# Model Generation Theorem Prover, *MGTP*

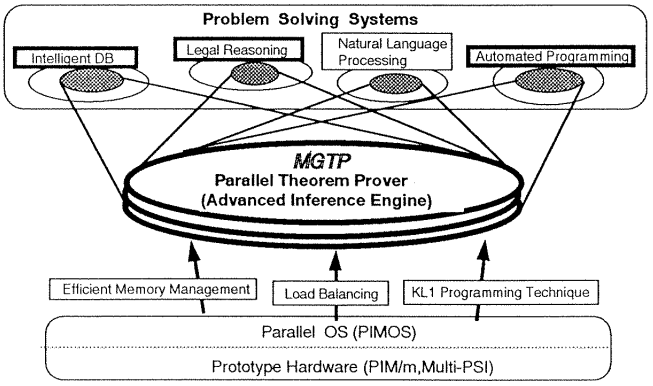

## ABSTRACT

The goal of our research is to build a parallel automated reasoning system on a parallel inference machine(PIM/m), using KL1 and PIMOS technologies. The MGTP prover, currently being developed, adopts the model generation method. In the development of MGTP, we aim to achieve the following:

1. Combine logic programming and automated reasoning, and develop parallelization techniques to implement an efficient first-order theorem prover.
2. Offer an advanced inference engine that can be applied to fields such as intelligent database systems, hypothetical reasoning, natural language processing, and automated programming.


## KEY FEATURES

- Development of clause compiling techniques and meta-programming utilities to implement an efficient prover in KL1.
- Development of AND/OR parallelization techniques, thereby achieving high scalability and linear speed up on PIM/m(256PEs).
- Development of a support environment to make it easy to use and develop MGTP.
- Development of MGTP applications (specification description, abductive reasoning, and automated programming).




**MGTP Relative to FGCS**

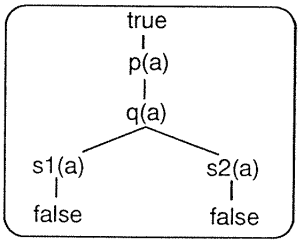
## Model Generation Method:

**- Tries to generate unit clauses (models) from a given set of clauses by forward reasoning.**

**A set of clauses**

```
Initial model:
    true -> p(a).
Model extension rule:
    p(X) -> q(X).
    q(X) -> s1(X);s2(X).
Goal:
    p(a),s1(a) -> false.
    q(a),s2(a) -> false.
```

**Model generation process**




**- Generated unit clauses which include variables**

→ **Non-ground problems**

**- Ganerated unit clauses which include no variables**

→ **Ground problems**

## Examples:

**(a) Party problem**

-Non-Horn and ground problem

We can always choose three persons who are either familiar with each other or not familiar with each other from six persons who meet at a party.

```
true-->person(1),person(2),person(3),person(4),
       person(5),person(6).

person(X),person(Y),X>Y-->f(X,Y);nf(X,Y).

f(X1,X2),f(X2,X3),f(X3,X1)-->false.
nf(X1,X2),nf(X2,X3),nf(X3,X1)-->false.
```

**(b) Condense detachment problems**

-Horn and non-ground problem
-Group theory,ring theory, and implicational logic

```
p(X),p(e(X,Y))-->p(Y).
      p(e(e(e(a,e(b,c)),c),e(b,a)))-->false.
      true-->p(e(A,e(e(B,e(C,A)),e(C,B)))).
```

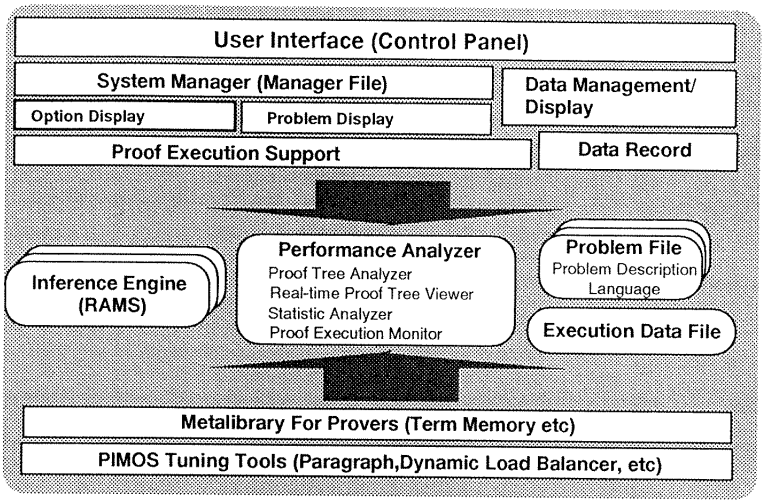
## Two Versions of MGTP:

