This paper presents the design and implementation of logic language interface on conventional network database systems which provide network data structure and navigational data manipulation language. This interface provides users with a unified view of various data structures in logics and inference facilities of theorem proving. Although most researchers try to combine logic systems like Prolog and relational database systems, we try to augment the conventional network database system with logic interface, since the network ones have been already mainly used in database applications. This also can be used as a common interface on heterogeneous database systems in distributed database systems. We show a refutation procedure which can reduce the search space by avoiding meaningless backtracking and redundant refutations.

1. Introduction

Recent computer systems provide two types of database management systems, relational (CODD) and network (CODA, OLLE) ones, which play a central role in most computer applications. The relational ones provide non-procedural data manipulation languages (DMLs) like SQL (DATE) on logical data structures and are used for ad hoc applications like CAD. The network ones provide navigational DMLs (OLLE) on the mixture of logical and physical data structures, and have been used for well-formed business applications which require high performance. At present, high level interfaces on the database systems are required. Relational interface systems which can provide non-procedural languages on the conventional network database systems have been realized by [TAKI80, DAYA].

Logic languages like Prolog (Kowa) have advantages to the relational model: unified and recursive treatment of data structure, procedures, queries and integrity constraints. Although most researchers (CHAN, LID, OHSU etc.) try to combine the relational model and logic languages, we discuss a pure Prolog interface on the network database system, because they have been mainly used in database applications and resolution is based on a tuple-at-a-time, navigational access like the DML (OLLE). In our refutation procedure, record occurrences are accessed according to sequences implemented in the network database system. Also, we try to reduce the search space in the resolution by avoiding meaningless backtracking and redundant refutations.

In chapter 2 and 3, we define a formal system of the network database system and semantics of the conventional DML. In chapter 4, 5 and 6, we discuss how to combine a refutation procedure and navigational data manipulation on the network database.

2. Navigational Database System

First, a navigational model is defined by extracting an essential, logical part of the
conventional network model [CODA73, 78, OLLE].

2.1 Network Data Structure

A network data structure $N$ is composed of a schema $N$, and a database $N$. $N$ is composed of two kinds of types, entity (E-) and function (F-) types. The E-type $E$ is a tuple of attributes $(\Omega E, A_1, \ldots, A_m)$. $\Omega E$ is an identifier and $A_i (i=1, \ldots, m)$ a data attribute. For an attribute $T$ of $E$, let $dom(T)$ denote a domain of $T$. An E-set for $E$ is $E \subseteq dom(\Omega E) \times dom(A_1) \times \ldots \times dom(A_m)$. Each element $e$ in $E$ is an E-tuple. Let $T(e)$ denote $T$'s value of $e$. Every E-tuple has different identifier.

If there exists a functional relation from an E-type $E_1$ to $E_2$, an F-type $F$ is defined as $F = (E_1, E_2)$. $E_1$ is the domain and $E_2$ the range. An F-set for $F$ is $F \subseteq E_1 \times E_2$. $F$ represents a partial function $F: E_1 \rightarrow E_2$.

A family of the E-sets and F-sets is a network database $N$. $N$ is represented by a graph whose nodes denote E-types and directed arcs F-types. Fig. 1 shows the network schema which represents information on departments (Dep), employees (Emp), and projects (Proj). For example, an arc DE from Emp to Dep indicates an F-type $DE = (\text{Emp}, \text{Dep})$.

**Fig. 1 Network Schema.**

2.2 Navigational Data Structure and Access Functions

A navigational data structure $V$ is defined by introducing ordering relations on the network data structure $N$. For an E-set $E$ and an attribute $T$ of $E$, an ordered E- (OE-) set $E/T$ is a totally ordered set (tos) $(E, <)$ where for every $e$ and $f$ in $E$, $e < f$ in $E/T$ if and only if (iff) $T(e) < T(f)$. $e << f$ (f is an immediate successor of e) iff $e < f$ and no $g$ in $E$ such that $e < g < f$.

$E^T$ is a partially ordered set (pos) $(E, <)$ where for every $e$, $f$ in $E$, $e < f$ in $E^T$ iff $T(e) < T(f)$ and $\Omega E(e) < \Omega E(f)$. $E^T$ is partitioned into a tos $E^T[v] = \{e | e \in E$ and $T(e) = v, <\}$ for every $v$ in $dom(v)$. $e << f$ in $E^T$ is defined like $E/T$.

For an F-set $F$ of $F = (E_1, E_2)$, some attribute $T$ of $E_1$, and $e_2$ in $E_2$, $F^T$ is a pos $(F, <)$ where for $f = e, g$ and $f' = e', g$ in $F$, $f < f'$ in $F^T$ iff $T(e) < T(e')$. $F^T$ is also partitioned into a tos $F^T[e_2] = \{e | e \in E_2$ and $T(e) = e_2, <\}$ for every $e_2$ in $E_2$. $e << f$ in $F^T$ is defined like $E/T$.

Here, $E/\Omega E$, $E^T$, and $F^T$ represent conventional record-type, CALC item $T$, and set-type whose sort key is $T$ [CODA73], respectively.

There are following access functions for $V$. Here, $E'$ is a set $E \cup \{\} \ (\cdot)$ means "undefined".

1) $topE/T : \rightarrow E$, $lastE/T : \rightarrow E$. $topE/T()$ (lastE/T()) gives the greatest (least) element in $E/T$.
2) $nextE/T : E \rightarrow E'$, $priorE/T : E \rightarrow E'$. For every $e$ in $E$, $nextE/T(e)$ (priorE/T(e)) gives $f$ such that $e << f$ (f <<< e) in E/T. Here, nextE/T(lastE/T()) = \perp and priorE/T(topE/T()) = \perp.
3) $dtopE^T : dom(T) \rightarrow E'$, $dlastE^T : dom(T) \rightarrow E'$. For every $v$ in dom(T), $dtopE^T(v)$ (dlastE^T(v)) gives the greatest (least) one in $E^T[v]$.