Dynamic Constraints and Object Migration*

Jianwen Su
Computer Science Department
University of California
Santa Barbara, CA 93106-5110
U.S.A.
su@cs.ucsb.edu

Abstract

In a class hierarchy, a "role set" is the set of classes where an object may reside simultaneously. A "migration pattern" is a sequence of role sets. A "migration inventory," which is a set of migration patterns, is viewed as a dynamic constraint on object migration. A set of transactions is "sound" wrt an inventory if it generates only patterns in the inventory; "complete" if all patterns in the inventory can be generated. An initial study on characterizing migration inventories of transactions is presented. Three update languages are considered: SL which contains five operators, CSL$^+$ which extends SL with positive conditionals, and CSL which allows both positive and negative conditionals. Four kinds of inventories are studied based on *laziness* and *immediate start*. It is shown that inventories produced by SL transactions are regular and every regular inventory can be generated by SL transactions. Soundness and completeness for SL transactions are decidable. Inventories generated by CSL (CSL$^+$) transactions are r.e. and every r.e. inventory can be generated by CSL$^+$ (CSL) transactions under nonimmediate start. It is also show that every r.e. immediate-start inventory can be obtained by a left quotient of the inventory of CSL$^+$ (CSL) transactions by a regular set. The exact characterizations are open. However, every context-free set can be generated. Soundness and completeness for CSL (CSL$^+$) transactions are undecidable.

1 Introduction

Database applications are becoming more and more complex. Techniques to model, organize and manipulate behaviors, and to incorporate behaviors into databases in a systematic way are increasingly desired. The growing popularity of OODBs has evidenced this trend. Important work on dynamic aspects of databases includes practically-oriented research on behavior modeling and transaction design [Bro81, MBW80, BR84, KM85, NCL$^+$87, BMSW89], encapsulating behaviors and structural data, (e.g., [CM84]); and theoretical studies on transactions as specification languages [AV89, AV88], and dynamic constraints [Via87, Via88]. Previous studies on modeling database behaviors can be roughly categorized into two approaches. One is to use behavioral constructs to describe semantic information in a way similar to the use of data constructs in modeling structural data. Examples are the transaction composition operators [Bro81, BR84], the inflow schemas of INSYDE [KM85], and the scripts in TAXIS [MBW80, NCL$^+$87]. The other is to specify database behaviors using "dynamic constraints." Temporal logic is a typical example of this approach [CF84, dCCF82]. In this paper; we study the interrelationships between dynamic constraints, transaction languages and behavior modeling primitives.

In an object-oriented database, an object belonging to a class can be also viewed as playing a role of that class [Sci89]. It is natural to allow objects to dynamically change their roles. For examples, a Ph.D. student must to go through "pre-screening," "post-screening," "candidate" stages in the specified order to obtain a doctorate degree; a plane may be in different status, but it can never get into the "flying" state directly from being at a "repair depot." Currently, there are some proposals for systems to support such "object migration" [Sci89, Per90, DL90, RS91]. In this paper, we initiate a theoretical investigation on object migration. Specifically, we consider "migration patterns" as a new type of dynamic constraints which specify the "good" sequences of states that a database object can possibly live through.

*Supported in part by NSF grant IRI-87-19875. Part of the work was done while the author was at the University of Southern California.

These constraints are similar to "path expressions" [CH74, AS83], which is used to control concurrent operations. We focus on the problems of characterizing migration patterns of transactions and testing (in)consistency of transactions and a set of migration patterns. The framework and techniques developed here can provide part of the basis for type checking of dynamic types on transactions [HJ90], transaction design [BR84, KM85, BMSW89], and the study of methods in OODBs. They lead to a new view of behavior specifications and dynamic constraints which extends the work in [AV89, AV88, Via87, Via88].

We consider a simple semantic data model, which contains class hierarchies and attributes ranging over printable values. The model can be viewed as a proper subset of many semantic models (IFO, SDM, GSM, etc.) [HK87]. In a class hierarchy, an object can belong to several classes simultaneously and objects can migrate between classes. The set of classes in which a object lives at a time instant if a *role set*. A *migration pattern* is a sequence of role sets. A *migration inventory* is a set of migration patterns which is closed under taking prefixes. A set of transactions (possibly with parameters) is *sound* with respect to (wrt) a migration inventory if it produces only migration patterns in the inventory; it is *complete* if it can produce all patterns in the inventory. The problems studied here are (1) whether a given set of transactions (with parameters) is sound and/or complete wrt a migration inventory; (2) expressiveness of transaction languages in terms of migration inventories that transactions can produce; and finally (3) expressiveness of a behavior modeling construct in which transactions are ordered, similar to inflow schema and scripts.

Three transaction languages are studied here: SL which contains five atomic operations: create, delete, modify, generalize, and specialize; CSL^+ which is SL extended with conditionals having positive test conditions; and CSL in which test conditions may be negative. The languages extend the relational language of [AV89] to incorporate objects. The major difference is that our languages allow object-based manipulations.

We study four kinds of migration pattern, based on two independent factors: *laziness* or not, *immediate* or *delay start*. A lazy pattern "discards" consecutively repeated role sets, i.e., it "records" only when an object migrates to a different role set. The idea of "immediate start" is to focus only on patterns in which objects are created at the first step starting from the empty database. The following results are obtained. For the language SL, the set of migration patterns generated by a given finite set of SL transactions is always a regular language under each kind of pattern. As a consequence, it is decidable whether it is sound and/or complete wrt a given migration inventory. Conversely, for each regular migration inventory there is a set of SL transactions which is both sound and complete wrt it. For the extended languages, the set of migration patterns generated by a finite set of CSL (and hence CSL^+) transactions is recursively enumerable (r.e.) under each kind of pattern. When considering delay-start patterns (lazy or non-lazy), wrt each r.e. migration inventory, there is a finite set of CSL+ (CSL) transactions which is both sound and complete. When considering immediate-start patterns (lazy or nonlazy), every r.e. migration inventory is a left quotient of the set of migration patterns generated by some set of CSL^+ (CSL) transactions by a regular set. In other words, each pattern can be generated with a padding. If padding is not allowed, the exact characterizations for immediate-start patterns of CSL and CSL^+ transactions are still open. However, it is shown that every context-free language of role sets can be generated by some CSL (or CSL^+) transactions with immediate-start patterns. Consequently, it is not decidable whether a set of transactions is sound (or complete) wrt a migration inventory. Finally, we apply the obtained results and techniques to analyzing a behavior modeling construct similar to transaction design methodologies of [KM85, NCL+87, BMSW89]. The construct imposes a (precedence) relation on the transactions so that only sequences of transactions defined by the relation can be executed. It is shown that this construct does not yield richer expressiveness in terms of migration patterns.