| | Ground MGTP (MGTP/G) | Non-Ground MGTP (MGTP/N) |
|---|---|---|
| **Problem** | Non-Horn and ground | Horn and non-ground |
| **Application** | E.g. Database problems | E.g. Mathematical theorems |
| **Programming Techniques** | -Direct use of KL1 variables<br>-Translation given clauses to KL1 clauses<br>-Efficient coding using head unification of KL1 | -Representing variables with ground terms<br>-Interpreting a given set of clauses<br>-Using"meta-library" E.g. Unification with occur check |
| **Parallelization** | AND/OR parallel | AND parallel |

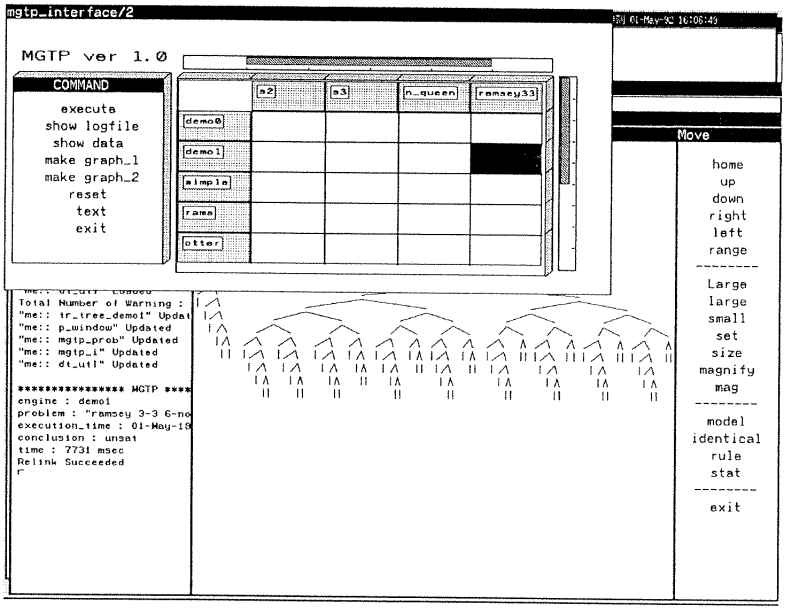
## Key Technologies to improve Efficiency:

| Problem | Technology |
|---|---|
| Redundancy in Conjunctive Matching | RAMS(RamifiedStack)<br>MERC(Multi-Entry Repeated Combination) |
| Irrelevant Clauses | Partial falsify relevancy test |
| Over Generation of Models | Lazy model generation |
| Parallelism | OR parallelization for non-Horn problems<br>AND parallelization(Model distributing / sharing ) for Horn problems |
| Unification/ Subsumption | Clause compiling technique<br>Term indexing memory |

## MGTP Development Supporting Environment




System Configuration




Control Panel and Proof Tree Analyzer

### Parallelization of MGTP

There are several resources for parallelization of MGTP:

- case splitting
- conjunctive matching in the antecedent part
- subsumption test

We focused on OR parallelization in case splitting and on AND parallelization in conjunctive matching and subsumption test.

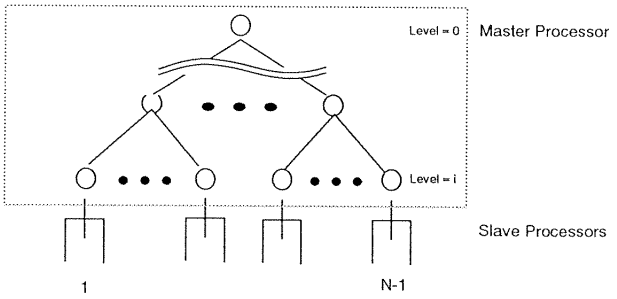
### OR Parallelization

For non-Horn and ground problems, it is sufficient to exploit OR parallelization induced by case splitting. We implemented an OR parallel version of MGTP based on the MERC method.

The processor allocation methods we have adopted achieve 'bounded-OR' parallelization in the sense that OR parallel forking in the proving process is suppressed so as to meet restricted resource circumstances. To do this, we adopted a scheme, called *level restricted allocation.*

We expanded model candidates, starting with an empty model, using a single master processor until the number of candidates exceeded the number of available processors. We, then, distributed the remaining tasks to slave processors. Each slave processor explored the branches assigned without further distributing tasks to any other processors.

