NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
TEMPORAL OBJECT-ORIENTED DATABASE: DATA MODEL, FORMALISM AND IMPLEMENTATIONS

Yan Zhen Qu

A Thesis in The Department of Computer Science

Presented in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy at Concordia University Montréal, Québec, Canada

June 1990

© Yan Zhen Qu, 1990
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-64755-8
ABSTRACT

TEMPORAL OBJECT-ORIENTED DATABASE: DATA MODEL, FORMALISM AND IMPLEMENTATIONS

Yan Zhen Qu, Ph.D.
Concordia University, 1990

This dissertation presents a data model for a temporal object-oriented database (TOODB) called the dynamic state machine model. The whole life of a real world entity is called a temporal object. In the dynamic state machine model, a temporal object is modeled by a multitype composite machine which consists of a set of primary machines, each of which represents a partial history of the temporal object when it is associated with an environment that is described by a type-version. Dynamic multityping, dynamic reference and dynamic extension mechanisms are adopted to model temporal object evolution. Three time notions (commit time, effective time and observation time) are used to capture different time semantics. Behavior constraints are an important modeling construct which enrich the temporal modeling power of the dynamic state machine model. Retroactive update affection identification is also addressed.
A linear time temporal logic, called dynamic state logic (DSL), and its proof system are developed. By using the dynamic state machine model as the model of this logic language, the behavior constraints on temporal objects can be specified naturally and precisely. The proof system of the DSL makes it possible to verify whether the given requirements or the expected properties can be derived from the behavior constraints defined in type-versions. A set of algorithms are presented, which efficiently check whether a TOODB satisfies behavior constraints specified by a certain class of temporal formulas of the DSL.

As for implementation issues, this dissertation concentrates on designing a suitable memory management system for the TOODB. A paged virtual memory management system has been designed, which has a two level secondary storage structure and is equipped with a scheme (called temporal clustering) for clustering temporal objects. Unlike most existing design methodologies, in addition to the features of data organization, possible user access patterns are also considered. An analysis model is developed to find the optimal design. Based on the analyses, a set of efficient algorithms to optimize parameters are developed. By changing the values of the parameters in the analysis model, the results of a series of experiments are described, which are helpful in understanding characteristics of clustering temporal objects. Based on these experimental results, some suggestions for efficiently applying the temporal clustering scheme are presented.
ACKNOWLEDGEMENTS

I am sincerely grateful to my two supervisors Dr. F. Sadri and Dr. P. Goyal for their guidance, advice and encouragement. From them I have learned and improved my skills in developing, presenting, and analyzing new ideas in a very competitive field. I feel extremely fortunate to have had the opportunity of working with them and sharing their experience and insight. They also aided and abetted most of my other adventures as a graduate student. The combination of freedom and unhesitating support they provided was invaluable.

Dr. M. Okada, one of my committee members, taught me how to apply the temporal logic correctly. He also provided insight and encouragement in many useful discussions. The formalism that I presented in the dissertation would not have been possible without his guidance.

I thank Dr. B. Gopinath of Rutgers University, Dr. H.F. Li and Dr. V.S. Alagar of Computer Science Department, and Dr. A.K. Elhakeem of the Computer Engineering and Electrical Engineering Department for their suggestions and comments.

My doctoral studies at Concordia University were made possible through the Concordia University Graduate Fellowship and the research grants of NSERC for which I am very grateful.

Special thanks go to my parents and wife, who have constantly given me support during the period of my studies.
Dedicated to

My Family
Contents

1 INTRODUCTION 1

1.1 THE DEFICIENCY OF TRADITIONAL OBJECT-ORIENTED DATA MODELS 1

1.2 TOODB: MODELING ISSUES 3

1.3 TOODB: IMPLEMENTATION ISSUES 7

1.4 THESIS OVERVIEW 11

2 RELATED WORK 14

2.1 MODELING TIME IN DATABASES 14

2.2 MODELING OBJECT EVOLUTION 15

2.2.1 GemStone 15

2.2.2 Iris 15

2.2.3 ENCORE 16

2.2.4 UCB Version Data Model 17

2.2.5 ORION 17

2.2.6 AVANCE 18

2.3 TEMPORAL LOGIC 19
2.4 IMPLEMENTATION OF TEMPORAL RELATIONAL DATABASE 21
2.5 OBJECT-BASED CLUSTERING ..................................... 22
2.6 OTHER RELATED WORK ........................................... 23

3 DYNAMIC STATE MACHINE MODEL 24
  3.1 INTRODUCTION .................................................. 24
  3.2 CONSTRUCTS OF THE DYNAMIC STATE MACHINE MODEL .................................................. 25
      3.2.1 Primary Machine ............................................ 25
      3.2.2 Multitype Composite Machine ............................... 35
      3.2.3 General Composite Machine ............................... 39
      3.2.4 Relationship between a $\delta_M$ and a $\delta_m$ ............... 41
      3.2.5 Integrity Rules ............................................. 41
  3.3 MODELING TEMPORAL OBJECTS BY DYNAMIC STATE MACHINES ............................................. 43
      3.3.1 Environment Modeling ...................................... 43
      3.3.2 Dynamic Multityping ...................................... 44
      3.3.3 Dynamic Extension ...................................... 46
      3.3.4 Dynamic References ...................................... 46
      3.3.5 Trace and Behavior History ................................. 48
      3.3.6 Retroactive Affection Identification ...................... 48
      3.3.7 Behavior Constraints ...................................... 50

4 DYNAMIC STATE LOGIC 53
4.1 INTRODUCTION ........................................ 53
4.2 DYNAMIC STATE LOGIC LANGUAGE ............... 54
4.3 SEMANTICS OF DSL ................................. 56
4.4 PROOF SYSTEM OF DSL ............................. 60

5 BEHAVIOR CONSTRAINTS ............................... 74
   5.1 INTRODUCTION .................................... 74
   5.2 SPECIFYING BEHAVIOR CONSTRAINTS BY THE DSL . . 75
   5.3 VERIFYING THE EXPECTED SYSTEM PROPERTIES ....... 78
   5.4 CHECKING BEHAVIOR CONSTRAINTS ................. 86

6 A DESIGN OF TOODB MEMORY MANAGEMENT .......... 92
   6.1 INTRODUCTION .................................... 92
   6.2 DATA ORGANIZATION .............................. 93
      6.2.1 Basic Data Unit ................................ 93
      6.2.2 Two-Level Storage Structure .................... 95
      6.2.3 Clustering Scheme .............................. 102
   6.3 USERS' ACCESS PATTERNS ......................... 106
      6.3.1 Primitive Transactions ......................... 106
      6.3.2 Processing Primary Temporal Queries ............. 108
      6.3.3 Relative Frequencies of PTs ..................... 109
   6.4 AN ANALYSIS MODEL ............................... 110
      6.4.1 Parameters of a Node ........................... 110
6.4.2 Average Distances Between History Records .......... 111
6.4.3 Average Page Access Number Expression .......... 114

7 OPTIMAL PARTITION .......... 116

7.1 INTRODUCTION .......... 116
7.2 DEFINITIONS OF SOME CONCEPTS .......... 117
7.3 ANALYSIS UNDER C = DEPTH-FIRST .......... 122
7.4 LOCAL OPTIMIZATION ALGORITHM .......... 127
7.5 ALGORITHM OF GLOBAL OPTIMIZATION .......... 132

8 EXPERIMENT RESULTS .......... 135

8.1 INTRODUCTION .......... 135
8.2 EXPERIMENT PARAMETERS .......... 136
8.3 CHARACTERISTICS OF A FIXED $P^*$ .......... 140
  8.3.1 Effect of the RP-Ratio .......... 141
  8.3.2 Effect of $\sum f_{it}/\sum f_{et}$ .......... 142
  8.3.3 Effect of the $z^*(H)$ .......... 144
8.4 CHARACTERISTICS OF THE OPTIMAL PARTITION .......... 147
  8.4.1 Characteristics of MiniR .......... 148
  8.4.2 Characteristics of MiniminiR .......... 151
  8.4.3 Effect of Changing $L$ .......... 154
  8.4.4 Characteristics of $P_{opt}$ .......... 155
  8.4.5 Effect of the Clustering Sequence .......... 158
List of Figures

1.1 Application of Time Notions ............................................. 5

5.1 Type Hierarchy of Example 5.2 ........................................... 77
5.2 Procedure of Product Manufacture ................................. 80
5.3 Type-version Hierarchy of Example 5.3 ............................. 80

6.1 The Public Object Base .................................................. 98
6.2 A Primary Machine Tree Schema .................................... 99
6.3 Examples of Some Concepts .......................................... 104

7.1 Examples of Definitions ............................................... 119
7.2 Construct a $PMT^q$ ...................................................... 122

8.1 Effect of RP-ratio on a $P^q$ ........................................... 140
8.2 Effect of $\sum f_{it}/\sum f_{et}$ on a $P^q - (1)$ ............... 143
8.3 Effect of $\sum f_{it}/\sum f_{et}$ on a $P^q - (2)$ ............... 144
8.4 Effect of $\sum f_{it}/\sum f_{et}$ on a $P^q - (3)$ ............... 145
8.5 Effect of $x^q(H)$ on a $P^q$ ........................................... 146
8.6 Effect of RP-ratio on MiniR .......................................... 149
8.7 Effect of $\sum f_{it}/\sum f_{et}$ on MiniR ............................ 150
8.8 Effect of $z^{s^k}(H)$ on MiniR ................................. 151
8.9 Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on MiniminiR ............. 153
8.10 Effect of $z^{s^k}(H)$ and on MiniminiR .......................... 153
8.11 Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on MaxminiR/MiniminiR .... 155
8.12 Effect of $z^{s^k}(H)$ and on MaxminiR/MiniminiR .................. 156
8.13 Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on 0.03B ................. 157
8.14 Effect of $z^{s^k}(H)$ and on L of MiniminiR ......................... 157
8.15 Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on $N_c$ of $P_{opt}$ ........ 159
8.16 Effect of $z^{s^k}(H)$ and on $N_c$ of $P_{opt}$ .......................... 159
8.17 Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on $P_{opt}$-LBand ........... 160
8.18 Effect of $z^{s^k}(H)$ and on $P_{opt}$-LBand .......................... 160
8.19 Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on Minimini$R_{brd}$ - Minimini$R_{dep}$ . 161
8.20 Effect of $z^{s^k}(H)$ and on Minimini$R_{brd}$ - Minimini$R_{dep}$ .......... 162
List of Tables

8.1 Parameters Used in Experiments .................................. 137
8.2 Three Types of RP-ratios ........................................... 137
8.3 $\Sigma f_{it}/\Sigma f_{et}$'s of Six Types of PT's .......................... 138
8.4 Three Types of RP-ratios ........................................... 139
8.5 Value Ranges of Parameters ....................................... 139
Chapter 1

INTRODUCTION

1.1 THE DEFICIENCY OF TRADITIONAL OBJECT-ORIENTED DATA MODELS

Object-oriented models recently have come of age with an increasing availability of object-oriented languages [BKKMSZ86, Co87, GR83, Me88] and database management systems [Ad85, B+87, CM84, Di86, EE87, MSP86]. The object-oriented models support inheritance and encapsulation allowing modeling of complex real world entities. As a result they have been widely applied in many non-traditional applications such as VLSI, multimedia data, office automation, CAD/CAM, and software engineering [CM84, B+87, BK85, Di86, LH88, PPT88, ZABCKM86]. They, nevertheless, lack some of the semantics that had been incorporated in record-based systems [CW83, SA86, Sn86]. The most significant deficiency of the traditional object-oriented models is the absence of temporal modeling ability. However, the applicability of the object model to a large number of application domains, for example, scheduling, project management, process planning, system controlling, etc., requires the ability to model general temporal relationships between objects [BK85, La87, LH88, KCB86, PPT88].
The last few years have witnessed a growing research interest in the object evolution. Most of work in this area have concentrated on object schema evolution [BKKK87, GOQS89, KC88, PS87, SZ86, SZ86, Zd86]. Existing models can be classified into two categories: snapshot and versioning. While snapshot models permit some changes on the definition of object class schema, they only maintain the last updated definition, and all object instances created under the old definition have to be transformed to conform to the new definition. Versioning models apply versions to record the changes on objects (either the schema or the instance). The following characteristics are common to most of existing versioning models:

- **an absence of a concept of time**: versions rather than the notion of time are emphasized, so that more general temporal relationships among the objects cannot be modeled;

- **state-orientation**: every version only contains “attribute values”, called states, there is no way to find why and how an object gets the state;

- **static typing**: if an object is an instance of a class (type), then it will be bound the type until it is deleted; the temporal semantics of the association between an object and a type or types is missed.

- **lacking behavior constraints**: constraints on the temporal relationships among object operations are not adopted as a modeling construct; this leaves a big semantic gap between the data model and the real world.
1.2 TOODB: MODELING ISSUES

We call an object-oriented database whose data model integrates the general temporal modeling capabilities a temporal object-oriented database (TOODB). The following issues must be captured by the data model of a TOODB.

The Notion of Time

Three notions of time have to be identified in the TOODB: commit time, effective time, and observation time. A query is an execution of an object operation (i.e., method) which retrieves information about the state of the object (i.e., attribute values), while an update is an execution of an object operation which modifies the state of the object. Commit time is the time instant when an object operation successfully completes (commits) an execution in the database. The effective time specified in an update is the time instant since then the data (i.e., the updated state of the object) made by the update takes effect (i.e., valid). It can be different from the commit time. Based on their commit times and effective times, updates can be classified into three categories: retroactive, normal, and post-dated. A retroactive update has an effective time which is less than its commit time. A normal update has an effective time which is equal to its commit time. A post-dated update has an effective time which is greater than its commit time. A query in a TOODB is called a temporal query, which has a temporal construct: the effective time and the observation time. A query without the temporal construct is considered a special case of temporal query. If the effective time and/or the observation time are not declared
in the query, they will be considered to have a default value which is the current time. Only those data with effective times matched with the effective time specified in the query are retrieved. Observation time is the time instant specified in a query at which the data will be retrieved with respect to the effective time specified in the query. Because a retroactive update may invalidate the data of a previous update, different answers may be given to the same query question at different observation times. These concepts can be demonstrated by the following example.

Example 1.1:

As shown in Figure 1.1, there is a set of data made by a sequence of updates. We assume that $d_i$ is the data made by the $i$-th update. In this example, $d_1$, $d_2$, $d_3$ and $d_8$ are data made by post-dated updates, since their effective times are greater than their commit times; $d_5$, $d_7$ and $d_9$ are data made by retroactive updates, since their effective times are less than their commit times; and $d_4$, $d_6$ and $d_{10}$ are data made by normal updates, since their effective times are equal to their commit times. We also assume that the $5$-th and the $7$-th updates invalidate the $1$st and the $2$nd updates respectively.

If the query "what are the valid data in the database at time $t_7$?" is associated with an observation time $t_8$, then the answer is "$d_2$, $d_4$, $d_5$ and $d_6$". If the observation time is changed as time $t_9$, for the same query, the answer out of the database will be "$d_4$, $d_5$, $d_6$ and $d_7$".
Features of Temporal Objects

We define a term *temporal object* to represent the whole life of an entity in the real world. In addition to the features modeled by the traditional OODB, the following new features must be modeled by temporal objects.

*Changing Environments* - We use a term *environment* to represent a set of structural and/or temporal relationships by which a group of entities are associated with each other at a certain (physical or conceptual) place and during a certain time period. An entity may migrate from one environment to another dynamically. The properties and behaviors of an entity and the relationships between itself and other entities are impacted by the environments where it is involved. For example, a person can be in many different environments, such as a family, a company, and/or a school during his/her life, and the relationships with other entities and the behaviors of a
person are different in these environments.

*Dynamic References* - If two entities are related to each other by a binary relationship, we say they refer to each other. In the same environment, for the same relationship, the references between entities may change frequently. For example, in an office, a manager and a secretary form a binary relationship. However, a manager may work with different secretaries at different times, or vice versa.

*Dynamic Revisiting of an Environment* - An entity may enter and leave an environment periodically or randomly. For example, a person usually goes to his/her office in the morning and leaves in the afternoon. However, someday, for some reason, he/she may not show up.

*Concurrently Exhibiting Multiple Properties and Behaviors* - Sometimes, an entity may possess properties and exhibit behaviors which are generally considered to be related to different environments. This also could be due to applying different points of view to look at an entity. For example, a person may be a student and a part time employee during the same time period.

**Temporal Relationships of Behaviors**

One of the important advantages of the object-oriented model is that the behaviors of objects are captured by their operations. Therefore, in the context of time, the temporal relationships of objects operations form an essential part of the behaviors of objects. The data model must answer how general temporal relationships of object operations can be modeled.
Temporal relationships among object operations capture additional semantics for the behaviors of objects. For example, some operations can occur concurrently, some operations are always causally related, and some operations always occur exclusively. For example, retroactive updates capture an important temporal relationship between object operations: some previous object operations may be invalidated by a later operation of an object.

Retroactive updates are often regarded as some kind of correction activity. For example, in engineering design applications retroactive updates may be applied for the correction of past mistakes. In some cases the effect of an invalidated operation is irreversible, e.g., a product already built based on an erroneous design can not be “undone” when the error is discovered and (retroactively) corrected. In such cases we can at least identify affected operations. An object operation invalidated by a retroactive update may cause an invalidation propagation to related objects operations. Making the system responsible for identification of object operations that need verification due to a retroactive update will greatly reduce the complexity of handling such cases.

1.3 TOODB: IMPLEMENTATION ISSUES

To implement a TOODB, the DBMS must take care of many issues such as memory management (main memory buffer management and secondary storage management), transaction management (concurrency control and crash recovery), high-level data management (schema management, configuration, security, data distribution,
constraint management and query optimization). In this dissertation we will concentrate on the secondary storage management.

The design goal of a memory management system is to attain an optimal system performance measured by data retrieval time. If the demanded page containing the data to be retrieved is not available in the main memory, a page fault occurs. Page fault processing time is a major cost of data retrieval. Therefore, we must minimize the average page fault rate by an appropriate design.

Conceptually, the average page fault rate is a function of the average page access number and the probability of page fault per page access. When a page is referred to, one page access is counted. The average page access number is a function of the characteristics of queries and data organization (i.e., storage organization, data structure and access methods) on the secondary storage. The probability of page fault per page access is determined by several factors such as buffer organization, buffer size, replacement algorithm, and data prefetching strategy.

In a TOODB, the characteristics of temporal queries issued by users influence the data organization as well. Since the temporal construct in a temporal query has an impact on the amount of data to be retrieved to answer the query, it is necessary to review all the aspects of memory management systems to see how they are influenced.

Two basic types of object storage organizations have been used in the non-temporal OODB: reference-based and copy-based. Here a reference means an object identifier. In reference-based storage organization, whenever an object is referred to, a reference is stored, while in copy-based storage organization, a copy of the details of
a referred object is always clustered together with the object which refers to it. When storing objects by the referenced-based storage organization, the memory space efficiency reaches its maximum. However, for queries which need to exploit the details of referred objects, the reference-based storage organization becomes inefficient. This situation becomes even worse in a TOODB, because a large number of objects may be involved in a query which has a long time interval or several different time intervals. Copy-based storage organization improves time efficiency, but it will consume more memory space. It is an unacceptable cost to store an entire TOODB by the copy-based storage organization. A related issue to copy-based storage organization is clustering. To design an appropriate clustering technique for the TOODB, we must consider some new factors introduced by the TOODB, for example, how to group the historical data of objects with different sizes and data updating rates so that the clustering can reach the optimal time efficiency for temporal queries.

The characteristics of queries or users' access patterns can be abstracted through primitive transactions and relative frequencies of primitive transactions. Primitive transactions can be typed according to their functions. In a TOODB, at least two basic types of primitive transactions are needed. One is used to retrieve data by going through consecutive data records of an object. The other one is used to first retrieve a data from an "owner" object, then to retrieve a data that is referred to by an "attribute" object. The relative frequencies of primitive transactions that occur in every object are important for describing characteristics of queries as well. The distribution of the relative frequencies of primitive transactions has an influence on
the design of the data organization of the physical object base (this will also be discussed in Chapter 6).

A page access may or may not become a page fault. The design goal of main memory management is to minimize the probability of the page fault per page access. High speed buffer storage devices can greatly improve system performance compared with a system without such devices. Buffer size in a main memory has a direct influence. Choosing the replacement algorithm and the data prefetching strategy is critical for main memory management. In this aspect most existing techniques used in the OODB can be applied to the TOODB with a few changes, since the TOODB may introduce more requirements, e.g. the data prefetching strategy needs to take the temporal construct into account as well.

In addition to the time spent on the page fault processing, in a TOODB, a new cost of query processing results from the characteristics of the event: growing data space of the object base. Since the disk-head seeking time is always proportional to the data space of the object base, insisting on the traditional one-level storage structure will make it very difficult to improve disk-head seeking time in the TOODB. However, from the application point of view, a user, in a certain time period, may be only interested in a very small part of the object base. For a multiuser system, different users may be interested in different parts of the object base. These facts should be considered.

These modeling and implementation issues need to be addressed in order to incorporate the capabilities of temporal information modeling and processing into object-
oriented databases.

1.4 THESIS OVERVIEW

This dissertation makes three contributions: developing a data model for a temporal object oriented database (TOODB), developing a formal system for the data model, and designing a secondary storage management system for the DBMS of TOODB.

Our data model is formed by incorporating object-oriented concepts and dynamic multityping, dynamic instance extension, dynamic reference, event sequence and behavior constraint mechanisms into a state machine model. Therefore it is called the dynamic state machine model. The basic constructs of the dynamic state machine model are dynamic state machines which are divided into three categories: primary machine, multitype composite machine and general composite machine. A multitype composite machine is used to model the whole life of a real world entity which is called a temporal object. It consists of a set of primary machines. Each primary machine is used to model a partial history of a temporal object when it is associated with a certain environment which is described by a type-version. There is only one general composite machine which consists of all multitype composite machines.

By extending the linear temporal logic developed by Z. Manna and A. Pnueli [MP83] to contain the temporal operators of past time, we developed a dynamic state logic (DSL). We also developed a proof system for the DSL. By using the dynamic state machine model as the model of this logic language, we can specify the behavior
constraints on temporal objects naturally and precisely. Based on the proof system of the DSL, we can verify whether the behavior constraints defined in type-versions can derive the given requirements or the expected properties. By presenting a set of algorithms, we have shown that whether the TOODB satisfies behavior constraints specified by a certain class of temporal formulas of the DSL can be efficiently checked.

To implement a TOODB according to the dynamic state machine model, many issues of DBMS, such as memory management, transaction management and high-level data management, need to be reviewed. In this dissertation, we concentrate on designing a suitable memory management system for the TOODB. Memory management for non-temporal object-oriented databases has been discussed widely [CDRS89, CK89, HZ87, St84]. However, in a TOODB, some new questions are waiting for answers. For example, one typical question is how to cluster temporal objects. Clustering is a popular technique used in the non-temporal OODB. In clustering, frequently co-referenced objects are grouped into the same cluster and placed close to each other on the secondary memory in order to reduce page access number. In a TOODB, all the temporal objects have their own database histories. To cluster temporal objects we must decompose these objects' database histories into "history pieces". How do we determine an appropriate "history piece length" so that the clustering can reach a minimal average page access number? This dissertation contributes one possible solution to this question. We present a design of secondary storage management system which has a two-level object base structure. Unlike most existing design methodologies, our design also takes into account possible user access patterns rather than only
the features of data organization. An analysis model is developed for the purpose of finding an optimal design. Based on the analyses, a set of efficient algorithms to optimize parameters are designed. By changing the values of the parameters in the analysis model, we carried out a series of experiments. The results of these experiments help us to understand the characteristics of clustering temporal objects.

This dissertation is organized as follows. Chapter 2 briefly surveys the related work in object evolution, temporal logic, and object-based memory management and clustering techniques. Chapter 3 defines the basic concepts of the dynamic state machine model, and discusses how the dynamic state machine model can be applied as a data model of the TOODB. Chapter 4 introduces dynamic state logic (DSL), and shows how to set up the proof system for the DSL according to the operational semantics of dynamic state machines. Chapter 5, through examples, shows how to specify the constraints on the behaviors of objects by the DSL, and how to verify the expected system properties by the proof system of DSL. A set of algorithms to check if a TOODB satisfies behavior constraints are provided. Chapter 6 presents a design of a paged virtual memory with a two-level storage structure in the secondary storage. A clustering scheme for temporal objects is proposed. An analysis model to obtain the optimal design is developed. Analyses and algorithms are provided in Chapter 7. The results of experiments are discussed in Chapter 8. Finally, the conclusions and future directions are presented in Chapter 9. Appendix A provides a set of theorems and derived inference rules of the DSL for the past time.
Chapter 2

RELATED WORK

2.1 MODELING TIME IN DATABASES

Time is an essential piece of information about a constantly evolving real world. There has been considerable interest in modeling time in databases [Sn86]. Many researchers have shown how data can be enriched with time attributes to reflect a history of data evolutions [SA86]. So far, the work to extend databases for supporting temporal information processing has mainly concentrated on extending the relational databases [CW83, Sn86]. Two approaches to extend the relational data model for temporal modeling have been suggested: (1) extending the semantics of the relational data model to incorporate time directly, and (2) adding time as an additional attribute and translating queries and updates involving time into retrievals and modifications on the underlying snapshot states through an extended query language. The work of incorporating time can also be found in deductive databases. R. Kowalski and M. Sergot developed the Event Calculus to formalize the semantics of valid time in deductive databases [KS86]. A later work extended the Event Calculus to accommodate the concept of transaction time thereby forming a theory of time for
temporal deductive databases [Sr88].

2.2 MODELING OBJECT EVOLUTION

2.2.1 GemStone

GemStone is an object-oriented database developed by Servio Logic Corporation [PS87, PSM87]. GemStone allows users to change types (class schemas) without versioning. The operations on type modification are restricted to adding or deleting an attribute variable, modifying the constraint on an attribute variable, and labeling a type either indexable or non-indexable. The changes in the type hierarchy are restricted to either adding a new type (class) as a leaf node to the type hierarchy, or removing a type from the type hierarchy. Without maintaining version history in GemStone, only the last version of a type remains in the system. When a type is changed, all instances of that type have to be converted to conform with the new version of the type. The conversion is done as an atomic operation, and all instances of the type are unavailable for use by others for the duration of the operation. GemStone’s approach is only suited to applications which do not use historical data. Very few temporal relationships can be captured by this approach.

2.2.2 Iris

Iris is an object-oriented DBMS developed at Hewlett-Packard database laboratory [F+87, F+89]. The Iris data model consists of three constructs: objects, types and functions. Attributes of objects, relationships among objects, and computa-
tions of objects are expressed in terms of functions, so that Iris can be considered a functional system. In addition to supporting evolutions of function expressions, Iris supports type evolution by allowing new types to be created and existing types to be deleted. However, new subtype/supertype relationships among existing types cannot be created. Unlike other systems, Iris objects may gain or lose types dynamically. But at each moment, an object can only be associated with one type and its supertypes. Iris objects can be either versioned or inversioned. Versioned object is represented by a version set containing all the versions of the object. Each version set has a generic instance which has properties whose values are uniform over all versions. Any reference to a versioned object can be either a specific reference to a particular version of the object, or a generic reference to the generic instance. Iris does not have time notions.

2.2.3 ENCORE

The data model of ENCORE, which was developed at Brown University, exploited a versioning approach to the type evolution of an object-oriented database [SZ86]. Every modification of a type forms a version of that type. The inclusive summary of all the versions of a type forms a version set interface of the type. If a version set interface contains more than one type version, every type version must define a set of "handlers" to filter the differences with other type versions. A type version acts just like a database view for users to look at an object in an expected "illusion". An object instance of a type can be created under any type version and remains bounded to that
type version until it is explicitly coerced into another type version by a corresponding screening handler. Effectively, this is a late binding to the representation of an object. An inefficient part of this approach is that whenever a new type version is added, all the existing type versions must be updated by adding a new screening handler corresponding to the new type version. How to preserve objects behaviors consistent in type evolution is not addressed by [SZ86]. The concept of time is absent in ENCORE.

2.2.4 UCB Version Data Model

The UCB Version Data Model, a data model for an object-oriented database which is specially used for engineering design applications, was developed at the University of California, Berkeley [KCB86]. In this model, both object type and object instance are versionable. The structural relationship of the object composition is called a configuration. The logical (conceptual) correspondence relationship of objects is called an equivalence. These two kinds of relationships are also based on the versions of objects. For example, a configuration, in fact, is a version of a composite object whose components are bounded to specific versions. The version derivation is a unique temporal relationship explicitly supported by this model. In this model as well, time is not addressed.

2.2.5 ORION

ORION, an object-oriented database management system developed by the database group at Microelectronics Computer Technology Corp., supports a version control
mechanism for object evolution on both schema (type and type hierarchy) and instance \([B+87, K+87, KC88]\). Consistency of schema is specified by a set of invariants that are enforced by semantic rules. Compared with other systems, ORION has more schema change facilities. Concepts such as default or shared values and composite objects in ORION are usually not supported by other systems. A composite object describes a dependency relationship between the components and the owner object. ORION also supports versions of composite objects, e.g. a version instance of an owner object and a generic instance of a component object are treated as a version of the entire composite object. However, the problem with ORION composite objects is that component objects are always created at the same time when the composite object is created. ORION has not incorporated a concept of time.

2.2.6 AVANCE

AVANCE is a type of an object-oriented database management system being developed at the University of Stockholm and the Swedish Institute for Systems Development, primarily aimed at applications in the field of Office Information Systems [Ki89]. AVANCE supports two kinds of version control mechanisms: application-level version control and system version control. For application-level version control, AVANCE supports not only tracing the historical state or identifiable versions of an object, but also deriving more than one successor from one "frozen" (immutable) version. Type versioning is also supported in AVANCE. The screening approach is applied to handle the problem of mismatching between a type version and an instance
version. One of the special features of AVANCE is that it is the first object-oriented database management system using version control to handle the following issues: (1) synchronizing access to shared objects, (2) providing rollback recovery from system crashes, and (3) providing greater concurrency. In AVANCE, the life of an object version is determined by the user's applications, i.e. an object version is deletable. The concept of time appears in AVANCE, but it is only used for ordering versions and concurrency control. How temporal relationships among objects are modeled is not addressed by the AVANCE data model.

2.3 TEMPORAL LOGIC

The work on temporal logic basically develops in two directions: (1) adding the concept of time into the first order classical logic, (2) restricting the modal logic interpretations.

James Allen's work [Al84], is one of the most significant contributions to the temporal logic, which uses first order classical logic and is based on the concept of time intervals. For Allen, the basic entities that are associated with time intervals are properties, events, and processes. The fact that a property holds for an interval is denoted by the formula $\text{HOLD}(p, i)$, where $p$ is a property type and $i$ is an interval. A property is true for an interval iff it holds for every subinterval. The fact that an event occurred over an interval is denoted by the formula $\text{OCCUR}(e, i)$, where $e$ is an event type. In contrast with properties, an event cannot occur over two intervals, one of which contains the other. An example Allen gives of an event type is $\text{CHANGE-}$
POS(ball, pos1, pos2). Finally, the fact that a process occurs over an interval iff it occurs over some subintervals is denoted by the formula \( \text{OCCURRING}(p, i) \). Some later work extended Allen's temporal logic to allow modeling periodical events and processing imprecise time [Pr85]. Y. Shoham improved Allen's work by "reifying" propositions and also allowed using time points [Sh87]. Although Allen's work is developed for AI applications, the research work of F. Sadri has concluded that under certain conditions it can be translated into R. Kowalski and M. Sergot's Event Calculus which was developed directly for deductive database applications [Sa86].