This paper is organized as follows. Section 2 introduces the simple semantic model and the language SL. The formal notion of migration patterns is given in Section 3, and the characterization of SL transactions is provided. In Section 4, the two extensions CSL^+ and CSL are informally defined, and the results concerning them are presented. The application of the techniques is discussed in Section 5. Due to space limitation, many detailed proofs and formal definitions are omitted.

2 Preliminaries

In this section we introduce a semantic data model and an update language used in the paper. We begin with the following definitions.

Let $G = (V, E)$ be a directed graph, where V is a (finite) set of vertices and $E \subseteq V \times V$ a set of edges. A pair of vertices in V is *weakly connected* if there is an undirected path between them. A subgraph of G is

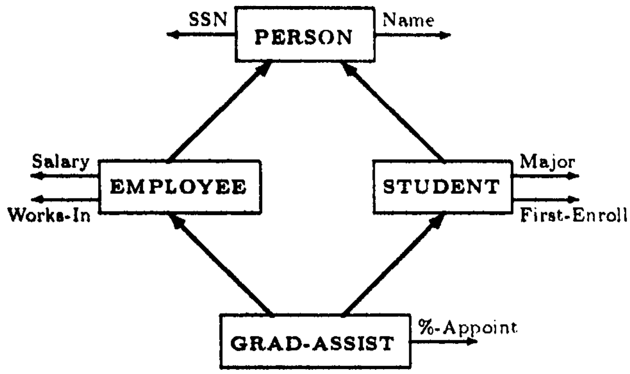

Figure 1: A Database Schema D

weakly connected if vertices are pairwise weakly connected and not weakly connected to any other vertices in G.

A graph G is a *specialization-graph* if it is acyclic and for each pair of weakly connected vertices u, v, there exists a vertex w which has directed paths from both u and v. Intuitively, a specialization graph consists of several weakly-connected components and each component has a single root which has directed paths from all other vertices in the component. This notion is motivated by ISA rules of IFO schemas [AH87].

For the formal development, we assume the existence of the following pairwise disjoint and countably infinite sets:

- $\mathcal{U} = \{a, b, c, ...\}$ of *constants*;
- $\mathcal{C} = \{P, Q, R, ...\}$ of *class names*;
- $\mathcal{A} = \{A, B, C, ...\}$ of *attribute names*;
- $\mathcal{O} = \{o_1, o_2, o_3, ...\}$ of *abstract objects*, with a total ordering $<_{\mathcal{O}}$ such that $o_i <_{\mathcal{O}} o_j$ iff $i < j$;
- $\mathcal{V} = \{x, y, z, ...\}$ of *variables*.

Definition: A *(semantic) database schema* is a triple $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$, where:

1. $\mathbf{C} \subseteq \mathcal{C}$ is a finite set of class names;
2. $\mathbf{isa} \subseteq \mathbf{C} \times \mathbf{C}$ such that $(\mathbf{C}, \mathbf{isa})$ is a specialization-graph. The reflexive and transitive closure of **isa** is denoted by $\mathbf{isa}^*$;
3. $\mathbf{A} : \mathbf{C} \to \text{powerset}^{\text{fin}}(\mathcal{A})$ is a total mapping such that $\mathbf{A}(P) \cap \mathbf{A}(Q) = \emptyset$ whenever $P \neq Q$. (This restriction is included for technical simplicity.)

Intuitively, a database schema consists of a set of classes, subclass relationships, and attributes which range over $\mathcal{U}$. Due to inheritance, the set of all attributes defined on class P is the set $\mathbf{A}^*(P) = \{A \mid \exists Q, P \ \mathbf{isa}^* \ Q \wedge A \in \mathbf{A}(Q)\}$.

(a) $\mathbf{o}(\text{PERSON}) = \{o_1, o_2, o_3, o_4, o_5\}$
$\mathbf{o}(\text{EMPLOYEE}) = \{o_1, o_3, o_4\}$
$\mathbf{o}(\text{STUDENT}) = \{o_1, o_2, o_4\}$
$\mathbf{o}(\text{GRAD-ASSIST}) = \{o_1\}$

(b) **a** is shown by the following tables

PERSON	SSN	Name
o_1	0302	Charles
o_2	3698	David
o_3	6657	Faith
o_4	9709	Chris
o_5	0067	Michelle

EMPLOYEE	Salary	Works-In
o_1	150	History
o_3	140	CS
o_4	200	EE

STUDENT	Major	First-Enroll
o_1	History	1986
o_2	CS	1988
o_4	EE	1989

GRAD-ASSIST	%-Appoint
o_1	49%

Figure 2: An Instance $d = \langle \mathbf{o}, \mathbf{a}, o_6 \rangle$

Notation: Let $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$ be a schema. A class $P \in \mathbf{C}$ is an *isa-root* if there does not exist a $Q \in \mathbf{C}$, P **isa** Q.

Definition: A *(database) instance* of a database schema $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$ is a triple $d = \langle \mathbf{o}, \mathbf{a}, o \rangle$, where

1. $\mathbf{o} : \mathbf{C} \to \text{powerset}^{\text{fin}}(\mathcal{O})$ such that $\mathbf{o}(P) \subseteq \mathbf{o}(Q)$ if P **isa** Q, and $\mathbf{o}(P) \cap \mathbf{o}(Q) = \emptyset$ if P, Q are not weakly connected;
2. $\mathbf{a}$: $\cup_{P \in \mathbf{C}}(\mathbf{o}(P) \times \mathbf{A}(P)) \to \mathcal{U}$ is a total mapping;
3. $o \in \mathcal{O}$ such that $\forall P, o'$, if $o' \in \mathbf{o}(P)$ then $o' <_{\mathcal{O}} o$.

The set of all instances of D is denoted by *inst*(D).

Intuitively, the mapping **o** assigns to a class a set of abstract objects, **a** specifies a value for each object and each appropriate attribute, and the object o is the next object to be used when objects are created into the database. In our model, each object in $\mathcal{O}$ can be "created" into a database at most once.

Example 2.1: A schema and its instance are shown in Figures 1 and 2. □

We now briefly introduce the manipulation language SL. The language extends the transaction language of [AV89] to incorporate objects and their manipulations. There are two major differences: (1) SL allows to manipulate "object identifiers" since it is based on an object-based model while [AV89] used the relational model and operations focus on tuple manipulations. For example, the operator **`create`** of SL always creates an object with an identifier, but the operator **`insert`** of [AV89] creates a tuple only when the tuple is not in the database. (2) SL has two new operators, **`specialize`** and **`generalize`**, to support object migration.