This allocation scheme for task distribution works fairly well, since the communication cost can be minimized.



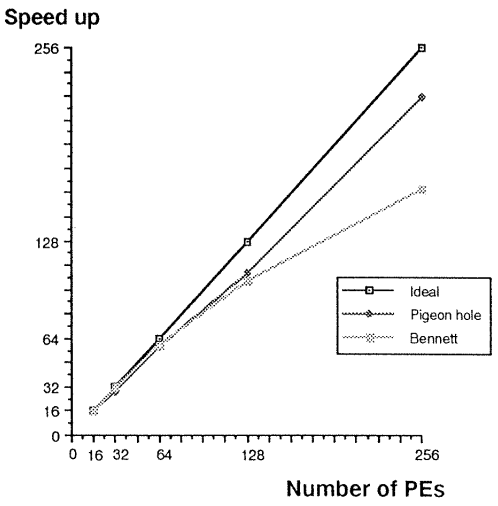

**Level restricted allocation**

**Pigeon hole problem**

$true \rightarrow pigeon(1), pigeon(2), pigeon(3), pigeon(4), pigeon(5),$
$pigeon(6), pigeon(7), pigeon(8), pigeon(9), pigeon(10), pigeon(11).$
$pigeon(M) \rightarrow hole(M,1); hole(M,2); hole(M,3); hole(M,4); hole(M,5);$
$hole(M,6); hole(M,7); hole(M,8); hole(M,9); hole(M,10).$
$hole(M1,N), hole(M2,N), M1 \neq M2 \rightarrow false.$

**Bennett's quasigroups**

$true \rightarrow dom(1), dom(2), dom(3), dom(4), dom(5),$
$dom(6), dom(7), dom(8), dom(9), dom(10), dom(11).$
$dom(M), dom(N) \rightarrow$
$p(M,N,1); p(M,N,2); p(M,N,3); p(M,N,4); p(M,N,5); p(M,N,6);$
$p(M,N,7); p(M,N,8); p(M,N,9); p(M,N,10); p(M,N,11).$
$p(X,11,Y), Y+1 < X \rightarrow false.$
$p(E,X,Y), p(Y,E,Z), p(Z,E,U), X \neq U \rightarrow false.$
$p(X,X,U), X \neq U \rightarrow false.$
$p(X,Y,U), p(X,Y1,U), Y \neq Y1 \rightarrow false.$
$p(X,Y,U), p(X1,Y,U), X \neq X1 \rightarrow false.$




MGTP/G on PIM/m

### AND Parallelization

AND parallelization for Horn problems is achieved by exploiting parallelisms inherent in conjunctive matchings and subsumption tests. We implemented an AND parallel version of MGTP based on lazy model generation. The system adopts the following schemas: the proof unchanging schema according to the number of PEs, the model sharing schema (copying in a distributed memory architecture), and the master-slave schema.

**Proof Unchanging:** Our policy in developing parallel theorem provers is to distinguish between the speedup effect caused by parallelization and the search-pruning effect caused by strategies. A proof-changing prover may achieve super-linear speedup whereas it may cause the strategy to be changed. On the other hand, a proof-unchanging prover allows us to obtain greater speedup as the number of PEs increases, without changing the strategy.

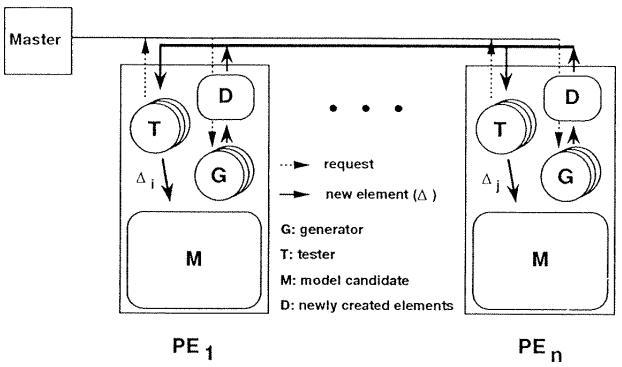
**Model Sharing:** The merit of model sharing is that time-consuming subsumption testing and conjunctive matching can be performed at each PE independently, with minimal inter-PE communication.

**Master-Slave:** The master-slave configuration makes it easy to build a parallel theorem prover by simply connecting a sequential version of MGTP on a slave PE to the master PE. Since slave processes spontaneously obtain tasks from the master, and the size of each task is equalized, good load balancing is achieved.

In this system, generator and subsumption processes run in a demand-driven mode, while tester processes run in a data-driven mode.