Another direction of temporal logic research is a branch of modal logic. Interpretations of modal logic consist of a universe of worlds and a basic accessibility relation between the worlds. Modal operators such as \( \Box \) (called necessity) and \( \Diamond \) (called possibility) can be used to describe dynamic change from one world to another. If we call each world a state and restrict the accessibility relation as the passage of time (i.e., a state \( s_i \) is accessible from another state \( s_j \) if through a process in time \( s_j \) can change to \( s_i \)), then a temporal logic is formed. Such temporal logic can be further classified into two categories: branching time and linear time. In branching time temporal logic, time may split into alternative courses representing different possible futures at each time instance, while in linear time temporal logic, each moment has only one possible future.

A typical linear time temporal logic is the one developed by Z. Manna and A. Pnueli [MP83], which only contains the temporal operators describing future time and is used for specifying and verifying concurrent program properties such as liveness,
safety and precedence.

2.4 IMPLEMENTATION OF TEMPORAL RELATIONAL DATABASE

Compared with the efforts on conceptual aspects such as modeling, query languages and semantics of time, few works have been published about the implementation of temporal relational databases. I. Ahn has investigated several issues for the implementation of a temporal relational database such as the handling of ever-growing storage size, the representation of temporal versions in physical storage, and efficient access methods for both temporal and non-temporal data [Ah86]. He argued that because of adopting the non-deletion policy, historical data in temporal relational databases have static characteristics. Also in general, historical data are not accessed as frequently as current data. Therefore, historical data and current data can be treated differently. Hence, a two-level storage scheme, one level for historical data and one level for current data, was suggested in [Ah86].

Another work on physical implementation was done by D. Rotem and A. Segev [RS87]. They proposed a partitioning scheme of files applicable to temporal relational databases.

More recently, a new indexing technique, called time index, was presented by R. Elmasri and G.T.J. Wuu [EW90], for improving the performance of certain classes of temporal queries in temporal relational databases.
2.5 OBJECT-BASED CLUSTERING

Clustering is a popular technique used in object-oriented databases for memory management [At85, Ma86, Zd84, HZ87, KBCGW87]. Object-based clustering can be classified into two categories: static and dynamic. Static clustering performs the clustering when the system is in a quiescent state, while dynamic clustering does its job each time an object is created or updated. Paging performances of five static object-based clustering schemes have been investigated by J. W. Stamos [St84]. This research concluded that static clustering substantially reduce the number of page faults caused by an unclustered initial placement, and performance differences between clustering schemes were not appreciable significant. E.E. Chang found that when the read/write ratio is high, dynamic clustering can improve the paging performance even more [Ch89]. Partition principles applied in existing clustering schemes can be based on users' hints, class hierarchy, common attribute structure, or object composition hierarchy. To attain an optimal partition, two types of heuristics, transaction-oriented and single-object evaluation of object usage, are used in the ENCORE system [HZ87]. Transaction-oriented heuristics monitor how objects are used together. Single-object heuristics use three measurements amassed over a period of time for monitoring: the access count, the open count, and the access ratio. These statistical data are used to adjust object partitions.

To our knowledge, the question of clustering temporal object has not been addressed before.
2.6 OTHER RELATED WORK

A clustering algorithm for the database with hierarchical structure was proposed by M.A. Schkolnick [Sc77], where all instances of a type are grouped into segments. These segments are then grouped in a hierarchical tree structure. Access patterns of data segments are captured by three transaction types and relative frequencies of transactions. Examining these patterns allows the algorithm to determine a partition of the tree. Once partitions have been produced, the available disk space is divided into linear address spaces (LASs), one for each partition (subtree). The instances of all segments for a given partition are placed in their corresponding LASs in the same order as they appear in the hierarchical order. In our work, we show that similar ideas can be applied to build an analysis model for clustering temporal objects. Moreover, our model takes into account alternative clustering orderings, and different history piece lengths. We use different transaction types, and also consider the semantics of various distributions of relative frequencies of transactions.
Chapter 3

DYNAMIC STATE MACHINE MODEL

3.1 INTRODUCTION

Lacking the temporal modeling constructs, the traditional data model of the OODB is not able to capture the many dynamic changes common to every day phenomena. Many real world applications involve the management of large amounts of time-dependent information. In order to apply an object-oriented database in an application with dynamic changes, incorporating temporal modeling capability into the data model is needed. In this chapter we will introduce a data model of the TOODB, which is called the dynamic state machine model. This model is an extension of the state machine model. In order to fully model temporal objects, the state machine model is extended in the following ways: (i) the time notion is introduced; (ii) state machines are typed; (iii) structural and temporal compositions of state machines are considered; (iv) behavior constraints are adopted as a modeling construct; (v) the behavior history of a state machine is maintained in the trace (i.e., event sequence); (vi) dynamic changes in a state machine are captured by dynamic multityping, dynamic
extension, dynamic reference, and identification of retroactive update affection mechanisms. After providing definitions of the concepts and rules used in the dynamic state machine model, we will discuss why and how the dynamic state machine model can capture the features of the TOODB.

3.2 CONSTRUCTS OF THE DYNAMIC STATE MACHINE MODEL

In this section, we will define the constructs of the dynamic state machine model: the concepts and rules. We assume the following disjoint countable symbol sets: \( I_t \)-types, \( I_v \)-type-versions, \( I_r \)-transitions, \( I_e \)-event identifiers, \( I_t \)-trace identifiers, \( I_{id} \)-dynamic state machine identifiers, and \( I_v \)-variables.

In the dynamic state machine model, every entity in the real world is modeled by a dynamic state machine. All dynamic state machines are assigned a unique identifier. A dynamic state machine can be either a primary machine or a composite machine. Therefore, \( I_{id} = I_{pmid} \cup I_{comid} \), where \( I_{pmid} \) contains identifiers of all existing primary machines and \( I_{comid} \) contains identifiers of all existing composite machines. All dynamic state machines form a three level hierarchy. At the bottom level there are many primary machines. A set of primary machines forms a multitype composite machine. All multitype composite machines form the general composite machine.

3.2.1 Primary Machine

A primary machine \( m \) is defined by a 5-tuple, i.e.,

\[
m = (M, \varphi^i_m, E_m, R_m, \hat{\Sigma}_m) \tag{3.1}
\]
where \( m \in I_{\text{mid}} \), \( M \in I_{\text{mid}} \) (\( M \) represents the multitype composite machine, defined in Section 3.2.2, which contains the primary machine \( m \)), \( \varphi^i_m \) is a type-version, \( E_m \) is the event set, \( R_m \) is the trace set which contains all traces formed by the events in \( E_m \), \( \hat{\Sigma}_m \) is a set of legal state sequences. These notions are further defined as follows.

**Type and Type-version**

A type \( \varphi \) (\( \varphi \in I_{\text{ty}} \)) can be either an atomic type (e.g. integer, character, boolean) or a non-atomic type (i.e., an abstract data type). A non-atomic type consists of a set of type-versions, each of which is a kind of definition of the type. Atomic types are treated as single-version types. We will use only the type-version to define primary machines.

A type-version \( \varphi^i \) is defined as a 6-tuple, i.e.,

\[
\varphi^i = (t_c, t_f, \text{sup}, V, T_r, C)
\]  
(3.2)

where \( \varphi^i \in I_{\text{tv}} \), \( t_c \) is the commit time, \( t_f \) is the effective time, \( \text{sup} \) points out the super type-version, \( V \) is a variable set, \( T_r \) is a transition set (i.e., \( T_r \subseteq I_r \)) and \( C \) is a constraint set.

The commit time is the instant of time when the definition of the type-version commits. The effective time is the instant of time from which the type-version can be used to create primary machines.

The variable set \( V \) contains two kinds of variables: attributes and parameters. Attribute variables are used to define attributes of a primary machine, while parameter variables are only used in transitions. Each variable \( v \) is assigned a type-version \( \varphi^i_v \).
where \( \varphi_v^i \in I_v \), which specifies a domain for \( v \). The domain of a type-version \( \varphi^i \) is a subset of \( I_{pmid} \) and is denoted as \( |\varphi^i| \), i.e., \( |\varphi^i| \subseteq I_{pmid} \). Two elements in the same domain means they are the primary machines with the same type-versions.

To ensure a type-version is well defined, we adopt the following convention: all type-versions which are used by a type-version being defined must have an effective time that is earlier than or equal to that of the type-version being defined.

Each transition \( \tau (\tau \in T_\tau) \) in the transition set \( T_\tau \) is a kind of operation defined on the attributes and consists of a specification and an implementation. The specification of a transition is a 3-tuple \( (\tau, \text{input.parameters}, \text{output.parameters}) \), whereas the implementation of a transition is a program consisting of a sequence of steps. Each step is either a local read, a local write, a local computation or a message. A local read retrieves a value of an attribute variable. A local write updates the value of an attribute variable. A local computation is a program segment which makes use of the functionality of the underlying programming language and the input parameters (i.e., arguments) to do some computation but does not modify any stored values outside of its own local memory. A result of the computation may be returned. A message is an invocation of a transition defined in another type-version. We assume each transition has at least one output parameter.

Three kinds of time parameters are used in every transition. The commit time \( t_c \) is the instant of time when all the steps in the transition \( \tau \) commit. The effective time \( t_f \) is the instant of time at which the data written to an attribute starts taking effect; The observation time \( t_o \) is the time when the database is observed. Every local
read step in \( \tau \) has an observation time. The data retrieved by a local read step should satisfy the following condition: the commit time and the effective time of the data are less than or equal to the effective time and the observation time of the transition. We assume that (1) in a transition containing no local read steps, \( t_o = \text{nil} \) and the data written by all local write steps have the same effective time; (2) all local read steps in a transition use the same effective time and the same observation time to retrieve data.

In this model, we adopt the notions of encapsulation and information hiding. Primary machines communicate each other through message passing. All the specifications of elements in \( T_r \) are the signatures of the messages that can be received by the primary machines defined by the type-version. Therefore \( T_r \) is an external interface.

Formally, a transition defines a mapping such that

\[
\tau : D_{\tau}^{\text{in}} \to D_{\tau}^{\text{out}} \quad (3.3)
\]

\( D_{\tau}^{\text{in}} \) is called the input domain of \( \tau \) which consists of the cross product of the domains of all attributes on which the \( \tau \) is defined and the domains of input parameters, i.e.,

\[
D_{\tau}^{\text{in}} = |\varphi_{x_1}^i| \times \ldots \times |\varphi_{x_1}^i| \times |\varphi_{y_1}^i| \times \ldots \times |\varphi_{y_k}^i| \times |\varphi_{t_c}^i| \times |\varphi_{t_f}^i| \times |\varphi_{t_s}^i| \quad (3.4)
\]

\( D_{\tau}^{\text{out}} \) is called the output domain of \( \tau \) which consists of the cross product of the domains of the attributes on which \( \tau \) is defined and the domains of output parameters, i.e.,

\[
D_{\tau}^{\text{out}} = |\varphi_{x_1}^i| \times \ldots \times |\varphi_{x_1}^i| \times |\varphi_{y_1}^i| \times \ldots \times |\varphi_{y_k}^i| \quad (3.5)
\]
In the above, \( |\mathcal{P}_x^i| \) is the domain of the attribute variable \( x_i \), \( |\mathcal{P}_y^i| \) is the domain of parameter variable \( y_j \), \( |\mathcal{P}_t^i_c| \), \( |\mathcal{P}_t^i_f| \) and \( |\mathcal{P}_t^i_o| \) are the domains of \( t_c \), \( t_f \) and \( t_o \) respectively.

The constraint set \( \mathcal{C} \) contains the behavior constraints. A precise definition of behavior constraints depends on the concrete language applied (e.g., a logic language).

Event

An event occurs when a transition \( \tau \) is executed. For each event \( e \), we define the following.

(1) The variable set of \( e \), denoted as \( V_e \), consists of all the variables that are used in the transition \( \tau \) of \( e \).

(2) The value set of \( e \), denoted as \( \psi_e \), consists of all the values of the variables in \( V_e \) after the event \( e \) occurs.

\[
\psi_e = \{ \psi_e(v_1), \psi_e(v_2), \ldots, \psi_e(v_n) \}
\]  
(3.6)

where \( \psi_e(v_i) \) represents the value assigned to the variable \( v_i \) (\( v_i \) can be either an attribute \( x \), or a parameter such as \( y \), \( t_c \), \( t_f \) or \( t_o \)) by the event \( e \).

Now we can formally define the notion of event as follows. An event \( e \) is a one-to-one mapping from \( V_e \) to \( \psi_e \), i.e., \( e : V_e \rightarrow \psi_e \), where \( e \in I_e \), \( V_e \) and \( \psi_e \) are as defined above.
Retroactive Event and Affected Event

Assume that $\psi_e(t_c)$ and $\psi_e(t_f)$ respectively represent the commit time and the effective time of an event $e$. The event is called a normal event if $\psi_e(t_c) = \psi_e(t_f)$. It is called a retroactive event if $\psi_e(t_c) > \psi_e(t_f)$. It is called a post-dated event if $\psi_e(t_c) < \psi_e(t_f)$.

A retroactive event may be used to explicitly invalidate a previous event or implicitly affect some previous data values and events. The concepts of affected value and affected event are defined as follows.

Let $e_i, e_j$ be two events, and $\tau_i, V_{e_i}, \psi_{e_i}(t_c), \psi_{e_i}(t_f)$ and $\tau_j, V_{e_j}, \psi_{e_j}(t_c), \psi_{e_j}(t_f)$ be transitions, variable sets, commit times and effective times of $e_i$ and $e_j$ respectively. Let $x$ be an attribute variable. We call the value $\psi_{e_i}(x)$ an affected value if the following conditions hold:

1. $e_j$ is a retroactive event;
2. $x \in V_{e_i}$ and $x \in V_{e_j}$;
3. $\psi_{e_i}(t_c) \leq \psi_{e_j}(t_c)$ and $\psi_{e_i}(t_f) \geq \psi_{e_j}(t_f)$;
4. $\tau_j$ is invoked to assign a value $\psi_{e_j}(x)$ to $x$.

Let $e_k$ be another event with a transition $\tau_k$, a variable set $V_{e_k}$, a commit time $\psi_{e_k}(t_c)$, an effective time $\psi_{e_k}(t_f)$ and an observation time $\psi_{e_k}(t_o)$. Let $y$ be a variable and $y \in V_{e_k}$. The value of $y, \psi_{e_k}(y)$, is also an affected value if the following conditions hold:

1. $\tau_k$ is invoked to read an affected value $\psi_{e_j}(x)$ at $\psi_{e_k}(t_o)$,
2. $\psi_{e_k}(t_e) \geq \psi_{e_j}(t_c)$ and $\psi_{e_k}(t_f) \geq \psi_{e_j}(t_f)$,

3. $r_k$ is invoked to assign a value to $y$ that depends upon $\psi_j(x)$.

An event $e$ is called an affected event if either at least one of its values becomes an affected value or it is explicitly invalidated by a retroactive event.

**Legal Event**

An event is legal until it becomes an affected event.

**Trace and State Evolution**

For all the events in the event set $E_m$ of a primary machine $m$, a total ordering relation $<$ is defined, such that $e_i \prec e_j$ iff $\psi_{e_i}(t_c) < \psi_{e_j}(t_c)$. If a set of events $\{e_0, e_1, ..., e_{n-1}\}$, which is a subset of $E_m$, satisfies the following conditions:

1. $e_0$ is the first event in the $E_m$,

2. $e_0 \prec e_1 \prec ... \prec e_{n-1}$,

3. there is no event $e_j, e_j \in E_m$, such that $e_i-1 \prec e_j \prec e_i, i = 1, 2, ..., n - 1$,

then these events form a trace which is denoted as

$$r_n = e_0 \bullet e_1 \bullet ... \bullet e_{n-1} \quad (3.7)$$

where $r_n \in I_r$.

All the traces of a primary machine $m$ are contained in the trace set $R_m$. 

31
The state-evolution of a primary machine \( m \) is an infinite sequence defined as follows:

\[
\sigma_m = s_m^0, s_m^1, s_m^2, \ldots \tag{3.8}
\]

where \( s_m^i = (r_i, e_i), \ i = 0, 1, \ldots \), is called a state, \( r_i \) is a trace, \( e_i \) is called a state transferring event which is the last event in the trace \( r_i \).

\( \sigma_m \) satisfies the following conditions:

1. \( r_0 = \text{null} \) and \( e_0 \) executes the transition \( \tau_0 \) which initializes all the variables in the \( V_m \), and \( \tau_0 \) can only be executed in the \( e_0 \).

2. for any two consecutive states \( (r_i, e_i) \) and \( (r_i+1, e_{i+1}) \), \( r_{i+1} = r_i \circ e_i \).

3. every state \( s^i \) has one and only one state transferring event \( e_i \) which executes a transition \( \tau_i \) which belongs to \( T_r \).

Legal Trace and State Evaluation

For each state \( s^i = (r_i, e_i) \), there is a legal trace which is formed by all the legal events in \( r_i \) according to a total ordering relation \( \prec^* \) defined by the following convention:

1. the first event is \( e_0 \);

2. for any two legal events \( e_{j_1} \) and \( e_{j_2} \) in \( r_i \), if \( \psi_{e_{j_1}}(t_f) < \psi_{e_{j_2}}(t_f) \), then \( e_{j_1} <^* e_{j_2} \);

3. for any two legal events \( e_{j_1} \) and \( e_{j_2} \) in \( r_i \), if \( \psi_{e_{j_1}}(t_f) = \psi_{e_{j_2}}(t_f) \) and \( \psi_{e_{j_1}}(t_c) < \psi_{e_{j_2}}(t_c) \), then \( e_{j_1} <^* e_{j_2} \).
The legal trace of a state $s^i$ is denoted as

$$r_i^* = e_0 \bullet e_1 \bullet ... \bullet e_{j_{\text{max}}} \quad (3.9)$$

Let $r$ be a trace variable and $e$ be an event variable. The legal variable set of the state $s^i$ $V_{s^i}$ is defined as follows:

$$V_{s^i} = \bigcup_{e_j \in r_i^*} V_{e_j} \cup \{r, e\}, \quad (3.10)$$

where $V_{e_j}$ is the variable set of a legal event in the legal trace $r_i^*$. The domain of $r$ is $R_m$ and the domain of $e$ is $E_m$. The legal domain set of the state $s^i$ $D_{s^i}$ is defined as follows:

$$D_{s^i} = \bigcup_{v \in V_{s^i}} |\varphi^i_v|. \quad (3.11)$$

The state evaluation of a state $s^i$, $i = 0, 1, ...$, is denoted as $\hat{s}^i$ and is defined as a mapping $\hat{s}^i : V_{s^i} \rightarrow D_{s^i}$, such that:

1. $\hat{s}^i(r) = r_i,$

2. $\hat{s}^i(e) = e_i,$

3. 

$$\hat{s}^i(v) = \begin{cases} 
\psi_{e_{j_{\text{max}}}}(v), & \text{for } v \in V_{e_{j_{\text{max}}}} \\
\psi_{e_{j_{\text{max}}-1}}(v), & \text{for } v \notin V_{e_{j_{\text{max}}} \land v \in V_{e_{j_{\text{max}}-1}}} \\
\psi_{e_0}(v), & \text{for } v \notin V_{e_{j_{\text{max}}} \land ... \land v \notin V_{e_1} \land v \in V_{e_0}} 
\end{cases} \quad (3.12)$$

where $\psi_{e_j}(v)$ is the value assigned to the variable $v$, $V_{e_j}$ is the variable set of the legal event $e_j$ in $r_i^*$.

We define the legal state sequence corresponding to $\sigma_m$ to be the sequence $\hat{\sigma}_m$ such that:

$$\hat{\sigma}_m = \hat{s}_m^0, \hat{s}_m^1, \hat{s}_m^2, ... \quad (3.13)$$
We define $\Sigma_m$ to be the set of all state evolutions of a primary machine $m$ and $\hat{\Sigma}_m$ to be the set of all corresponding legal state sequences.

**Is-a Relationship**

A primary machine is often denoted simply by $m_p^M$, which means this primary machine has a type-version $\varphi^i$ and belongs to a composite machine $M$.

If a type-version is a generalization (or specialization) of another one, it is called the super (or sub) type-version of the other one. This kind of generalization/specialization relationship is called *is-a* relationship. The *is-a* relationship is anti-symmetric and transitive. If $\varphi^1$ is the super type-version of $\varphi^2$ which is the super type-version of $\varphi^3$, then both $\varphi^1$ and $\varphi^2$ are called an *ancestor type-version* of $\varphi^3$. In this model, we assume that a type-version fully inherits the definition of its super type-version. A group of type-versions, according to the *is-a* relationships that they have, form a *type-version tree*. If $\varphi^1$ is the super type-version of $\varphi^2$, then $\varphi^1$ is an ancestor type-version of $\varphi^2$. If $\varphi^1$ is the super type-version of $\varphi^2$, and $\varphi^2$ is the super type-version of $\varphi^3$, then $\varphi^1$ is an ancestor type-version of $\varphi^3$. All type-version trees form a *type-version forest*.

Given two primary state machines $m_p^M$ and $m_p^M$, if $\varphi^1$ is the super type-version of the $\varphi^2$, $m_p^M$ is called a *basic machine* of $m_p^M$, and $m_p^M$ is called an *extension machine* of $m_p^M$. The part of the $m_p^M$ which is not in the $m_p^M$ is called an *extension part* of $m_p^M$. The basic/extension relationship is anti-symmetric and transitive. If $m_p^M$ is the basic machine of $m_p^M$ which is the basic machine of $m_p^M$, then $m_p^M$ is also a basic machine.
of $m_p^M$. A primary machine may not exist at the same time when its basic machine exits. The *dynamic extension* mechanism is applied to capture this characteristics, which consists of two integrity rules 3.6 and 3.7 (see Section 3.2.5). An extension machine may have an event which has the same commit time as that of an event occurred in one of its basic machine. We treat these events as a complex event which will be defined in the next section.

3.2.2 Multitype Composite Machine

A multitype composite machine $M$ is defined by a 5-tuple, i.e.,

$$M = (m_c, P_M, E_M, R_M, \hat{\Sigma}_M)$$  \hspace{1cm} (3.14)

where $M \in I_{mid}$ and $m_c$, $P_M$, $E_M$, $R_M$ and $\hat{\Sigma}_M$ are respectively the cooperating machine, primary machine set, event set, trace set and legal state sequence set of $M$.

Cooperating Machine

The cooperating machine $m_c$ is a *pseudo primary machine*, which functions as a manager of the multitype composite machine $M$. It is defined as a 5-tuple, i.e.,

$$m_c = (M, \varphi_c^l, E_c, R_c, \hat{\Sigma}_c)$$  \hspace{1cm} (3.15)

$m_c \in I_{pmid}$. $M$ is the identifier of the composite machine which the $m_c$ belongs to. $\varphi_c^l = (t_c, t_f, V_c, T_c, C_c)$ is a *pseudo type-version*, where $t_c$ is the commit time, $t_f$ is the effective time, $V_c$ is the *composite variable set*, $T_c$ is the *composite transition set*, and $C_c$ is the *composite behavior constraint set*.
Basically, in the variable set $V_c$, there are at least two set-variables: $a$, which is called the component machine activity state variable, and $b$, which is called the component machine variable. Each element of $a$ corresponds to one component machine and has a value of either active or stopped. The value of $b$ is a set of identifiers of component machines which are currently associated with the composite machine $M$, i.e., the domain of $b$ is the power set of $I_{p_{mid}}$ (denoted as $P^{fin}(I_{p_{mid}})$). $V_c$ also contains the parameter variables used in the transitions defined below.

$T_c \subseteq I_r$. It contains at least the following transitions:

1. **Create**: used to add a new component machine;
2. **Stop**: used to stop a component machine;
3. **Resume**: used to resume a stopped component machine;
4. **Compo**: used to test whether a primary machine is a component;
5. **Is-a**: used to test whether two component primary machines have the is-a relationship;
6. **Affected event identification**: used to identify all the events which are affected by a retroactive event.

$C_c$ consists of the behavior constraints that define temporal relationships among the r's of different component machines and/or those in the $T_c$.

The events whose transitions are in $T_c$ form an event set $E_c$, and the total ordering relation based on the commit times of events in $E_c$ form the traces in the trace set $R_c$. 

36
Therefore, we also can define the state, state evolution and state evaluation similarly to a real primary machine.

**Complex Event**

In a multitype composite machine, it is possible that several events from different primary machines (including the pseudo primary machine) commit at the same time. If an event in a multitype composite machine commits at a time when no other events commit at the same time, then it is called a *simple event*. If several events have identical commit times, they are technically treated as a *complex event*. Assume that $e_1, e_2, ..., e_n$ are events with identical commit times. That is, $e_1 : V_{e_1} \rightarrow \psi_{e_1}$, $e_2 : V_{e_2} \rightarrow \psi_{e_2}, ..., e_n : V_{e_n} \rightarrow \psi_{e_n}$, and $\psi_{e_1}(t_e) = \psi_{e_2}(t_e) = ... = \psi_{e_n}(t_e)$. The transition invoked in the event $e_i$ is $\tau_i : D_{\tau_i}^{in} \rightarrow D_{\tau_i}^{out}$.

The complex event, denoted as $e$, formed by these events is defined as follows:

$$e : V_e \rightarrow \psi_e \quad (3.16)$$

where $V_e = \bigcup_{i=1}^n V_{e_i}$ which is a set containing all the variables used in the $n$ events; and $\psi_e = \bigcup_{i=1}^n \psi_{e_i}$, which is a set containing all the values assigned to the variables in $V_e$ such that $\psi_e(v) = \psi_{e_i}(v)$ if $v \in V_{e_i}$.

The transition $\tau$ used by the complex event $e$ is defined as

$$\tau : D_{\tau_1}^{in} \times D_{\tau_2}^{in} \times ... \times D_{\tau_n}^{in} \rightarrow D_{\tau_1}^{out} \times D_{\tau_2}^{out} \times ... \times D_{\tau_n}^{out}$$

such that

$$\tau(x_{11}, ..., t_{o_1}, ..., x_{n1}, ..., t_{o_n}) = \begin{cases} \tau_1(x_{11}, ..., t_{o_1}), & \text{for } (x_{11}, ..., t_{o_1}) \in D_{\tau_1}^{in} \\ \vdots \\ \tau_n(x_{n1}, ..., t_{o_n}), & \text{for } (x_{n1}, ..., t_{o_n}) \in D_{\tau_n}^{in} \end{cases} \quad (3.17)$$

37
The transition of a complex event is called a \textit{complex transition}, while the transition of a simple event is called a \textit{simple transition}.

The events $e_1, e_2, \ldots, e_n$ are called \textit{element events} of the complex event $e$. Now the event set of the multitype composite machine $M$ can be defined as follows:

$$E_M = \bigcup_{t=t_0}^{t'} \{ E_{C_M} \cup E_{S_M} \}, \quad (3.18)$$

where $t'$ represents the current time, and $E_{C_M}$ and $E_{S_M}$ represent a \textit{complex event set} and a \textit{simple event set} of $M$ respectively.

Correspondingly, the transition set of the multitype composite machine $M$ can be defined as follows:

$$T_M = \bigcup_{t=t_0}^{t'} \{ T_{C_M} \cup T_{S_M} \}, \quad (3.19)$$

where $t'$ represents the current time, and $T_{C_M}$ and $T_{S_M}$ represent a \textit{complex transition set} and a \textit{simple transition set} of $M$ respectively.

A total ordering relation $\prec$ can be defined on all the events in $E_M$, such that for any $e_i, e_j \in E_M (i \neq j)$, $e_i \prec e_j$ if and only if $\psi_{e_i}(t_c) < \psi_{e_j}(t_c)$. Therefore, similar to the primary machine, the notion of trace can be defined. All of traces of the multitype composite machine $M$ are contained in the trace set $R_M$.

The concepts of state and state evolution of the multitype composite machine $M$ can be defined similarly to those of the primary machine as well. The state evolution $\sigma_M$ of the composite machine $M$ is formed by the traces in $R_M$ and events in $E_M$.

To define the legal trace and state evaluation for a state of a multitype composite machine, we must first define the concept of \textit{legal complex event}. Assume that $s^i =$
\((r_i, e_i)\) is a state of the multitype composite machine \(M\). We call a simple event or an element event of a complex event in \(r_i\) an \textit{individual event} of \(r_i\). If more than one legal individual events in \(r_i\) have the same effective times, they form a legal complex event, which can be formally defined similarly to the way to define the complex event (i.e., (3.16) and (3.17)). If an legal individual event in \(r_i\) does not belong to any legal complex event, then it is called an legal simple event. The \textit{legal trace} of state \(s^f\) is formed by all legal simple and complex events in \(r_i\) based on the total ordering relation \(<^*\) defined as follows:

1. the first event is \(e_0\);
2. for any two legal events \(e_{j_1}\) and \(e_{j_2}\) in \(r_i\), if \(\psi_{e_{j_1}}(t_f) < \psi_{e_{j_2}}(t_f)\), then \(e_{j_1} <^* e_{j_2}\).

The legal variable set of a state, the legal domain set of a state, and the state evaluation as well as legal state sequence can be defined similarly to those defined for the primary machine (i.e., (3.9), (3.10), (3.11), (3.12) and (3.13)). We define \(\Sigma_M\) to be the set of all state evolutions of a multitype composite machine \(M\) and \(\hat{\Sigma}_M\) to be the set of all legal state sequences correspondingly.

### 3.2.3 General Composite Machine

The general composite machine \(M_g\) is defined by a 5-tuple, i.e.,

\[
M_g = (m_g, P_g, E_g, R_g, \hat{\Sigma}_g)
\]

where \(M_g \in I_{mid}\), and \(m_g, P_g, E_g, R_g\) and \(\hat{\Sigma}_g\) are respectively the cooperating machine, the multitype composite machine set, event set, trace set and legal state sequence.
set of \( M_{\phi} \).

The cooperating machine \( m_{\phi} \) is similar to the cooperating machine \( m_{c} \) defined in the multitype composite machine. The difference is that here the domain of the component machine variable \( b \) is a subset of \( P^{fin}(I_{mid}) \). In the general composite machine, a multitype composite machine, which is called schema machine and is denoted as \( M_{tp} \), is specially used to model the type and type-version evolution. In the \( M_{tp} \), each type \( \phi \) has a corresponding primary machine which is identified as \( m_{\phi} \). In the variable set of \( m_{\phi} \), there are two attribute variables: type-version-id \( tv \) and type-version-definition \( def \). \( def \) is a tuple-variable which consists of six elements: \( t_{c}, t_{f}, sup, V, T_{r}, \) and \( C \). The domain of \( tv \) is \( I_{tv} \). The domain of \( def \) is defined as

\[
D_{def} = |\varphi_{t_{c}}^{i}| \times |\varphi_{t_{f}}^{i}| \times |I_{tv}| \times P^{fin}(I_{u}) \times P^{fin}(I_{r}) \times |C|,
\]

where \( |C| \) represents all the constraints. In the transition set of \( m_{\phi} \), the following transitions are contained:

(1) **New-version**: used to create a new type-version of the type.

(2) **Parent-Of**: used to test whether a type-version is the super type-version of another one.

(3) **Ancestor**: used to test whether a type-version is the ancestor type-version of another one.

In the constraint set of \( m_{\phi} \), the following constraints are contained:

(1) each type-version belongs to only one type.
(2) each type-version has only one super type-version.

(3) the effective time of a type-version must be later than or equal to that of its super type-version.

Except for these things, all the other concepts in $M_g$ can be defined similarly to those defined in the multitype composite machine.

3.2.4 Relationship between a $\hat{\sigma}_M$ and a $\hat{\sigma}_m$

Let $\hat{\sigma}_M = \hat{\sigma}_M^0, \hat{\sigma}_M^1, \hat{\sigma}_M^2, \ldots$ be a legal state sequence of a composite machine $M$; let $m_i$ be a component machine of the $M$. To produce the legal state sequence of $m_i$ based on the legal state sequence of $M$, a projection function is carried out by the following steps: (i) check the results of each state evaluation $\hat{\sigma}_M^i$, and select only those state evaluations in which the legal trace $\tau_i^*$ contains events that belong to the component machine $m_i$, (ii) renumber these selected state evaluations, (iii) remove all the legal events which do not belong to the $m_i$ from the legal traces of these selected state evaluations.