An *atomic condition* is in one of the following forms: '$A=a$,' '$A \neq a$,' '$A=x$,' or '$A \neq x$,' where $A \in \mathcal{A}$ is an attribute, $a \in \mathcal{U}$ a constant, and $x \in \mathcal{V}$ a variable. A *condition* is a set of atomic conditions. Let Γ be a condition. Γ is *ground* if it does not contain any variables. Define $Att(\Gamma) = \{A \in \mathcal{A} \mid A \text{ appears in } \Gamma\}$. For $A \in Att(\Gamma)$, A is *defined* in Γ if $A=s \in \Gamma$ for some $s \in \mathcal{V} \cup \mathcal{U}$. Let $Att_{\text{def}}(\Gamma) = \{A \mid A \text{ is defined in } \Gamma\}$.

Let $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$ be a database schema and $d = \langle \mathbf{o}, \mathbf{a}, o_i \rangle \in inst(D)$. Suppose $P \in \mathbf{C}$ is a class and Γ is a ground condition with $Att(\Gamma) \subseteq \mathbf{A}^*(P)$. For each object $o \in \mathbf{o}(P)$, the notion of "o *satisfies* Γ" is defined in the natural manner. Define $Sat(\Gamma, d, P) = \{o \in \mathbf{o}(P) \mid o \text{ satisfies } \Gamma\}$.

Intuitively, an "atomic update" with respect to some database schema is an operation on instances of the schema which satisfies some syntactic restrictions. A "transaction" is then a sequence of atomic updates.

Definition: Let $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$ be a database schema. An *atomic update* on D is an expression in one of the following forms: (The names are the initials of "create," "delete," "modify," "generalize," and "specialize.")

1. $\mathbf{c}(P, \Gamma)$, where $P \in \mathbf{C}$ is an isa-root, Γ is a condition, and $Att(\Gamma) = Att_{\text{def}}(\Gamma) = \mathbf{A}(P)$;
2. $\mathbf{d}(P, \Gamma)$, where $P \in \mathbf{C}$ is an isa-root, Γ is a condition, and $Att(\Gamma) \subseteq \mathbf{A}(P)$;
3. $\mathbf{m}(P, \Gamma, \Gamma')$, where $P \in \mathbf{C}$, Γ, Γ' are conditions, $Att(\Gamma), Att(\Gamma') \subseteq \mathbf{A}^*(P)$, and $Att_{\text{def}}(\Gamma') = Att(\Gamma')$;
4. $\mathbf{g}(P, \Gamma)$, where $P \in \mathbf{C}$ is not an isa-root, Γ is a condition, and $Att(\Gamma) \subseteq \mathbf{A}^*(P)$;
5. $\mathbf{s}(P, Q, \Gamma, \Gamma')$, where $P, Q \in \mathbf{C}$, Q **isa** P, Γ, Γ' are conditions, $Att(\Gamma) \subseteq \mathbf{A}^*(P)$, and $Att_{\text{def}}(\Gamma') = Att(\Gamma') = \mathbf{A}^*(Q) - \mathbf{A}^*(P)$.

An atomic update is *ground* if Γ (and Γ') are ground.

Definition: Let $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$ be a database schema. A *transaction* T on D is a sequence $\theta_1; \ldots; \theta_n$, where $n \geq 0$ and θ_i is an atomic update for each $i \in [1..n]$. T is *ground* if θ_i is ground for each $i \in [1..n]$. A *transaction schema* is a finite set of transactions.

In the following, we informally describe the semantics of atomic updates. Generally, the semantics of each update is a mapping on $inst(D)$. We first consider ground transactions. If the conditions are not satisfiable, then the atomic update yields the identity mapping. Now suppose condition(s) are satisfiable and $d = \langle \mathbf{o}, \mathbf{a}, o_i \rangle \in inst(D)$.

1. $[\![\mathbf{c}(P, \Gamma)]\!](d) = \langle \mathbf{o}', \mathbf{a}', o_{i+1} \rangle$ where $\mathbf{o}'$ adds the new object o_i to class P, whose attribute values are assigned by Γ.
2. $[\![\mathbf{d}(P, \Gamma)]\!](d) = \langle \mathbf{o}', \mathbf{a}', o_i \rangle$ where $\mathbf{o}'$ deletes all objects in $Sat(\Gamma, d, P)$ from d.
3. $[\![\mathbf{m}(P, \Gamma, \Gamma')]\!](d) = \langle \mathbf{o}, \mathbf{a}', o_i \rangle$ where the attribute values of objects in $Sat(\Gamma, d, P)$ are changed according to Γ'.
4. $[\![\mathbf{g}(P, \Gamma)]\!](d) = \langle \mathbf{o}', \mathbf{a}', o_i \rangle$ where $\mathbf{o}'$ removes all objects in $Sat(\Gamma, d, P)$ from P and its subclasses. Since P is not an isa-root, the objects are not completely deleted from d.
5. $[\![\mathbf{s}(P, Q, \Gamma, \Gamma')]\!](d) = \langle \mathbf{o}', \mathbf{a}', o_i \rangle$ where $\mathbf{o}'$ adds to class Q all objects in $Sat(\Gamma, d, P)$ but not in class Q, whose new attribute values are assigned by Γ'.

Definition: The *semantics* of a ground transaction $T = \theta_1; \ldots; \theta_n$ on D is a mapping on $inst(D)$ defined by $[\![T]\!] = [\![\theta_1]\!] \circ \cdots \circ [\![\theta_n]\!]$, where $f \circ g(x) = g(f(x))$.

An *assignment* is a mapping from $\mathcal{V}$ to $\mathcal{U}$, i.e., from variables to constants. In general, a transaction T may contain variables. We also denote T as $T(x_1, \ldots, x_n)$ if $x_1, \ldots, x_n$ are all variables occurring in T. If α is an assignment, $T[\alpha]$ is the transaction obtained from T by substituting all occurrences of each variable x by $\alpha(x)$. Thus, $T[\alpha]$ is a ground transaction.

Definition: The *semantics* of a transaction $T(x_1, \ldots, x_n)$ is a mapping from assignments to mappings on $inst(D)$ defined by: $[\![T(x_1, \ldots, x_n)]\!](\alpha) = [\![T[\alpha]]\!]$ for all assignments α.