The main factor in the degradation of system performance is sequentiality in subsumption testing. This can be minimized by utilizing the synchronization mechanism supported by KL1. Demand-driven control can also be easily and efficiently implemented by utilizing the KL1 stream.

By using demand-driven control, we can not only suppress unnecessary model extensions and subsumption tests but also maintain the high running rate which is the key to achieving linear speedup.




**Load allocation for MGTP/N**

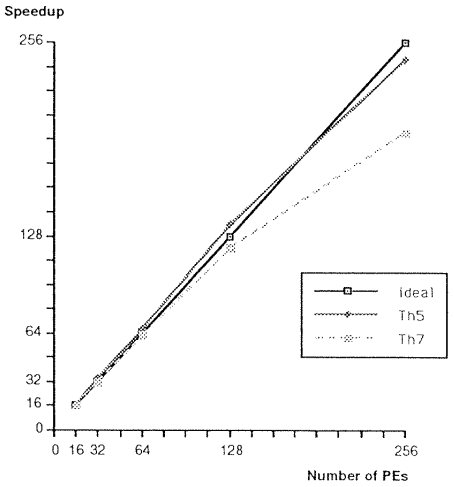
### Theorem 5

$true \rightarrow p(i(i(i(X,Y),Z),i(i(Z,X),i(U,X)))).$
$p(X), p(i(X,Y)) \rightarrow p(Y).$
$p(i(i(a,b),i(i(b,c),i(a,c)))) \rightarrow false.$

### Theorem 7

$true \rightarrow p(i(X,i(Y,X))).$
$true \rightarrow p(i(i(n(X),n(Y)),i(Y,X))).$
$p(X), p(i(X,Y)) \rightarrow p(Y).$
$p(i(i(a,b),i(n(b),b(a)))) \rightarrow false.$

$true \rightarrow p(i(i(X,Y),i(i(Y,Z),i(X,Z)))).$
$true \rightarrow p(i(i(i(X,Y),Y),i(i(Y,X),X)))$




MGTP/N on PIM/m

## MGTP Applications (1)
## Abductive Reasoning System


### Abstract

We have developed several parallel abductive reasoning systems using the MGTP. We will describe two implementations of the abductive reasoning system shown in Figure 1: one is the *MGTP+MGTP* method, and the other is the *Skip* method. In each method, the given input formulas are translated into MGTP rules in a different way. We demonstrated them by applying them to a logic circuit design problem.

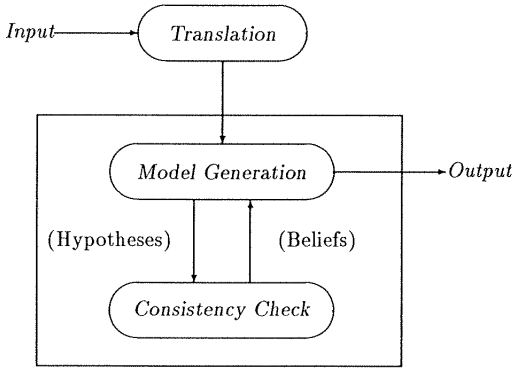




Figure 1: Abductive Reasoning System based on the MGTP

### Abduction

We considered the first-order abductive framework $(\Sigma, \Gamma)$, where $\Sigma$ is a set of Horn clauses and $\Gamma$ is a set of atoms (*abducibles*). A set $E$ of ground instances of elements of $\Gamma$ is an *explanation* of a closed formula $G$ from $(\Sigma, \Gamma)$ if: (1) $\Sigma \cup E \models G$, and (2) $\Sigma \cup E$ is consistent. Given $\Sigma$, $\Gamma$ and $G$, the task of abduction is to find the explanations of $G$ from $(\Sigma, \Gamma)$.

### MGTP+MGTP

Each ground hypothesis $H$ from $\Gamma$ is represented by $fact(H, \{H\})$, and each Horn clause in $\Sigma$ of the form:

$$A_1 \wedge \ldots \wedge A_n \supset C\,,$$

is translated into the MGTP rule of the form:

$$fact(A_1, E_1),\ \ldots,\ fact(A_n, E_n) \rightarrow fact(C,\ \mathrm{cc}(\bigcup_{i=1}^{n} E_i))\,,$$

where $E_i$ is a set of ground hypotheses from $\Gamma$ on which $A_i$ depends, and the function cc is defined as:

$$\mathsf{cc}(E) = \begin{cases} E & \text{if } \Sigma \cup E \text{ is consistent;} \\ \mathsf{nil} & \text{otherwise.} \end{cases}$$

Each time MGTP-1 derives a new ground atom, the consistency of the combined hypotheses is checked by MGTP-2 (see Figure 2). The parallelism comes from calling multiple MGTP-2s at once.

