths.

```

circuit_2::
  component(dpa_ctrl_box, D_CTRL_BOX, CN1),
  component(register, REG1, CN2),
  component(register, REG2, CN3),
  component(not, NOT, CN4),
  component(multiplexer, MUX1, CN5),
  component(multiplexer, MUX2, CN6),
  component(comparator, COMP, CN7),
  component(f_subtractor_with_mux, SUB_with_MUX, CN8),
  cell_number(CELL_C),
  (CN1+CN2+CN3+CN4+CN5+CN6+CN7+CN8 = CELL_NUM,
   CELL_C >= CELL_NUM),
  delay_constraint(Delay_constraint),
  {
    :delay(#circuit_constraint, [D_CTRL_BOX, REG1, REG2, NOT, MUX1, MUX2, C
OMP, SUB_with_MUX], Delay),
    Delay_constraint >= Delay,
    :plot(#plot, CELL_NUM, Delay)
  }
->
circuit(2, [D_CTRL_BOX, REG1, REG2, NOT, MUX1, MUX2, COMP, SUB_with_MUX], Delay
CELL_NUM).

```

Fig. 2: A rule representing knowledge shown in Fig. 1.

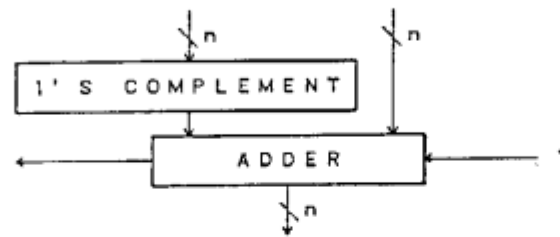


Fig. 3: An example of knowledge for component design about a subtractor.

```

component_design_f_Subtractor::
    necessary_element(subtractor),
    component(adder, ADDER, CELL_NUM1),
    component(f_1_Complementer, CMPL_1, CELL_NUM2),
    { CELL_NUM1 + CELL_NUM2 = CELL_NUM }
->
    assume(component(subtractor, subtractor([ADDER, CMPL_1]), CELL_NUM)).
    
```

Fig. 4: A rule representing knowledge shown in Fig. 3.

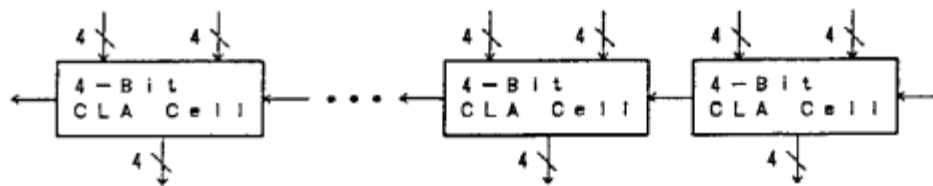


Fig. 5: An example of knowledge about an adder mapped by the CMOS standard cells.

```

technology_mapping_f_Adder_1::
    necessary_element(adder),
    cell(a4h, NUM),
    { CELL_NUM = NUM*2 }
->
    assume(component(adder, adder([a4h, a4h]), CELL_NUM)).
    
```

Fig. 6: A rule representing knowledge shown in Fig. 5.

Constraints on area are expressed as inequalities in the basic cell count, for example: "The total basic cell count must not exceed 400". Constraints on time are expressed as inequalities in the delay, for example: "The delay must not exceed 60 nanoseconds".

APRICOT/0 prunes the search tree earlier by the rules (Fig. 7) for incremental evaluation of constraints on area.