3 Object Migration

In this section, we initiate the theoretical study on migration patterns in object-based models. We first introduce the notion of migration patterns and four kinds of inventory. A new type of dynamic constraints is then defined. As the main result of the section, Theorem (Theorem 3.7) states that the set of migration patterns of SL transaction schema is always a regular set, and conversely, every regular set of migration patterns can be "simulated" by some SL transaction schema.

We begin with the following informal discussion. Within a class hierarchy, an object can belong simultaneously to a set of classes, called its "role set," and can migrate to a different role set. The sequence of role sets in the object's life span is its "migration pattern." For example, an object residing in classes *PERSON* and *STUDENT* plays roles both as a person and as a student. If the object is added into *EMPLOYEE* and deleted from *STUDENT*, its role set is changed to {*PERSON*, *EMPLOYEE*}. One possible migration pattern for this object is from {*PERSON*, *STUDENT*} to {*PERSON*, *EMPLOYEE*} to

$\{PERSON\}$. A "migration inventory," which is a set of migration patterns, restricts the patterns through which objects can migrate. Here we view each migration inventory as a dynamic constraint on database updates. Our focus is to study the relationship between transaction schemas and migration inventories. the essential problem is to characterize migration inventories of transaction schemas.

To simplify the formal presentation, we assume in this section that the schema graphs are weakly connected. This is because operations in SL on one class do not depend on the contents of other unrelated classes, and objects cannot migrate to classes which are not weakly connected, The assumption is similar to focusing on a single relation in [AV89] and will be relaxed when we consider richer languages in Section 4.

Definition: A *role set* ω on a database schema $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A}\rangle$ is a subset ω of $\mathbf{C}$ such that for each class $P \in \omega$, all ancestors of P are also in ω, i.e., $\{Q \in \mathbf{C} \mid P \ \mathbf{isa}^* \ Q\} \subseteq \omega$. The *empty* role set is denoted by ω_ϕ. The set of all role sets on D is denoted by Ω. The set of non-empty role sets is denoted by Ω_+ $(= \Omega - \{\omega_\phi\})$.

Example 3.1: Consider the database schema shown in Example 2.1. The set of role sets is $\{\omega_\phi$, (G), (S), (E), (SE), (P)$\}$ where (G) means $\{GRAD\text{-}ASSIST\}$, ..., (SE) means $\{STUDENT, EMPLOYEE\}$, etc. In the instance shown in Figure 2, the role sets of o_1, o_4, and o_5 are (G), (SE), and (P) (respectively). □

Let $d = \langle \mathrm{o}, \mathbf{a}, o_i\rangle$ be an instance of D. For each object o define $RoleSet(o, d) = \{P \mid o \in \mathrm{o}(P)\}$. Note that if o does not occur in d, $RoleSet(o, d) = \omega_\phi$. The following fact states that the two operations **s** and **g** are sufficient to migrate objects between role sets.

Proposition 3.2: Let D be a database schema and $\omega_1, \omega_2 \in \Omega_+$ two nonempty role sets on D. There is a ground transaction T consisting of only $\{\mathbf{s}, \mathbf{g}\}$-operations such that if $d \in inst(D)$ and $o \in \mathcal{O}$ with $RoleSet(o, d) = \omega_1$, then $RoleSet(o, [\![T]\!](d)) = \omega_2$. □

We now consider object migration patterns, i.e., sequences of role sets through which objects can pass in their life cycles, in the context of a given transaction schema. Migration patterns are viewed as words over the alphabet Ω. In this study, we focus on patterns starting from the empty database $\langle \emptyset, \emptyset, o_1\rangle$. In general, a migration pattern of an object may start with an element in[1] ω_ϕ^* (before being created), be followed by an element in Ω_+^* (while in database), and end in an element in ω_ϕ^* again (after being deleted).

[1] We also use regular expressions to denote languages.

Definition: Suppose D is a schema and Ω the set of all role sets on D. An *object migration pattern* is a word over Ω which is in the set $\omega_\phi^* \Omega_+^* \omega_\phi^*$.
An *object migration inventory* is a set L of migration patterns such that $INIT(L) \subseteq L$, where $INIT(L) = \{x \mid \exists y \in \Omega^*, xy \in L\}$ is the set of initial words of L.

Example 3.3: Consider Example 3.1. Suppose that each person will live through exactly one continuous time period as a student, perhaps receive assistantships from some point on, and eventually be employed. This can be expressed as a migration inventory: $INIT(L)$ where[2] $L = \omega_\phi^*[\mathrm{P}]^*[\mathrm{S}]^*[\mathrm{G}]^*[\mathrm{E}]^+[\mathrm{P}]^*\omega_\phi^*$. □

Before we define the notion of a transaction schema "satisfying" a migration inventory, we discuss two orthogonal decisions that allow us to study four different kinds of migration patterns: laziness and immediate start.

Laziness concerns whether consecutively repeated role sets are included or not. In reality an object may not migrate or even be updated frequently. Lazy patterns discard all consecutively repeated role sets. Formally, we define the function ('r'emove 'r'epeats) $f_{\mathrm{rr}} : \Omega^* \to \Omega^*$ as[3]: (1) $f_{\mathrm{rr}}(\lambda) = \lambda$; (2) $f_{\mathrm{rr}}(a) = a$ if $a \in \Omega$; (3) $f_{\mathrm{rr}}(waa) = f_{\mathrm{rr}}(wa)$ if $a \in \Omega$ and $w \in \Omega^*$; and (4) $f_{\mathrm{rr}}(wab) = f_{\mathrm{rr}}(wa)b$ if $a, b \in \Omega$, $w \in \Omega^*$, and $a \neq b$.

Immediate-start patterns are patterns of those objects which are created by the first transaction executed. Thus, the first role set is not empty.

Definition: Suppose D is a schema, Ω the set of all role sets on D, and w is a migration pattern. Then, w is *nonlazy* and *delay start*. If $f_{\mathrm{rr}}(w) = w$, w is called *lazy*. If $w \in \Omega_+^* \omega_\phi^*$, w is called *immediate start*. An inventory L is *lazy* (*nonlazy*) and/or *immediate* (*delay*) *start* if L is a set of *lazy* (*nonlazy*) and/or *immediate* (*delay*) *start* patterns.

Example 3.4: Continuing from Example 3.1, (P)(S)(G)(E) is a lazy and immediate-start object migration pattern. (P)(S)(S)(S)(G)(G) is an immediate-start but not lazy migration pattern. Also, $\omega_\phi\omega_\phi$ (P)(P)(P)(S) is neither lazy nor immediate start.