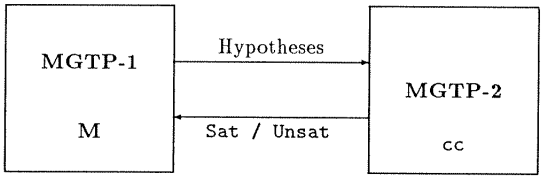



Figure 2: MGTP+MGTP

### Skip Method

When a clause in $\Sigma$ contains the negative occurrences of abducibles $H_1, \ldots, H_m$ ($H_i \in \Gamma$, $m \geq 0$) and is in the form:

$$A_1 \wedge \ldots \wedge A_l \wedge \underbrace{H_1 \wedge \ldots \wedge H_m}_{abducibles} \supset C\,,$$

we translate it into the following MGTP rule:

$$A_1, \ldots, A_l \rightarrow H_1, \ldots, H_m, C \mid \neg \mathsf{K} H_1 \mid \ldots \mid \neg \mathsf{K} H_m\,.$$

In this translation, each hypothesis in the premise part is *skipped* instead of being resolved, and is moved to the right-hand side. A model candidate containing both $H$ and $\neg \mathsf{K} H$ is rejected by the schema:

$$\neg \mathsf{K} H,\ H \rightarrow \qquad \text{for every hypothesis } H\,.$$

This method utilizes the extension and rejection of model candidates supplied by the MGTP. An OR-parallelism can be obtained in the way that multiple model candidates are kept in distributed memories. In order to avoid possible combinatorial explosion in constructing model candidates for the skip method, we also showed a way to "cut" model candidates that cannot contribute to providing solutions.

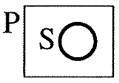
## MGTP Applications (2)
## Protocol Specification Description System


### Abstract

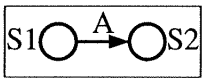
In order to describe a complex protocol specification easily, we proposed a protocol specification description language, Ack. We developed a processing system for this language using the parallel theorem proving technique, and applied it to the description of several services for a telephone switching system.


### Ack Notation

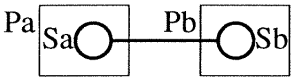
- Atomic Formulas



: The state of process P is S.

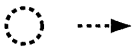


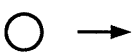
: The state of process P transits from S1 to S2 by action A.



: At the same time, the states of processes Pa and Pb are Sa and Sb, respectively.

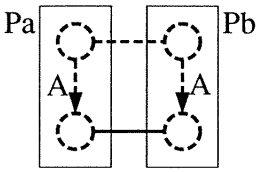
- Clause $(A_1 \wedge \cdots \wedge A_m \rightarrow B_1 \wedge \cdots \wedge B_n)$

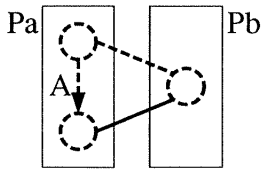

: Antecedent $(A_1, \cdots, A_m)$


: Consequent $(B_1, \cdots, B_n)$

### Axioms for Ack

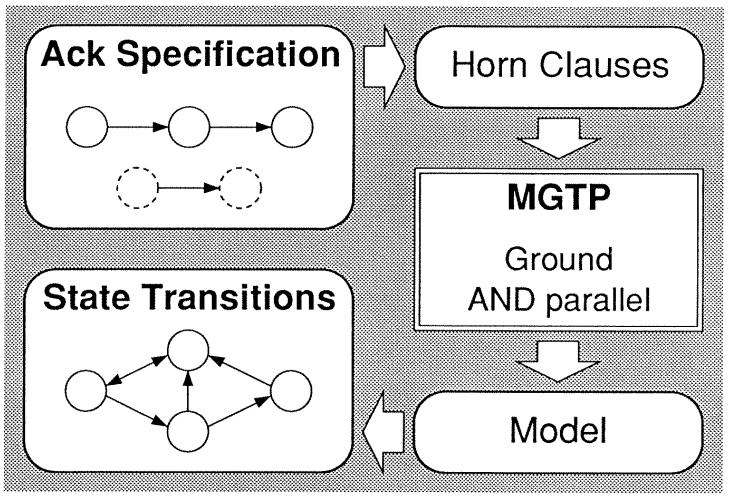
- Synchronization




where A $\notin$ Action(Pb)

### Synthesis of the State Transitions




### Example of Description and Synthesis