The result is denoted as $\hat{\sigma}_M \downarrow_{m_i} = \hat{\sigma}_M^0 \downarrow_{m_i}, \hat{\sigma}_M^1 \downarrow_{m_i}, \ldots$.

We claim that $\hat{\sigma}_{m_i} = \hat{\sigma}_M \downarrow_{m_i}$. Correspondingly, we define $\hat{\Sigma}_M \downarrow_{m_i}$ as a set which contains all $\hat{\sigma}_M \downarrow_{m_i}$'s, and claim that $\hat{\Sigma}_{m_i} = \hat{\Sigma}_M \downarrow_{m_i}$.

3.2.5 Integrity Rules

Rule 3.1: All dynamic state machines share the same type-version forest.

Rule 3.2: In a composite machine, when a component machine $m_i$ is created, the corresponding activity state variable is assigned as active; when the $m_i$ is stopped,
the corresponding activity state variable is assigned as \textit{stopped}; when the stopped \(m_i\) is resumed, the corresponding activity state variable is assigned as \textit{active} again.

\textit{Rule 3.3:} In a composite machine, if the activity state variable of a component machine \(m_i\) has a value of \textit{active}, then the component machine \(m_i\) can execute any transitions defined in the transition set \(T_{m_i}\). If the value is \textit{stopped}, then the component machine \(m_i\) can execute only retrieval transitions.

\textit{Rule 3.4:} When a primary machine is stopped, all its active extension machines will be stopped at the same time; when a composite machine is stopped, all its component machines will be stopped at the same time.

\textit{Rule 3.5:} When a primary machine is to be resumed, if its basic machine is stopped, then the basic machine must resume at the same time.

\textit{Rule 3.6:} A dynamic state machine is created according to the following procedures:

1. The general machine is assumed to be created in advance by the system.

2. Any multitype composite machines are created by executing the \textit{Create} transition in the general machine.

3. Any primary machine \(m_p^M\) is created by executing the \textit{Create} transition in the corresponding multitype composite machine by the following steps:

   (1) if \(\varphi^i\) is a type-version of a root type, then \(m_p^M\) is created directly,

   (2) if \(\varphi^i\) is a type-version of a nonroot type, then whether an \(m_p^M\) exists such that \(\varphi^j\) is the super type-version of the \(\varphi^i\) is checked,
(3) if the answer in (2) is "Yes", then (a) if the \( m_{p}^{M} \) is active, it is taken as the basic machine of \( m_{p}^{M} \), (b) if the \( m_{p}^{M} \) is stopped, it resumes and is taken as the basic machine of \( m_{p}^{M} \).

(4) if the answer in (2) is "No", then a \( m_{p}^{M} \) is inductively created and taken as the basic machine of \( m_{p}^{M} \).

(5) for all conditions, the extension part of \( m_{p}^{M} \) is created.

**Rule 3.7:** A primary machine shares only the histories of its basic machines that existed after the primary machine has created.

**Rule 3.8:** Every retroactive event will trigger the affected event identification function to identify all the events and values which are affected by the retroactive event (directly or indirectly).

### 3.3 MODELING TEMPORAL OBJECTS BY DYNAMIC STATE MACHINES

The following section will discuss how the dynamic state machine model can capture the features of temporal objects.

#### 3.3.1 Environment Modeling

At any time, an temporal object must be associated with one or more environment. An environment is defined as a set of structural and/or temporal relationships with which a set of objects are associated at a certain (physical or conceptual) place and during a certain time period. The notion of type is suitable to model environments. Structural relationships can be modeled by is-attribute-of and is-a (i.e.,
generalization/specialization) hierarchies. Temporal relationships can be modeled by the time notions used in transitions and behavior constraints. Because the definition of a type may involve evolution, types are versionable (atomic types such as integer and boolean are treated as single-version types). Every type-version is a node in an is-a hierarchy which is a tree. In our data model, the semantics of the multiple inheritance is captured by applying temporal composition (e.g. a multitype composite machine). Therefore, only a tree structure is needed for the type-version hierarchy. Since each type may have more than one version, at a given time there may exist a set of such hierarchies, which is called the type-version forest. Therefore, any change in environments can be captured by a type-version forest.

3.3.2 Dynamic Multityping

In the dynamic state machine model, a temporal object is modeled by a multitype composite machine which applies a dynamic multityping mechanism. Essentially, the dynamic multityping mechanism is used to capture another kind of composition relationship among a set of objects, which is called temporal composition, i.e., a temporal object is treated as a composition based on the temporal relationship hold by its different historical parts. A primary machine is used to model the experience of the temporal object under a certain environment. Whenever a temporal object moves into a new environment, a primary machine defined by the type-version corresponding to the new environment is created and added into the multitype composite machine which models that temporal object. The behavior evolutions of the temporal object under different environments are recorded in the corresponding primary machines.
After its creation, the primary machine is active. An active primary machine can execute all the transitions defined in it. When a temporal object leaves an environment, the corresponding primary machine must stop. A stopped primary machine can only respond to the queries about its history. A stopped primary machine can resume again later on if the temporal object reenters the environment represented by the type-version of that primary machine. A primary machine may stop and resume many times.

In a multitype composite machine, more than one active primary machine may exist at a given time. This is because of the following reasons: (1) the semantics of the is-a hierarchy of type-versions determine that if a primary state machine is active, then all the ancestor primary machines also have to be active, (2) more than one environment may be associated with a temporal object in the same time period.

We can demonstrate the concepts above by an example. Assume that a person is named John which is used as the multitype composite machine identifier in this example. At the beginning, John can only be described by the type-version Person¹, so that only the primary machine mJohn Person¹ is in the composite machine John. Later on, John goes to school; hence, another primary machine with type-version Student¹, i.e., mJohn Student¹, is created and added into the multitype composite machine John. These two primary machines concurrently remain active after the creation of the mJohn Student¹. After his graduation from a university, John is hired by a computer company as a programmer, and therefore a new primary machine mJohn Programmer¹ is created and added into the multitype composite machine John. However, due to John's leaving school,
the primary machine $m_{\text{John}}^{\text{Student1}}$ stops. After working for two years, John quits his job and goes back to university to pursue his master's degree. At that time, the stopped $m_{\text{John}}^{\text{Student1}}$ resumes, and the $m_{\text{John}}^{\text{Programmer1}}$ stops.

3.3.3 Dynamic Extension

To support dynamic multityping, the dynamic state machine model applies a dynamic extension mechanism which permits us to model the connections among the histories of a temporal object in different environments. The basic idea of dynamic extension is that when a new primary machine is created, if in the corresponding multitype composite machine there exits a primary machine whose type-version is the super type-version of this new primary machine, then this existing primary machine is taken as the basic machine of the new one. The newly created primary machine is called the extension machine of the basic machine. The history of the extension machine begins at the time of its creation, and it starts to share the history of the basic machine from this point on.

Let's go back to the previous example of modeling the person named John. When the new primary machine $m_{\text{John}}^{\text{Student1}}$ is created, it takes the existing primary machine $m_{\text{John}}^{\text{Person1}}$ as its basic machine. However, the past history of $m_{\text{John}}^{\text{Person1}}$, that existed before the $m_{\text{John}}^{\text{Student1}}$ was created, will not be shared by the $m_{\text{John}}^{\text{Student1}}$.

3.3.4 Dynamic References

By applying the dynamic reference mechanism, the dynamic state machine model can easily capture the enriched semantics of aggregation in the TOODB. In the
TOODB, the *is-attribute-of* relationship between two temporal objects becomes a time function. Let's consider the concept of composition, a special case of aggregation, where a composite object is related to the components that make it up via the *is-part-of* relationship. This relationship is a special case of the *is-attribute-of* relationship. For example, a car can be considered as the composition of the engine, body, tires and transmission system. In the traditional OODB, the *is-part-of* relationship is modeled by a composite object and component objects. Because of the lack of temporal modeling, the connection between a component object and the composite object is fixed: the component objects are created together with the composite object, and they are always connected to each other. This causes a semantic gap between the object in the database and the entity in reality. For example, for most car manufacturers, body parts and engine parts are designed and fabricated independently. Also in real situations, cars may be refitted with new engines. On the other hand, an engine itself may experience several stages: a new engine, an engine in use, a reconditioned engine, and a worn out engine. Also an engine may be connected to different cars at different times, e.g. the new engine is connected to car I, but after it is reconditioned, it is connected to car J. In the dynamic state machine model, the car and engine are treated as independent multitype composite machine; both of them consist of different primary machines and maintain their own histories. The *is-part-of* relationship makes them related to each other in a certain time period. This experience is recorded in their histories.
3.3.5 Trace and Behavior History

We define the execution of a transition as an event. From user's point of view, an event carries out either a query or an update on the "database state" of a dynamic state machine. In our model, the database state determined by the event. Whenever an event occurs the "database state" of the dynamic state machine is updated. That is, we emphasize behaviors. The behavior history of a primary machine is represented by the traces consisting of the events associated with the primary machine since its creation. The behavior history of a composite machine consists of all the behavior histories of its component machines.

The behavior history provides more information than that provided by the history which only provide attribute values. For example, we may model a "cursor" as an object with an attribute "location" and four methods: "move up", "move down", "move left" and "move right". From a behavior history, we can analyze cursor movement as a function of time. This allows us to know not only where the cursor is, but also how the cursor moves.

3.3.6 Retroactive Affection Identification

Retroactive updates occur often in many applications. For example, retroactive salary adjustments, retroactive design corrections. A retroactive update may invalidate some of the actions that took place before the update. That is a retroactive update may result in a cascade of invalidations. The first step in handling retroactive updates is to identify the actions and data that are affected by a retroactive update.
In applications that involve large amounts of historical data, the system rather than the __ser takes care of the retroactive update affection identification is more preferable. Including the concepts and rules with regard to retroactive update affection identification in our model serves this purpose.

Let's consider an example. We assume that John has a checking account in the bank ANYNAME, and that John's monthly salary is 1600 dollars which will be directly deposited into John' bank account on payday once every two weeks. We also assume that the bank ANYNAME has the following conventions: (1) if every day during a month, the balance of a checking account is equal to or greater than 600 dollars, then there is no charge on all the checks or cash transactions made by the client during that month. Otherwise, each check or cash transaction will be charged a 1 dollar service fee; (2) if an account is overwithdrawn, then a 5 dollars service fee will be charged and the amount of money that is overwithdrawn will be charged on a 20 percent per year interest rate. Let's assume that somewhere between March 1, 1987, and March 31, 1987, the following sequence of events took place:

(1) On March 1, 1987, John's checking account had a balance of 600 dollars.

(2) On March 3, 1987, John cashed 300 dollars from his checking account.

(3) On March 7, 1987, John purchased a dishwasher by writing a check for 450 dollars post-dated to March 14, 1987 which is his payday.

(4) On March 14, 1987, because the company where John was working made a mistake, John's salary payment was not deposited into his bank account so that when his check was cashed, the account was overwithdrawn by 150
dollars.

(5) On March 15, 1987, John cashed 500 dollars from his account because he did not know his payment was not available, so that the account was further overwithdrawn.

(6) On March 25, 1987, John went to the branch of the bank to request a larger credit limit for his credit card, and got a rejection because of his record of overwithdrawals.

The event (6) made John realize there was something wrong with his company's salary payment. On March 26, 1989, the company corrected the mistake by issuing a retroactive deposit John’s salary into his account effective on March 14. Because the database of the bank ANYNAME had the retroactive updates affection identification function,¹ the events (4), (5), and (6) were identified as the affected events. Therefore, John avoided the charges due to overwithdrawing and got an extended credit limit of his credit card.

3.3.7 Behavior Constraints

The data stored in a database is used to model real world entities. To insure that only correct data is stored, it must satisfy certain types of consistency constraints, e.g. the number of working hours in one week of an employee may not exceed 40 hours.

However, in the TOODB, the consistency constraints also have to include behavior

¹This function actually belongs to the futuristic databases
constraints, which specify the temporal relationships among the events. Whether data is correct or not depends not only on the scope of the data but also on the temporal relationships of the events which produce the data.

For example, in a OODB for a library, we assume that the type “book” is defined as follows:

Type: Book
Supertype: Print-Material
Attributes:
  BID: Book-Id-type
  Call-No.: Call-No-type
  Title: Title-type
  Author: Author-type
  Publisher: Publisher-type
Methods:
  Borrow (BID)
  Return (BID)
  Renew (BID)

The system should not permit the following event sequences to be stored into database, because they have obvious semantic mistakes.

\((Borrow(b_1), t_1), (Borrow(b_1), t_2), (Return(b_1), t_3), (Renew(b_1), t_4)\)

\((Borrow(b_2), t_1), (Return(b_2), t_2), (Return(b_2), t_3), (Renew(b_2), t_4)\)
Therefore, behavior constraints are an important part of temporal modeling in our data model. They allow the modeling of consistency beyond what is allowed by the traditional type system, and capture more semantics of application domains which are missed by the traditional OODB. Their function cannot be replaced by other things.

In summary, the dynamic state machine model not only includes all the modeling power of all the existing approaches related to object evolution, but also captures the generic temporal relationships of objects and basic dynamic features of object evolution which so far have not been dealt with by any existing approaches.
Chapter 4

DYNAMIC STATE LOGIC

4.1 INTRODUCTION

In this chapter we will introduce a linear time temporal logic, called the Dynamic State Logic (DSL), for specifying behavior constraints of dynamic state machines. The DSL is an extension of the linear time temporal logic developed by Z. Manna and A. Pnueli [MP83]. We assume familiarity with their work presented in [MP83] and only stress the extensions.

There are two aspects in which the DSL departs from the temporal logic presented in [MP83]. First, the DSL needs to consider the past as well as the future, in order to trace the histories of objects. Thus, we also use the following temporal operators: $\ominus$(Previous), $\Xi$(Past-always), $\mathtt{S}$(Since). Correspondingly, we extend the definitions of term and formula. Second, unlike the formal system presented in [MP83], the formal system of the DSL is a composite system with a hierarchical structure, which consists of many component formal systems. This corresponds to the composite structure of dynamic state machines.

We will adopt the following conventions: $M$ represents a composite state machine,
$m$ or $m^M_p$ represents a component machine of a composite machine, and $\mu$ represents a dynamic state machine (either a primary machine or a composite machine).

4.2 DYNAMIC STATE LOGIC LANGUAGE

Definition 4-1:

In the DSL, we use the following alphabets

Logical Symbols:

Connectives: $\neg$, $\rightarrow$

Universal quantifier: $\forall$

Parentheses: $()$

Temporal Operators: $\bigcirc$(Next), $\blacksquare$(Henceforth), $\bigcirc\bigcirc$(Until), $\ominus$(Previous), $\Theta$(Past-always), $S$(Since)

Other Symbols:

All the symbols used in the definition of a dynamic state machine (i.e., $I_{tp}$, $I_e$, $I_r$, $I_{tv}$, $I_{id}$, $I_r$ and $I_{ov}$) are also used as symbols in the DSL with the same meaning as they have in the dynamic state machine model. To distinguish DSL symbols, we will put a $\sim$ over them.

DSL is a multi-sorted logic with a set $\Psi$ of sorts. Each sort is a domain for a variable in the variable set $V_\mu$ of a dynamic state machine $\mu$. Transition set $T_\mu$, event set $E_\mu$, and trace set $R_\mu$ of a dynamic state machine are also considered as sorts in $\Psi$. Variables are divided into global variables and local variables. $\tilde{r}$, $\tilde{r}$, and $\tilde{e}'$ are used
as local variable symbols of sorts $T_\mu$, $R_\mu$, and $E_\mu$, respectively.

For each n-tuple of sorts there are predicate and function symbols, and for each sort there is a set of constant symbols. In addition to the functions and relations defined in Chapter 3, the following predicates will also be used in the DSL.

$Parent-Of(\varphi^1, \varphi^2)$: $\varphi^1$ is the super type-version of $\varphi^2$.

$Ancestor(\varphi^1, \varphi^2)$: $\varphi^1$ is the ancestor type-version of $\varphi^2$.

$Belong\_To(\varphi^i, \varphi)$: $\varphi^i$ is a type-version of a type $\varphi$.

$Compo(m, M)$: $m$ is a component machine of the composite machine $M$.

$Typed(m, \varphi^i)$: the primary machine $m$ has the type-version $\varphi^i$.

$Is\_a(m_1, m_2)$: the primary machine $m_1$ is an extended machine of the primary machine $m_2$.

$In(m, b)$: a dynamic state machine identifier $m$ occurs in the component machine variable $b$ of the manager machine of a composite machine.

$Stopped(\mu)$: the dynamic state machine $\mu$ is stopped.

We will use predicate $\tilde{e}' = \tilde{\tau}$ to represent an event which executes a transition $\tau$.

Definition 4-2:

We define terms as follows:

1. all the constant symbols are terms;

2. all the variable symbols are terms;
3. \( f(t_1, \ldots, t_n) \) is a term, if \( t_1, \ldots, t_n \) are terms and \( f \) is an n-ary function symbol;

4. \( \bigcirc t \) is a term, if \( t \) is a term;

5. \( \Theta t \) is a term, if \( t \) is a term.

Definition 4-3:

We define formulas as follows:

1. An atomic formula is a string of the form \( p(t_1, \ldots, t_n) \), where \( p \) is an n-ary predicate symbol and \( t_1, \ldots, t_n \) are terms. We write \( t_1 = t_2 \) instead of \( =(t_1, t_2) \).

2. If \( w_1 \) is a formula, then \( \lnot w \), \( \bigcirc w \), \( \Box w \), \( \Theta w \), and \( \exists w \) are formulas.

3. If \( w_1 \) and \( w_2 \) are formulas, then \( (w_1 \rightarrow w_2) \), \( (w_1 \lor w_2) \) and \( (w_1 \land w_2) \) are formulas.

4. If \( \nu \) is a global variable and \( w \) is a formula, then \( (\forall \nu : w) \) is a formula.

4.3 SEMANTICS OF DSL

Definition 4-4:

The interpretation \( I_\mu = (S_\mu, A_\mu, \hat{\sigma}_\mu) \), where \( S_\mu \) is the fixed structure for \( \mu \), which assigns to all global constant symbols, function symbols and predicate symbols the corresponding constants, functions and relations used in the description of \( \mu \); \( A_\mu \) is a global assignment which assigns a value to each global free variable; \( \hat{\sigma}_\mu \) is a legal
state sequence in $\hat{\Sigma}_\mu$, as defined in Chapter 3, which assigns values to all local free variables.

If we assume that a composite machine $M$ consists of a set of component machines $m_i$, then $I_M = \bigcup I_{m_i}$. For every $m_i$, we have $I_{m_i} = (S_{m_i}, A_{m_i}, \hat{\sigma}_{m_i})$, where $S_{m_i} = S_{M\downarrow m_i}$, $A_{m_i} = A_{M\downarrow m_i}$, and $\hat{\sigma}_{m_i} = \hat{\sigma}_{M\downarrow m_i}$. $S_{m_i} = S_{M\downarrow m_i}$ means that $S_{m_i}$ is formed by those assignments in the $S_M$ which are used only for the constants, functions and relations for describing the $m_i$. $A_{m_i} = A_{M\downarrow m_i}$ means that $A_{m_i}$ is formed by the global assignments in the $A_M$ which assign values to the global variables of $m_i$. We claim that $I_{m_i} = I_{M\downarrow m_i}$.

The set of all the interpretations which have the same global assignment $A_\mu$ but different legal state sequences $\hat{\sigma}_\mu$ is denoted by $\Pi_\mu$. $\Pi_\mu$ is exactly that class of interpretations which are models for describing the properties and behaviors of the dynamic state machine $\mu$.

Definition 4-5:

Let $I_\mu = (S_\mu, A_\mu, \hat{\sigma}_\mu)$ be an interpretation. Then we define the following:

1. 

\[
I_\mu = \begin{cases} 
I_M, & \text{for } \mu = M \\
I_{m_i}, & \text{for } \mu = m_i
\end{cases}
\]

2. $I_\mu[f]$ is a function assigned to the function symbol $f$ by $I_\mu$, and $I_\mu[f] = S_\mu(f)$.

3. $I_\mu[p]$ is the relation assigned to the predicate symbol $p$ by $I$, and $I_\mu[p] = S_\mu(p)$. 

57
4. \( I^{(k)} = (S_\mu, A_\mu, \hat{\sigma}_\mu^{(k)}) \), where \( k \geq 0 \), \( \hat{\sigma}_\mu^{(k)} = \hat{\sigma}_\mu^k, \hat{\sigma}_\mu^{k+1}, \ldots \) which is the \( k \)-truncated suffix of the \( \hat{\sigma}_\mu = \hat{\sigma}_\mu^0, \hat{\sigma}_\mu^1, \ldots \), i.e., \( I^{(k)} \) is the interpretation obtained from \( I_\mu \) by replacing \( \hat{\sigma}_\mu \) with \( \hat{\sigma}_\mu^{(k)} \). \( I_\mu \) is the abbreviation of \( I^{(0)}_\mu \).

5. \( I_\mu(v/d) = (S_\mu, A_\mu(v/d), \hat{\sigma}_\mu) \), where \( A_\mu(v/d) \) is the assignment which maps \( v \) to \( d \) and agrees with \( A_\mu \) on all other global variables distinct from \( v \).

**Definition 4-6:**

We associate with each interpretation \( I_\mu \) and every term \( t \) an element \( I_\mu[t] \), by using induction on terms.

1. for each constant symbol \( c \), \( I_\mu[c] = S_\mu(c) \),
2. for each global variable \( v \), \( I_\mu[v] = A_\mu(v) \),
3. for each local variable \( x \), \( I_\mu^{(k)}[x] = \hat{\sigma}_\mu^k(x) \), where \( k \geq 0 \),
4. for any term \( t \), \( I_\mu^{(k)}[\overline{t}] = I_\mu^{(k+1)}[\overline{t}] \), \( (k \geq 0) \),
5. for any term \( t \), \( I_\mu^{(k)}[\overline{t}] = I_\mu^{(k-1)}[\overline{t}] \), \( (k \geq 1) \),
6. for each n-ary function symbol \( f \) and terms \( t_1, \ldots, t_n \) (all of appropriate sorts), \( I_\mu[f(t_1, \ldots, t_n)] = I_\mu[f](I_\mu[t_1], \ldots, I_\mu[t_n]) \).

**Definition 4-7:**

For all interpretations \( I_\mu^{(k)} \), we use \( \models_{I_\mu^{(k)}} w \) to represent that \( I_\mu^{(k)} \) satisfies \( w \), where \( w \) is a formula. We define the satisfaction inductively as follows:

1. \( \models_{I_\mu^{(k)}} (t_1 = t_2) \) iff \( I_\mu^{(k)}[t_1] = I_\mu^{(k)}[t_2] \),

58
2. \( \models_{\mu}^k p(t_1, ..., t_n) \) iff \( (I_\mu^{(k)}[t_1], ..., I_\mu^{(k)}[t_n]) \in I_\mu^{(k)}[p] \),

3. \( \models_{\mu}^k (\neg w) \) iff \( \not\models_{\mu}^k w \),

4. \( \models_{\mu}^k (w_1 \rightarrow w_2) \) iff \( \models_{\mu}^k w_2 \) or \( \models_{\mu}^k (\neg w_1) \),

5. \( \models_{\mu}^k (\bigcirc w) \) iff \( \models_{\mu}^{k+1} w \),

6. \( \models_{\mu}^k (\Box w) \) iff for every \( i \geq k \), \( \models_{\mu}^i w \),

7. \( \models_{\mu}^k (w_1 \cup w_2) \) iff for some \( j \geq k \), \( \models_{\mu}^j w_2 \) and for all \( i, k \leq i < j \), \( \models_{\mu}^i w_1 \),

8. \( \models_{\mu}^k (\Theta w) \) iff \( \models_{\mu}^{k-1} w \), where \( k \geq 1 \),

9. \( \models_{\mu}^k (\exists w) \) iff every \( 0 \leq i \leq k \), such that \( \models_{\mu}^i w \),

10. \( \models_{\mu}^k (w_1 \mathop{S} w_2) \) iff for some \( j, 0 \leq j \leq k \), \( \models_{\mu}^j w_2 \) and for all \( i, j < i \leq k \), \( \models_{\mu}^i w_1 \),

11. \( \models_{\mu}^k (\forall v : w) \) iff for all \( d \in D_j \), \( \models_{\mu}^{k}(v/d) w \) where \( v \) and its quantifier are of sort \( j \).

Definition 4-8:

If for any \( \models_{\mu}^k = (S_\mu, A_\mu, \sigma_\mu^{(k)}) \) in \( \Pi_\mu \), \( \models_{\mu}^k w \), where \( k \geq 0 \), we say \( w \) is \( \Pi_\mu \)-valid, and denote it as \( \models_{\Pi_\mu} w \).

**Abbreviations:**

- **disjunction** \( w_1 \lor w_2 \) for \( \neg w_1 \rightarrow w_2 \)
- **conjunction** \( w_1 \land w_2 \) for \( \neg(w_1 \rightarrow \neg w_2) \)
- **bicondition** \( w_1 \leftrightarrow w_2 \) for \( (w_1 \rightarrow w_2) \land (w_2 \rightarrow w_1) \)
eventually \( \Diamond w \) for \( \neg \Box \neg w \)
past-sometime \( \nabla w \) for \( \neg \exists \neg w \)
before \( w_1 B w_2 \) for \( \neg (\neg w_1 U w_2) \)
after \( w_1 A w_2 \) for \( \neg (\neg w_1 S w_2) \)
existential \( \exists v : w \) for \( \neg \forall v : \neg w \)

4.4 PROOF SYSTEM OF DSL

The proof system of the DSL consists of four parts.

Part (a) of the DSL proof system formalizes the pure temporal logic properties of sequences in general. In this part, all of the axioms and inference rules which are related only to the future time are inherited from the proof system of [MP83]. Since we have some temporal operators related to the past time, here, we list ten new axioms and one inference rule. All these extended axioms and the inference rule of Part (a) will be denoted by the prefix "APA" or "APR".

\[(APA1) \quad \vdash \mu \neg \nabla w \leftrightarrow \exists \neg w\]
\[(APA2) \quad \vdash \mu (\exists w_1 \to w_2) \to (\exists w_1 \to \exists w_2)\]
\[(APA3) \quad \vdash \mu \exists w \to w\]
\[(APA4) \quad \vdash \mu \neg \Theta w \leftrightarrow \Theta \neg w\]
\[(APA5) \quad \vdash \mu \Theta (w_1 \to w_2) \to (\Theta w_1 \to \Theta w_2)\]
\[(APA6) \quad \vdash \mu \exists w \to \Theta w\]
\[(APA7) \quad \vdash \mu \exists w \to \Theta \exists w\]
\[(APA8) \quad \vdash \mu (w \to \Theta w) \to (w \rightarrow \exists w)\]
(APA9) \[ \vdash_{\mu} (w_1 \ S \ w_2) \leftrightarrow (w_2 \lor (w_1 \land \Theta(w_1 \ S \ w_2))) \]

(APA10) \[ \vdash_{\mu} (w_1 \ S \ w_2) \rightarrow \nabla w_2 \]

In addition to axioms, a new inference rule is also added as follows:

(APR1) \[ \Xi I - \Xi Insertion \]

\[ \vdash_{\mu} w \]

\[ \vdash_{\mu} \Xi w \]

Part (b) of the DSL proof system consists of the axioms that describe the properties of primary machines defined by a type-version. These axioms can be considered as domain axioms, since all the primary machines defined by a type-version form a domain. The properties of a domain can be written as formulas of DSL. If a formula of the DSL is valid for a domain, then that formula is an axiom of part (b).

For example, the definition of a type-version Queue\(^1\) contains the following:

\[ V = \{\text{header}, \text{tail}, \text{length}\} \]

\[ T_r = \{\text{Insertion}, \text{Deletion}\} \]

where the type-versions of three attribute variables are Integers, and the two transitions are defined as follows:

\[ \text{Insertion} : \text{tail} := \text{tail} + 1; \text{length} := \text{length} + 1 \]

\[ \text{Deletion} : \text{header} := \text{header} - 1; \text{length} := \text{length} - 1 \]

To express the properties of the primary machines defined by the Queue\(^1\), we have the following DSL formulas:
$\vdash_{\mathcal{M}_{Queue}} \bar{e}' = \text{Insertion} \rightarrow ((\text{length} = \Theta \text{length} + 1) \wedge (\text{tail} = \Theta \text{tail} + 1)),$

$\vdash_{\mathcal{M}_{Queue}} \bar{e}' = \text{Deletion} \rightarrow \Theta \text{length} \geq 1,$

Based on the semantics of the queue and the properties of integers, it is easy to show the formulas above are valid for primary machines with type-version $Queue^1$, so that they are axioms of Part (b). In fact, these formulas can be considered as the constraint part of the definition of type-version $Queue^1$.

All the axioms of Part (b) will be denoted by the prefix “BA”.

Part (c) of the DSL proof system contains axioms and inference rules that describe the relationships among the component machines of composite state machines. Part (c) can be further divided into two subparts: (i) axioms and inference rules for the $i$-$a$ hierarchy, (ii) axioms that describe the special relationships among component machines and the relationships between a specific composite machine and its component machines.

All the axioms and inference rules of Part (c) will be denoted by the prefix “CA” or “CR”.