Let $L = \{(\mathrm{P})(\mathrm{S})^n(\mathrm{G})^m(\mathrm{E})^k(\mathrm{P}) \mid n, m, k \geq 1\}$. Then $INIT(L)$ is an (immediate-start) migration inventory. $f_{\mathrm{rr}}(INIT(L)) = \{$(P), (P)(S), (P)(S)(G), (P)(S)(G)(E), (P)(S) (G)(E)(P)$\}$ is a lazy migration inventory. □

Example 3.5: In concurrent programming, operations on shared resources are synchronized to ensure

[2] $a^+ = aa^*$.

[3] λ is the empty word.

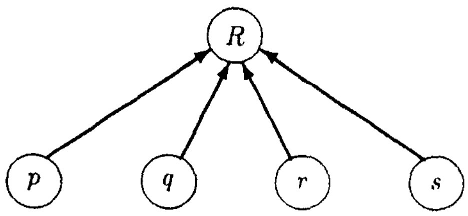

Figure 3: A Class Hierarchy for Four Operations

correctness. One synchronization mechanism is to associate with each resource a "path expression", which are regular expressions (over the set of available operations) [CH74]. Intuitively, path expressions specify the order in which operations can be executed without causing inconsistency of resources. The following illustrates that the path expressions are a special case of inventories.

Suppose B is an abstract data type with four operations p, q, r, and s. Using a schema (Figure 3) to represent four operations by four subclasses of the root, each path expression is "converted" into a migration inventory in the natural fashion. For example, suppose $(p(q \cup r)s)^*$ is a path expression of the four operations. Then, the nonlazy inventory $L = INIT(\omega_\phi^* \cdot (p(q \cup r)s)^* \cdot \omega_\phi^*)$ specifies the restriction that each transaction which simulates one operation has to obey the path expression. □

Definition: Let $\mathbf{T}$ be a transaction schema on a database schema D. An *(object) migration pattern* of $\mathbf{T}$ is an element $\omega_1 \cdots \omega_n$ in $\omega_\phi^* \cdot \Omega_+^* \cdot \omega_\phi^*$ with the following property:

1. $n \geq 0$; and
2. $\exists o \in \mathcal{O}, T_1, ..., T_n \in \mathbf{T}$, and assignments $\alpha_1, ..., \alpha_n$ such that $RoleSet(o, d_i) = \omega_i$ for $i \in [1..n]$, where $d_0 = \langle \emptyset, \emptyset, o_1 \rangle$ and $d_i = [\![T_i[\alpha_i]]\!](d_{i-1})$ for $i \in [1..n]$.

The migration inventory *generated by* $\mathbf{T}$, denoted $\mathcal{L}(\mathbf{T})$, is the set of all migration patterns of $\mathbf{T}$.

A migration pattern of $\mathbf{T}$ is *immediate start* if it is an element of $\Omega_+^* \cdot \omega_\phi^*$. A(n immediate-start) migration pattern u of $\mathbf{T}$ is *lazy* if $u = f_{rr}(u')$ for some (immediate-start) migration pattern u' of $\mathbf{T}$.

The immediate-start inventory *generated by* $\mathbf{T}$, denoted $\mathcal{L}_{\mathrm{imm}}(\mathbf{T})$, is the set of all immediate-start migration patterns of $\mathbf{T}$. The lazy (or lazy and immediate-start) migration inventory *generated by* $\mathbf{T}$ is $\mathcal{L}^{\mathrm{lazy}}(\mathbf{T}) = f_{rr}(\mathcal{L}(\mathbf{T}))$ (or $\mathcal{L}_{\mathrm{imm}}^{\mathrm{lazy}}(\mathbf{T}) = f_{rr}(\mathcal{L}_{\mathrm{imm}}(\mathbf{T}))$).

The semantics of the new family of dynamic constraints is now defined as follows:

Definition: Let $\mathbf{T}$ be a transaction schema over a database schema D and L a migration inventory over D. $\mathbf{T}$ is ***sound wrt*** L if $\mathcal{L}(\mathbf{T}) \subseteq L$; $\mathbf{T}$ is ***complete wrt*** L if $L \subseteq \mathcal{L}(\mathbf{T})$. $\mathbf{T}$ is sound [complete] ***under lazy (immediate-start) patterns*** wrt L if L is lazy (immediate start) and the lazy (immediate-start) inventory generated by $\mathbf{T}$ is contained in [contains] L.

Example 3.6: Consider the database schema shown in Figure 3 and the regular expression $p(qqp)^*$, where p (q) is a role set containing a singleton class p (q). Further let $L = INIT(\omega_\phi^* p(qqp)^* \omega_\phi^*)$. L is now a (nonlazy and delay-start) migration inventory.

We can design a transaction schema $\mathbf{T}$ to generate the inventory L. Specifically, $\mathbf{T}$ contains a single transaction: $T(x) = T_0(x); T_1(x); T_2; T_3; T_4(x)$, where

$$
\begin{aligned}
T_0(x) &= \mathtt{m}(q, \{A{=}c, B{=}x\}, \{A{=}d\}); \mathtt{d}(R, \emptyset),\\
T_1(x) &= \mathtt{g}(q, \{A = c, B{\neq}x\}); \mathtt{m}(R, \{A{=}c\}, \{A{=}a\});\\
&\qquad \mathtt{s}(R, p, \{A{=}a\}, \emptyset),\\
T_2 &= \mathtt{m}(q, \{A{=}b\}, \{A{=}c\}),\\
T_3 &= \mathtt{g}(p, \{A{=}a\}); \mathtt{s}(R, q, \{A{=}a\}, \emptyset);\\
&\qquad \mathtt{m}(q, \{A{=}a\}, \{A{=}b\}),\\
T_4(x) &= \mathtt{c}(R, \{A{=}a, B{=}x\}); \mathtt{s}(R, p, \{A{=}a\}, \emptyset).
\end{aligned}
$$

Here a, b, c, d are constants in $\mathcal{U}$. They are used to "control" migration of objects. Intuitively, the transaction T_4 will create an object in the class p. The transaction T_3 will migrate object(s) in the class p to q and T_2 will let object(s) stay in q. Transaction T_1 will finally migrate those objects whose attribute B values are not x to p to enter another migration cycle. For those objects having x as their value for attribute B, the transaction T_0 simply deletes them from the database. The parameter x is used to "randomly" determine whether objects will continue to migrate or to be deleted. □

We now present the main theorem which states that inventories of SL transaction schemas are regular.

Theorem 3.7: Let D be a database schema and Ω the set of role sets on D.