All solutions that satisfy all given constraints (Fig. 8) are obtained by APRICOT/0.

```

circuit_constraint000::
  component(dpg_ctrl_box, D_CTRL_BOX, CN1),
  component(register, REG1, CN2),
  cell_number(CELL_NUM),
  {
    CN1+CN2 > CELL_NUM
  }
->
  [].

circuit_constraint00::
  component(dpg_ctrl_box, D_CTRL_BOX, CN1),
  component(register, REG1, CN2),
  component(register, REG2, CN3),
  component(not, NOT, CN4),
  component(multiplexer, MUX1, CN5),
  component(multiplexer, MUX2, CN6),
  cell_number(CELL_NUM),
  {
    CN1+CN2+CN3+CN4+CN5+CN6 > CELL_NUM
  }
->
  [].

circuit_constraint10::
  component(dpg_ctrl_box, D_CTRL_BOX, CN1),
  component(register, REG1, CN2),
  component(register, REG2, CN3),
  component(not, NOT, CN4),
  component(multiplexer, MUX1, CN5),
  component(multiplexer, MUX2, CN6),
  component(comparator, COMP, CN7),
  cell_number(CELL_NUM),
  {
    CN1+CN2+CN3+CN4+CN5+CN6+CN7 > CELL_NUM
  }
->
  [].

```

Fig. 7: Rules for incremental evaluation of constraints on area.

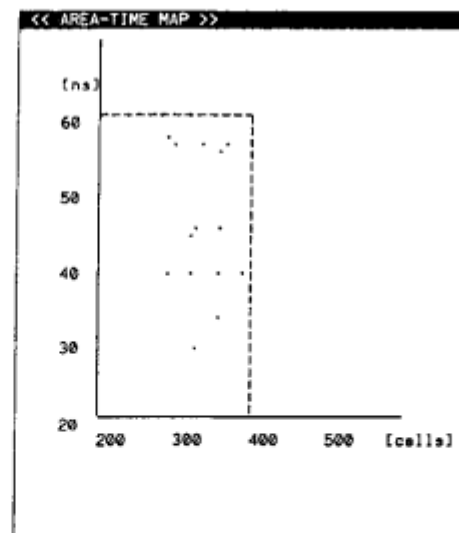
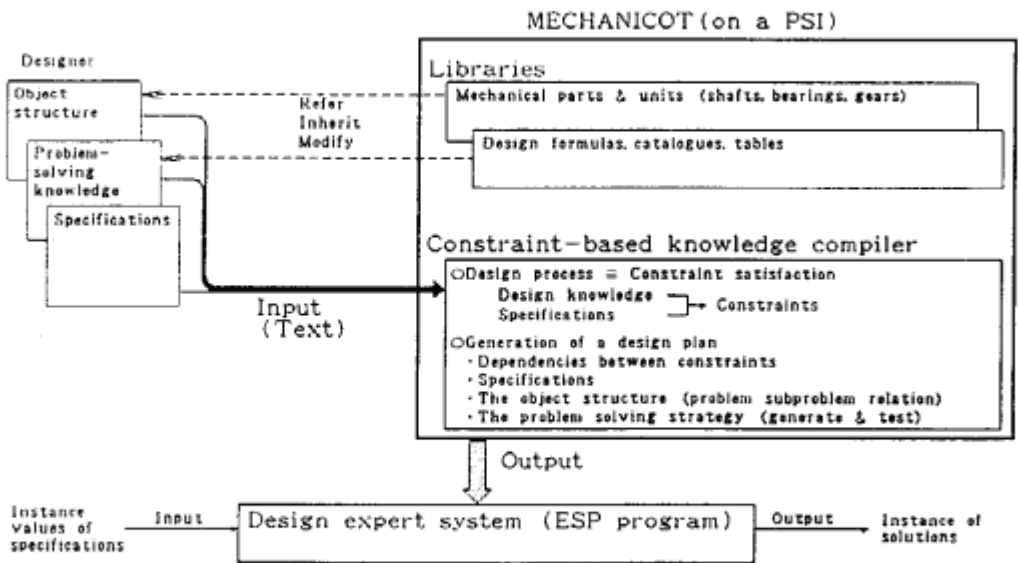


Fig. 8: Mapping all solutions on time-area.