The axioms and inference rules of Part (c-i) are listed as follows:

(CA1) COPA - Component Axiom

Let $\tilde{m}$ be a component machine of a composite machine $\tilde{M}$ and $\tilde{b}$ be a component machine variable of $\tilde{M}$, then

$\vdash_{\tilde{M}} \text{Comp}(\tilde{m}, \tilde{M}) \leftrightarrow \text{In}(\tilde{m}, \tilde{b})$

(CR1) COMP - Composition Rule

Let $\tilde{M}$ be a composite machine, $\tilde{m}$ be a component machine of the $\tilde{M}$, and $\omega$ be
a temporal formula, then

\[ \vdash_{\bar{m}} w \]
\[ \vdash_{\bar{M}} \text{Compo}(\bar{m}, \bar{M}) \]
\[ \vdash_{\bar{M}} w \]

(CA2) TATA - Type and Type-version Axiom

Let \( \bar{\varphi}^i \) be a type-version, \( \bar{\varphi}_1 \) and \( \bar{\varphi}_2 \) be two different types, then

\[ \vdash_{\bar{M}_{\text{tp}}} \text{Belong To}(\bar{\varphi}_1, \bar{\varphi}_1) \rightarrow \neg \text{Belong To}(\bar{\varphi}_1, \bar{\varphi}_2) \]

(CA3) SUTA - Super Type-version Axiom

Let \( \bar{\varphi}_1, \bar{\varphi}_2 \) and \( \bar{\varphi}_3 \) be three different type-versions, and let \( \bar{\varphi}_2,\text{.sup} \) be a variable which value is the identifier of its super type-version, then

1. \[ \vdash_{\bar{M}} \text{Parent Of}(\bar{\varphi}_1, \bar{\varphi}_2) \leftrightarrow (\bar{\varphi}_2,\text{.sup} = \bar{\varphi}_1) \]
2. \[ \vdash_{\bar{M}} \text{Parent Of}(\bar{\varphi}_1, \bar{\varphi}_2) \rightarrow \neg \text{Parent Of}(\bar{\varphi}_2, \bar{\varphi}_1) \]
3. \[ \vdash_{\bar{M}} \text{Parent Of}(\bar{\varphi}_1, \bar{\varphi}_3) \rightarrow \neg \text{Parent Of}(\bar{\varphi}_2, \bar{\varphi}_3) \]

(CA4) ASTA - Ancestor Type-version Axiom

Let \( \bar{\varphi}_1, \bar{\varphi}_2 \) and \( \bar{\varphi}_3 \) be three different type-versions. We have

1. \[ \vdash_{\bar{M}} \text{Parent Of}(\bar{\varphi}_1, \bar{\varphi}_2) \rightarrow \text{Ancestor}(\bar{\varphi}_1, \bar{\varphi}_3) \]
2. \[ \vdash_{\bar{M}} \text{Ancestor}(\bar{\varphi}_1, \bar{\varphi}_2) \rightarrow \neg \text{Ancestor}(\bar{\varphi}_2, \bar{\varphi}_1) \]
3. \[ \vdash_{\bar{M}} (\text{Ancestor}(\bar{\varphi}_1, \bar{\varphi}_2) \land \text{Ancestor}(\bar{\varphi}_2, \bar{\varphi}_3)) \rightarrow \text{Ancestor}(\bar{\varphi}_1, \bar{\varphi}_3) \]

63
(CA5) TYP A - Typed Axiom

Let \( \tilde{m} \) be a primary machine, \( \tilde{p}^1 \) be a type-version and \( \tilde{m}.\varphi^i \) be a variable which value is the identifier of \( m \)'s type-version, then

\[
\vdash_{\tilde{m}} \ T_{\text{Typed}}(\tilde{m}, \tilde{p}^1) \rightarrow \tilde{m}.\varphi^i = \tilde{p}^1
\]

(CA6) ISAA - Is-a Axiom

Let \( \tilde{m}_1 \) and \( \tilde{m}_2 \) be two different primary machines, \( \tilde{p}^1 \) and \( \tilde{p}^2 \) be two different type-versions, then

1. \( \vdash_{\tilde{M}} \ Is.a(\tilde{m}_1, \tilde{m}_2) \rightarrow ((\tilde{m}_1.\varphi^i = \tilde{p}^1) \land (\tilde{m}_2.\varphi^i = \tilde{p}^2)) \land ((\tilde{P}_{\text{arent.O}}(\tilde{p}^1, \tilde{p}^3)) \lor (\tilde{A}_{\text{cestor}}(\tilde{p}^1, \tilde{p}^2))) \)

2. \( \vdash_{\tilde{M}} \ \tilde{I}s.a(\tilde{m}_1, \tilde{m}_2) \rightarrow \neg \tilde{I}s.a(\tilde{m}_2, \tilde{m}_1) \)

3. \( \vdash_{\tilde{M}} \ (\tilde{I}s.a(\tilde{m}_1, \tilde{m}_2) \land \tilde{I}s.a(\tilde{m}_2, \tilde{m}_3)) \rightarrow \tilde{I}s.a(\tilde{m}_1, \tilde{m}_3) \)

(CR2) ISAR - Is-A Rule

Let \( \tilde{m}_1 \) and \( \tilde{m}_2 \) be two primary machines in the composite machine \( \tilde{M} \), let \( \tilde{p}^1 \) and \( \tilde{p}^2 \) be two different type-versions, and let \( w \) be any temporal formula, then

\[
\vdash_{\tilde{m}_1} w
\]

\[
\vdash_{\tilde{M}} w \ A (\tilde{e}' = \tilde{C}_{\text{reate}}(\tilde{m}_2))
\]

\[
\vdash_{\tilde{M}} \tilde{C}_{\text{ompo}}(\tilde{m}_1, \tilde{M}) \land \tilde{C}_{\text{ompo}}(\tilde{m}_2, \tilde{M})
\]

\[
\vdash_{\tilde{M}} \tilde{I}s.a(\tilde{m}_2, \tilde{m}_1)
\]

\[
\vdash_{\tilde{M}} w
\]

The axioms and inference rules of Part (c-ii) are listed as follows:
(CR3) CREA - Creation Rule

Let \( \bar{m} \) be a component machine of a composite machine \( \bar{M} \), and \( \bar{\tau}_0 \) be the initialization transition of the \( \bar{m} \), then

\[
\vdash_{\bar{M}} \bar{\epsilon}' = \text{Create}(\bar{m})
\]

\[\vdash_{\bar{m}} \bar{\epsilon}' = \bar{\tau}_0\]

(CA7) STOP - Stop Axiom

Let \( \bar{m}, \bar{m}_1 \) and \( \bar{m}_2 \) be component machines of a composite machine \( \bar{M} \), then

1. \[\vdash_{\bar{M}} \bar{\epsilon}' = \text{Stop}(\bar{m}) \rightarrow \text{Stopped}(\bar{m})\]

2. \[\vdash_{\bar{M}} (\text{Stopped}(\bar{m}_2) \land \text{Is.a}(\bar{m}_1, \bar{m}_2)) \rightarrow \text{Stopped}(\bar{m}_1)\]

3. \[\vdash_{\bar{M}} (\text{Stopped}(\bar{M}) \land \text{Compo}(\bar{m}, \bar{M})) \rightarrow \text{Stopped}(\bar{m})\]

(CA8) RESU - Resume Axiom

Let \( \bar{m}, \bar{m}_1 \) and \( \bar{m}_2 \) be component machines of a composite machine \( \bar{M} \), then

1. \[\vdash_{\bar{M}} \bar{\epsilon}' = \text{Resume}(\bar{m}) \rightarrow \neg\text{Stopped}(\bar{m})\]

2. \[\vdash_{\bar{M}} (\bar{\epsilon}' = \text{Resume}(\bar{m}_1) \land \text{Is.a}(\bar{m}_1, \bar{m}_2) \land \text{Stopped}(\bar{m}_2)) \rightarrow \bar{\epsilon}' = \text{Resume}(\bar{m}_2)\]

In Part (c-ii) we allow additional new axioms according to concrete applications. For example, assume that a multitype composite machine \( M \) consists of two component machines \( m_1 \) and \( m_2 \), and that \( V_{m_1} = (x_{11}, x_{12}) \), \( T_{m_1} = (\tau_{11}, \tau_{12}) \), \( V_{m_2} = (x_{21}, x_{22}) \), \( T_{m_2} = (\tau_{21}, \tau_{22}) \). Suppose we have the following requirements: (i)
when $x_{11} = 5$, the execution of $\tau_{11}$ will trigger the execution of $\tau_{22}$, (ii) $\tau_{12}$ is allowed to be executed once.

These two requirements can be translated into formulas of DSL as follows:

$$(F_1) \quad \vdash_{\mathcal{M}} (\bar{e}' = \bar{\tau}_{11} \land \bar{x}_{11} = \bar{5}) \rightarrow \bar{e}' = \bar{\tau}_{22}$$

$$(F_2) \quad \vdash_{\mathcal{M}} \nabla (\bar{e}' = \bar{\tau}_{11}) \rightarrow \neg(\bar{e}' = \bar{\tau}_{11})$$

Obviously, according to the requirements above, we can easily show that $F_1$ and $F_2$ are $\Pi_{\mathcal{M}}$-valid, so that we can treat them as axioms.

Part (d) of the DSL proof system consists of all the axioms and inference rules that reflect the common properties of dynamic state machines. All the axioms and inference rules of Part (d) will be denoted by the prefix "DA" or "DR" respectively. The axioms and rules of Part (d) are as follows:

(DR1) INIT - Initialization Rule

Let $\bar{\tau}_0$ be the initialization transition of a dynamic state machine $\bar{\mu}$ and $\bar{w}$ be a state formula (i.e., $\bar{w}$ does not contains any temporal operators), then

$$\vdash_{\bar{\mu}} \bar{w}S(\bar{e}' = \bar{\tau}_0)$$

$$\vdash_{\bar{\mu}} \bar{\Xi}w$$

(DA1) TEXT- Trace Extension Axiom

Let $\bar{\tau}$ be a trace and $\bar{\tau}$ is the transition executed by the most recent event, then

$$\vdash_{\bar{\mu}} (\bar{e}' = \bar{\tau}) \rightarrow (\bar{\tau} = \Theta \bar{\tau} \cdot \bar{e}')$$

(DA2) OSTE - One State Transferring Event Axiom
Let \( \tilde{\tau}_1 \) and \( \tilde{\tau}_2 \) be two transitions in the \( T_\mu \) which is the transition set of a dynamic state machine \( \mu \), then
\[
\vdash_{\mu} (\tilde{e}' = \tilde{\tau}_1) \rightarrow (\tilde{e}' = \tilde{\tau}_2)
\]

**Definition 4-9:**

The proof system of DSL is sound iff \( \vdash_{\mu} w \) implies \( \models_{\Pi_\mu} w \) for any \( w \).

**Theorem 4.1:**

The proof system of DSL is sound.

**Proof:**

Since \( P(a) \) is extended from the Z. Manna and A. Pnueli's linear time temporal logic, we need only prove that all the axioms (AAP1) to (AAP10) and the inference rule of \( \Xi \) Insertion are \( \Pi_\mu \)-valid. The proof of these axioms and inference rule are as follow.

\( \Pi_\mu \)-validity of (APA1):
\[
\models_{\Pi_\mu} \neg \forall w \iff \models_{\Pi_\mu} \neg (\neg \Xi \neg w) \\
\iff \models_{\Pi_\mu} \Xi \neg w
\]

\( \Pi_\mu \)-validity of (APA2):
\[
\models_{\Pi_\mu} \Xi (w_1 \rightarrow w_2) \Rightarrow \models_{\Pi_\mu} (w_1 \rightarrow w_2) \\
\Rightarrow \models_{\Pi_\mu} (w_1 \rightarrow w_2), \quad 0 \leq i \leq k
\]
\[ \Rightarrow \models_{j^{(i)}} \neg w_1 \text{ or } \models_{j^{(i)}} w_2, \ 0 \leq i \leq k \]

\[ \Rightarrow \models_{j^{(k)}} \exists w_1 \text{ or } \models_{j^{(k)}} \exists w_2, \]

\[ \Rightarrow \models_{\Pi_{\mu}} \exists w_1 \text{ or } \models_{\Pi_{\mu}} \exists w_2, \]

\[ \Rightarrow \models_{\Pi_{\mu}} \neg \exists w_1 \text{ or } \models_{\Pi_{\mu}} \exists w_2, \]

\[ \Rightarrow \models_{\Pi_{\mu}} (\exists w_1 \rightarrow \exists w_2) \]

**$\Pi_{\mu}$-validity of (APA3):**

\[ \models_{\Pi_{\mu}} \exists w \Rightarrow \models_{j^{(k)}} \exists w \]

\[ \Rightarrow \models_{j^{(i)}} w, \ 0 \leq i \leq k \]

\[ \Rightarrow \models_{j^{(k)}} w, \ \text{for } i = k \]

\[ \Rightarrow \models_{\Pi_{\mu}} w \]

**$\Pi_{\mu}$-validity of (APA4):**

\[ \models_{\Pi_{\mu}} \neg \Theta w \iff \models_{j^{(k)}} \neg \Theta w \ k \geq 1 \]

\[ \iff \neg \models_{j^{(k)}} \Theta w \ k \geq 1 \]

\[ \iff \neg \models_{j^{(k+1)}} w \ k \geq 1 \]

\[ \Rightarrow \models_{j^{(k+1)}} \neg w \ k \geq 1 \]

\[ \iff \models_{j^{(k)}} \Theta \neg w \ k \geq 1 \]

\[ \Rightarrow \models_{\Pi_{\mu}} \Theta \neg w \]

**$\Pi_{\mu}$-validity of (APA5):**

\[ \models_{\Pi_{\mu}} \Theta (w_1 \rightarrow w_2) \Rightarrow \models_{j^{(k)}} \Theta (w_1 \rightarrow w_2), \ k \geq 1 \]

68
\[\Rightarrow \models_{\mu^{k-1}} (w_1 \rightarrow w_2), \quad k \geq 1\]
\[\Rightarrow \models_{\mu^{k-1}} \neg w_1 \lor \models_{\mu^{k-1}} w_2, \quad k \geq 1\]
\[\Rightarrow \models_{\mu^{k}} \theta \neg w_1 \lor \models_{\mu^{k}} \theta w_2, \quad k \geq 1\]
\[\Rightarrow \models_{\Pi_{\mu}} \theta \neg w_1 \lor \models_{\Pi_{\mu}} \theta w_2,\]
\[\Rightarrow \models_{\Pi_{\mu}} \theta w_1 \lor \models_{\Pi_{\mu}} \theta w_2, \]
\[\Rightarrow \models_{\Pi_{\mu}} (\theta w_1 \rightarrow \theta w_2)\]

**$\Pi_{\mu}$-validity of $(APA6)$:**

\[\models_{\Pi_{\mu}} \Sigma w \quad \Rightarrow \quad \models_{\mu^{i}} \Sigma w\]
\[\Rightarrow \quad \models_{\mu^{i}} w, \quad 0 \leq i \leq k\]
\[\Rightarrow \quad \models_{\mu^{(i-1)}} w, \quad i = (k - 1)\]
\[\Rightarrow \quad \models_{\mu^{k}} \theta w\]
\[\Rightarrow \quad \models_{\Pi_{\mu}} \theta w\]

**$\Pi_{\mu}$-validity of $(APA7)$:**

\[\models_{\Pi_{\mu}} \Sigma w \quad \Rightarrow \quad \models_{\mu^{k-1}} \Sigma w, \quad k \geq 1\]
\[\Rightarrow \quad \models_{\mu^{k}} \theta \Sigma w, \quad k \geq 1\]
\[\Rightarrow \quad \models_{\Pi_{\mu}} \theta \Sigma w\]

**$\Pi_{\mu}$-validity of $(APA8)$:**

\[\models_{\Pi_{\mu}} \Sigma (w \rightarrow \theta w) \quad \Rightarrow \quad \models_{\mu^{k}} \Sigma (w \rightarrow \theta w)\]
\[\Rightarrow \quad \models_{\mu^{i}} (w \rightarrow \theta w), \quad 0 \leq i \leq k\]
\[ \Rightarrow \vdash_{\mu^{(i)}} \neg w, \ 0 \leq i \leq k, \]
\[ \text{or } \vdash_{\mu^{(i)}} w, \ 1 \leq i \leq k \]
\[ \Rightarrow \vdash_{\mu^{(i)}} \neg w, \ 0 \leq i \leq k, \]
\[ \text{or } \vdash_{\mu^{(i-1)}} w, \ 1 \leq i \leq k \]
\[ \Rightarrow \vdash_{\mu^{(i)}} \neg w, \ 0 \leq i \leq k, \]
\[ \text{or } \vdash_{\mu^{(i)}} w, \ 0 \leq i \leq (k+1) \]
\[ \Rightarrow \vdash_{\mu^{(i)}} \neg w, \ 0 \leq i \leq k, \]
\[ \text{or } \vdash_{\mu^{(i)}} w, \ 0 \leq i \leq k \]
\[ \Rightarrow \vdash_{\mu^{(i)}} \neg w \text{ or } \vdash_{\mu^{(i)}} w, \ 0 \leq i \leq k \]
\[ \Rightarrow \vdash_{\mu^{(k)}} \neg w \text{ or } \vdash_{\mu^{(k)}} w, \text{ for } i = k \]
\[ \Rightarrow \vdash_{\Pi_{\mu}} \neg w \text{ or } \vdash_{\Pi_{\mu}} \Xi w \]
\[ \Rightarrow \vdash_{\Pi_{\mu}} (w \rightarrow \Xi w) \]

\( \Pi_{\mu} \)-validity of (APA9):

\[ \vdash_{\Pi_{\mu}} (w_1 Sw_2) \iff \vdash_{\mu^{(k)}} (w_1 Sw_2) \]
\[ \iff \vdash_{\mu^{(1)}} w_2, \text{ for some } j, \ 0 \leq j \leq k, \]
\[ \text{and } \vdash_{\mu^{(j)}} w_1, \text{ for all } i, j \leq i \leq k \]
\[ \iff [ \vdash_{\mu^{(j)}} w_2, \text{ for some } j, \ 0 \leq j \leq (k-1); \text{ or } \vdash_{\mu^{(k)}} w_2 ], \]
\[ \text{and } [ \vdash_{\mu^{(j)}} w_1, \text{ for all } i, j \leq i \leq (k-1); \text{ and } \vdash_{\mu^{(k)}} w_1 ] \]
\[ \iff \vdash_{\mu^{(k)}} w_2 \lor (w_1 \land \Theta(w_1 Sw_2)) \]
\[ \iff \vdash_{\Pi_{\mu}} w_2 \lor (w_1 \land \Theta(w_1 Sw_2)) \]
\( \Pi_\mu \)-validity of (APA10):

\[ \models_{\Pi_\mu} (w_1 S w_2) \Rightarrow \models_{\pi^{(j)}} w_2, \text{ for some } j, 0 \leq j \leq k, \]

and \( \models_{\pi^{(i)}} w_1, \text{ for all } i, j \leq i \leq k \)

\[ \Rightarrow \models_{\pi^{(k)}} \neg \neg w_2 \]

\[ \Rightarrow \models_{\pi^{(k)}} \neg w_2 \]

\[ \Rightarrow \models_{\Pi_\mu} \neg w_2 \]

\( \Pi_\mu \)-validity of (APR1):

\[ \models_{\Pi_\mu} w \Rightarrow \models_{\pi^{(k)}} w, \text{ for any } \pi^{(k)} \in \Pi_\mu \]

\[ \Rightarrow \models_{\pi^{(k)}} \neg w \]

\[ \Rightarrow \models_{\Pi_\mu} \neg w \]

The \( \Pi_\mu \)-validities of the axioms in Part (b) directly follow from concrete applications.

The \( \Pi_\mu \)-validities of axioms and inferences rules in the Part (c) and Part (d) can easily be proved according to the operational semantics of the dynamic state machine model. We will give the detailed proves to some axioms and inference rules. For the other axioms and rules, we only provide the proof outline.

\( \Pi_M \)-validity of (CA1):

Let \( \pi^{(k)} = (S_M, A_M, \sigma^{(k)}_M) \) be any interpretation in the \( \Pi_M \), and assume that \( \models_{\pi^{(k)}} \widehat{\text{Comp}}(\widehat{m}, \widehat{M}) \), i.e., \( \pi^{(k)} \) satisfies \( \widehat{\text{Comp}}(\widehat{m}, \widehat{M}) \). According to the definition of the multitype and general composite machines, when a new component machine \( m \)
is created, the identifier of the component machine will become one element of the value of the component machine variable \( b \). That is, \( \models I^k(\bar{m}, \bar{b}) \). Since \( I^k_M \) is any interpretation in the \( \Pi_M \), we have proved that

\[
\models_{\Pi_M} \hat{Compo}(\bar{m}, \bar{M}) \rightarrow \hat{I}(\bar{m}, \bar{M})
\]

Similarly, we can prove that

\[
\models_{\Pi_M} \hat{I}(\bar{m}, \bar{M}) \rightarrow \hat{Compo}(\bar{m}, \bar{M})
\]

Hence, the \( \Pi_M \)-validity of (CA1) has been proved.

\( \Pi_\mu \)-validity of (CR1):

Assume that \( \hat{Compo}(\bar{m}, \bar{M}) \) is provable in the \( \bar{M} \), and that \( w \) is provable in \( \bar{m} \). According to the soundness theorem, \( \models_{\Pi_M} Compo(\bar{m}, \bar{M}) \) and \( \models_{\Pi_m} w \). We must show \( \models_{\Pi_M} w \). Since \( m \) is a component machine of \( M \), according to the related definitions in Chapter 3, a component machine is a part of a composite machine. Therefore, a formula of \( \bar{m} \) is a formula of \( \bar{M} \). Thus, if \( \models_{\bar{m}} w \) is provable in \( \bar{m} \), then \( \models_{\bar{M}} w \) is provable in \( \bar{M} \). Also, for every \( I_m \), we have \( I^{(k)}_m = I^{(k)}_{M \cup m} \), where \( I^{(k)}_m \in \Pi_m \) and \( I^{(k)}_M \in \Pi_M \). Thus, from \( \models_{\Pi_m} w \), we have \( \models_{\Pi_M} w \). That is, the \( \Pi_\mu \)-validity of (CR1) has been proved.

The proof of the \( \Pi_\mu \)-validities of (CA2) is based on the convention that every type-version belongs to only one type.

The proves (CA3), (CA4), (CA5) and (CA6) are based on the related definitions and conventions in the dynamic state machine model.

\( \Pi_\mu \)-validity of (CR2):
Assume \( w \) is provable in the \( \tilde{m}_1 \); \( wA(\tilde{e}' = \tilde{C}reate(\tilde{m}_2)), \tilde{I}s.a(\tilde{m}_2, \tilde{m}_1) \), and \\
\( \tilde{C}ompo(\tilde{m}_1, \tilde{M}) \land \tilde{C}ompo(\tilde{m}_2, \tilde{M}) \) are provable in the \( \tilde{M} \). According to the soundness theorem, \( \models_{\Pi_{m_1}} w \), \( \models_{\Pi_{m}} wA(\tilde{e}' = \tilde{C}reate(\tilde{m}_2)) \), \( \models_{\Pi_{M}} \tilde{I}s.a(\tilde{m}_2, \tilde{m}_1) \) and \\
\( \models_{\Pi_{M}} \tilde{C}ompo(\tilde{m}_1, \tilde{M}) \land \tilde{C}ompo(\tilde{m}_2, \tilde{M}) \). We must show \( \models_{\Pi_{m_2}} w \).

Since \( m_2 \) is an extension machine of \( m_1 \) (or \( m_1 \) is the basic machine of \( m_2 \)), according to the dynamic extension mechanism defined in the dynamic state machine model, after \( m_2 \) is created, it will inherit everything of the \( m_1 \). This means that, after \( m_2 \) is created, (i) a formula of \( \tilde{m}_1 \) is a formula of \( \tilde{m}_2 \), so that \( w \) is provable in the \( \tilde{m}_1 \) implies \( w \) is provable in the \( \tilde{m}_2 \); (ii) \( I_{m_1}^{(k)} \) is a subset of a \( I_{m_2}^{(k)} \). However, \( I_{m_1}^{(k)} \in \Pi_{m_1} \) and \( I_{m_2}^{(k)} \in \Pi_{m_2} \). Thus, from \( \models_{\Pi_{m_1}} w \), \( \models_{\Pi_{m}} wA(\tilde{e}' = \tilde{C}reate(\tilde{m}_2)) \), \( \models_{\Pi_{M}} \tilde{I}s.a(\tilde{m}_2, \tilde{m}_1) \) and \( \models_{\Pi_{M}} \tilde{C}ompo(\tilde{m}_1, \tilde{M}) \land \tilde{C}ompo(\tilde{m}_2, \tilde{M}) \) we have \( \models_{\Pi_{m_2}} w \). That is, we have proved the \( \Pi_{\mu} \)-validity of (CR2).

The \( \Pi_{\mu} \)-validity of (CR3) directly follows Rule 3.6 and the definition of state-evolution in the dynamic state machine model.

The \( \Pi_{\mu} \)-validity of (CA7) directly follows Rule 3.2 and Rule 3.4 in the dynamic state machine model.

The \( \Pi_{\mu} \)-validity of (CA8) directly follows Rule 3.5 in the dynamic state machine model.

The \( \Pi_{M} \)-validities of (DR1), (DA1) and (DA2) directly follow from the definition of state-evolution in the dynamic state machine model.

Hence the proof system of the DSL is sound.

73
Chapter 5

BEHAVIOR CONSTRAINTS

5.1 INTRODUCTION

In the dynamic state machine model, constraints are a part of a type-version. We call constraints behavior constraints. These consist of not only the structural relationships among objects such as membership and range of data, but also the temporal relationships among the object operations such as causality and concurrency. To specify the behavior constraints, we need a language capable of describing dynamic changes. The DSL is developed to serve this purpose. Behavior constraints are specified as formulas of the DSL. Specifying behavior constraints by the DSL has an additional advantage. That is, based on the proof system of the DSL, we are able to verify whether the behavior constraints defined in type-versions can derive the expected properties. While the time complexity for searching a proof may be exponential, the time complexity for checking whether the database history of a dynamic state machine satisfies a given behavior constraint is linear in the number of event records of the dynamic state machine. Since every dynamic state machine has a database history consisting of a finite number of event records, each of which deter-
mines a state. In this chapter, we will demonstrate, in Section 5.2, some examples of how behavior constraints can be specified by the DSL formulas. Then, in Section 5.3, through another example, we will show how to verify whether the design of behavior constraints in type-versions can derive the expected system properties based on the proof system of the DSL. Finally, in Section 5.4, by presenting a set of algorithms, we will show that a certain class of DSL formulas that contains various kinds of temporal operators for past time can be checked efficiently.

5.2 SPECIFYING BEHAVIOR CONSTRAINTS BY THE DSL

We classify behavior constraints into two categories: internal and external. Internal behavior constraints describe the structural and temporal relationships between attributes or events in the same primary machine. External behavior constraints describe the structural relationships and temporal relationships between attributes or events of different primary machines. Because we are describing temporal relationships, DSL becomes a suitable language to specify behavior constraints in TOODB's. Through the following examples, we will see how to use behavior constraints to incorporate additional temporal modeling capabilities in our model.

Example 5.1:

Consider the example of the library database used in Chapter 3. To avoid incorrect event sequences to appear in the database, we can add the following behavior constraints into the definition of type of Book:

75
Constraints:

(1) $\vdash \tilde{m} \tilde{e}' = \tilde{\text{Borrow}}(\tilde{m}) \rightarrow (\Theta \tilde{e}' = \tilde{\text{Return}}(\tilde{m}) \vee \exists \tilde{e}' = \tilde{\text{Borrow}}(\tilde{m}))$

(2) $\vdash \tilde{m} \tilde{e}' = \tilde{\text{Return}}(\tilde{m}) \rightarrow (\Theta \tilde{e}' = \tilde{\text{Borrow}}(\tilde{m}) \vee \Theta \tilde{e}' = \tilde{\text{Renew}}(\tilde{m}))$

(3) $\vdash \tilde{m} \tilde{e}' = \tilde{\text{Renew}}(\tilde{m}) \rightarrow (\Theta \tilde{e}' = \tilde{\text{Borrow}}(\tilde{m}) \vee \Theta \tilde{e}' = \tilde{\text{Renew}}(\tilde{m}))$

where $\tilde{m}$ is a primary machine which represents a book. According to these behavior constraints, a book has been borrowed and not returned yet cannot be borrowed again; only a borrowed book or a renewed book can be returned.

The example above is also an example of internal constraints. The following example will show the use of external constraints.

Example 5.2:

Assume a company has three departments: Research Department, Production Department, and Sales Department. Every employee at a given time works in only one department. We design a TOODB for the company. Because the employees of different departments have different behaviors, it is natural to treat them as different types which are subtypes of a type Employee (for convenience, in this example, we treat these types as single-version types). The type hierarchy is as shown in Figure 5.1.

It is possible for an employee to transfer from one department to another department. For example, an employee who works in the Research Department is able to transfer into the Production Department. In the type $\text{Research.E}$ we give the
Figure 5.1: Type Hierarchy of Example 5.2

Following formulas:

1. \[ \vdash_{Research_E} (\Theta \tilde{e} = \text{Transfer}_To(\tilde{m}^M_{Research_E}, \tilde{Production}_E) \]
   \[ \land \Theta \neg \text{Existing}(\tilde{m}^M_{Production_E}) \]
   \[ \rightarrow (\tilde{e}' = \text{Create}(\tilde{m}^M_{Production_E}) \]
   \[ \land \tilde{e}' = \text{Stop}(\tilde{m}^M_{Research_E}) \]

2. \[ \vdash_{Research_E} (\Theta \tilde{e}' = \text{Transfer}_To(\tilde{m}^M_{Research_E}, \tilde{Production}_E) \]
   \[ \land \Theta \neg \text{Stopped}(\tilde{m}^M_{Production_E}) \]
   \[ \rightarrow (\tilde{e}' = \text{Resume}(\tilde{m}^M_{Production_E}) \]
   \[ \land \tilde{e}' = \text{Stop}(\tilde{m}^M_{Research_E}) \]

\( M \) represents a composite machine identifier, here an employee. \text{Transfer}_To is a method defined in the type \text{Research}_E. \text{Create}, \text{Stop}, and \text{Resume} are the functions defined in the temporal object \( M \). \( \Theta \) operator is used to express the causal relationship between two events such as \text{Transfer}_To and \text{Create}: the \text{Transfer}_To occurs ahead of the \text{Create}.
The two formulas above are the transferring constraints in two cases. The first formula is for the first case: an employee transferring from the Research Department into the Production Department where he/she never worked before. In this case, Transfer_To event in the \( m^M_{\text{Research}_E} \) will trigger the Create event to add a new primary machine \( m^M_{\text{Production}_E} \) into the composite machine \( M \) and trigger the Stop event to stop the primary machine \( m^M_{\text{Research}_E} \), because at a given time an employee belongs only to one department. The second formula is used for another case: an employee transfers from the Research Department to the Production Department where he/she used to work before. In this case, instead of triggering the Create event, the Transfer_To event in the \( m^M_{\text{Research}_E} \) will trigger the Resume event to reactivate the primary machine \( m^M_{\text{Production}_E} \). Meanwhile the \( m^M_{\text{Research}_E} \) itself will be stopped.

Since the above formulas involve events outside of the primary machine \( m^M_{\text{Research}_E} \), they are the external constraints of \( m^M_{\text{Research}_E} \).

5.3 VERIFYING THE EXPECTED SYSTEM PROPERTIES

Since behavior constraints are specified by the formulas of the DSL, we can verify whether the expected system properties are held by or can be derived from the behavior constraints defined in type-versions through the proof system of the DSL. This is very helpful for checking the correctness of the design of a TOODB. We demonstrate this point by the following example.
Example 5.3:

Assume a company is going to set up a TOODB to store the information about the products manufactured by the company. The following is a brief description. A product is made by two steps: design and production. The production can be further divided into three substeps: assembling, testing, and packing. If a product fails the testing, it will be sent for repair. If a product cannot be repaired, then it is discarded. Whenever a product is transferred from one step/substep to another, it may wait some time before the next treatment. The whole product manufacture procedure can be described by Figure 5.2.

Some expected system properties of this TOODB are as follows:

$F_1$: Every tested product has a test report stored in the database.

$F_2$: For every product, only one copy of data is stored in the database.

$F_3$: Every product which need to be repaired has to tested after it is repaired.

In the following, we will show how the type-version hierarchy for this TOODB can be designed, and how we can check if our design meets the above expected system properties. First we design a type-version hierarchy which is as shown in Figure 5.3, where we treat every type as a single-version type.

In this example, we will consider only the production step. We will use notation $\bar{e}' = \bar{\tau}_i(\bar{m}_i^M)$ as the abbreviation of $\bar{e}' = \bar{\tau} \land \bar{Compo}(\bar{m}, \bar{M}) \land \bar{Typed}(\bar{m}, \bar{\varphi}_i)$. The definitions of some type-versions are given as follows.