1. For any transaction schema $\mathbf{T}$:
   (a) $\mathcal{L}(\mathbf{T})$, $\mathcal{L}_{\mathrm{imm}}(\mathbf{T})$, $\mathcal{L}^{\mathrm{lazy}}(\mathbf{T})$, and $\mathcal{L}_{\mathrm{imm}}^{\mathrm{lazy}}(\mathbf{T})$ are all regular.
   (b) $\mathcal{L}(\mathbf{T}) = \omega_\phi^* \cdot \mathcal{L}_{\mathrm{imm}}(\mathbf{T})$.
2. For every regular set $L \subseteq \Omega_+^*$, there is a transaction schema $\mathbf{T}$ such that $\mathcal{L}(\mathbf{T}) = \omega_\phi^* \cdot INIT(L \cdot \omega_\phi^*)$.

Corollary 3.8: It is decidable whether a given transaction schema is sound (complete) wrt a given migration inventory represented by a regular expression. □

Define a function (remove empty initial) f_{rei} : $\Omega^* \rightarrow \Omega^*$ as: $f_{\text{rei}}(\omega_1...\omega_n) = \omega_k...\omega_n$ where $k \geq 1$, $\omega_k \neq \omega_\phi$, and for $i \in [1..k-1]$, $\omega_i = \omega_\phi$.

Corollary 3.9: (1) $\mathcal{L}^{\text{lazy}}(\mathbf{T}) = f_{\text{rr}}(\mathcal{L}(\mathbf{T}))$. (2) $\mathcal{L}^{\text{lazy}}_{\text{imm}}(\mathbf{T}) = f_{\text{rr}}(\mathcal{L}_{\text{imm}}(\mathbf{T}))$. (3) $\mathcal{L}_{\text{imm}}(\mathbf{T}) = f_{\text{rei}}(\mathcal{L}(\mathbf{T}))$. (4) $\mathcal{L}^{\text{lazy}}_{\text{imm}}(\mathbf{T}) = f_{\text{rei}}(\mathcal{L}^{\text{lazy}}(\mathbf{T}))$. □

In other words, the diagram commutes:

$$\begin{array}{ccc} \mathcal{L}(\mathbf{T}) & \xrightarrow{f_{rr}} & \mathcal{L}^{\text{lazy}}(\mathbf{T}) \\ f_{\text{rei}} \downarrow & & \downarrow f_{\text{rei}} \\ \mathcal{L}_{\text{imm}}(\mathbf{T}) & \xrightarrow{f_{rr}} & \mathcal{L}^{\text{lazy}}_{\text{imm}}(\mathbf{T}) \end{array}$$

The remainder of the section is devoted to the discussion on the proof of Theorem 3.7. Due to space limitation, only a sketch is presented.

For Part 2, it can be shown by induction that a "migration graph" can be constructed from a given regular expression. Informally, a *migration graph* is a directed graph with a distinct source and sink such that the source and sink are labelled by ω_ϕ and the rest vertices are labelled by nonempty role sets. Generalizing the argument in Example 3.6, it is straightforward to prove that a single transaction can be constructed which generates the inventory from the migration graph.

The proof of Part 1 involves constructing a migration graph for $\mathbf{T}$. In the following, we first show that objects in databases behave independently as far as updates in SL are concerned; and then present the main steps of constructing the migration graph.

Let $D = \langle \mathbf{C}, \mathbf{isa}, \mathbf{A} \rangle$ be a schema, $d = \langle \mathbf{o}, \mathbf{a}, o_i \rangle \in inst(D)$, and $I \subseteq \mathcal{O}$ be finite. The *restriction* of d onto I, denoted by $d|_I$, is an instance of D: $d|_I = \langle \mathbf{o}', \mathbf{a}', o_i \rangle$, where for each $P \in \mathbf{C}$, $\mathbf{o}'(P) = \mathbf{o}(P) \cap I$ and $\mathbf{a}' = \{((P, o, A), a) \in \mathbf{a} \mid o \in I\}$.

Lemma 3.10: If $d \in inst(D)$, T is a ground transaction, and $I \subseteq \mathcal{O}$ such that every object in I appears in d, then $[\![T]\!](d|_I) = ([\![T]\!](d))|_I$. □

Since each object behaves independently of the others, it is easy to see that if an object o has a migration pattern $\omega_\phi^i u$ generated by a sequence of transactions $T_1, ..., T_n$ (with assignments $\alpha_1, ..., \alpha_n$) where $n \geq i$, then the sequence of transactions $T_{i+1}, ..., T_n$ will generate the migration pattern u for some object o'. Thus, Part 1(b) holds. Furthermore, the subsequence of all transactions in $T_1, ..., T_n$ which changed the role set of o generates the lazy migration pattern $f_{\text{rr}}(\omega_\phi^i u)$. Hence, Corollary 3.9 follows. Consequently, if $\mathcal{L}_{\text{imm}}(\mathbf{T})$ is regular, then $\mathcal{L}(\mathbf{T})$ is regular. Since the family of regular sets is closed under homomorphism [Har78], $\mathcal{L}^{\text{lazy}}(\mathbf{T})$ and $\mathcal{L}^{\text{lazy}}_{\text{imm}}(\mathbf{T})$ are also regular. It remains to be shown that $\mathcal{L}_{\text{imm}}(\mathbf{T})$ is indeed regular.

Lemma 3.10 allows to focus on each individual object when studying migration patterns. To construct the migration graph, we extend the notion of "hyperplane" [AV89] to partition the object space (with respect to role sets and attribute values) so that elements in the same subspace are not distinguishable by $\mathbf{T}$. Now suppose $S = \{A_1, ..., A_n\}$ is a set of attributes and $C = \{a_1, ..., a_k\}$ is a set of constants. The partition $\pi_C(S)$ is obtained through the following procedure.

First, define a ***hyperplane*** on S wrt C to be a condition $\{\xi_1, ..., \xi_n\}$, where $\forall i \in [1..n], \xi_i \in \{(A_i{=}a_1), ..., (A_i{=}a_k), (A_i{\neq}a_1, ..., a_k)\}\}$.

Let Γ to be a hyperplane. Define $Att^{\neq}(\Gamma) = \{A \mid (A{\neq}a_1, ..., a_k) \in \Gamma\}$ and $E_\Gamma = \{(A, A') \mid A, A' \in Att^{\neq}(\Gamma) \wedge A \neq A'\}$. For each $r \subseteq E_\Gamma$, let r^* denote the reflexive and transitive closure of r (relative to $Att^{\neq}(\Gamma)$). Define an equivalence relation on r's: $r_1 \equiv_\Gamma r_2$ if $r_1^* = r_2^*$. If $Att^{\neq}(\Gamma)$ is nonempty, $\equiv_\Gamma$ yields a partition $\{[r_1^*], ..., [r_k^*]\}$ and Γ is partitioned further into $\{(\Gamma, [r_j^*]) \mid j \in [1..k]\}$. Otherwise, Γ is not partitioned and it is also denoted as $(\Gamma, \emptyset)$ for technical convenience.