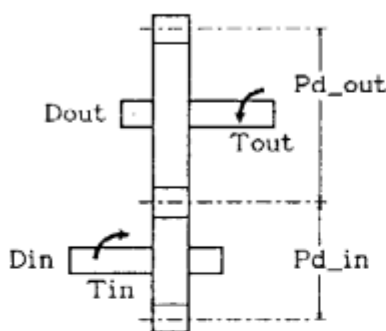
Reference

F. Maruyama, et al., "co-LODEX: A Cooperative Expert System for Logic Design", Proc. of FGCS '88 Vol. 3, pp.1299-1306 (1988).

Title	Constraint-based Knowledge Compiler – MECHANICOT –
Purpose	The purpose of this research is to consider constraints as a knowledge representation and to apply the constraint problem-solving paradigm to practical problems. The focus of the research is to generate a plan for problem solving from declarative knowledge (constraint) representation.
Outline & Features	When knowledge is described as constraints, it is easy to understand and maintain it, since knowledge is expressed in a declarative form. However, it is hard to obtain efficiency in problem solving, because most declarative representations lack control information such as the strategy of problem solving. The constraint-based knowledge compiler (KC) provides facilities to overcome this problem. The KC regards knowledge as constraints, and generates a plan for problem solving by analyzing dependencies between constraints. The inputs to the KC are explicit knowledge written in a frame-like language and the output is a program written in ESP, which is a <i>problem-solving plan</i> generated by the KC. To generate a <i>problem-solving plan</i>, the KC uses: • Constraints • Input parameters • The hierarchy of problem decomposition (problem subproblem relations) • A problem-solving strategy (generate and test, in this case) MECHANICOT is a prototype KC, which is applied to a mechanical design.
System Configuration	 The diagram illustrates the system configuration of MECHANICOT (on a PSI). It shows the interaction between a Designer, Libraries, the Constraint-based knowledge compiler, and the Design expert system (ESP program). Designer: Contains three components: Object structure, Problem-solving knowledge, and Specifications. It interacts with the Libraries via 'Refer', 'Inherit', and 'Modify' operations. Libraries: Contains two components: Mechanical parts & units (shafts, bearings, gears) and Design formulas, catalogues, tables. Constraint-based knowledge compiler: Receives 'Input (Text)' from the Designer's Specifications. It performs two main tasks: ○ Design process = Constraint satisfaction: Design knowledge and Specifications are used to generate Constraints. ○ Generation of a design plan: This involves Dependencies between constraints, Specifications, The object structure (problem subproblem relation), and The problem solving strategy (generate & test). Design expert system (ESP program): Receives 'Input' (Instance values of specifications) and 'Output' (from the compiler). It produces the final 'Output' (Instance of solutions).

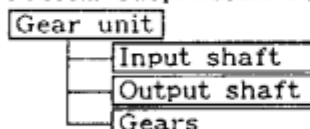
- Purpose of this research
 - Constraints $\equiv$ Declarative knowledge representation
 - To apply the constraint problem-solving paradigm to practical problems
- Declarative knowledge (constraints) $\Leftrightarrow$ Efficiency of problem solving
 - ◎ To generate a plan for efficient problem solving from given constraints
- Constraint-based knowledge compiler
 - Regarding knowledge as constraints
 - Generation of a problem-solving plan
 - Constraints
 - Input parameters
 - The hierarchy of problem decomposition (problem subproblem relations)
 - The problem solving strategy (generate and test)
- MECHANICOT
 - Application of the knowledge compiler to a mechanical design
 - Characteristic of routine design
 - Design knowledge
 - Design specification $\rightarrow$ Constraints
 - Each design leads to a different design plan

[Example of design (gear unit)]



- Specification (Input)
 - Moment: T_{in}, T_{out}
 - Shearing strength: G_{in}, G_{out}
 - Torsion angle: $\theta_{in}, \theta_{out}$
 - Input revolution: R_{in}
- Design parameter (Output)
 - Gear ratio: R_g
 - Gear pitch diameter: Pd_{in}, Pd_{out}
 - Shaft diameter: D_{in}, D_{out}
 - Output revolution: R_{out}