Type-version: $P.In_PRODUCTION (\varphi_3)$

Sup: Product ($\varphi_1$)
Figure 5.2: Procedure of Product Manufacture

Object($p_0$) → Product($p_1$) → $P_{In\_Design}(p_2)$, $P_{In\_Production}(p_3)$, $P_{In\_Ass.}(p_4)$, $P_{In\_Wait}(p_5)$, $P_{In\_Test}(p_6)$, $P_{In\_Rep.}(p_7)$, $P_{In\_Pack}(p_8)$

Figure 5.3: Type-version Hierarchy of Example 5.3

80
Attributes:

\((x_{31})\) Product-Schematic-Number: Integer;

\((x_{32})\) Product-Production-Place: String;

\((x_{33})\) Product-Production-Manager: Name;

\((x_{34})\) Total Working Hours: Real;

Methods:

\((\tau_{31})\) Starting Production;

\((\tau_{32})\) Reporting Production;

Constraints:

Internal:

\((BA_{31})\) \(\vdash_{n_{p_{3}}} \tilde{e}_{0} = \tilde{\tau}_{31}(\tilde{m}_{p_{3}}^{M})\);

\((BA_{32})\) \(\vdash_{n_{p_{3}}} \bigvee(\tilde{e}' = \tilde{\tau}_{31}(\tilde{m}_{p_{3}}^{M})) \rightarrow \neg(\tilde{e}' = \tilde{\tau}_{31}(\tilde{m}_{p_{3}}^{M}))\);

\((BA_{33})\) \(\vdash_{n_{p_{3}}} \bigvee(\tilde{e}' = \tilde{\tau}_{32}(\tilde{m}_{p_{3}}^{M})) \rightarrow \neg(\tilde{e}' = \tilde{\tau}_{32}(\tilde{m}_{p_{3}}^{M}))\);

External:

\((CA_{31})\) \(\vdash_{M} \Theta(\tilde{e}' = \tilde{\tau}_{31}(\tilde{m}_{p_{i}}^{M})) \rightarrow (\tilde{e}' = \tilde{\text{Create}}(\tilde{m}_{p_{i}}^{M}))\);

\((CA_{32})\) \(\vdash_{M} \Theta(\tilde{e}' = \tilde{\tau}_{32}(\tilde{m}_{p_{i}}^{M})) \rightarrow (\tilde{e}' = \tilde{\text{Stop}}(\tilde{m}_{p_{i}}^{M}))\);

\((CA_{33})\) \(\vdash_{M} ((\tilde{e}' = \tilde{\tau}) \land (\tilde{\tau} \in \tilde{m}_{p_{i}}^{M})) \rightarrow \neg\tilde{\text{Stopped}}(\tilde{m}_{p_{i}}^{M})\) \(i = 3, 4, 5, 6, 7, 8;\)

\((CA_{34})\) \(\vdash_{M} \neg\tilde{\text{Stopped}}(\tilde{m}_{p_{i}}^{M}) \rightarrow \forall j \neq i : \tilde{\text{Stopped}}(\tilde{m}_{p_{j}}^{M})\);

\((CA_{35})\) \(\vdash_{M} \tilde{\text{Stopped}}(\tilde{m}_{p_{i}}^{M}) \rightarrow \forall k : (\neg(\tilde{e}' = \tilde{\tau}_{k}(\tilde{m}_{p_{i}}^{M}))\) \(i = 3, 4, 5, 6, 7, 8.\)

Type-version: \(P.In\_Testing (\varphi_{6})\)

Sup: \(P.In\_Production (\varphi_{3})\)

Attributes:
(x_{61}) \text{ Number of Tests: Integer;}

(x_{62}) \text{ Testing Hours: Real;}

(x_{63}) \text{ Testing Person: Name;}

(x_{64}) \text{ Testing Conclusion: String;}

Methods:

(\tau_{61}) \text{ Testing;}

(\tau_{62}) \text{ Wait-Repairs;}

(\tau_{63}) \text{ Test-Report;}

Constraints

Internal:

(BA_{61}) \vdash_{\kappa_{p_6}} (\hat{e}' = \check{\tau}_{61}(\check{m}^M_{p_6})) \rightarrow (\check{x}_{64} = \check{P}ass \lor \check{x}_{64} = \check{D}iscard \lor \check{x}_{64} = \check{R}epair);

(BA_{62}) \vdash_{\kappa_{p_6}} \check{x}_{64} = \check{R}epair \rightarrow (\hat{e}' = \check{\tau}_{62}(\check{m}^M_{p_6}));

(BA_{63}) \vdash_{\kappa_{p_6}} \nabla (\check{x}_{64} = \check{R}epair) \rightarrow (\hat{e}' = \check{\tau}_{63}(\check{m}^M_{p_6}));

(BA_{64}) \vdash_{\kappa_{p_6}} (\check{x}_{64} = \check{P}ass \lor \check{x}_{64} = \check{D}iscard) \rightarrow (\hat{e}' = \check{\tau}_{63}(\check{m}^M_{p_6}));

(BA_{65}) \vdash_{\kappa_{p_6}} (\check{x}_{64} = \check{R}epair \land \neg(\hat{e}' = \check{\tau}_{61}(\check{m}^M_{p_6}))) \rightarrow \Theta(\check{x}_{64} = \check{R}epair);

External:

(CA_{61}) \vdash_{\kappa_{\check{x}}} (\hat{e}' = \check{C}reate(\check{m}^M_{p_6}) \lor \hat{e}' = \check{R}esume(\check{m}^M_{p_6})) \rightarrow (\check{e}' = \check{\tau}_{61}(\check{m}^M_{p_6}));

(CA_{62}) \vdash_{\kappa_{\check{x}}} (\hat{\check{e}}' = \check{\tau}_{62}(\check{m}^M_{p_6}) \lor \hat{\check{e}}' = \check{\tau}_{63}(\check{m}^M_{p_6})) \rightarrow (\check{e}' = \check{R}esume(\check{m}^M_{p_6}));

(CA_{63}) \vdash_{\kappa_{\check{x}}} (\hat{\check{e}}' = \check{R}esume(\check{m}^M_{p_6})) \rightarrow (\check{e}' = \check{S}top(\check{m}^M_{p_6}));

(CA_{64}) \vdash_{\kappa_{\check{x}}} (\hat{\check{e}}' = \check{R}esume(\check{m}^M_{p_6})) \rightarrow (\check{x}_{64} = \check{x}_{62}).

Type: P.In.Rep. (\rho_7)

Sup: P.In.Production (\rho_3)
Attributes:

($x_{71}$) Number of Repairs: Integer;

($x_{72}$) Repairs Hours: Real;

($x_{73}$) Rep-Person: Name;

($x_{74}$) Repairs Conclusion: String;

Methods:

($\tau_{71}$) Repair;

($\tau_{72}$) Send-To-Test;

Constraints:

Internal:

($BA_{71}$) \( \vdash_{\tilde{m}^{M}_{p_1}} \nabla (\tilde{e}' = \tilde{\tau}_{71}(\tilde{m}^{M}_{p_1})) \rightarrow (\tilde{e}' = \tilde{\tau}_{72}(\tilde{m}^{M}_{p_1})) \);

($BA_{72}$) \( \vdash_{\tilde{m}^{M}_{p_1}} (\tilde{x}_{71} = \tilde{M}ax\tilde{N}umber) \rightarrow (\tilde{x}_{74} = \tilde{D}iscard) \);

External:

($CA_{71}$) \( \vdash_{\tilde{M}} (\tilde{e}' = \tilde{C}reate(\tilde{m}^{M}_{p_1}) \lor \tilde{e}' = \tilde{R}esume(\tilde{m}^{M}_{p_1})) \rightarrow (\tilde{e}' = \tilde{\tau}_{71}(\tilde{m}^{M}_{p_1})) \);

($CA_{72}$) \( \vdash_{\tilde{M}} (\tilde{e}' = \tilde{\tau}_{72}(\tilde{m}^{M}_{p_1})) \rightarrow (\tilde{e}' = \tilde{R}esume(\tilde{m}^{M}_{p_1})) \);

($CA_{73}$) \( \vdash_{\tilde{M}} (\tilde{e}' = \tilde{R}esume(\tilde{m}^{M}_{p_1})) \rightarrow (\tilde{e}' = \tilde{S}top(\tilde{m}^{M}_{p_1})) \);

Now we translate the requirements above into formulas of the DSL as follow:

($F_1$): \( \vdash_{\tilde{M}} \nabla \Theta (\tilde{e}' = \tilde{C}reate(\tilde{m}^{M}_{p_1})) \rightarrow \nabla (\tilde{e}' = \tilde{\tau}_{63}(\tilde{m}^{M}_{p_1})) \);

($F_2$): \( \vdash_{\tilde{M}} \tilde{e}' = \nabla \tilde{\tau}_{32}(\tilde{m}^{M}_{p_0}) \rightarrow \neg (\tilde{e}' = \tilde{\tau}_{32}(\tilde{m}^{M}_{p_2})) \);

($F_3$): \( \vdash_{\tilde{M}} (\tilde{x}_{64} = \tilde{R}epair) \rightarrow ((\tilde{e}' = \tilde{\tau}_{61}(\tilde{m}^{M}_{p_0})) \land ((\tilde{e}' = \tilde{\tau}_{71}(\tilde{m}^{M}_{p_1}))) \).

83
Proof:

We have to show $F_1 - F_3$ are provable.

$F_1 : (1) \vdash_{A_6} (\bar{e}' = \overline{\text{Create}(\bar{m}_{p,e}^{M})}) \rightarrow \bar{e}' = \bar{r}_{61}(\bar{m}_{p,e}^{M})$ \hspace{1cm} (CA61)

(2) $\vdash_{A_6} \forall \theta (\bar{e}' = \overline{\text{Create}(\bar{m}_{p,e}^{M})}) \rightarrow \forall \theta (\bar{e}' = \bar{r}_{61}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (1)(APR1)(APR2)

(3) $\vdash_{n_{M}} \theta (\bar{e}' = \bar{r}_{61}(\bar{m}_{p,e}^{M}))$

$\rightarrow (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \lor \bar{x}_{64} = \bar{r}_{\text{repair}})$ \hspace{1cm} (BA61)

(4) $\vdash_{A_6} \theta (\bar{e}' = \bar{r}_{61}(\bar{m}_{p,e}^{M}))$

$\rightarrow (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \lor \bar{x}_{64} = \bar{r}_{\text{repair}})$ \hspace{1cm} (3)(CR1)

(5) $\vdash_{A_6} \forall \theta (\bar{e}' = \bar{r}_{61}(\bar{m}_{p,e}^{M}))$

$\rightarrow \forall (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \lor \bar{x}_{64} = \bar{r}_{\text{repair}})$ \hspace{1cm} (4)(APR2)

(6) $\vdash_{A_6} \forall \theta (\bar{e}' = \bar{r}_{61}(\bar{m}_{p,e}^{M}))$

$\rightarrow (\forall (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \lor \bar{x}_{64} = \bar{r}_{\text{repair}}))$ \hspace{1cm} (5)(APT8)

(7) $\vdash_{A_6} \forall \theta (\bar{e}' = \overline{\text{Create}(\bar{m}_{p,e}^{M})})$

$\rightarrow (\forall (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \lor \bar{x}_{64} = \bar{r}_{\text{repair}}))$ \hspace{1cm} (2)(6)(PR)

(8) $\vdash_{n_{M}} (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \rightarrow \bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (BA64)

(9) $\vdash_{M} (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \rightarrow \bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (8)(CR1)

(10) $\vdash_{A_6} (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}} \rightarrow \nabla (\bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (9)(APR2)

(11) $\vdash_{A_6} (\nabla (\bar{x}_{64} = \bar{p}_{\text{ass}} \lor \bar{x}_{64} = \bar{d}_{\text{discard}}) \rightarrow \nabla (\bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (10)(APT8)

(12) $\vdash_{n_{M}} (\bar{x}_{64} = \bar{r}_{\text{repair}} \rightarrow \bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (BA63)

(13) $\vdash_{A_6} (\bar{x}_{64} = \bar{r}_{\text{repair}} \rightarrow \bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (12)(CR1)

(14) $\vdash_{A_6} (\nabla (\bar{x}_{64} = \bar{r}_{\text{repair}}) \rightarrow \nabla (\bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (13)(APR2)

(15) $\vdash_{A_6} (\nabla (\bar{x}_{64} = \bar{r}_{\text{repair}}) \rightarrow \nabla (\bar{e}' = \bar{r}_{63}(\bar{m}_{p,e}^{M}))$ \hspace{1cm} (14)(APT4)
(16) \[ \vdash M \nabla \Theta (\tilde{c} = \text{Create} (\tilde{m}_{p}^{M})) \rightarrow \nabla (\tilde{c} = \tau_{83}(\tilde{m}_{p}^{M})) \]  
(6)(11)(15)(PR)

That is, \( F_{1} \) is provable.

\( F_{2} \):

1. \[ \vdash M \nabla \tilde{c} = \tau_{32}(\tilde{m}_{p}^{M}) \rightarrow \neg (\tilde{c} = \tau_{32}(\tilde{m}_{p}^{M})) \]  
\((BA_{32})\)

2. \[ \vdash M \nabla \tilde{c} = \tau_{32}(\tilde{m}_{p}^{M}) \rightarrow \neg (\tilde{c} = \tau_{32}(\tilde{m}_{p}^{M})) \]  
\((2)(CR1)\)

That is, \( F_{2} \) is provable.

\( F_{3} \):

1. \[ \vdash M (\tilde{x}_{64} = \tilde{\text{Repair}}) \rightarrow (\tilde{c} = \tau_{62}(\tilde{m}_{p}^{M})) \]  
\((BA_{62})\)

2. \[ \vdash M (\tilde{x}_{64} = \tilde{\text{Repair}}) \rightarrow (\tilde{c} = \tau_{62}(\tilde{m}_{p}^{M})) \]  
\((1)(CR1)\)

3. \[ \vdash M (\tilde{c} = \tau_{62}(\tilde{m}_{p}^{M})) \rightarrow \neg \tilde{\text{Stopped}} (\tilde{m}_{p}^{M}) \]  
\((CA_{33})\)

4. \[ \vdash M \neg \tilde{\text{Stopped}} (\tilde{m}_{p}^{M}) \rightarrow \tilde{\text{Stopped}} (\tilde{m}_{p}^{M}) \]  
\((CA_{34})\)

5. \[ \vdash M (\tilde{x}_{64} = \tilde{\text{Repair}}) \rightarrow \neg (\tilde{c} = \tau_{71}(\tilde{m}_{p}^{M})) \]  
\((2)(3)(4)(CA_{35})\)

6. \[ \vdash M (\tilde{x}_{64} = \tilde{\text{Repair}} \land \neg (\tilde{c} = \tau_{61}(\tilde{m}_{p}^{M}))) \rightarrow \Theta (\tilde{x}_{64} = \tilde{\text{Repair}}) \]  
\((BA_{65})\)

7. \[ \vdash M (\tilde{x}_{64} = \tilde{\text{Repair}} \land \neg (\tilde{c} = \tau_{61}(\tilde{m}_{p}^{M}))) \rightarrow \Theta (\tilde{x}_{64} = \tilde{\text{Repair}}) \]  
\((6)(CR1)\)

8. \[ \vdash M (\tilde{x}_{64} = \tilde{\text{Repair}}) \rightarrow ((\tilde{c} = \tau_{61}(\tilde{m}_{p}^{M})) A (\tilde{c} = \tau_{71}(\tilde{m}_{p}^{M})) \]  
\((5)(7)(APR15)\)

That is, \( F_{3} \) is provable.

The inference rules APR1, APR2 and APR15 as well as the theorems APT4 and APT8 that we have used in the proof above are listed in the Appendix A. PR represents the Propositional Reasoning Rule, i.e.,

\[ \vdash \mu w_{1} \land w_{2} \land ... \land w_{n} \rightarrow w_{n+1} \]

\[ \vdash \mu w_{1} \land w_{2} \land ... \land w_{n} \]

\[ \vdash \mu w_{n+1} \]
5.4 CHECKING BEHAVIOR CONSTRAINTS

We use the database history of a composite machine as a model for a formal system of the composite machine in the DSL, and the database history of a primary machine as a model for a formal system of the primary machine in the DSL. We assume that every event of a dynamic state machine leaves an event record in the TOODB. At any time, in the TOODB, every dynamic state machine has a finite number of event records. Therefore, checking whether the database history of a dynamic state machine satisfies a given behavior constraint specified by a DSL formula is decidable. If we assume all the data in an event record can be retrieved once, then the time complexity of a decision procedure of checking whether the database history of a dynamic state machine satisfies a behavior constraint is linear in the number of event records of the corresponding dynamic state machine which are stored in the TOODB.

Example 5.4:

Let $w_i$'s $i = 1, 2, 3$ be different formulas specifying states (i.e., $w_i$ does not contain any temporal operators), and $\mu$ be a dynamic state machine. Suppose we are going to check the following behavior constraints: (i) $\vdash \mu \; w_1 \rightarrow \Xi w_2$, (ii) $\vdash \mu \; w_1 \rightarrow \nabla w_2$, (iii) $\vdash \mu \; w_1 \rightarrow w_2 S w_3$, (iv) $\vdash \mu \; \Xi w_1 \rightarrow w_2$ and (v) $\vdash \mu \; \nabla w_1 \rightarrow w_2$.

We can check these five behavior constraints by going through the database history of $\mu$. Assume for each behavior constraint, there is a checking procedure denoted by $Procedure_i$, $i = 1, 2, ..., 5$. The algorithms for these procedures are presented as follows.
We need two variables: state and result. The record variable state is used to contain the event record which is currently reviewed. The boolean variable result is used to remember the conclusion of the checking. We use “endstate” as a technical symbol to represent that the checking procedure has passed through all the event records. Here, we assume that all the event records have been sorted according to their ordering in the legal trace.

(i) Procedure\textsubscript{1}

\begin{verbatim}
BEGIN
  result := false;
  state := the last recent event record;
  IF \( w_1 \) and \( w_2 \) hold in the state
    THEN BEGIN
      result := true;
      state := the initial event record;
      WHILE (state \neq \text{endstate}) \land (\text{result} = \text{true}) DO
        IF \( w_2 \) holds in the state
          THEN state := next event record
        ELSE result := false;
      END;
    IF result = \text{true} THEN output \textit{SUCCESS}
      ELSE output \textit{FAIL};
\end{verbatim}
END \{Procedure_1\}.

(ii) Procedure_2

BEGIN

result := false;

state := the last recent event record;

IF \( w_1 \) holds in the state

THEN BEGIN

state := the initial event record;

WHILE (state \( \neq \) endstate) \& (result = false) DO

 IF \( w_2 \) does not hold in the state

 THEN state := next event record

 ELSE result := true;

END;

IF result = true THEN output SUCCESS

ELSE output FAIL;

END \{Procedure_2\}.

(iii) Procedure_3

BEGIN

result := false;

state := the last recent event record;

IF \( w_1 \) holds in the state

THEN
BEGIN

state: = the initial event record;

WHILE (state \neq endstate) \land (result = false) DO

BEGIN

IF \( w_3 \) holds in the state

THEN

IF \( w_2 \) holds in the state

THEN

BEGIN

result: = true;

state: = next event record;

WHILE (state \neq endstate) \land (result = true) DO

IF \( w_2 \) hold in the state

THEN state: = next event record;

ELSE

BEGIN

result: = false;

EXIT;

END;

END;

ELSE EXIT;

ELSE state: = next event record;

END;
END;

END;

IF result = true THEN output SUCCESS
ELSE output FAIL

END\{Procedure_3\}.

(iv) Procedure_4

BEGIN

result := false;

state := the last recent event record;

IF w_2 holds in the state

THEN BEGIN

result := true;

state := the initial event record;

WHILE (state ≠ endstate)∧( result = true) DO

IF w_1 holds in the state

THEN state := next event record

ELSE result := false;

END;

IF result = true THEN output SUCCESS
ELSE output FAIL

END\{Procedure_4\}.

(v) Procedure_5
BEGIN

result := false;

state := the last recent event record;

IF $w_2$ holds in the state

THEN BEGIN

state := the initial event record;

WHILE (state $\neq$ endstate)$\land$( result = false) DO

IF $w_1$ does not hold in the state

THEN state := next event record

ELSE result := true;

END;

IF result = true THEN output SUCCESS

ELSE output FAIL

END{Procedure_3}.

Obviously, all these checking procedures need to go through the database history of the dynamic state machine $\mu$ once, so that the time complexity of them is linear in the number of the event records of the database history of the dynamic state machine $\mu$. These algorithms show only the design principles. In real application, the algorithm design should take the concrete data storage organization into account, e.g. the data of an event record of a complex event may be stored separately.
Chapter 6

A DESIGN OF TOODB MEMORY MANAGEMENT

6.1 INTRODUCTION

Since a TOODB stores the historical data of objects, the data volume in the TOODB is ever growing. Therefore, designing a suitable memory management system becomes very important to implement a TOODB. In this chapter, we will present the design of a paged virtual memory management system for the TOODB. Our design goal is to minimize the cost of processing page faults by minimizing the average page access number. The main problems to overcome are: (a) ever growing data space, (b) clustering data having different updating rates and sizes, and (c) increased complexity of design optimization due to temporal constructs. Because these problems mainly influence the secondary storage management, we direct our attention to the secondary storage management. Our design takes into account the characteristics of both data organization and users' access patterns. To handle problem (a), a two-level storage structure physical database is adopted. To provide a solution to problems (b) and (c), we develop a clustering scheme and an analysis model which demonstrates the
relationship between the average page access number and the parameters reflecting the characteristics of both data organization and users' access patterns. Based on the analysis model, we are able to find the optimal design in terms of minimal average page access number. In this chapter, we will first introduce a set of data organization constructs, then present a way to abstract users' access patterns. Then the analysis model will be presented.

In this chapter, some previous notations have been redefined, whenever this is done, the new definition is given.

6.2 DATA ORGANIZATION

6.2.1 Basic Data Unit

In the physical database, each dynamic state machine has two kinds of data to be stored: description information and history records. For a primary machine, the description information means the definition of a type-version. Every type-version is stored in a type-version record which consists of seven fields, i.e., a field for type-version identifier plus other six fields, each of which corresponds to one part of the type-version definition. Whenever a simple event or an element event of a complex event occurs, a history record will be stored into database. A history record comprises the following fields:

- Transaction-id: The identifier of the database transaction which invokes the transition in the event;
• Sequence-no: The number to indicate the order of execution of the transition invoked within a database transaction;

• Invoker-id: The identifier of the dynamic state machine which invokes the transition of the event;

• Receiver-id: The identifier of the dynamic state machine where the event occurs;

• Transition-id: The identifier of the transition of the event;

• Commit-time: Commit time of the event;

• Effective-time: Effective time of the event;

• Observation-time: Observation time used in the event;

• Write-Set: for every variable in the event variable set $V_e$, there is a variable-value pair such that the value is written to the variable by the transition of the event.

• Read-Set: for every variable in the event variable set $V_e$, there is a variable-value pair such that the value is read from the variable by the transition of the event.

A multitype composite machine can be treated as a set of primary machines (including a pseudo primary machine). Therefore the data of a multitype composite machine is the collection of the data of its component primary machines. The data of the general composite machine is all of data stored in the database.
6.2.2 Two-Level Storage Structure

In the physical database, we adopt a two-level storage structure. The first level is a public object base which stores all the data of all existing dynamic state machines. The second level consists of a set of private object bases, each of which contains the data related to a specific time period of some dynamic state machines that are queried often by a user during a certain time period. In the public object base, the reference-based storage organization is adopted, while in private object bases, a revised copy-based storage organization is adopted.

The Public Object Base

The public object base consists of two parts: type-version forest storage and machine storage.

In the type-version forest storage, every type-version record is stored in a segment which consists of pages. All segments of the type-versions of a type-version tree form a linked-segment-list. There is an Id-index in the type-version forest storage, from which a type-version identifier is mapped into the starting address of the segment that contains the type-version.

The machine storage consists of an Id-index and a set of segment-list-groups. The data of a multitype composite machine is stored in a segment-list-group which consists of a set of linked-segment-lists. All history records of a primary machine are stored in one linked-segment-list. The history records of a primary machine are sorted based on the ordering of their commit times. Every linked-segment-list consists of a set of
paged segments, each of which has a *segment descriptor* and a *data chain*. The segment descriptor contains three fields: *number of history records*, *pointer to the next segment*, and *the starting address of the first history record*. Every element of the data chain is a pair: a *history record* and *the length of the history record*. The data that are related only to the multitype composite machine are technically treated as the data of a pseudo primary machine, and are stored in one linked-segment-list. Every segment-list-group has a *group descriptor* and a *time index*. The group descriptor is a 4-tuple \((Id, PMnum, IndexAdd, PMs)\), where \(Id\) contains the identifier of the multitype composite machine, \(PMnum\) contains the number of primary machines, \(IndexAdd\) contains the physical address of the root node of the *time index*, and \(PMs\) is a set of pairs \((PMid, FSAdd)\), where \(PMid\) contains the identifier of a primary machine, \(FSAdd\) contains the physical starting address of the first segment of the primary machine. The *Time-index* is a \(B^+\) tree. The key value of the index represent effective time. In the leaf level of the time index, every key value represents the starting time of a time interval, and every pointer points to an ID (identifier) block, i.e., every ID block corresponds to a time interval. If the primary state machine associated with the multitype composite machine has at least one history record whose effective time overlaps with a time interval, then it has one entry in the corresponding ID block. Each entry is a 4-tuple \((id, type, address, next)\), where \(id\) is the identifier of the primary machine, \(type\) contains a type or a type-version identifier of a primary machine, \(address\) contains a starting physical address of the page containing the history records of the primary machine, and \(next\) contains a physical address of an ID
block which corresponds to the next time interval in which the primary machine of type *type* has some history records. The Id-index maps an identifier of a multitype composite machine into the physical address of the segment that contains the corresponding segment-list-group descriptor. Figure 6.1 shows the structure of the public object base.

**The Private Object Base**

The purpose of building private object bases is to reduce the data space and speed data retrieval. Since we assume a private object base has only one user, the clustering can be applied to improve paging performance. Conceptually, a private object base is a view of the public object base. The view maintenance problem will not discussed in this dissertation. We will concentrate on how to cluster the database history of a primary machine.

Because a primary machine may have a composition hierarchy in which a circular reference may occur (e.g., a primary machine $m_p^M$ may be referred to by a primary machine $m_p^M$ which is referred to by the $m_p^M$ as an attribute value), the pure copy-based storage organization cannot work. Therefore, a revised copy-based storage organization is adopted, which is described as follows.

A tree type data structure which is called the *primary machine tree* is used to store the data of a primary machine, which is the basic storage unit of a private object base.

A primary machine tree has a schema which is a type-version composition hi-
Figure 6.1: The Public Object Base
Figure 6.2: A Primary Machine Tree Schema

erarchy formed by the type-version of a primary machine and the type-versions of the attribute variables, each of which may have their own attributes. A type-version composition hierarchy can be represented as a directed graph (each edge directs from the parent node to the child node). To avoid circular reference structure, we have the following convention: for any path starting from the root node, if a non-atomic type-version has been a node on a upper level, then it becomes a terminal node.

Example 6.1:

An example of type-version composition hierarchy or a primary machine tree schema is shown in Figure 6.2. In this example, the root node is “Univ.” which occurs in other level as well. When “Univ.” occurs again on a lower level, it becomes a terminal node.

For convenience, the notation $q$ is used to represent the root node of a primary machine tree schema, and $q^i$ to represent a child node of $q$. The notation $PMT^q$ is
used to represent a primary machine tree rooted at node \( q \), which sometimes means only the schema of a primary machine tree and sometime means both the primary machine tree schema and the data that are stored in the primary machine tree.

To store all the data effective in a time period \( H \) of a primary machine by a primary machine tree, the following conventions are adopted:

(1) if an history record of the primary machine with an effective time that falls in \( H \), this history record is stored in the root node \( q \).

(2) If a primary machine is referred to by an attribute value of the attribute \( q^i \) in a history record with an effective time \( t_f \), then the history record of this referred primary machine which effective time overlaps with the \( t_f \) is stored in the node \( q^i \).

(3) (2) is recursively applied until the leaf level.

(4) If a node being referred to represents the same primary machine that is represented by a node at a upper level, then a pseudo history record is formed in which the physical addresses of the history records of the primary machine being referred to are stored such that the effective times of these history records are matched with that of the history record which refers to the node.

(5) In all nodes, history records are sorted according to their commit times.
Delta vs. Checkpoint

Each history record only stores new data which is different from those stored in the last history record and is called a delta. Conceptually, to reply for a temporal query, all the history records of the machine whose commit times are less than or equal to the observation time of the query need to be reviewed. Obviously, if a sequence of \( n \) history records needs to go through, the time complexity is \( O(n) \). However, a recent research result has concluded that by a suitable data structure which requires three times memory space compared with the the memory space required by a linear data structure such as an array, the time complexity of going through \( n \) deltas is \( O(\log \log n) \)[Go90].

Another alternative is to adopt the checkpoint mechanism. At a selected instant of time which is called a checkpoint, for each primary machine, the system summarizes all deltas whose commit times are less than or equal to the checkpoint and materializes the resulting summation as a checkpoint record. In an extreme case, every history record becomes a checkpoint record so that the time complexity of processing a temporal query is \( O(1) \), i.e., a constant. However, the checkpoint mechanism will consume much more memory space than those of the delta mechanism.

A trade off between the delta mechanism and the checkpoint mechanism is possible. For example, a checkpoint is set up after every \( m \) deltas. While a checkpoint record is used as an initial base for the query processing, deltas will help to save memory space. In this case, the time complexity of a query is bound by \( O(\log \log m) \).

We apply the mixture mechanism above in both the public object base and private
object bases. We assume that all primary machines stored in the public object base or in the same private object base will follow the same checkpoints. In the public object base, all checkpoint records of a primary machine will be inserted into the linked-segment-list of the primary machine. In the private object base, the first record stored in every node of a $PMT^q$ must be a checkpoint record. In fact, in both public and private object bases, a checkpoint record is treated the same as a history record.

Retroactive updates occurring later may invalidate some of the checkpoint records generated. Query processing will account for invalidated checkpoint records and use the most recent unaffected checkpoint record as its initial base. To handle the affected checkpoint records, the system may include a mechanism for resetting them periodically.

6.2.3 Clustering Scheme

To achieve a better paging performance, we present a clustering scheme to organize data in the private object base. To describe the clustering scheme, the following concepts need to be defined.

For every pair of parent-child nodes of a $PMT^q$, a partition variable is defined, which can be assigned either 0 or 1. If the child node is denoted as $q^k$, then the corresponding partition variable is denoted as $p_k$.

A $PMT^q$ with $n$ nodes has a set of $(n - 1)$ partition variables. Therefore there are $2^{(n-1)}$ possible combinations for the values of $(n - 1)$ partition variables.

Every combination of the values of all partition variables of a $PMT^q$ is called a partition of the $PMT^q$, and is denoted as $P^q$. We use $\text{Partition-Set}(PMT^q)$ to
represent all the possible \( P^a \)'s for a \( PMT^a \).

If a partition \( P^a \in \text{Partition-Set}(PMT^a) \) contains \( k \) 0's, then the \( PMT^a \) is decomposed into \((k + 1)\) subtrees, and each such subtree is called a cluster.

The value of a \( p_k \) assigned by a partition \( P^a \) has the following meaning:

\[
p_k = \begin{cases} 
1, & q^k \text{ and its parent node are in the same cluster} \\
0, & q^k \text{ and its parent node are not in the same cluster.}
\end{cases}
\]

For convenience, a partition \( P^a \) can be written as a string consisting of 0's and/or 1's.

A clustering sequence is an ordering in which the nodes in a cluster (i.e., a subtree of a \( PMT^a \)) are traversed. We use \( C \) to denote a clustering sequence.

There are two basic types of clustering sequences: \textit{depth-first traversal} and \textit{breadth-first traversal}. A clustering sequence is used as the storage ordering of nodes in the same cluster of a partition in the secondary memory.

\textbf{Example 6.2:}

As shown in Figure 6.3 (a), a \( PMT^a \) consists of five nodes: \( a, b, c, d, \) and \( e \). Assume that \( a \) contains three history records \( a_1, a_2, a_3 \); \( b \) contains four history records \( b_1, b_2, b_3, b_4 \); \( c \) contains three history records \( c_1, c_2, c_3 \); \( d \) contains five history records \( d_1, d_2, d_3, d_4, d_5 \); and \( e \) contains four history records \( e_1, e_2, e_3, e_4 \).

Since there are five nodes, we have four partition variables \( \{p_a, p_c, p_d, p_e\} \). The total number of possible partitions is \( 2^{(5-1)} = 2^4 = 16 \). Two examples shown in Figure 6.3 (b) are

\[
P_1^a : \{p_a = 1, p_c = 0, p_d = 1, p_e = 1\}, \quad \text{or} \quad P_1^a = 1011,
\]
Figure 6.3: Examples of Some Concepts

\[ P_1^a \colon \{ p_b = 0, p_c = 0, p_d = 1, p_e = 1 \}, \quad \text{or} \quad P_2^a = 0011. \]

\( P_1^a \) partitions  \( PMT^a \) into two subtrees (clusters): one consists of  \( a, b, d \) and \( e \); the other one consists of \( c \). \( P_2^a \) partitions  \( PMT^a \) into three subtrees (clusters): the first one consists of \( a \); the second one consists of \( b, d \) and \( e \); the third one consists of \( c \).