Now let $\pi_C(S) = \{(\Gamma, [r^*]) \mid$ a hyperplane and $r \subseteq E_\Gamma$ or $r^* = \emptyset\}$.

Let $Att(\omega)$ be the set of all attributes defined on classes in a role set ω. Suppose C contains exactly all constants in $\mathbf{T}$. Define $V_\mathbf{T} = \{(\omega, p) \mid \omega \in \Omega_+, p \in \pi_C(Att(\omega))\}$.

Given $v = (\omega, (\Gamma, [r^*])) \in V_\mathbf{T}$, for an object o occurring in a database d, o *matches* v if $RoleSet(o, d) = \omega$, $o \models \Gamma$, and the attributes whose value is not in C satisfies the equality relation r^*. It is obvious that every object of d matches exactly one v in $V_\mathbf{T}$.

Lemma 3.11: Let $v_1 = (\omega_1, (\Gamma_1, [r_1^*]))$ and $v_2 = (\omega_2, (\Gamma_2, [r_2^*]))$ be two vertices in $V_\mathbf{T}$. It can be decided if there exist a database instance d consisting of a single object o matching v_1, an assignment α, and a transaction $T \in \mathbf{T}$ such that[4] $[\![T[\alpha]]\!](o)$ matches v_2.

Proof: (Sketch) We first construct an object o such that its role set is ω_1 and its attribute values satisfy Γ_1 and r_1^* with new values for attribute in $Att^{\neq}(\Gamma)$.

Claim: There exist a database instance containing o which matches v_1 and an assignment α such that $[\![T[\alpha](o)]\!]$ matches v_2 iff there exists an assignment α' such that $[\![T[\alpha'](o_1)]\!]$ matches v_2.

[4] $[\![T[\alpha]]\!](o)$ denotes the object in the output.

The if part is trivial. For the only if part, suppose there exist a d and α which satisfy the above conditions. Since both o, o_1 match v_1, there is a isomorphism ρ between o, o_1. We now define $\alpha'(x) = \rho(\alpha(x))$ if $\alpha(x) = o.A$ for some attribute A and a new value otherwise, such that $\forall x, y,\ \alpha(x) = \alpha(y)$ iff $\alpha'(x) = \alpha'(y)$. It can then be shown that $[\![T[\alpha']]\!](o_1)$ matches v_2.

By the above claim, only finite number of assignments need to be examined. The decidability result follows easily. □

Define $E = \{(u, v) \mid$ there exist a database d containing an object o matching u, an assignment α and a transaction $T \in \mathbf{T}$, such that $[\![T[\alpha]]\!](o)$ matches $v\}$. The edges of the migration graph to be constructed include E and also the edges corresponding to object creations and deletions. Since the constructions for these cases are similar to the case discussed, we leave out the construction (details in full paper). Let v_s, v_t be two distinct vertices representing the source and the sink and E' the set of edges emanating from v_s to $V_{\mathbf{T}}$ and from $V_{\mathbf{T}}$ to v_t. Finally, the graph $G = (V_{\mathbf{T}} \cup \{v_s, v_t\}, E \cup E', Label)$ is the migration graph of $\mathbf{T}$, where *Label* assigns ω_ϕ to v_s, v_t and ω to $(\omega, p) \in V_{\mathbf{T}}$.

Using an induction, it can be shown that:

Lemma 3.12: (1) Let G be the constructed migration graph and $o \in \mathcal{O}$. Then the migration pattern of o is $\omega_\phi^* \cdot p_s$, where p_s is a walk[5] in G starting from v_s. (2) If v_s has at least one outgoing edge, then for each walk $v_s = v_0, \ldots, v_n$, each $o \in \mathcal{O}$, there exist $m \geq 0$, assignments $\alpha'_1, \ldots, \alpha'_m, \alpha_1, \ldots, \alpha_n$, and transactions $T'_1, \ldots, T'_m, T_1, \ldots, T_n \in \mathbf{T}$ such that if

$$d = [\![T'_1[\alpha'_1]; \ldots; T'_m[\alpha'_m]]\!](\langle \emptyset, \emptyset, o_1 \rangle),$$

then o does not occur in d and for each $i \in [1..n]$,

$$([\![T_1[\alpha_1]; \cdots; T_i[\alpha_i]]\!](d))|_{\{o\}} \text{ matches } v_i. \ \square$$

From the above lemma, it is straightforward that the patterns in $\mathcal{L}_{\mathsf{imm}}(\mathbf{T})$ is the set of all walks starting from v_s in G. It is then easy to construct from G a regular grammar corresponding to all walks departing from v_s. This concludes the proof of Theorem 3.7.

4 Extended Languages

We consider two extensions to SL, which has the ability to test before executing an update. We informally describe the languages first and then state the results.

[5]A *walk* of a graph (V, E) is a sequence of vertices (not necessarily distinct) $v_1, \ldots, v_n$, where $\forall i \in [1..n], v_i \in V$ and $\forall i \in [1..(n-1)], (v_i, v_{i+1}) \in E$.

Let D be a database schema. A ***positive literal*** (***negative literal***) is of form '$P(\Gamma)$' ('$\neg P(\Gamma)$'), where P is a class in D and Γ a condition such that $Att(\Gamma) \subseteq \mathbf{A}^*(P)$. A *literal* is either a positive literal or a negative literal. For any $d \in inst(D)$, ***satisfaction*** of literals is defined in the natural manner.

A *conditional update* is of the form: '$\delta_1, \ldots, \delta_n \rightarrow \theta$,' where $n \geq 1$, δ_i's are literals, and θ is an atomic update. The conditional update is ***positive*** if δ_i's are positive. The semantics is defined in the natural fashion: the atomic update is executed if all literals are satisfied. Note that they are restricted conditionals since variables local to a conditional are not allowed. A CSL (CSL$^+$) transaction is a sequence of (positive) conditional or atomic updates. A CSL (CSL$^+$) transaction schema is a finite set of CSL (CSL$^+$) transactions.