- Knowledge of object structure
 - Problem subproblem relation



- Connective relation
 - To build the unit,
 - D_{in} = (Hole diameter of input gear)
 - D_{out} = (Hole diameter of output gear)

□ Knowledge for problem solving

○ Gear ratio
 $R_g = T_{out} / T_{in}$

○ Output shaft revolution
 $R_{out} = R_{in} / R_g$

○ Shaft diameter

$$D_{in} = \sqrt[4]{\frac{32 T_{in} \times 180 \times 1000}{\pi G_{in} \times \theta_{in}}}$$

$$D_{out} = \sqrt[4]{\frac{32 T_{out} \times 180 \times 1000}{\pi G_{out} \times \theta_{out}}}$$

○ Gear module

$m_{gen}(\text{Module})$ % Select a value from the standard value set
 % {2.0, 2.5, 4.0, 5.0, 6.0}

○ Gear pitch diameters

$Pd \geq (\text{shaft diameter}) + 7 \times (\text{module})$

$Pd \leq 2000 / \pi \times (\text{shaft revolution})$



$Pd_{min_in} = D_{in} + 7m$

$Pd_{min_out} = D_{out} + 7m$

$Pd_{max_in} = 2000 / \pi R_{in}$

$Pd_{max_out} = 2000 / \pi R_{out}$

$pd_gen(Pd_{min}, Pd_{max}, Pd)$

% pd_gen : Generate a value between Pd_{min} and Pd_{max}

○ Gear ratio

$R_g = Pd_{out} / Pd_{in}$

[Input of MECHANICOT]

```
class_name
gear_unit:

consist_of % design object structure
input_shaft.
output_shaft.
gear:

parameter % input & output parameter
tin, tout, rin, rg, rout:

constraint % structural constraint
tin = #input_shaft.tin,
% (#) class (!) parameter
tout = #output_shaft.tout,
rin = #gear.rin,
rout = #gear.rout,
rg = #gear.rg:

design_method
([rg],
rg=tout/tin
),
([rout],
rout=rin/rg
):

end.
```

```
class_name
input_shaft:

parameter
tin, gin, din, dmin:

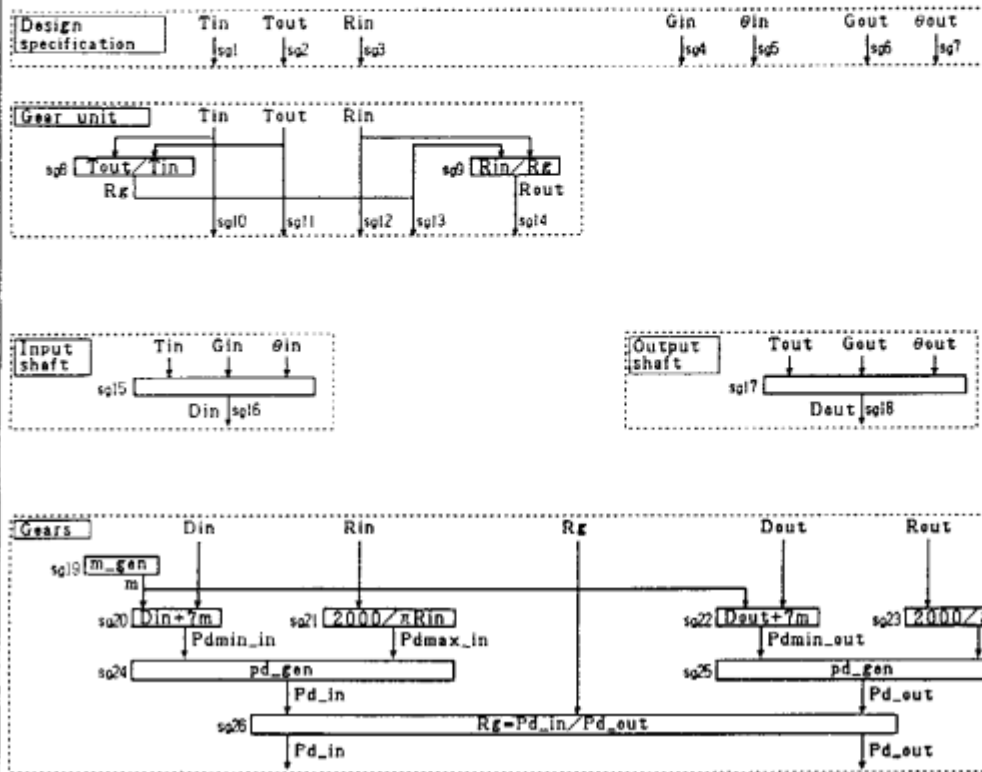
constraint
din = #gear.din:
% (#) class (!) parameter

design_method
([din],
din = ((32*tin*180000)
/ (9.87*gin*din))**0.25
):

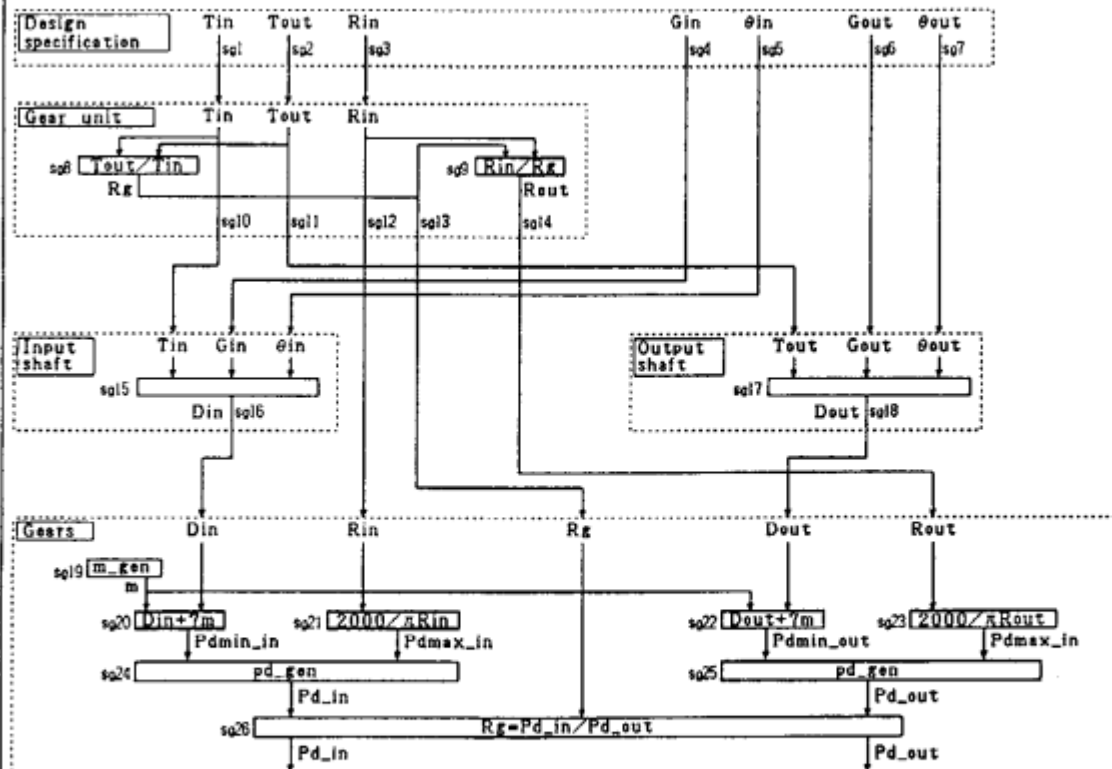
end.
```