Under the  \( P_1^a \), the history records in the first cluster are stored in the ordering of depth-first traversal as follows: \( a_1, a_2, a_3, b_1, b_2, b_3, b_4, d_1, d_2, d_3, d_4, d_5, e_1, e_2, e_3, e_4 \).

Let  \( H \) represent the total time interval of a private object base. That is the
effective times of all the history records stored in the private object base must fall into the $H$. Let $L$ represent a length of reference time interval. By the $L$, $H$ is divided into $N$ reference time intervals which is half opened (i.e., an reference interval is represented as $[t_1, t_2)$, where $t_1, t_2$ are two time points). $N = \lfloor H/L \rfloor$.

Based on their effective times, all history records stored in a node of a $PMT^q$ can be distributed into $N$ reference time intervals, so that a $PMT^q$ can be decomposed into $N$ pieces, each of which is associated with one reference time interval. We call each piece of the $PMT^q$ an $hp$-tree and denote the $j$-th $hp$-tree as $PMT^q_j$. Every node in an $hp$-tree is called a history piece and is denoted as $hp$.

In summary, based on the concepts defined above, our clustering schema consists of the following rules.

(1) $Hp$-tree Clustering Rule:

All $hp$-trees of a $PMT^q$ follow the same partition $P^q$ on the $PMT^q$.

(2) Page Assignment Rule:

(a) If two $hp$'s of an $hp$-tree do not belong to the same cluster of a partition, their history records are stored on different pages.

(b) If two $hp$'s do not belong to the same $hp$-tree, their history records are stored on different pages.

(c) All the $hp$'s in the same cluster of an $hp$-tree are stored on a set of pages with contiguous physical addresses.

(3) Data Storing Ordering Rule:
All \( hp\)-trees of a \( PMT^q \) follow the same clustering sequence \( C \) to store \( hp \)'s in the secondary memory, and history records keep their original ordering within an \( hp \).

(4) \textit{Data Distribution Rule:}

(a) If the effective time of a history record of the root node \( q \) overlaps with the \( j \)-th standard time interval \((j = 1, 2, ..., N)\), a copy of this history record is stored into the \( j \)-th \( hp \) of the \( q \).

(b) If a history record in a child node of the \( q \) is referred to by a history record in the \( j \)-th \( hp \) of the \( q \), then it is stored in the \( j \)-th \( hp \) of the corresponding child node.

(c) Recursively apply (b) until the leaf level.

6.3 \textbf{USERS' ACCESS PATTERNS}

To abstract users' access patterns, we define a set of concepts in this section.

6.3.1 \textbf{Primitive Transactions}

We define a \textit{primary temporal query} (PTQ) to be a query which only accesses one \( PMT^q \) to retrieve some data of a primary machine that are related to a time point or time interval. To process a primary temporal query, the system will first select a suitable checkpoint record in the root node of the \( PMT^q \) as the initial base and accesses the history records with effective times greater than the selected checkpoint and a commit time less than or equal to the observation time. The same idea will be used when a node at a lower level needs to be accessed.
In a private object base, we treat a checkpoint record the same as a history record. From now on, when we say a history record it means a history or checkpoint record, except when they are clearly distinguished.

We define the basic operations to process a primary temporal query to be \textit{primitive transactions}. A primitive transaction (PT) retrieves data from the history records that are stored in a \textit{PMT}^q. It accesses at least one record but at most two history records.

We further classify all primitive transaction into three types:

\textbf{IT1:} a primitive transaction first visits a history record in an \textit{hp} of a node, and then either stops or accesses the successor history record in the same \textit{hp} of the same node.

\textbf{IT2:} a primitive transaction first visits a history record in an \textit{hp} of a node, and then accesses the successor history record in a successor \textit{hp} of the same node.

\textbf{ET1:} a primitive transaction first visits a history record in an \textit{hp} of a node, and then accesses a history record in the same \textit{hp} of one of its child nodes.

For convenience, we sometimes use ITx to represent IT1 and/or IT2.

IT means \textit{internal transactions} and ET means \textit{external transactions}. Two history records accessed by an ITx (type) primitive transaction are in the same node of a \textit{PMT}^q, while two history records accessed by an ET1 (type) primitive transaction are in a pair of parent-child nodes of a \textit{PMT}^q. The ITx primitive transactions are used
to get more history records of one object when a query involves a time interval. The ET1 primitive transactions are used to get the details of a primary machine referred to by a history record. Technically, the root node can be considered as having a pseudo parent node user, so that the temporal queries issued from users to the \( PMT^u \) can be simulated as ET1 primitive transactions between users and the root node \( q \).

The following example is used to clarify the definition above.

**Example 6.3:**

This example is a continuation of Example 6.2. Primitive transactions that involve two history records in the same nodes such as \( (a_1, a_2) \), or \( (b_3, b_4) \), or \( (e_2, e_3) \) are examples of ITx primitive transactions. Primitive transactions that involve two history records in a pair of parent-child nodes such as \( (a_1, b_1) \), or \( (b_3, d_4) \), or \( (b_2, e_3) \) are examples of ET1 primitive transactions.

### 6.3.2 Processing Primary Temporal Queries

By using the concepts that we have defined, we can think of processing a primary temporal query as an execution of a sequence of primitive transactions. The processing procedure starts from an ET1 primitive transaction between the user and the root \( hp \) (node) of an \( hp\text{-}tree \) and is followed by a series of ITx and/or ET1 primitive transactions in a set of \( hp \)'s. We have the following rules:

1. if the next primitive transaction can be either an ET1 or an ITx, then the ET1 has priority.
2. the ordering of accessing nodes of the hp-tree is identical to the clustering sequence C which is chosen.

6.3.3 Relative Frequencies of PTs

We assume that the statistical data about how many IT1, IT2 and ET1 primitive transactions occur in every node of a \( PMT^q \) are available. From these data we can easily find out corresponding to every primary temporal query issued to a \( PMT^q \) by the users, on average, the average numbers of IT1, IT2 and ET1 primary transitions at every node of the \( PMT^q \) during a certain time period, which is called the \textit{primitive transaction distribution}. Based on the primitive transaction distribution, for every node of a \( PMT^q \), we defined three \textit{average relative frequencies}, each of which is for one type of primitive transaction. We used \( \bar{f}_{it1}^q \), \( \bar{f}_{it2}^q \) and \( \bar{f}_{et1}^q \) respectively to represent the average relative frequencies of IT1, IT2 and ET1 primitive transactions in a node \( q^k \in PMT^q \). The definition of the average relative frequencies is as follows.

\[
\begin{align*}
\bar{f}_{it1}^q & = \frac{\bar{c}_{it1}^q}{\sum_{k=1}^{n}[\bar{c}_{it1}^k + \bar{c}_{it2}^k + \bar{c}_{et1}^k]} \\
\bar{f}_{it2}^q & = \frac{\bar{c}_{it2}^q}{\sum_{k=1}^{n}[\bar{c}_{it1}^k + \bar{c}_{it2}^k + \bar{c}_{et1}^k]} \\
\bar{f}_{et1}^q & = \frac{\bar{c}_{et1}^q}{\sum_{k=1}^{n}[\bar{c}_{it1}^k + \bar{c}_{it2}^k + \bar{c}_{et1}^k]} 
\end{align*}
\]

(6.1)

where \( n \) represents the total number of nodes in the \( PMT^q \); \( \bar{c}_{it1}^q \), \( \bar{c}_{it2}^q \) and \( \bar{c}_{et1}^q \) represent the average number of IT1, IT2 and ET1 primitive transactions in the node \( q^k \) corresponding to every one primary temporal query respectively.

If a \( PMT^q \) is decomposed into \( N = \lceil \frac{H}{L} \rceil h_i \)-trees, then for the \( j \)-th hp-tree we
have

\[ f_{ii1}^k = \bar{f}_{ii1}^k / N \]
\[ f_{ii2}^k = \bar{f}_{ii2}^k / N \]
\[ f_{eli}^k = \bar{f}_{cli}^k / N \quad (6.2) \]

6.4 AN ANALYSIS MODEL

To find out how to minimize the average page access number, an analysis model which is an expression of the average page access number of a PMT is set up. We first define a set of parameters which reflect the characteristics of data organization.

6.4.1 Parameters of a Node

Assume the length of reference time interval \( L \) and the total time interval \( H \) have been given. Let \( s \) represent the page size in bytes.

For every \( hp \) of a node \( q^k \in PMT \), we defined four parameters: (i) \( z_{v}^k(H) \), the total number of history records in \( H \), (ii) \( v_{v}^k(H) \), the summation of history record sizes in \( H \), (iii) \( \bar{z}_{v}^k = \lceil z_{v}^k(H)/N \rceil \), where \( N = \lceil H/L \rceil \), the average number of history record in an \( hp \), and (iv) \( \bar{v}_{v}^k = \lceil v_{v}^k(H)/z_{v}^k(H) \rceil \), the average history record size. Thus, the average \( hp \) size of \( q^k \) is \( \bar{z}_{v}^k \cdot \bar{v}_{v}^k \). The units of \( v_{v}^k(H) \) and \( \bar{v}_{v}^k \) are bytes.

\( z_{v}^k(H) \) is a parameter representing the updating rate of data. \( v_{v}^k(H) \) is a parameter representing the requirement of data space. \( \bar{v}_{v}^k \) is an integer and a constant. \( \bar{z}_{v}^k \) is a function of \( L \), and can be used to adjust the size of \( hp \)'s.

110
6.4.2 Average Distances Between History Records

The distance between two history records is the offset in terms of bytes between the starting addresses of these two history records.

Two history records can be either in the same node or in two different nodes of a PMTv. The two nodes can be either in the same cluster or in different clusters of a partition Pq. We assume that only the history records that are in the same cluster are stored continuously in a physical memory space. Technically, we assume that the distance of two history records that are not in the same cluster is equal to the page size s because whenever the distance between two history records greater than or equal to a page size, one more page access is needed. This technical assumption and the following definition of path partition expression allow us to have a general formula for distance.

Considering a PMTv, let (q1, q2), (q2, q3), ..., (qk-1, qk) be the edges which form a path from node q1 to node qk, and let p2, p3, ..., pk be partition variables associated with the corresponding edges of the path.

A path partition expression of two nodes q1 and qk, denoted as: \( \lambda^{q_1 q_k} \), is defined as

\[
\lambda^{q_1 q_k} = p_2 * p_3 * ... * p_k \tag{6.3}
\]

where * represents the arithmetic multiplication operator.

Let \( \overline{d}_{ij} \) be the average distance between two contiguous history records in the node qk of j-th hp-tree of a PMTv, which is defined as follows:

\[
\overline{d}_{ij} = \overline{v}^k_i \tag{6.4}
\]
We use $d_{e_1}^{i_1}$ to denote the average distance between two history records in the node $q^k$ and its parent node $q^{k'}$ of the $j$-th hp-tree of a $PMT^q$.

(i) For $C =$ depth-first traversal, we have

$$d_{e_1}^{i_1} = (1/2)(\tilde{z}_{j}^{k'} \cdot \tilde{u}_{j}^{k'} + \tilde{z}_{j}^{k} \cdot \tilde{u}_{j}^{k}) \cdot p_k + s \cdot (1 - p_k)$$

$$+ \sum_{q'' \in S(q^k)} \tilde{u}_{j}^{q''} \cdot p_k$$

(6.5)

where

$$\tilde{u}_{j}^{q''} = \tilde{z}_{j}^{q''} \cdot \tilde{u}_{j}^{q''} \cdot \lambda_{q^k q^{k''}} + \sum_{q''' \in S(q''')} \tilde{z}_{j}^{q''} \cdot \tilde{u}_{j}^{q'''} \cdot \lambda_{q^k q^{k''}}$$

(6.6)

is called the size of an hp-tree $PMT_j^{q'}$.

In the above, $S(x)$ represents the set of child nodes of the node $x$, and these nodes are in the same cluster with node $x$. The expression $x < y$ means that the node $x$ is on the left side of node $y$.

(ii) If $C =$ breadth-first traversal, we assign every node in a $PMT^q$ a number according to the breadth-first traversal ordering, then we have

$$d_{e_1}^{i_1} = (1/2)(\tilde{z}_{j}^{k'} \cdot \tilde{u}_{j}^{k'} + \tilde{z}_{j}^{k} \cdot \tilde{u}_{j}^{k}) \cdot p_k + s \cdot (1 - p_k)$$

$$+ \sum_{q'' \in S(q^k)} \tilde{z}_{j}^{q''} \cdot \tilde{u}_{j}^{q''} \cdot \lambda_{q^k q^{k''}} \cdot \lambda_{q'' q^{k''}} \cdot p_k$$

$$+ \sum_{q'' \in S(q^k)} \tilde{z}_{j}^{q''} \cdot \tilde{u}_{j}^{q''} \cdot p_{q''} \cdot p_k$$

(6.7)

112
where \( q \) is the root node of a \( PMT^a \), \( n(x) \) represents the ordering number of node \( x \). We assign every node in a \( PMT^a \) a number according to the breadth-first traversal ordering.

**Example 6.4:**

Assume \( N = 1 \). Considering the \( PMT^a \) in Figure 6.3 (a), the distance expressions of each pair of parent-child nodes are given as follows:

(i) \( C = \text{depth-first traversal} \)

\[
\begin{align*}
d^b &= (1/2)(z^a v^a + z^b v^b)p_b + s(1 - p_b) \\
d^c &= (1/2)(z^a v^a + z^c v^c)p_c + s(1 - p_c) \\
&\quad + [z^c v^a p_b + z^d v^c p_b p_d + z^e v^c p_b p_c] p_c \\
d^d &= (1/2)(z^b v^b + z^d v^d)p_d + s(1 - p_d) \\
d^e &= (1/2)(z^b v^b + z^e v^e)p_e + s(1 - p_e) + z^d v^d p_d p_e
\end{align*}
\]

(ii) \( C = \text{breadth-first traversal} \)

\[
\begin{align*}
d^b &= (1/2)(z^a v^a + z^b v^b)p_b + s(1 - p_b) \\
d^c &= (1/2)(z^a v^a + z^c v^c)p_c + s(1 - p_c) + z^a v^a p_b p_c \\
d^d &= (1/2)(z^b v^b + z^d v^d)p_d + s(1 - p_d) + z^c v^c p_b p_c p_d \\
d^e &= (1/2)(z^b v^b + z^e v^e)p_e + s(1 - p_e) \\
&\quad + z^c v^c p_b p_c p_e + z^d v^d p_d p_e
\end{align*}
\]

113
6.4.3 Average Page Access Number Expression

Because most of the time a history record is involved in two contiguous primitive transactions, we assume that for every primitive transaction the first involved history record has been already in the main memory.

To determine the page access number, we consider each type of primitive transaction.

(i) For IT1, two history records involved are in the same cluster and also in the same hp-tree. If we assume that the first history record can be placed with uniform probability anywhere within a page, the average page access number for the second history record can be determined by the average distance between two history records and the page size $s$. Therefore, by denoting the average page access number $F_{i_1i_1}^{s_i^k}$, we have the following definition:

$$F_{i_1i_1}^{s_i^k} = \begin{cases} \frac{s_i^k}{s}, & \text{for } d_{i_1i_1}^{s_i^k} < s \\ 1, & \text{for } d_{i_1i_1}^{s_i^k} \geq s \end{cases} \quad (6.8)$$

where $d_{i_1i_1}^{s_i^k}$ is determined by (6.4).

(ii) For IT2, the two history records involved are not in the same hp-tree, so every IT2 primitive transaction needs one page access for the second history record.

(iii) For ET1, there are two possibilities: (a) the two history records involved are not in the same cluster, (b) the two history records involved are in the same cluster. For (a), we technically assume that $d_{c_1l}^{s_i^k} = s$. For (b), $d_{c_1l}^{s_i^k}$
is determined by (6.5) or (6.7). Therefore, for ET1, we have the following definition:

\[
\tilde{r}_{ei1}^h = \begin{cases} 
\frac{d_{ei1}^h}{s}, & \text{for } d_{ei1}^h < s \\
1, & \text{for } d_{ei1}^h \geq s
\end{cases} \quad (6.9)
\]

Based on the parameters defined above and the relative frequencies of primitive transactions, the average page access number of the queries on a database history of a PMT\(^q\), denoted as \(\bar{R}^q\), can be determined by the following expression:

\[
\bar{R}^q = N \sum_{q_j^h \in PMT_j^q} \{ f_{i1}^q \ast \tilde{r}_{ei1}^h + f_{i2}^q \ast [\tilde{r}_{ei1}^h \ast (r_{ei1}^q \ast p_k + (1 - p_k))] \} \quad (6.10)
\]

We will always assume that for the root node \(q\) we have \(p_q = 0\). \(\bar{R}^q\) is a function of multiple parameters that are related to each other. In the next chapter, we will discuss how to find suitable parameters to minimize the \(\bar{R}^q\).
Chapter 7

OPTIMAL PARTITION

7.1 INTRODUCTION

Among all the parameters used in the expression of average page access number (i.e., (6.10)), the length of reference time interval $L$ plays a dominant role. The relative frequencies in an $hp$ of a node $q^k$ (i.e., $f_{it1}^{q^k}, f_{it2}^{q^k}$ and $f_{ct1}^{q^k}$) are a function of $L$. The average page access number of a node $q^k$ (i.e., $\bar{r}_{ct1}^{q^k}$) is a function of $L$ as well. For a fixed value of $L$, if there is a partition $P^q$ such that the $\bar{R}^q(L, P^q)$ is minimized, then we say that the $\bar{R}^q(L, P^q)$ is the local optimal $\bar{R}^q$, and that the $P^q$ is the local optimal partition. The global optimal partition is one of the local optimal partitions such that its corresponding local optimal $\bar{R}^q$ is the minimum of all local optimal $\bar{R}^q$'s. Different values of $L$ may result in different local optimal partitions. In this chapter, we will first discuss, for a fixed value of $L$, how to design an efficient algorithm to find the local optimal partition. For a $PMT^*$ with $n$ nodes, there are $2^{(n-1)}$ possible partitions. A trivial algorithm to find the local optimal partition is enumerating all possible partitions and comparing the resulting average page access numbers, which is essentially an exponential algorithm. To apply our design methodology in practical
applications, we must find a better algorithm. To discuss the clustering, we need to choose a clustering sequence \( C \). From the experiments that we carried out, we found that in all cases, the average page access under the condition of \( C = \text{breadth-first traversal} \) is greater than that under the condition of \( C = \text{depth-first traversal} \). Based on this finding, \( C = \text{depth-first traversal} \) is used in the our analysis. After examining the properties of \( C = \text{depth-first traversal} \), an efficient algorithm to find the local optimal partition and its corresponding local optimal \( \tilde{R}^g \) will be presented. Conceptually, to find the global optimal partition and the global optimal \( \tilde{R}^g \), we need to choose many values of \( L \) and compare all the corresponding local optimal \( \tilde{R}^g \)'s. However, from the experiments that we carried out, we found that the local optimal \( R^g \) is a nonlinear function of \( L \), and that the curve of the local optimal \( R^g \) against the \( L \) has only one minimum in the interval \((0,H)\), where \( H \) is the length of total time interval. That is the local minimum is the global minimum in the interval \((0,H)\). This fact allows us to design the algorithm for the global optimization by applying any existing "line search" algorithms, which are used in the numerical analysis to find the minimum of a nonlinear function with one variable.

All the notations defined in the Chapter 6 will be maintained.

### 7.2 DEFINITIONS OF SOME CONCEPTS

Definition 7.2.1:

Let \( q^i \in PMT^g \) and \( PMT^{g'} \) represent a subtree of the \( PMT^g \), where the \( q^i \) is the root of the subtree. If \( q^i \neq q \), then \( PMT^{g'} \) is called a true subtree.
Definition 7.2.2:

Let \( P^q \in \text{Partition-Set}(\text{PMT}^q) \) and \( P^q \in \text{Partition-Set}(\text{PMT}^q) \), where \( \text{PMT}^q \) is a true subtree of the \( \text{PMT}^q \). If \( P^q \) is a substring of \( P^q \), denoted as \( \text{sub}(P^q, P^q) \), then we say that (a) \( P^q \) is a restriction of \( P^q \), denoted as \( P^{q\mid q} \), i.e., \( P^q = P^{q\mid q} \), (b) \( P^q \) is an extension of the \( P^q \), (c) the rest part of \( P^q \) after removing the \( P^q \), denoted as \( \text{rmv}(P^q, P^q) \), is an extension-head of the \( P^q \).

Let \( \text{PMT}^q \) be a true subtree of a \( \text{PMT}^q \), \( P^q \in \text{Partition-Set}(\text{PMT}^q) \), \( P^q \in \text{Partition-Set}(\text{PMT}^q) \), and \( P^q = P^{q\mid q} \). Assume the clustering sequence \( C \) has been given. First we decompose the \( \text{PMT}^q \) based on the \( P^q \), then we decompose the \( \text{PMT}^q \) based on the \( P^q \). We adopt the following definition.

Definition 7.2.3:

If in the two decompositions above, the relative position of any two history records in \( \text{PMT}^q \) and the distance between them are identical, then \( P^q \) is called a sequence-preserving extension of \( P^q \) under the \( C \). Otherwise \( P^q \) is called a non-sequence-preserving extension of \( P^q \) under the \( C \).

We also define the concept of root-involving cluster and its size.

Definition 7.2.4:

Let \( P^q \in \text{Partition-Set}(\text{PMT}^q) \). We call a cluster of the \( P^q \) which contains the root node \( q \) a root-involving cluster or \( q \)-involving cluster. The size of a root-involving cluster is defined as the summation of the average node size of all the nodes in the
cluster. Similar definition is adopted for each \textit{hp-tree} of the $PMT^q$.

Figure 7.1 shows some examples to demonstrate the definitions above. Two primary machine trees are shown in Figure 7.1(a). $PMT^b$ is a true subtree of $PMT^a$. Two partitions on $PMT^a$ are shown in Figure 7.1(b). Two partitions on $PMT^b$ are shown in Figure 7.1(c). If we assume that $C = \text{depth-first traversal}$, then $P_1^a$ and $P_2^a$ are sequence-preserving extensions of the $P_1^b$ and $P_2^b$ respectively. However, if $C = \text{breadth-first traversal}$, then only $P_1^a$ is a sequence-preserving extension of the $P_1^b$.

In the following discussion, we assume $L$ is known. For convenience, $L$ will not appear in the formulas within the following lemmas, corollaries and theorems. Also, $^-$ is removed from the related parameters.
Lemma 7.2.1:

Assume that $C =$ depth-first traversal, $PMT^q_i$ is a true subtree of $PMT^q$, $P^q_i \in Partition-Set \ (PMT^q)$, and $P^q \in Partition-Set \ (PMT^q)$. If $\text{sub}(P^q_i, P^q)$, then the $P^q$ is a sequence-preserving extension of the $P^q_i$.

Proof:

Consider two possible cases: (a) the $PMT^q_i$ is an independent cluster in the $P^q$, (b) the $PMT^q_i$ is a part of a cluster in the $P^q$. For case (a) the claim of the lemma is straightforward. For case (b), based on the characteristics of the depth-first traversal, the relative position and distance of any two history records in the subcluster formed by the $PMT^q_i$ are not influenced by any other parts in the same cluster, so that the statement is also true. Therefore, the lemma is proved.

Lemma 7.2.2:

Under the condition of $C =$ depth-first traversal, if $q^i$'s are child nodes of the $q$, then the average page access of the $PMT^q$, denoted as $R^q(P^q)$, can be described as follows:

$$R^q(P^q) = f^q_{i1} * r^q_{i1} + f^q_{i2} + \sum_{q^i \in S(q)} \{ [f^q_{i1} * (r^q_{i1} * p_{q^i} + (1 - p_{q^i}))] + R^{q^i}(P^{q^i}) \} \quad (7.1)$$

where $R^{q^i}(P^{q^i})$ is defined as (6.10).
Proof:

We assume that a set of PMT$q^i$'s exists, where $i = 1, 2, ..., m$, and the PMT$q$ is formed by adding a node $q$ and connecting it to the roots of PMT$q^i$'s as shown in Figure 7.2. We choose a $P^q$ such that it is the sequence-preserving extension for all the $P^q^i$'s, i.e., $\forall i : P^q = P^q^i$. Therefore, according to the definition of the sequence-preserving extension, the following expression holds:

$$R^q(P^q) = R^q(P^q)$$

where $R^q(P^q)$ is defined as (6.10).

From Figure 7.2, we can write down the following formula,

$$f_{i1}^q * r_{i1}^q + f_{i2}^q + \sum_{q^i \in S(q)} \{ [f_{ei1}^q * (r_{ei1}^q * p_{q^i} + (1 - p_{q^i}))] + R^q(P^q) \}$$

or

$$f_{i1}^q * r_{i1}^q + f_{i2}^q + \sum_{q^i \in S(q)} \{ [f_{ei1}^q * (r_{ei1}^q * p_{q^i} + (1 - p_{q^i}))] + R^q(P^q) \}$$

That is, the lemma is proved.

Lemma 7.2.3:

Let PMT$q^i$ be a true subtree of a PMT$q$, and let $P_1^q, P_2^q$ be two different partitions of the PMT$q^i$, and $P_1^q, P_2^q$ be two sequence-preserving extensions of the $P_1^q$ and $P_2^q$, respectively. If (1) the extension-heads of the $P_1^q$ and $P_2^q$ are the same, (2) $R(P_1^q) \leq R(P_2^q)$, and (3) in both $P_1^q$ and $P_2^q$, $p_{q^i} = 0$, then $R^q(P_1^q) \leq R^q(P_2^q)$. 

121
Proof:

According to (3), we can decompose the $PMT^q$ into two parts: the first part consists of the $PMT^q'$, and the second part consists of the rest part of the $PMT^q$ which excludes the $PMT^q'$. Therefore, we can express the $R^q(P^q_l)$ as follows:

$$R^q(P^q_l) = f^q_{stl} + R^q(P^q_l) + R(rmv(P^q_l', P^q_l)), \quad l = 1, 2$$

Based on (1), the second parts from the $P^q_1$ and $P^q_2$ are identical, so that we have $R(rmv(P^q_1', P^q_1)) = R(rmv(P^q_2', P^q_2))$. From (2) and the analyses above, we can conclude that $R^q(P^q_1) \leq R^q(P^q_2)$, so that the lemma is proved.

### 7.3 Analysis Under $C = \text{Depth-First}$

In this section, we are only concerned with the case of $C = \text{depth-first traversal.}$

122
Theorem 7.3.1:

If $C$ is fixed as depth-first traversal, then (7.1) holds for any true subtrees of a $PMT^q$.

Proof:

The proof of the theorem is straightforward by induction on the height of the $PMT^q$ based on Lemma 7.2.1 and Lemma 7.2.2.

Theorem 7.3.2:

Assume that $C =$ depth-first traversal and $P^q \in Partition-Set(PMT^q)$. If $q^i$ is a child node of the node $q$ and $p_{q^i} = 0$ ($i = 1, 2, ..., m$, $p_{q^i} \in P^q$), then

$$min(R^{i}(P^q)) = f^{q}_{it1} \ast r^{q}_{it1} + f^{q}_{it2} + \sum_{q^i \in S(q)} [r^{q^i}_{it1} + min(R^{q^i}(P^{q1q^i}))] \quad (7.2)$$

Proof:

The proof of the theorem is straightforward by applying Theorem 7.3.1 and Lemma 7.2.3.

Let $P^{q^i}_1$ and $P^{q^i}_2$ be two partitions on a true subtree rooted at $q^i$ of a $PMT^q$, and $w^{q^i}_{1}(P^{q^i}_1)$ and $w^{q^i}_{2}(P^{q^i}_2)$ be the sizes of $q^i$-involving clusters in $P^{q^i}_1$ and $P^{q^i}_2$, respectively.

Theorem 7.3.3:

If $R^{q^i}(P^{q^i}_1) \leq R^{q^i}(P^{q^i}_2)$ and $w^{q^i}_{1}(P^{q^i}_1) \leq w^{q^i}_{2}(P^{q^i}_2)$, then for any $PMT^{q^k}$, where $q^k$ is an ancestor of the $q^i$, $R^{q^k}(P^{q^k}_1) \leq R^{q^k}(P^{q^k}_2)$, where $rmv(P^{q^k}_1, P^{q^k}_2) = rmv(P^{q^i}_1, P^{q^i}_2)$. 

123
Proof:

First, we assume \( q^k \) is the parent of the \( q^i \). Let \( P_2^{q^k} \) be any extension of the \( P_2^{q^i} \). For any such \( P_2^{q^k} \), we can find a \( P_1^{q^k} \) which is an extension of the \( P_1^{q^i} \) such that \( rmv(P_1^{q^i}, P_1^{q^k}) \) is identical to \( rmv(P_2^{q^i}, P_2^{q^k}) \). Because \( C = \text{depth-first traversal} \), according to Lemma 7.2.1, \( P_1^{q^k} \) and \( P_2^{q^k} \) are sequence-preserving extensions, so that the formula (7.1) can be applied. We only need to consider two kinds of cases: (1) in both \( P_1^{q^k} \) and \( P_2^{q^k} \), \( p_{q^k} = 0 \), and i.e., the node \( q^k \) is not in the same cluster with \( PMT^{q^i} \); (2) in both \( P_1^{q^k} \) and \( P_2^{q^k} \), \( p_{q^k} = 1 \), i.e., the node \( q^k \) is in the same cluster with the \( PMT^{q^i} \). For the case (1), according to Lemma 7.2.3, we only need to prove \( R(P_1^{q^k}) \leq R(P_2^{q^k}) \). Since this is known to be true according to the given condition, the theorem holds for case (1). For case (2), by (7.1), the following expression holds:

\[
R^k(P_1^{q^k}) = f_{i1}^{q^k} \ast r_{i1}^{q^k} + f_{i2}^{q^k} + \sum_{q^i \in S(q^i)} [f_{i1}^{q^i} \ast r_{i1}^{q^i}(P_1^{q^i}) + R^i(P_1^{q^i})], \quad l = 1, 2
\]

Since \( rmv(P_1^{q^i}, P_1^{q^k}) = rmv(P_2^{q^i}, P_2^{q^k}) \), we have

\[
R^k(P_1^{q^k}) - R^k(P_2^{q^k}) = (f_{i1}^{q^k} \ast r_{i1}^{q^k}(P_1^{q^k}) - f_{i1}^{q^k} \ast r_{i1}^{q^k}(P_2^{q^k})) + (R^i(P_1^{q^i}) - R^i(P_2^{q^i})).
\]

By (6.5) and (6.6) and the assumption of \( w_{q^i}^{q^k}(P_1^{q^i}) \leq w_{q^i}^{q^k}(P_2^{q^i}) \), we have \( d_{i1}^{q^k}(P_1^{q^k}) \leq d_{i1}^{q^k}(P_2^{q^k}) \), and hence by (6.9) we have \( r_{i1}^{q^k}(P_1^{q^k}) \leq r_{i1}^{q^k}(P_2^{q^k}) \). We have known that \( R^i(P_1^{q^i}) \leq R^i(P_2^{q^i}) \) so that \( R^i(P_1^{q^k}) \leq R^i(P_2^{q^k}) \). That is, the theorem holds for the case (2). Since the average page access expression has a recursive structure under the condition of \( C = \text{depth-first traversal} \), by induction, we can prove the claim of the theorem holds for \( q^k \) being any ancestor of the \( q^i \). Hence the theorem has been proved.
Theorem 7.3.4:

If $R^s(P^q_1) \leq R^s(P^q_2)$ and $w^{q^i}(P^q_1) \geq s$, then for any $PMT^q$ where $q^k$ is an ancestor of the $q^i$, $R^s(P^q_1) \leq R^s(P^q_2)$, where $rmt(P^q_1, P^q_2) = rmt(P^q_1') , P^q_2)$. 

