2F-6

推論向き高並列データフローマシンとそのユニフィケーション機能

後藤 厚宏　田中 英彦　元岡 達
(東京大学　工学部)

1. Introduction

Knowledge information processing, especially inference processing, will be a main requirement of computer ability in near future. In this paper, we describe a basic architecture of highly parallel inference system based on parallel processing of Prolog.

2. Prolog and Proof Diagram

A Prolog program is a set of definite clauses (Horn clauses). For a given program D and a goal statement G, a derivation is a sequence of goal statements Notice that often a given derivation can be extended in several different ways. The set of all derivations starting at G can be arranged in the form of a tree of goal statements and called a "search tree". A "proof tree (PT)" is a data structure which stores the derivating sequence from a root node to one node in a search tree.

(1) Proof Diagram　We introduce Proof Diagram (PD). PD is analogous to Ferguson Diagram [Emd81a] but PD can more clearly represent a construction of PT in problem-solving.

```
[ example 1 ]   D1    app[ nil,*x,*x ] <-.
                D2    app[ (*u.*x),*y,(*u.*z) ] <- app[ *x,*y,*z ].
                D3    subl[ *x,*y ] <- app[ *u,*x,*v ] app[ *v,*w,*y ]
                G     <- subl[ (a.*x),(*y.nil) ]
```

A goal and each definition clause of example 1 is represented by a template. Fig.1 shows an active PT consisting of four templates. A unification is represented by an upper half circle meeting a lower half circle. (U1~U3 in Fig.1) This diagram shows explicitly the unifications and the structures borrowed from the program.

Program execution in Prolog is to construct PTs. Major distinction between Prolog and ordinary languages is that the sequence of problem-solving is given in a deterministic way or not. In Prolog, each procedure call is non-deterministic and a lot of PTs are tried, many of them fail and some succeed.

(2) Unification　If alternatives of PTs are derived in parallel, each PT can be reduced. Fig.2 shows a reduced PT of Fig.1. Therefore each unification has two roles, one is pattern matching between a call and a heading and the other is reducing a derived PT. This unification is rather complicated, but can be done in parallel if variables are linear.

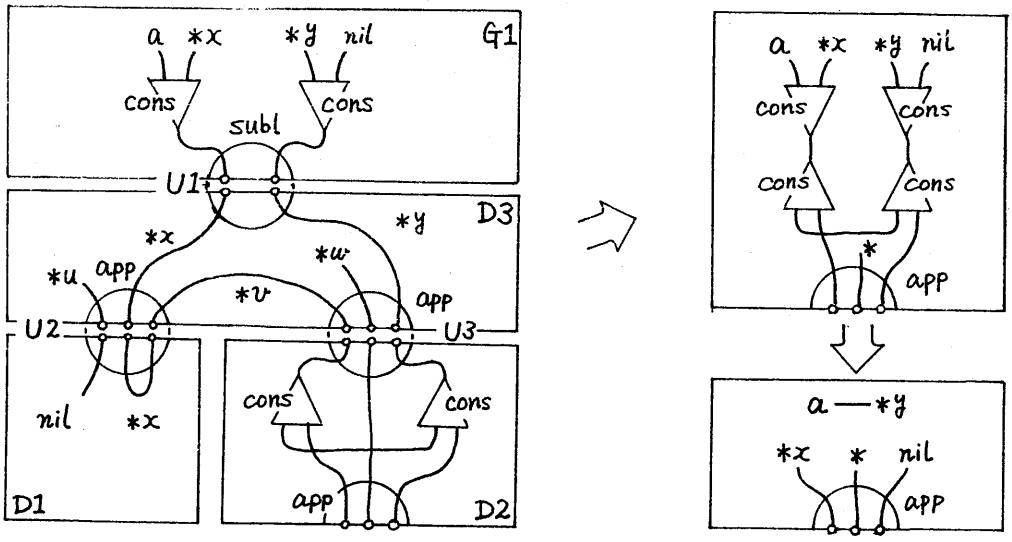

Fig.1 Proof Tree represented by Proof diagram

Fig.2 Reduced Proof Tree

3. Parallel Processing and its Control in Prototype Machine

As seen before there are three parallelisms in the execution of Prolog.

P1. Parallelism on deriving alternatives of PTs (OR-parallelism)
P2. Parallelism on unifying active calls in a PT (AND-parallelism)
P3. Parallelism on unifying arguments

Prototype Machine (Fig.3) is based on P1 and P3 parallelism. Processing elements are Unify Processors (UP). Each UP inputs a PT, unifies one call with all alternatives of its definition clauses and outputs several new PTs. Other components of this machine are Goal Pool (GP) and Activity Controller (AC).

(1) Processor allocation & inactivation of active PTs It is natural to suppose that the number of active PTs is larger than the number of UPs, because P1 parallelism is so enormous. In this machine, active PTs are allocated to the UPs according to not only the specification in a program but the priority of PTs. The priority of a PT is higher if the number of calls is fewer than others or just decreased.

When one of PTs succeeds and other alternatives become needless, it is necessary to inactivate those PTs. For this reason, each active PT has PT-name representing the relationship among active PTs.

The entities of active PTs are stored in Goal Pool but their control information (priority and PT-name)are represented by tokens. Tokens are managed by Activity Controller and used for allocating and inactivating PTs.

Tokens flow out from Activity Controller to Token Loop. Each token carries a pointer to the entity of a PT. A UP catches one token, and starts unification processing. This UP outputs the entities of new PTs to Goal Pool and simultaneously outputs the corresponding tokens to Token Loop.

(2) Selection of calls in UPs If there are no explicit control of active calls in an allocated PT (ex. mode declaration of variables, evaluable predicates, etc.), each UP can select a unified call to minimize the output PTs. Therefore, a call whose definitions are facts or deterministic (no bodies) is selected first in each UP.

4. Future Research

The next step in our research is to build simulators for this model. We are planning to use the data flow machine TOPSTAR [Aid82a] for simulation.

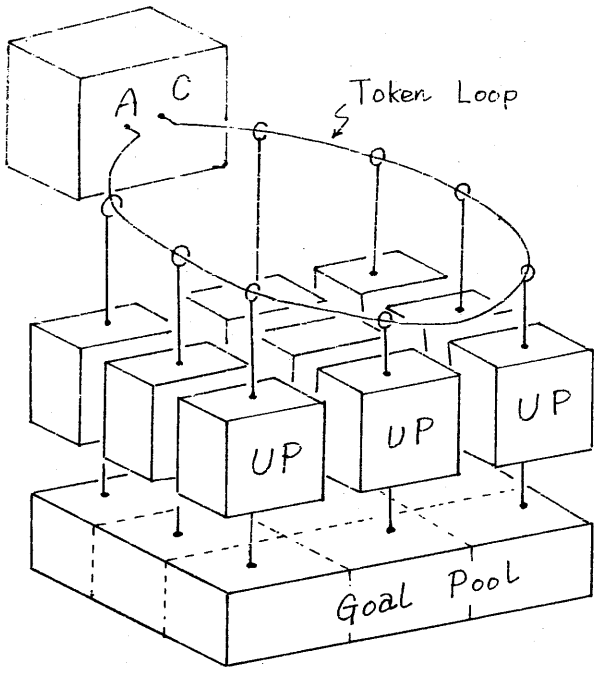

Fig.3 Prototype Machine

References

Emd81a. M.H. van Emden, "An Algorithm for Interpreting PROLOG Programs," RRCS, (Sept. 1981).

Aid82a. 相田 他、『並列PROLOGシステム "Paralog" の性能測定』、第24回情処全大