D e t a i l s (3/3)

[Intermediate result of analysis]



[Result of analysis]



Title	A Software Library for Japanese Language Processing LTB
Purpose	LTB(Language Tool Box) provides an efficient development environment for basic and general modules for analysis, generation etc.
Outline & Features	◦ <u>Modules</u> LAX: The morphological and semantic analyzer. SAX: Syntactic and semantic analyzer. The grammar is written in DCG. GEN: Japanese sentence generator. The input is a frame structure written in CIL. CIL: Basic programming language of the LTB. Every tool in the LTB has access to CIL and its programming environments. DataBase: The dictionary and the thesaurus, both of which are referred by both analysis modules and the generation module. Shell: The LTB-shell coordinates the operation of every module in a uniform way, and it manages communications between any two modules. ◦ <u>Development environment</u> Editors: edit rules, dictionary, input/output data. Tracers: trace the execution of LTB modules. Inspectors: inspect the system state during/after the execution. Pretty Printers: display the data on the screen.
System Configu- ration	

A Scenario of the Demonstration

Analyzing sentences:

- ◇ LAX analyzes an input sentence, and transforms it into a sequence of phrases.
- ◇ SAX analyzes this sequence of phrases, which is the output of LAX, and constructs its semantic structure.

Generating sentences:

- ◇ GEN generates a Japanese sentence. The input is a frame structure, called Intermediate Representation.

Sample sentences:

Japanese: 人間にとって自然は限り無い資源の宝庫である。

pronunciation: Ningen-ni totte sizen-wa kagirinai sigen-no hôko-dearu.

English: For man, nature is a treasure house of unlimited resources.

intermediate representation:

```
{関係 / {語彙 / である },
rel    lex    'be'
ロール / {記述対象 / {補語 / {語彙 / 自然 }},
role   topic   comp lex 'nature'
内容 / { 補語 / {被修飾 / {被修飾 / {語彙 / 宝庫 },
content comp modifee modifee lex 'treasure-house'
連体修飾 / {語彙 / 資源 }},
modifier lex 'resource'
連体修飾 / {関係 / {語彙 / 限り無い }}}} ,
modifier rel    lex 'unlimited'
状態 / { 表層 / にとって,
manner surf 'for'
補語 / {語彙 / 人間 }}}}
comp lex 'man'
```

Development environment:

LTB-shell:

```
icpsi244:~>sys>user>demo
LTB> goal $gen consult -d "jg_dic_r, c!
mp"

icpsi244:~>sys>user>jgsh_v10>dictionary
ry>jg_dic_r, emp, 2 .... consult on
LTB> goal $gen consult -d "jg_dic_v, c!
mp"

icpsi244:~>sys>user>jgsh_v10>dictionary
ry>jg_dic_v, emp, 2 .... consult on
LTB> create ltb_sax#sax sax
LTB> goal $sax -u "sizen, dcg"
LTB> create lax_vile#lax_shell lax
LTB> goal $lax -d "sizen" -x
LTB> create jgsh_v10#jgsh_top top
LTB>
LTB>
LTB> cat me:sizen.txt

icpsi244:~>sys>user>demo>sizen.txt, 2
人間にとって自然な読み易い表現の言葉がある
end of type
LTB> goal $lax -a ( me:sizen.txt -t
[人間にとって、自然な、読み易い、表現の、言葉、がある]
LTB> □
```

LAX - dictionary editor:

LAX(interpreter)	
Dict:sizen	command
LAX > editor:	exit
	close

	p-printer
	inspector

	analyze

	editors
	new
	continue

	translate
	rev_trans

LAX dictionary editor	
Dict:sizen	command
(editing) 1/4 < * >	inspector
	TOP

	search
	new
	continuous
	category
	surface
	select
	with-buffer
	with-menu
	push-folder
	pop-folder
	clear-all
	clear-data
	add

```

end of type
LTB> goal $lax -x

```