Proof:

The proof is similar to that of Theorem 7.3.3. First, we assume $q^k$ is the parent of the $q^i$. Let $P^q_2$ be any extension of the $P^q_2$. For any such $P^q_2$, we can find a $P^q_1$ which is an extension of the $P^q_2$ such that $rmt(P^q_1', P^q_2)$ is identical to $rmt(P^q_1', P^q_2')$. Because $C =$ depth-first traversal, according to Lemma 7.2.1, $P^q_1$ and $P^q_2$ are sequence-preserving extensions, so that the formula (7.1) can be applied. We only need to consider two possible cases: (1) in both $P^q_1$ and $P^q_2$, $p^q = 0$, i.e., the node $q^k$ is not in the same cluster with $PMT^q$; (2) in both $P^q_1$ and $P^q_2$, $p^q = 1$, i.e., the node $q^k$ and the $PMT^q$ are in the same cluster. For case (1), according to Lemma 7.2.3, we only need to prove $R(P^q_1) \leq R(P^q_2)$. Since this is known to be true according to the given condition the theorem holds for the case (1). For case (2), we have two possibilities: (i) $w^{q^i}(P^q_1) < s$ and (ii) $w^{q^i}(P^q_1) \geq s$. If it is case (i), according to the formula (6.5) and (6.6), we have $d^{q^i}_{e11}(P^q_1) \leq d^{q^i}_{e11}(P^q_2)$, which is the situation of theorem 7.3-3. If it is case (ii), by (6.5) and (6.6), we have $d^{q^i}_{e11}(P^q_1) \geq s$ and $d^{q^i}_{e11}(P^q_2) \geq s$, so that $r^{q^i}_{e11}(P^q_1) = r^{q^i}_{e11}(P^q_2) = 1$. By (7.1), we can derive

$$R^q(P^q_1) = f_{q^i}^* \cdot r^{q^i}_{e11} + f_{q^i}^2 + \sum_{q^i \in S(q^k)} [f_{e11} + R^i(P^q_1^q)]$$

so that

$$R^q(P^q_1) - R^q(P^q_2) = R^q(P^q_1^q) - R^q(P^q_2^q) = R^q(P^q_1') - R^q(P^q_2')$$

125
We know that $R^s(P^q_s) \leq R^s(P^q_1)$ so that $R^s(P^q_1) \leq R^s(P^q_2)$. That is, the theorem holds for case (2). Since the average page access expression has a recursive structure under the $+$ condition of $C = \text{depth-first traversal}$, by induction, we can prove the claim of the theorem holds for $q^k$ being any ancestor of the $q^l$. Hence, the theorem has been proved.

Let $q^k$, a node in a $PMT^q$, be the parent of the nodes $q^{k_1}, q^{k_2}, ..., q^{k_m}$. Assume for all $k_i, i=1, 2, ..., m$, the minimal $R^s(P^{q^k_i})$'s are known, where every $P^{q^k_i}$ represents a partition on the $PMT^{q^k_i}$ such that $R^s(P^{q^k_i})$ is the minimum of all possible $R^s$'s.

**Theorem 7.3.5:**

If $\forall k_i : (1/2) \ast (z^{q^k_i} + v^{q^k_i} + z^{q^k_i} + v^{q^k_i}) \geq s_i$, then one of optimal partition will contain a $P^{q^k}$ such that $P^{q^k} = \{p_{q^{k_1}} = 0, p_{q^{k_2}} = 0, ..., p_{q^{k_m}} = 0\} \cup P^{q^k_1} \cup P^{q^k_2} \cup ... \cup P^{q^k_m}$, where $\cup$ represents the union operator.

**Proof:**

First we prove by the assumption of the theorem that if $P^{q^k} = \{p_{q^{k_1}} = 0, p_{q^{k_2}} = 0, ..., p_{q^{k_m}} = 0\} \cup P^{q^k_1} \cup P^{q^k_2} \cup ... \cup P^{q^k_m}$, then $R^s(P^{q^k})$ is the minimum. Since we know that $\forall k_i : (1/2) \ast (z^{q^k_i} + v^{q^k_i} + z^{q^k_i} + v^{q^k_i}) \geq s_i$, by (6.9) for any partition $P^{q^k}$ on the $PMT^{q^k}$, then $r^{q^k_i}_{et_1}(P^{q^k}) = 1$. Hence, we can derive

$$R^s(P^{q^k}) = f_{et_1}^{q^k} + f_{et_2}^{q^k} + \sum_{q^{k_i} \in \mathcal{E}(q^k)} [f_{et_1}^{q^{k_i}} + R^s(P^{q^{k_i}})]$$

By the assumption of the theorem and Theorem 7.3.2, we know that if $P^{q^k} = \{p_{q^{k_1}} = 0, p_{q^{k_2}} = 0, ..., p_{q^{k_m}} = 0\} \cup P^{q^k_1} \cup P^{q^k_2} \cup ... \cup P^{q^k_m},$

126
then

\[
\min(R^k(P^k)) = f_{i_1}^{k_1}r_{i_1}^{k_1} + f_{i_2}^{k_2} + \sum_{q^{k_i} \in S(t^i)} [f_{i_1}^{k_1} + \min(R^k(P^k))] 
\]

Since in the \(P^k\), we have \(p_{q^1} = 0, p_{q^2} = 0, \ldots, p_{q^{m}} = 0\), in this case the size of the root-involving cluster \(w^{q^k} = z^{q^k} \cdot v^j\) is also the minimum of all possible choices. According to Theorem 7.3.3, we can conclude that the claim of the theorem is true.

### 7.4 LOCAL OPTIMIZATION ALGORITHM

Based on the theorems above, we present an efficient algorithm which can find an optimal \(p^k\) on a \(PMT^k\) which, under the predefined \(F^r\) and \(L\) and the assumption that \(C\) is depth-first traversal, produces a minimal \(R^q\). Since this algorithm is based on the assumption that \(C\) is the depth-first traversal, we call it DEPTHFIRST. The data structures used in the DEPTHFIRST are as follows.

(i) \((R, W, P)\): for every possible partition \(P^k\) on a \(PMT^k\), a triple is used, where \(R\) contains \(R(P^k)\), \(W\) contains \(w^{q^k}(P^k)\), and \(P\) contains \(P^k\).

(ii) \(Q^k\): for a \(PMT^k\), a queue is used to contain triples \((R, W, P)\)'s, and all of them are sorted according to the increasing value of \(R\).

(iii) \(PC\text{-}Vector^k\): for any node \(q^k\) and all of its child nodes \(q^{k_1}, q^{k_2}, \ldots, q^{k_m}\), the partition variables \(p_{q^{k_1}}, p_{q^{k_2}}, \ldots, p_{q^{k_m}}\) form a parent-child vector (denoted as PC-Vector). Since every \(p_{q^{k_i}}\) can only be 1 or 0, for a PC-Vector with \(n\) partition variables, there are \(2^n\) possible instances. Every instance is called an \textit{element} of the PC-Vector.
(iv) ZV .Set^q^: for a PMT^q^, there is a set-variable to contain the average number of history records z^q^i(H) and the average size of history records v^q^i of every node q^i in the PMT^q^.

(v) for a PMT^q^, N^q^ is a variable to contain N.

(vi) for a PMT^q^, F^q^ is a set-variable to contain elements of F.

The following is the algorithm DEPTHTFIRST.

Algorithm: DEPTHTFIRST(PMT^q^, ZV .Set^q^, N^q^, F^q^)

Function Description:

This will output a triple (R,W,P) such that P contains a P^q^ which makes the R^q(P^q^)

contained in R a minima of all possible R^q^'s.

BEGIN

Step 1:

IF q^k^ has any children

THEN FOR every child q^k^i of q DO

DEPTHTFIRST(PMT^q^i^, ZV .Set^q^i^, N^q^i^, F^q^i^)

ELSE computing (R,W,P) and insert it into the Q^q^;

Step 2:

IF \forall q^i : (1/2)(z^q^i \cdot v^q^i + z^q^i \cdot v^q^i) \geq s

THEN from every Q^q^i take the first triple (R,W,P) and, based on Theorem 7.3.5 to compute a new triple (R,W,P) by using the fetched R's and P's

128
ELSE

Step 3:

FOR every element of the $PC\_Vector^{s^k}$ DO

BEGIN

Case(i): FOR every $p_{q^k_i} = 0$ DO

take the first $(R,W,P)$ from the $Q^{q^k}$;

Case(ii): FOR every $p_{q^k_i} = 1$ DO

take every $(R,W,P)$ in the $Q^{q^k}$, one by one;

FOR all the combinations of the $(R,W,P)$'s fetched above DO

BEGIN

(1) Compute a new $(R,W,P)$ for $PMT^{s^k}$;

(2) Compare it with every $(R,W,P)$ in the $Q^{q^k}$;

IF a comparison finds a pair of $(R,W,P)$'s satisfies Theorem 7.3.3

or Theorem 7.3.4

THEN insert or leave the $(R,W,P)$ whose $R$ and $W$ are smaller into the $Q^{q^k}$

END;

END;

Step 4:

Sort the $(R,W,P)$'s in the $Q^{q^k}$ according to the increasing values of $R$'s and

output the $Q^{q^k}$

END{DEPTHFIRST}.
Theorem 7.3.6:

The best time complexity of the DEPTHFIRST is $O(n)$, where $n$ is the number of nodes in a $PMT^\theta$.

Proof:

If every internal node has only child one node, we get a trivial case for the claim of the theorem. We assume every internal node may have more than one child node. In this general case, if for every pair of parent-child nodes the condition of Theorem 7.3.5 holds, then we only need go through step 2 of the algorithm for every subtree rooted at the child node which results one computation for the $(R,W,P)$ of every node, so that there are totally $n$ computations. That is, the best time complexity is $O(n)$.

Theorem 7.3.7:

The worst time complexity of the DEPTHFIRST is $O(cn)$, where $c = 1 + s^\theta / \beta$, $s$ is the page size, $\beta$ is the fanout number of an internal node of the $PMT^\theta$.

Proof:

The worst case occurs when a $PMT^\theta$ is a balanced tree and every internal node has the same fanout number. Let $\beta$ be an integer constant to represent the fanout number of an internal node in a $PMT^\theta$, $n$ be the total number of nodes of the $PMT^\theta$,
and \( h \) be the height of the \( PMT^q \). Then we have

\[
n = \frac{(\beta^h - 1)}{(\beta - 1)}. \quad (7.3)
\]

Let \( T(n, \beta) \) represent the time spent in the DEPTHFIRST. Since there are \( \beta \) child nodes fanned out from the root node \( q \), each subtree rooted at one child node of the \( q \) contains \( (n - 1)/\beta \) nodes. Due to the recursive property of the DEPTHFIRST, the time spent in each subtree is \( T(n - 1/\beta, \beta) \). The time spent on Step 3, in the worst case, is

\[
\begin{array}{c}
\binom{\beta}{\alpha} * \binom{\alpha}{1} * \binom{\alpha}{1} = \alpha^\beta, \\
\end{array}
\quad (7.4)
\]

where \( \alpha \) is the longest length of \( Q^{r^k} \). According to Theorem 7.3.4, every \( Q^{r^k} \) contains at most one \((R,W,P)\) with a value of \( W \) greater or equal to page size \( s \). Since all \( W \)'s are integers, there can be at most \( s \) elements in each \( Q^{r^k} \). Thus \( \alpha \leq s \). The time spent on Step 4 is \( O(s \log s) \). Therefore, we have

\[
T(n, \beta) = \beta \cdot T(n - 1/\beta, \beta) + s^\beta + O(s \log s)
\]

\[
= \beta^2 \cdot T(n - 1/\beta^2, \beta) + \beta \cdot s^\beta + s^\beta + O(\beta \cdot s \log s)
\]

\[
= \begin{array}{c}
\vdots
\end{array}
\]

\[
\leq \beta^{h-1} \cdot T(n - 1/\beta^{h-1}, \beta) + \beta^{h-2} \cdot s^\beta + O(\beta^{h-2} s \log s)
\]

\[
= \beta^{h-1} \cdot (n - 1/\beta^{h-1}) + \beta^{h-2} \cdot s^\beta + O(\beta^{h-2} s \log s)
\]

\[
= (n - 1) + \beta^{h-2} \cdot s^\beta + O(\beta^{h-2} s \log s)
\]

\[
= (n - 1) + \beta^{-2}[(\beta - 1) \cdot n + 1] \cdot s^\beta + O(\beta^{h-2} s \log s)
\]
\[ \leq O((1 + s^\alpha / \beta) \ast n) \]

Since both \( \beta \) and \( s \) are constants, Therefore, we have

\[ T(n, \beta) \leq O(cn) \] (7.5)

where \( c = 1 + s^\alpha / \beta \). That is, the theorem is proved.

### 7.5 ALGORITHM OF GLOBAL OPTIMIZATION

In the last section, we have discussed that under the condition that the value of \( L \) is given, by the DEPTHFIRST algorithm we can find the local optimal partition and the local optimal \( R^q \). Since the local optimal partition is a function of \( L \), to find a global optimal partition we need to concern many different values of \( L \). Conceptually, the range for choosing \( L \) is \((0, H)\), where \( H \) is the total time interval. If \( H \) is very large, searching an appropriate \( L \) by going through all the possible points in \((0, H)\) is too time consuming. Through experiments that we carried out, we found that the local optimal \( R^q \) is a nonlinear function of \( L \) and the curve of the local optimal \( R^q \) against the \( L \) has only one minimum in the interval \((0, H)\). That is, in the interval \((0, H)\), the local minimum is the global minimum. Based on this fact, to search an appropriate value of \( L \), we can apply any existing “line search” algorithms, e.g. Fibonacci Search and Gold-Cutting Search \([C+73, L+82]\) that are used in the numerical analysis to find the minimum of a nonlinear function with one variable.

In the following, we present an algorithm to find the global optimal \( R^q \) which is based on the Gold-Cutting algorithm.
Algorithm: OPTIMIZATION($PMT^q$, $H$, $L_0$, $\delta$)

Function Description:

Based on the input $PMT^q$ with all necessary parameters, the length of total time interval $H$, the starting value of $L$, $\text{mini}L$, and a predefined tolerance requirement $\delta$, this algorithm will output an $L$ and a $P^q$ such that $R^q(L, P^q)$ is the optimal one.

Subprocedures:

(1) PARAMETERS($PMT^q$, L, H)

Function Description:

This will output a set of parameters such as N, F, and ZV-Set which contains $z_q^q(L)$ and $v_q^q; \forall q^q \in PMT^q$.

(2) DEPTHFIRST($PMT^q$, ZV-Set, F, N)

Function Description:

This will output a local optimal partition and local optimal $R^q$.

Procedure Of OPTIMIZATION:

Local Variables: $a, b, L_1, L_2, R_1, R_2$;

BEGIN

\[ a := \text{mini}L; \ b := H; \]

\[ L_1 := a; \ L_2 := b; \]

FOR $L_1$ and $L_2$ DO

BEGIN

Call PARAMETERS;

Call DEPTHFIRST;

END FOR

END

133
Assign the resulted $R^v$'s to $R_1$ and $R_2$ respectively;

END

WHILE $|L_2 - L_1| > \delta$ DO

IF $R_1 > R_2$

THEN

BEGIN

$L_1 := a + 0.382 \times (b - a); a := L_1;$

Call PARAMETERS;

Call DEPTHFIRST and assign the result to $R_1$

END

ELSE { i.e., $R_1 \leq R_2$ }

BEGIN

$L_2 := a + 0.618 \times (b - a); b := L_2;$

Call PARAMETERS;

Call DEPTHFIRST and assign the result to $R_2$

END;

Output the $L$ and $P^o$ of the $R_i$ which is the minimum between $R_1$ and $R_2$;

END{OPTIMIZATION}. 

134
Chapter 8

EXPERIMENT RESULTS

8.1 INTRODUCTION

In the expression of average page access number (i.e. (6.10)), the parameters are not independent of each other. Changing the values of parameters may have an interactive, even a contradictory, influence on the reduction of average page access number $R^q$. For example, as the value of $L$ gets greater, $f_{it1}^q$'s get greater which will increase the average page access number, but $f_{it2}^q$'s get smaller which will reduce the average page access number. To have a close view of how these parameters affect each other and how they comprehensively affect $R^q$, we carried out a series of experiments. The experiments can be divided into two groups according to whether the parameter partition $P^q$ is fixed. Each group of experiments consists of a set of subgroups, each of which to investigate one specific characteristic of the clustering scheme that we proposed.

The parameters used in the experiments will be introduced in Section 8.2. In Section 8.3 we will discuss the relationships between a fixed partition $P^q$ and other parameters under the condition that $L$ is a variable. Then, in Section 8.4, we will
consider the properties of the local optimal partition and the global optimal partition and analyze the relationships among various parameters that influence the optimal partitions. Last, in Section 8.5, we will present some suggestions for applying the clustering scheme in practical applications based on the results of the experiments.

8.2 EXPERIMENT PARAMETERS

The parameters used in our experiments are listed in Table 8.1. They can be divided into two categories. The first category consists of the parameters that appear in (6.10). The second category consists of newly defined parameters which are mainly related to various kinds of performance measurements.

In the experiments, we only consider the cases that the page size is larger enough compared with the average size of history records in a node of a $PMT^γ$. A parameter called the RP-ratio (i.e., the ratio between the average size of history records $v^γ_j$ and the page size $s$) is used. In the experiments, we assumed that all nodes in a $PMT^γ$ have the same RP-ratio and we only considered three different kinds of RP-ratios: great, medium and small. Their definitions are listed in Table 8.2.

$\Sigma f_{iτ}/\Sigma f_{et}$ is a parameter used to study the impact of relative frequencies of primitive transitions. To demonstrate the relationship between the distribution of relative frequencies of primitive transactions and the primary temporal queries, we roughly classify primary temporal queries against a $PMT^γ$ into six types: shallow-level short-interval (SS), shallow-level long-interval (SL), deep-level short-interval (DS), deep-
<table>
<thead>
<tr>
<th>NOTATION</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^q$</td>
<td>a partition of a $PMT^q$</td>
</tr>
<tr>
<td>C</td>
<td>clustering sequence</td>
</tr>
<tr>
<td>L</td>
<td>length of reference time interval</td>
</tr>
<tr>
<td>H</td>
<td>length of total time interval</td>
</tr>
<tr>
<td>s</td>
<td>page size</td>
</tr>
<tr>
<td>$z^{q^k}(H)$</td>
<td>number of history records in a node $q^k$ during $H$</td>
</tr>
<tr>
<td>$z^{q^k}_{ij}$</td>
<td>average number of history records in an $hp$ of $q^k$</td>
</tr>
<tr>
<td>$v^{q^k}_{ij}$</td>
<td>average history record size in an $hp$ of $q^k$</td>
</tr>
<tr>
<td>RP-ratio</td>
<td>$v^{q^k}_{ij}/s$, ratio of average history record size to page size</td>
</tr>
<tr>
<td>$N_c$</td>
<td>number of clusters in a partition</td>
</tr>
<tr>
<td>$\sum f_{it}$</td>
<td>summation of all $f_{itx}$'s, where $x = 1,2$</td>
</tr>
<tr>
<td>$\sum f_{et}$</td>
<td>summation of all $f_{et1}$'s</td>
</tr>
<tr>
<td>$\sum f_{it}/\sum f_{et}$</td>
<td>ratio of $\sum f_{it}$ to $\sum f_{et}$</td>
</tr>
<tr>
<td>$MiniR$</td>
<td>the minimal $R^q$ under a given $L$</td>
</tr>
<tr>
<td>MaxminiR</td>
<td>maximal miniR($L$)</td>
</tr>
<tr>
<td>MiniminiR</td>
<td>minimal MiniR</td>
</tr>
<tr>
<td>MaxminiR/MiniminiR</td>
<td>ratio of MaxminiR to MiniminiR</td>
</tr>
<tr>
<td>0.03B</td>
<td>range of $L$ in which $</td>
</tr>
<tr>
<td>$P_{opt}(L)$</td>
<td>the local optimal partition causing MiniR</td>
</tr>
<tr>
<td>$P_{opt}$</td>
<td>the global optimal partition causing MiniminiR</td>
</tr>
<tr>
<td>$P_{opt}-LB$</td>
<td>range of $L$ where $P_{opt}(L) = P_{opt}$</td>
</tr>
<tr>
<td>$L_{opt}$</td>
<td>the value of $L$ where MiniminiR is found</td>
</tr>
</tbody>
</table>

Table 8.1: Parameters Used in Experiments

<table>
<thead>
<tr>
<th>Type of RP-ratio</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great</td>
<td>RP-ratio = 0.1</td>
</tr>
<tr>
<td>Medium</td>
<td>RP-ratio = 0.01</td>
</tr>
<tr>
<td>Small</td>
<td>RP-ratio = 0.001</td>
</tr>
</tbody>
</table>

Table 8.2: Three Types of RP-ratios
Table 8.3: $\sum f_{it} / \sum f_{et}$'s of Six Types of PT's

<table>
<thead>
<tr>
<th>Type of PT</th>
<th>$\sum f_{it} / \sum f_{et}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TL</td>
<td>0.01</td>
</tr>
<tr>
<td>DL</td>
<td>0.05</td>
</tr>
<tr>
<td>SL</td>
<td>0.21</td>
</tr>
<tr>
<td>TS</td>
<td>0.32</td>
</tr>
<tr>
<td>DS</td>
<td>0.66</td>
</tr>
<tr>
<td>SS</td>
<td>1.00</td>
</tr>
</tbody>
</table>

level long-interval (DL), traversal short-interval (TS), and traversal long-interval (TL).

Assume $h$ represents the height of a $PMT^q$ and the root node is at the top level (i.e., the first level). Shallow-level primary temporal queries mainly access the nodes which are at the top $[h/2]$ levels. Deep-level primary temporal queries access many nodes which are distributed on the levels that are below the $[h/2]$-th level. Traversal primary temporal queries access all nodes of the $PMT^q$. It is an extreme case of the deep-level primary temporal queries. If the time interval used in a primary temporal query is less or equal to $0.05H$, where $H$ is the total time interval, we call it a short-interval. Otherwise, we call it a long-interval. The concrete definition for the departure of the short-interval and long interval depends on the applications. For each type of primary temporal queries defined above, we chosen one example. Their corresponding $\sum f_{it} / \sum f_{et}$'s are listed in Table 8.3.

As defined in Chapter 6, $z^{q^k}(H)$ represents the number of history records that are stored in a node $q^k$ of a $PMT^q$ during the time period $H$. When the reference time interval $L$ is chosen, a $PMT^q$ is decomposed into $N$ $hp$-trees, so is $z^{q^k}(H)$. Each piece of $z^{q^k}(H)$ is denoted as $z^{q^k_i}$ and associated with one $hp$-tree. As a matter of fact,
<table>
<thead>
<tr>
<th>Type of $\bar{s} , (H)$</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very great</td>
<td>( \bar{s} , (H) = 500 )</td>
</tr>
<tr>
<td>Great</td>
<td>( \bar{s} , (H) = 100 )</td>
</tr>
<tr>
<td>Medium</td>
<td>( \bar{s} , (H) = 50 )</td>
</tr>
<tr>
<td>Small</td>
<td>( \bar{s} , (H) = 10 )</td>
</tr>
</tbody>
</table>

Table 8.4: Three Types of RP-ratios

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RANGE OF VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{P}^l )</td>
<td>all possible partitions of a PMT(^g)</td>
</tr>
<tr>
<td>C</td>
<td>depth-first, breadth-first</td>
</tr>
<tr>
<td>L</td>
<td>0.01(H) — 1.0(H)</td>
</tr>
<tr>
<td>( N_c )</td>
<td>1 — 21</td>
</tr>
</tbody>
</table>

Table 8.5: Value Ranges of Parameters

like \( \nu_h \), \( \bar{s} \) is an important factor that determines the value of \( d_{e_1} \) which directly influences the average page access number. Although \( \bar{s} \, (H) \) cannot be changed, \( \bar{s} \) can be adjusted by changing the value of \( L \). In the experiments, we assumed that all nodes in a PMT\(^g\) have the same \( \bar{s} \, (H) \), and four different kinds of \( \bar{s} \, (H) \)'s were considered. Their definitions are presented in Table 8.4.

The usage of the other newly defined parameters for the performance measurement purpose will be further described when they are used. The value ranges of some other parameters are listed in Table 8.5.
\[
C = \text{Depth-first} \quad L \in [0.01H, 1.0H] \\
P^q \text{ is fixed} \quad z^* (H) = 100 \quad \sum f_{it} / \sum f_{et} = 0.66 \\
\text{RP-ratio} = 0.001 \quad \text{RP-ratio} = 0.01 \quad \text{RP-ratio} = 0.1
\]

![Graph](image)

Figure 8.1: Effect of RP-ratio on a \( P^q \)

### 8.3 Characteristics of a Fixed \( P^q \)

The purpose of our first group of experiments was to find the characteristics of a fixed partition under the condition that the reference time interval \( L \) is a variable. The motivation behind this group of experiments is to answer the question: if we do not change an existing partition, how do the changes of the other parameters influence the average page access number. To see the effect of different parameters, the experiments are further divided into three subgroups. In each subgroup, in addition to \( L \), only one parameter is a variable. The variable used in the subgroups are the RP-ratio, the \( \sum f_{it} / \sum f_{et} \) and the \( z^* (H) \).
8.3.1 Effect of the RP-Ratio

Figure 8.1 demonstrates the effect of the RP-ratio on the performance of a given partition $P^v$ under the condition that all the other parameters are fixed but the $L$ is a variable. Each $R^v$-$L$ curve is associated with one kind of RP-ratio.

From Figure 8.1, we observe the following.

(1) The greater the RP-ratio, the greater the average page access number.

(2) The $L_{opt}$ (i.e. the value of $L$ at which the MiniminIR is found) has a certain relationship with the RP-ratio: the smaller the RP-ratio, the greater the $L_{opt}$.

(3) When the RP-ratio is small, as $L$ gets greater, the average page access number curve first shows a steep reduction, then slows until it reaches the $L_{opt}$, after which it shows a very slow increment.

(4) When the RP-ratio is medium, as $L$ gets greater, the average page access number curve first shows a fast reduction until it reaches the $L_{opt}$, then it shows a fast increment;

(5) When the RP-ratio is great, the average page access number curve first shows a fast reduction until it reaches the $L_{opt}$, then it shows a fast increment until it reaches the maximum, when it begins a very slow reduction.

Case(1) means that when all the other parameters are fixed, the average page access number is determined by the RP-ratio. Under the condition that the other parameters are fixed, changing the value of $L$ has two functions: (i) changing the
number of history records in a node of an \textit{hp-tree}, and \(\text{(ii) changing the percentage distribution between } f_{i1}^{+} \text{ and } f_{i2}^{+}. \) Case(2) means that if the RP-ratio is smaller, more history records can be contained in a node of an \textit{hp-tree}. Case(3) tells us that under the conditions that all the other parameters are fixed, if the RP-ratio is small the average page access number is not sensitive to the altered values of \(L\) in a quite large range. However, a medium RP-ratio is quite sensitive to the changes of the value of \(L\) (i.e., case(4)). Case(5) tells us that when the RP-ratio is great, if all the other parameters are fixed, then changing the value of \(L\) will almost not influence the average page access number. This is because after all the nodes of an \textit{hp-tree} have enough history records, all \(d^{+}\)'s may become equal or greater than one page size, so that all \(r^{+}\)'s become "1". In this situation, increasing the value of \(L\) is to change only the percentage distribution between \(f_{i1}^{+}\) and \(f_{i2}^{+}\), which has no big influence on the average page access number when both the value of \(L\) and the RP-ratio are greater.

According to the observations above, we may conclude that a smaller RP-ratio or a greater page size is preferred. This is because it allows us to avoid the sensitivity to the changes of the value of \(L\) and to get fewer \textit{hp-trees}. Also in this situation the average page access number becomes smaller.

\subsection*{8.3.2 Effect of \(\Sigma f_{it}/\Sigma f_{et}\)}

To investigate the influence of the relative frequencies, a group of experiments with different \(\Sigma f_{it}/\Sigma f_{et}\)'s was made. They are summarized by Figures 8.2, 8.3 and 8.4. Each of the figures reports the experiments with one kind of RP-ratio.

From these figures, we find that under a given partition, if the other parameter are
\[ C = \text{Depth-first} \quad L \in [0.01H, 1.0H] \]
\[ P^0 \text{ is fixed} \quad \mathcal{E}^k(H) = 100 \quad \text{RP-ratio} = 0.001 \]
\[ \sum f_{it}/\sum f_{et} = 0.05 \quad \sum f_{it}/\sum f_{et} = 0.21 \quad \sum f_{it}/\sum f_{et} = 0.66 \]

Figure 8.2: Effect of $\sum f_{it}/\sum f_{et}$ on a $P^0 - (1)$

fixed, then the relationship between the average page access number and the value of
$\sum f_{it}/\sum f_{et}$ is a function of $L$. A greater value of the $\sum f_{it}/\sum f_{et}$ does not necessarily
mean a greater average page access number. Both Figure 8.2 and Figure 8.3 show
that when $L$ is very small, the greater $\sum f_{it}/\sum f_{et}$, the greater average page access
number, but in a quite large range of $L$, this relationship does not hold.

The fact above can be explained as follows. When $L$ is very small, the number
of history records is greatly reduced, e.g. if each node only contains one history
record, then all IT primitive transactions are IT2's. This means that $f_{it2}^k$'s become
an important factor for the average page access number of every node $q^k$. A greater
$\sum f_{it}/\sum f_{et}$ means a greater $\sum f_{it}$, that is, a greater $\sum f_{it2}^k$. When the value of $L$
become larger, the percentage distribution between a $f_{it1}^k$ and a $f_{it2}^k$ changes. The
C = Depth-first

\[ P^\alpha \text{ is fixed} \quad z^{\alpha}(H) = 100 \quad \text{RP-ratio} = 0.01 \]

\[ \sum \frac{f_{it}}{f_{et}} = 0.05 \quad \sum \frac{f_{it}}{f_{et}} = 0.21 \quad \sum \frac{f_{it}}{f_{et}} = 0.66 \]

\[ R^\alpha \]

\[ \log_{10} L + 2 \]

Figure 8.3: Effect of \( \sum f_{it}/\sum f_{et} \) on a \( P^\alpha \) – (2)

\( \sum f_{it}^{\alpha} \) is getting smaller but it still has certain important influence on the average page access number. At this time, \( r^{\alpha}_{it1} \)'s and \( r^{\alpha}_{it1} \)'s also have very important influence on the average page access number.

### 8.3.3 Effect of the \( z^{\alpha}(H) \)

A group of experiments was done to investigate the effect of \( z^{\alpha}(H) \) on the average page access number, assuming that the values for \( z^{\alpha}(H) \) are the same for any node. The results of the experiments are reported in Figure 8.5.

From Figure 8.5, we observe the following.

(1) No matter what the values of the \( z^{\alpha}(H) \)'s, through changing \( L \), we can get the same minimal average page access number for a given partition \( P^\alpha \).
C = Depth-first \quad L \in [0.01H, 1.0H] \\
P^q \text{ is fixed} \quad z^q(H) = 100 \quad \text{RP-ratio} = 0.1 \\
\sum f_{it}/\sum f_{et} = 0.05 \quad \sum f_{it}/\sum f_{et} = 0.21 \quad \sum f_{it}/\sum f_{et} = 0.66

Figure 8.4: Effect of $\sum f_{it}/\sum f_{et}$ on a $P^q$ – (3)

(under the condition that all the other parameters are fixed).

(2) The greater the $z^q(H)$, the smaller the value of the $L_{opt}$.

(3) For a given $L$, a greater $z^q(H)$ may or may not imply a greater value of the average page access number.

(4) When the $z^q(H)$ is very small, as $L$ gets greater, then the average page access number gets smaller.

(5) When the $z^q(H)$ is getting greater but not too great, as $L$ is gets greater, the average page access number curve gets smaller at first, until the $L_{opt}$ is reached, when it gets greater.

(6) When the $z^q(H)$ is very great, the average page access number curve shows
C = Depth-first

$L \in [0.01H, 1.0H]$

$P^q$ is fixed

RP-ratio = 0.001

$\sum f_t / \sum f_{et} = 0.66$

$z^q(H) = 10 \quad z^q(H) = 50 \quad z^q(H) = 100 \quad z^q(H) = 500$

$\log_{10} L + 2$

Figure 8.5: Effect of $z^q(H)$ on a $P^q$
several local minima.

Since the effect of the $z^k(\overline{H})$ on the average page access number functions together with the RP-ratio, we can change $L$ to adjust $z^k$'s and get the same effect as by changing the page size to adjust the RP-ratio.

8.4 CHARACTERISTICS OF THE OPTIMAL PARTITION

The purpose of the second group of experiments was to investigate the characteristics of optimal partitions. There are two kinds of optimal partitions: local optimal and global optimal. A local optimal partition is the partition that can cause the minimal average page access number compared with all possible partitions under the value of $L$. A local optimal partition is denoted as $P_{opt}(L)$. The corresponding average page access number is denoted as $MiniR$. The global optimal partition is the local optimal partition of which the corresponding MiniR is the minimum of all possible MiniR's when $L$ is variable. The global optimal partition is denoted as $P_{opt}$. The corresponding MiniR is denoted as MiniminiR.

Several other parameters are used in the experiments. MaxminiR/MiniminiR is used to measure the effect of the value of $L$. The greater the MaxminiR/MiniminiR, the more important is the choice of $L$. 0.03B is used to measure the sensitivity of MiniminiR on the changes of $L$. If the 0.03B is greater, the MiniminiR is less sensitive to the changes of $L$. $P_{opt}$-LB and is used to measure the sensitivity of the $P_{opt}$ on the
changes of $L$. If the $P_{\text{opt}}$-LBandis greater, the $P_{\text{opt}}$ is less sensitive to the changes of $L$. $N_c$ is used to measure the average number of nodes in a cluster of a partition.

The second group of experiments was further divided into five subgroups. The first subgroup investigated the characteristics of MiniR. The second subgroup observed the characteristics of MiniminiR. The third subgroup studied the comprehensive effects of the changes of RP-ratio, $z^{*}(H)$, and relative frequencies of various types of primitive transactions on the MaxminiR/MiniminiR and 0.03B. The fourth subgroup unveiled the characteristics of the optimal partition $P_{\text{opt}}$ under the changes of RP-ratio, $z^{*}(H)$, and relative frequencies of various types of primitive transactions. The fifth subgroup compared the paging performances when the clustering sequence is depth-first traversal with that when the clustering sequence is breadth-first traversal. The results of these experiments are as follows.

8.4.1 Characteristics of MiniR

If we fix the other parameters such as $\sum f_{it}/\sum f_{et}$, RP-ratio and $z^{*}(H)$, and treat the MiniR as a function of $L$, we can draw a nonlinear curve. Figure 8.6 shows the effect of changing the RP-ratio on the MiniR-L curve. Figure 8.7 shows how the changing $\sum f_{it}/\sum f_{et}$ influences the MiniR-L curve under the condition that RP-ratio and $z^{*}(H)$ are fixed. Figure 8.8 shows the influence of changing the $z^{*}(H)$ on the MiniR-L curve.

From Figure 8.6, we observe that, when $z^{*}(H)$ and $\sum f_{it}/\sum f_{et}$ are fixed, the following facts hold.
Figure 8.6: Effect of RP-ratio on MiniR

(1) For all kinds of RP-ratios, except at the two ends of the $L$, the MiniR-L curve shows only one minimum.

(2) A greater RP-ratio means a greater average page access number.

(3) A greater RP-ratio also means a smaller $L_{opt}$.

From Figure 8.7, we observe that, when the $z^k(H)$ and RP-ratio are fixed, the following phenomena occur.

(4) For all kind of $\sum f_{it}/\sum f_{et}$, except at the two ends of the $L$, the MiniR-L curve shows only one minimum.

(5) The effect of $\sum f_{it}/\sum f_{et}$ on MiniR is a function of $L$.  

149
C = Depth-first  \quad L \in [0.01H, 1.0H] 
\mu is variable  \quad z^{\mu}(H) = 50  \quad \text{RP-ratio} = 0.01 
\sum f_{it} / \sum f_{et} = 0.05  \quad \sum f_{it} / \sum f_{et} = 0.21  \quad \sum f_{it} / \sum f_{et} = 0.66

Figure 8.7: Effect of \( \sum f_{it} / \sum f_{et} \) on MiniR

(6) A greater \( \sum f_{it} / \sum f_{et} \) means a greater \( L_{opt} \).

From Figure 8.8, where the RP-ratio and \( \sum f_{it} / \sum f_{et} \) are fixed, we have the following findings.

(7) For all kinds of \( z^{\mu}(H) \)'s, except at the two ends of the \( L \), the MiniR-L curve shows only one minimum.

(8) For all kinds of \( z^{\mu}(H) \)'s, MiniminiR remains the same.

(9) A greater \( z^{\mu}(H) \) means a smaller \( L_{opt} \).

(10) A greater \( z^{\mu}(H) \) means a fast increment when \( L > L_{opt} \).
Based on cases (1), (4) and (7), we conclude that, in any cases, except at two ends of $L$, the MiniR-L curve has only one minimum. This finding has been used to design the algorithm OPTIMIZATION in Chapter 7.

### 8.4.2 Characteristics of MiniminiR

The results of the experiments to observe the characteristics of MiniminiR under the changes of related parameters are presented in Figure 8.9 and Figure 8.10. Figure 8.9 summarizes the comprehensive influence of changing the RP-ratio and the $\sum f_{it}/\sum f_{et}$ on MiniminiR.

In Figure 8.9, the $\Sigma f_{it}/\Sigma f_{et}$'s of the six types of primitive transactions have been sorted to form an increasing order based on their values. From Figure 8.9, we can find the following:
(1) For each fixed $\Sigma f_{it}/\Sigma f_{et}$, as the RP-ratio gets greater, the MiniminiR gets greater as well.

(2) If the RP-ratio is very small, as the $\Sigma f_{it}/\Sigma f_{et}$ gets greater, the MiniminiR get greater as well.

(3) If the RP-ratio is not too small, a greater $\Sigma f_{it}/\Sigma f_{et}$ may not mean a greater MiniminiR.

In Figure 8.9, the $z^s(H)$ is fixed. For each fixed $z^s(H)$, the observations above are similar. However, if we fix the $\Sigma f_{it}/\Sigma f_{et}$ and allow the RP-ratio and $z^s(H)$ to change, then we get the picture shown in Figure 8.10. This is an interesting result in that if the RP-ratio and the $\Sigma f_{it}/\Sigma f_{et}$ are fixed, then the MiniminiR is independent of the changes of $z^s(H)$. This implies that if we only change the number of history records stored in a private object base, assuming that the sizes of history records are not changed and the way the user uses the object base is not varied, then we can expect the same MiniminiR by a suitable clustering. As a matter of fact, the reason for this is very simple, i.e., we can change the $z^s$ through changing $L$ to get the same value for two MiniR's which have different values of $L$. For example, assume that we have $z^s(H_1) = 50$ and $z^s(H_2) = 500$, $H_2/H_1 = 5$, that all the other parameters are the same. If we can have a MiniR which corresponds to $z^s(H_1)$ and $L = L_1$, then under the condition $L = L_1/20$ we can find a MiniR which is equal to the previous one for the case of $z^s(H_2)$.
Figure 8.9: Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on MiniminIR.

Figure 8.10: Effect of $z^{\theta}(H)$ and on MiniminIR.
8.4.3 Effect of Changing L

MaxminiR/MiniminiR and 0.03B are the parameters used to measure the effect of changing L. While MaxminiR/MiniminiR shows an abstract picture of the effect on the MiniminiR of changing L, 0.03B presents a close view of this effect. The influence of the RP-ratio, $z^a(H)$, and $\sum f_{it}/\sum f_{et}$ on these two parameters are summarized by Figures 8.11, 8.12, 8.13 and 8.14.

From Figure 8.11, we have the following observations

(1) MaxminiR/MiniminiR is very sensitive to the changes of $\sum f_{it}/\sum f_{et}$.

(2) MaxminiR/MiniminiR is very sensitive to the changes of RP-ratio.

From Figure 8.12, we observe that

(3) For a fixed $\sum f_{it}/\sum f_{et}$, MaxminiR/MiniminiR is not sensitive to the changes in $z^a(H)$, especially when the RP-ratio is small.

From Figure 8.13, we can find the following:

(4) 0.03B is very sensitive to the changes of $\sum f_{it}/\sum f_{et}$.

(5) 0.03B is very sensitive to the changes of RP-ratio.

(6) For all kinds of $\sum f_{it}/\sum f_{et}$, as the RP-ratio gets greater, 0.03B gets smaller.

(7) For all kinds of $\sum f_{it}/\sum f_{et}$, as the RP-ratio gets greater, 0.03B gets closer to the lower values of L.

From Figure 8.14, we observe the following:
Figure 8.11: Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on MaxminiR/MiniminiR

(8) When $\sum f_{it}/\sum f_{et}$ is fixed, for every kind of RP-ratio, the location of 0.03B is very sensitive to the changes of $z^*(H)$.

These observations imply that whenever any changes occur in the RP-ratio, $z^*(H)$, and/or the user access pattern, if we still want to get the MiniminiR, then we need to readjust the value of $L$, i.e., we need to redistribute the history records into $hp$-trees.

### 8.4.4 Characteristics of $P_{opt}$

The characteristics of the global optimal partition $P_{opt}$ were also investigated in our experiments. $N_c$ and $P_{opt}$-$LB$ and are two parameters defined to measure the effects on $P_{opt}$ when the RP-ratio, $z^*(H)$, and $\sum f_{it}/\sum f_{et}$ change. The results of the experiments are summarized in Figures 8.15, 8.16, 8.17 and 8.18.
From Figure 8.15, if the $z^*(H)$ is fixed, we can find the following:

1. The change of $P_{opt}$ is dependent on the $\sum f_{it}/\sum f_{et}$.

2. When the RP-ratio is not very great, then for all kinds of $\sum f_{it}/\sum f_{et}$, the $P_{opt}$ does not changed as the RP-ratio gets greater.

3. When the RP-ratio is very great, then for all kinds of $\sum f_{it}/\sum f_{et}$, the $P_{opt}$ changes as the RP-ratio gets greater.

From Figure 8.16, we have the following observations:

4. Under the condition that the RP-ratio and $z^*(H)$ are fixed, changing $\sum f_{it}/\sum f_{et}$ may cause $P_{opt}$ to change.

5. Under the condition that RP-ratio and $\sum f_{it}/\sum f_{et}$ are fixed, changing
Figure 8.13: Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on 0.03B

C = Depth-first $z^{*k}(H) = 100$

Figure 8.14: Effect of $z^{*k}(H)$ and on L of MiniminiR
$z^*(H)$ does not cause $P_{opt}$ to change.

From Figure 8.17, under the condition that the $z^*(H)$ is fixed, we observe the following:

(6) For all kinds of $\sum f_{il}/\sum f_{el}$, as the RP-ratio gets greater, the $P_{opt}$-LBand gets smaller and close to the lower values of $L$.

(7) If the RP-ratio is not great, then the $P_{opt}$-LBand is not very sensitive to the changes of $\sum f_{il}/\sum f_{el}$.

From Figure 8.18, we have the following observations:

(8) Under the condition that the RP-ratio and $z^*(H)$ are fixed, the $P_{opt}$-LBand is not sensitive to the changes of $\sum f_{il}/\sum f_{el}$.

(9) Under the condition that the RP-ratio and $\sum f_{il}/\sum f_{el}$ are fixed, as the $z^*(H)$ gets greater, the $P_{opt}$-LBand gets smaller.

### 8.4.5 Effect of the Clustering Sequence

In addition to the experiments with depth-first traversal clustering sequence, we also made a series of experiments with breadth-first traversal clustering sequence. After comparing the results of the experiments, we find that in all aspects, two kinds of clustering sequences have similar characteristics. The most interesting finding is that in all the experiments that we made, if all the other parameters are the same, then the MiniminiR in the case of depth-first traversal is always smaller or equal to the
Figure 8.15: Effect of RP-ratio and $\sum f_{it}/\sum f_{it}$ on $N_c$ of $P_{opt}$

Figure 8.16: Effect of $z^*(H)$ and on $N_c$ of $P_{opt}$
Figure 8.17: Effect of RP-ratio and $\sum f_u / \sum f_t$ on $P_{opt}$-LBand

Figure 8.18: Effect of $z^q(H)$ and on $P_{opt}$-LBand
Figure 8.19: Effect of RP-ratio and $\sum f_{it}/\sum f_{et}$ on $\text{Minimini}R_{brd} - \text{Minimini}R_{dep}$

MiniminiR in the case of breadth-first traversal. Some of results of these experiments are reported in Figure 8.19 and Figure 8.20.

### 8.5 SOME SUGGESTIONS

A user applies the clustering scheme in two cases: to create a private object base or to reorganize a private object base. The experiments provided clearer ideas about how the various kinds of related parameters affect the average page access number. Although the algorithms presented are efficient from the theoretical point of view, based on the results of our experiments more time can be saved under certain situations. The following are some suggestions about how to apply the clustering scheme efficiently in both cases above.
8.5.1 Creating a Private Object Base

When a private object base is created for the first time, in order to apply the clustering scheme, some parameters such as \( H, z^a(H), \) RP-ratio and relative frequencies must be known in advance, and some parameters such as \( C, s, P^q \) and \( L \) need to be chosen by the user and the algorithms.

Since the results of the experiments have shown that the average page access number under \( C = \) depth-first traversal is better than that under \( C = \) breadth-first traversal, we can fix our choice of \( C \) on the depth-first traversal.

The principle for choosing \( s \) is to choose as large an \( s \) as possible. This is because a larger \( s \) can cause a smaller RP-ratio. Form the experiments, when RP-ratio is small, we find that

- The \( P_{opt} \)-LBand is large and in most cases all the partition variables in \( P_{opt} \) have a value "1" (see Figures 8.15, 8.16, 8.17 and 8.18);
• The choice of $P_{opt}$ is independent of the changes of relative frequencies of primitive transaction (see Figure 8.15 and Figure 8.17);

• For a given partition, the average page reference is not sensitive to the changes of the value of $L$ around the $L_{opt}$ (see Figure 8.2).

Therefore, when the RP-ratio is small, we can assume the $P_{opt}$ is that consists of all "1" 's. Then only the value of $L$ need to be changed to find a MiniR under this partition. When the MiniR is found, at that $L$, call DEPTHFIRST to check whether the partition is still optimal.

In the experiments, regardless of the other parameters, under the condition that the $z^*(H)$ is not too large, we observed the following:

• If the RP-ratio is not too great, when $L$ is variable in $(0, H)$, an average page access number curve shows only one minimum, i.e., the MiniR is the MiniminiR, except at the two ends of the $L$ (see Figures 8.1, 8.2 and 8.3).

• If the RP-ration is great, when $L$ is variable in $(0, H)$, an average page access number curve shows only one minimum and one maximum. Still, the MiniR is the MiniminiR, except at the two ends of the $L$ (see Figures 8.1, 8.6, 8.7 and 8.8).

These facts imply that in certain conditions, we may choose very few values of $L$. This means we can save many invocations to the DEPTHFIRST. To utilize this advantage, if the $H$ is too long such that $z^*(H)$ is too large, we can decompose the $H$ into several pieces to maintain the $z^*(H)$ in an appropriate value.

163
8.5.2 Reorganizing a Private Object Base

A user may need to reorganize a private object base due to some parameters being changed. Based on the experiment results, under many situations, we need not totally destroy the existing private object base.

If the RP-ratio of the data during a given period $H$ is widely varied, i.e., in different intervals the RP-ratios may be very different, then we should treat these intervals individually. For the intervals in which the RP-ratio is small, we can treat it as a stable part. When the user access pattern changes, the stable part need not to be reorganized.

When the length of period $H$ increases or decreases, the $z^{s^k}(H)$ may change correspondingly. From the experiments, when the RP-ratio and user access pattern are fixed, we have the following observations:

- $P_{opt}$ and MimininiR are independent with the changes of $z^{s^k}(H)$ (see Figure 8.9 and Figure 8.15).

- The greater the $z^{s^k}(H)$, the smaller the $L_{opt}$ (see Figure 8.8).

Therefore, under the condition that the RP-ratio and user access pattern are fixed, the effect of $z^{s^k}(H)$ on the average page access number can be adjusted by changing $L_{opt}$ which is equivalent to changing the number of hp-trees. This allows us to suggest that if a user only wants to change the $H$ and keep the previous access pattern, then only the RP-ratio needs to be checked. If the user wants to increase the $H$ and hold the RP-ratio almost unchanged in the new time interval, then the existing part of
the corresponding private object base can be maintained as it was. The same $P_{opt}$ and $L_{opt}$ can be used to cluster the newly added history records into a set of new $hp$-trees. If the user wants $H$ to be decreased, then the corresponding $hp$-trees are removed. The rest of the $hp$-trees do not need any changes.

In practical applications, users may change their access patterns. The effect of changing the access pattern on the average page access number can be viewed from the following aspects:

- When the RP-ratio is not great, a change in user access pattern does not affect the $P_{opt}$. However, the $L_{opt}$ and MaxminiR/MiniminiR vary greatly (see Figures 8.11, 8.12, 8.13 and 8.15).

- When the RP-ratio is great, the $P_{opt}$ needs to change. However, the $L_{opt}$'s of different access patterns are very close. Also the MaxminiR/MiniminiR's of different access patterns greatly decrease their differences (see Figures 8.11, 8.12, 8.13, 8.14, 8.15 and 8.16).

- When the RP-ratio is great, for a given partition, different user access pattern changes do not affect the value of the $L_{opt}$ (see Figure 8.4).

Based on these facts, we suggest that if a user does not often change the access pattern and the RP-ratio is great, then when the user access pattern temporally changes, it may not be necessary to reorganize the private object base because the performance may not greatly decrease.
Chapter 9

CONCLUSIONS AND FUTURE DIRECTIONS

The applicability of the object model to a large number of applications requires the ability to model general temporal relationships between objects. Traditional object models lack temporal modeling constructs. They can only capture specific case of object evolution, e.g. versioning. The work in this dissertation represents an effort to overcome the deficiency in the aspect of the traditional object models.

A dynamic state machine model was developed to capture general temporal relationships between objects. In the dynamic state machine model, the whole life of an entity in the real world (called a temporal object) was modeled by a multitype composite machine consisting of a set of primary machines, each of which modeled a partial history of the temporal object when it was associated with an environment that was described by a type-version. Dynamic multityping, dynamic reference, and dynamic extension mechanisms support basic necessities of modeling changes in objects. Behavior-orientation style was adopted. The history of an object was captured by event sequence. The behavior constraints were an important construct of temporal
modeling. Three time notions, i.e., commit time, effective time and observation time, permitted us to capture different time semantics needed in databases.

By extending the linear temporal logic developed by Z. Manna and A. Pnueli to contain the temporal operators of past time, a dynamic state logic (DSL) and its proof system were developed. By using the dynamic state machines as the models of this logic language, behavior constraints on temporal objects can be specified naturally and precisely. The proof system of the DSL made it possible for us to search a proof or derive the given requirements or the expected properties from the behavior constraints defined in type-versions. A set of algorithms were presented, which could efficiently check whether a TOODB satisfies behavior constraints specified by the certain class of temporal formulas of DSL.

Among the implementation issues such as memory management, concurrency control, constraints maintenance, and recovery, we concentrated on the memory management. A paged virtual memory management system of a TOODB was developed, which consists of a two-level storage on the secondary storage and a scheme for clustering temporal objects. Unlike most existing design methodology, our design of memory management also concerned possible user access patterns rather than only the features of data organization. An analysis model is developed for the purpose of finding the optimal design. Based on the analyses, a set of efficient algorithms to optimize related parameters were designed. By changing the values of the related parameters of the analysis model, a series of experiments was carried out, describing the experimental results is helpful in understanding the characteristics of temporal
clustering. Based on the experiments results, suggestions were made for efficiently applying the temporal clustering in practical applications.

Future research will mainly have at least four directions:

- Working on other implementation issues, e.g. transaction management, high-level data management etc.

- Prototyping our data model and algorithms in real world object-oriented DBMS products.

- Extending the DSL with additional inference rules or efficient decision procedures.

- Studying an appropriate way to merge the object-oriented database and the deductive database.
Bibliography


Appendix A

THEOREMS AND DERIVED INference RULES OF DSL

Let \( w, w_1, w_2, \ldots \) be any formulas unless otherwise specified.

Theorems For Past Time

(APT1)  \( \vdash \mu w \rightarrow \nabla w \)

(APT2)  \( \vdash \emptyset w \rightarrow \nabla w \)

(APT3)  \( \vdash \Xi w \leftrightarrow \Xi \Xi w \)

(APT4)  \( \vdash \nabla w \leftrightarrow \nabla \nabla w \)

(APT5)  \( \vdash \mu (\nabla \neg w) \leftrightarrow (\neg \Xi w) \)

(APT6)  \( \vdash \Xi (w_1 \rightarrow w_2) \rightarrow (\nabla w_1 \rightarrow \nabla w_2) \)

(APT7)  \( \vdash \Xi (w_1 \wedge w_2) \leftrightarrow (\Xi w_1) \wedge (\Xi w_2) \)

(APT8)  \( \vdash \nabla (w_1 \lor w_2) \leftrightarrow (\nabla w_1) \lor (\nabla w_2) \)

(APT9)  \( \vdash (\Xi w_1 \lor \Xi w_2) \rightarrow \Xi (w_1 \lor w_2) \)

(APT10) \( \vdash \nabla (w_1 \wedge w_2) \rightarrow (\nabla w_1 \wedge \nabla w_2) \)
(APT11) \( \vdash \mu (\exists w_1 \land \nabla w_2) \rightarrow \nabla (w_1 \land w_2) \)

(APT12) \( \vdash \mu \ (w_1 \land w_2) \leftrightarrow (\forall w_1 \land \exists w_2) \)

(APT13) \( \vdash \mu \ (w_1 \lor w_2) \leftrightarrow (\forall w_1 \lor \exists w_2) \)

(APT14) \( \vdash \mu \ (w_1 \rightarrow w_2) \leftrightarrow (\forall w_1 \rightarrow \exists w_2) \)

(APT15) \( \vdash \mu \ (w_1 \leftrightarrow w_2) \leftrightarrow (\forall w_1 \leftrightarrow w_2) \)

(APT16) \( \vdash \mu \ (\exists w \leftrightarrow \exists \Theta w) \)

(APT17) \( \vdash \mu \ (\nabla w \leftrightarrow \nabla \Theta w) \)

(APT18) \( \vdash \mu \ (\exists \nabla \exists w \leftrightarrow \nabla \exists w) \)

(APT19) \( \vdash \mu \ (\nabla \exists \nabla w \leftrightarrow \exists \nabla w) \)

(APT20) \( \vdash \mu \ (\exists w \leftrightarrow (w \land \Theta \exists w)) \)

(APT21) \( \vdash \mu \ (\nabla w \leftrightarrow (w \lor \Theta \nabla w)) \)

(APT22) \( \vdash \mu \ (w \land \nabla \neg w) \rightarrow \nabla (w \land \Theta \neg w) \)

(APT23) \( \vdash \mu \ (\neg w) S w \leftrightarrow \nabla w \)

(APT24) \( \vdash \mu \ (\exists w_1 \land \nabla w_2 \rightarrow w_1 S w_2) \)

(APT25) \( \vdash \mu \ (w_1 S w_2) S w_2 \leftrightarrow (w_1 S w_2) \)

(APT26) \( \vdash \mu \ (w_1 S w_2 \rightarrow w_1 S (w_1 S w_2)) \)

(APT27) \( \vdash \mu \ (\exists w_1 \land w_2 S w_3) \rightarrow (w_1 \land w_2) S (w_1 \land w_3) \)

(APT28) \( \vdash \mu \ (\forall w_1) S (\forall w_2) \leftrightarrow \forall (w_1 S w_2) \)

(APT29) \( \vdash \mu \ (w_1 \land w_2 S w_3) \leftrightarrow (w_1 S w_3) \land (w_2 S w_3) \)

(APT30) \( \vdash \mu \ (w_1 S (w_2 \lor w_3) \leftrightarrow (w_1 S w_2) \lor (w_1 S w_3) \)

(APT31) \( \vdash \mu \ (\nabla w_1 \lor w_2) \rightarrow [(\neg w_1) S w_2 \lor (\neg w_2) S w_1] \)

(APT32) \( \vdash \mu \ (w_1 S (w_2 \land w_3) \rightarrow (w_1 S w_2) \land (w_1 S w_3) \)

176
(APT33) \( \vdash \mu (w_1 S w_3 \lor w_2 S w_3) \rightarrow (w_1 \lor w_2) S w_3 \)

(APT34) \( \vdash \mu (w_1 \rightarrow w_2) S w_3 \rightarrow (w_1 S w_3 \rightarrow w_2 S w_3) \)

(APT35) \( \vdash \mu w_1 S w_2 \land (\neg w_2) S w_3 \rightarrow w_1 S w_3 \)

(APT36) \( \vdash \mu w_1 S (w_2 \land w_3) \rightarrow (w_1 S w_2) S w_3 \)

(APT37) \( \vdash \mu (w_1 S w_2) S w_3 \rightarrow (w_1 \lor w_2) S w_3 \)

(APT38) \( \vdash \mu w_1 S (w_2 S w_3) \rightarrow (w_1 \lor w_2) S w_3 \)

(APT39) \( \vdash \mu (\forall v : \Theta w) \leftrightarrow \Theta (\forall v : w) \)

(APT40) \( \vdash \mu (\exists v : \Theta w) \leftrightarrow \Theta (\exists v : w) \)

(APT41) \( \vdash \mu (\forall v : \Xi w) \leftrightarrow \Xi (\forall v : w) \)

(APT42) \( \vdash \mu (\exists v : \nabla w) \leftrightarrow \nabla (\exists v : w) \)

(APT43) \( \vdash \mu (\exists v : \Xi w) \rightarrow \Xi (\exists v : w) \)

(APT44) \( \vdash \mu \nabla (\forall v : w) \rightarrow (\forall v : \nabla w) \)

(APT45) \( \vdash \mu (\forall v : w_1) S (\forall v : w_2) \rightarrow (\forall v : w_1 S w_2) \)

(APT46) \( \vdash \mu (\exists v : w_1 S w_2) \rightarrow (\exists v : w_1) S (\exists v : w_2) \)

(APT47) \( \vdash \mu \Xi (w_1 \lor w_2) \leftrightarrow (w_1 \lor \Xi w_2) \), where \( w_1 \) contains no local variables.

(APT48) \( \vdash \mu \nabla (w_1 \land w_2) \leftrightarrow (w_1 \land \nabla w_2) \), where \( w_1 \) contains no local variables.

177
Drived Inference Rules

(APR1) $\Theta$ Insertion Rule

\[
\Gamma \vdash \Theta \quad w \\
\hline
\Gamma \vdash \Theta \quad w
\]

(APR2) $\triangledown$ Insertion Rule

\[
\Gamma \vdash \triangledown \\
\hline
\Gamma \vdash \triangledown \\
\hline
\Gamma \vdash \triangledown w
\]

(APR3) $\Xi\Xi$ Rule

\[
\Gamma \vdash w_1 \leftrightarrow w_2 \\
\hline
\Gamma \vdash \Xi w_1 \leftrightarrow \Xi w_2
\]

(APR4) $\triangledown\triangledown$ Rule

\[
\Gamma \vdash \triangledown w_1 \leftrightarrow \triangledown w_2
\]

(APR5) $\Theta\Theta$ Rule

\[
\Gamma \vdash \Theta w_1 \leftrightarrow \Theta w_2
\]

(APR6) Computational Induction Rule

\[
\Gamma \vdash w \rightarrow \Theta w \\
\hline
\Gamma \vdash w \rightarrow \Xi w
\]

178
(APR7) Consequence Rules

(a) $\Xi$ Q Rule

\[
\Gamma \mu w_1 \rightarrow w_2 \\
\Gamma \mu w_2 \rightarrow \Xi w_3 \\
\Gamma \mu w_3 \rightarrow w_4 \\
\hline
\Gamma \mu w_1 \rightarrow \Xi w_4
\]

(b) $\nabla$ Q Rule

\[
\Gamma \mu w_1 \rightarrow w_2 \\
\Gamma \mu w_2 \rightarrow \nabla w_3 \\
\Gamma \mu w_3 \rightarrow w_4 \\
\hline
\Gamma \mu w_1 \rightarrow \nabla w_4
\]

(c) $\Theta$ Q Rule

\[
\Gamma \mu w_1 \rightarrow w_2 \\
\Gamma \mu w_2 \rightarrow \Theta w_3 \\
\Gamma \mu w_3 \rightarrow w_4 \\
\hline
\Gamma \mu w_1 \rightarrow \Theta w_4
\]
(APR8) Concatenation Rules

(a) \( \Xi C \) Rule

\[
\begin{align*}
\Gamma_{\mu} w_1 & \rightarrow \Xi w_2 \\
\Gamma_{\mu} w_2 & \rightarrow \Xi w_3 \\
\hline
\Gamma_{\mu} w_1 \rightarrow \Xi w_3
\end{align*}
\]

(b) \( \nabla C \) Rule

\[
\begin{align*}
\Gamma_{\mu} w_1 & \rightarrow \nabla w_2 \\
\Gamma_{\mu} w_2 & \rightarrow \nabla w_3 \\
\hline
\Gamma_{\mu} w_1 \rightarrow \nabla w_3
\end{align*}
\]

(APR9) Right Since Introduction Rule

\[
\begin{align*}
\Gamma_{\mu} w_1 & \rightarrow \nabla w_3 \\
\Gamma_{\mu} w_1 & \rightarrow [w_3 \lor (w_2 \land \Theta w_1)] \\
\hline
\Gamma_{\mu} w_1 \rightarrow (w_2 Sw_3)
\end{align*}
\]

(APR10) Left Since Introduction Rule

\[
\begin{align*}
\Gamma_{\mu} [w_3 \lor (w_2 \land \Theta w_1)] & \rightarrow w_1 \\
\hline
\Gamma_{\mu} (w_2 Sw_3) \rightarrow w_1
\end{align*}
\]

(APR11) SS Rule

\[
\begin{align*}
\Gamma_{\mu} w_1 & \leftrightarrow w_2 \\
\Gamma_{\mu} w_3 & \leftrightarrow w_4 \\
\hline
\Gamma_{\mu} w_1 Sw_3 \leftrightarrow w_2 Sw_4
\end{align*}
\]

180
(APR12) S Insertion Rule

\[ \Gamma, w_2 \vdash \Gamma, w_1 S w_2 \]

(APR13) S Concatenation Rule

\[ \Gamma, w_1 \rightarrow w_2 S w_3 \]
\[ \Gamma, w_3 \rightarrow w_2 S w_4 \]

\[ \Gamma, w_1 \rightarrow w_2 S w_4 \]

(APR14) Right After Introduction Rule

\[ \Gamma, w_1 \rightarrow \neg w_3 \land (w_2 \lor \Theta w_1) \]

\[ \Gamma, w_1 \rightarrow (w_2 A w_3) \]

(APR15) Left After Introduction Rule

\[ \Gamma, w_1 \rightarrow \neg w_3 \]
\[ \Gamma, w_1 \land \neg w_2 \rightarrow \Theta w_1 \]

\[ \Gamma, w_1 \rightarrow (w_2 A w_3) \]