In CSL$^+$ (CSL), isolated classes in a schema can be "connected" by testing literals, so results similar to Lemma 3.10 do not hold. Thus, weak connectivity of schema is not assumed. However, we still focus on migration patterns with respect to some weakly connected component. In the formal framework, we extend the relevant definitions. For example, if D is a database schema, a role set must be a subset of a weakly-connected component; if G is a weakly-connected component, Ω_G denotes all nonempty role sets on G. If $\mathbf{T}$ is a transaction schema, $\mathcal{L}(\mathbf{T}, G)$ denotes $\mathcal{L}(\mathbf{T}) \cap \omega_\phi^* \cdot \Omega_G^* \cdot \omega_\phi^*$.

Theorem 4.1: For a CSL$^+$ (CSL) transaction schema $\mathbf{T}$, $\mathcal{L}(\mathbf{T})$, $\mathcal{L}_{\mathsf{imm}}(\mathbf{T})$, $\mathcal{L}^{\mathsf{lazy}}(\mathbf{T})$, and $\mathcal{L}^{\mathsf{lazy}}_{\mathsf{imm}}(\mathbf{T})$ are r.e.

Proof: We consider only $\mathcal{L}(\mathbf{T})$, the other cases leaving similar. Notice that $inst(D)$ is r.e., the number of variables in $\mathbf{T}$ is finite, and for each $d \in inst(D)$ there are only finitely many assignments which are not isomorphic to each other. It is easy to construct a Turing machine M which checks if a pattern is in $\mathcal{L}(\mathbf{T})$. □

Theorem 4.2: If D be a schema containing at least two weakly-connected components G, S, where S has at least four attributes, then, for each r.e. set $L \subseteq \Omega_G^*$:

1. there exists a CSL$^+$ (CSL) transaction schema $\mathbf{T}$ such that $\mathcal{L}(\mathbf{T}, G) = \omega_\phi^* \cdot INIT(L \cdot \omega_\phi^*)$ and $\mathcal{L}^{\mathsf{lazy}}(\mathbf{T}, G) = \omega_\phi \cdot f_{\mathrm{rr}}(INIT(L \cdot \omega_\phi^*))$;
2. if G has at least two classes then there exists a CSL$^+$ (CSL) transaction schema $\mathbf{T}$ such that $\{\omega_1^* \omega_2\}^{-1} \mathcal{L}_{\mathsf{imm}}(\mathbf{T}, G) = INIT(L \cdot \omega_\phi^*)$ and $\{\omega_1 \omega_2\}^{-1} \mathcal{L}^{\mathsf{lazy}}_{\mathsf{imm}}(\mathbf{T}, G) = f_{\mathrm{rr}}(INIT(L \cdot \omega_\phi^*))$ for some $\omega_1, \omega_2 \in \Omega_G$, where $X^{-1}Y$ is the left quotient [Har78] of Y by X.

Corollary 4.3: There exist non-recursive migration inventories for CSL$^+$ (CSL) transaction schemas. □

Corollary 4.4: There is some inventory L such that it is not decidable if a CSL$^+$ (CSL) transactions schema is sound (complete) wrt L. □

The proof of Theorem4.2 is based on simulating Turing machines. The objects in class S will hold an encoding of a Turing computation. There are transactions to generate an input word, simulate each move, and if the computation halts, create an object in classes in G and migrate it according to the accepted word. When we consider immediate-start patterns, ω_1 is used while simulating computations. If the computation halts, ω_2 sets a mark and the pattern is then produced. Thus, each word in the r.e. set is produced with a padding. What if padding is not allowed? The exact characterizations are remain open at this point. The following theorem partially answers the question. The proof uses Greibach normal forms, where each application of a production generates at least one terminal — a "real time" property that CSL$^+$ (CSL) transactions can simulate.

Theorem 4.5: Let D be a schema containing at least two weakly-connected components G, S, where S has at least three attributes. Then, for each context-free set $L \subseteq \Omega_G^*$, there is a CSL$^+$ transaction schema $\mathbf{T}$: $\mathcal{L}_{\text{imm}}(\mathbf{T}, G) = INIT(L \cdot \omega_\phi^*)$. □

5 Applications

We now illustrate through two examples how the techniques and results obtained here can be applied to many practical problems. The two examples are motivated by the INSYDE model [KM85] and scripts in the TAXIS data model [MBW80, NCL$^+$87]. The essential ingredient in both frameworks is to introduce a precedence relation over update transactions.

Definition: An *SL* (*CSL*$^+$, *CSL*) *inflow schema* over a schema D is a pair $\mathbf{T}^{\text{inf}} = (\mathbf{T}, E)$, where $\mathbf{T}$ is an SL (CSL$^+$, CSL) transaction schema on D and $E \subseteq \mathbf{T} \times \mathbf{T}$.

A sequence of transactions $T_1, ..., T_n$ is *well-formed* under $\mathbf{T}^{\text{inf}}$ if T_i's are in $\mathbf{T}$ and $(T_i, T_{i+1}) \in E$ for each $i \in [1..(n-1)]$.

It turns out that this precedence relation does not increase expressive power in terms of producing migration patterns.

Theorem 5.1: The inventories of SL (CSL$^+$, CSL) inflow schemas correspond to regular (r.e.) sets (respectively).

Within this framework, it is interesting to answer questions such as "will a student currently majoring in history work in business office with salary $\geq 35K$ in the future?" and "will an airplane which belongs to United Airlines be in the repair depot at Los Angeles International Airport?" In some situations such information can be used to detect mistakes in the data to be added into a database.

Example 5.2: Consider a database used by an office of Immigration Service in country X. According to the immigration law, before a person entering the country with a type C visa can be allowed to immigrate, s/he has to go back to his/her own country (defined as the country s/he was a citizen of just before s/he entered the country X) and stay for at least 3 years. The transactions designed for this application have to guarantee that no one can *directly* change his/her status from visa type C to an immigrant. □

For a class P, a *property* on P is a set of formulas of form "$A{=}a$" or "$A{=}B$," where $A, B \in \mathbf{A}^*(P)$ are attributes and $a \in \mathcal{U}$ a constant.

Reachability Problem: Given a database schema D, an inflow schema $\mathbf{T}^{\text{inf}}$, classes P, Q, and properties ρ_1, ρ_2 on P, Q (respectively), for any $d \in inst(D)$ which contains an object o in class P satisfying ρ_1, can o be updated by a well-formed sequence of transactions to class Q with property ρ_2?

Theorem 5.3: Reachability is decidable for SL inflow schemas and undecidable for CSL$^+$ (CSL) inflow schemas. □

The precedence relation on transactions is sometimes unnatural. We can also refine it to specify the order on updates for each object, motivated by scripts in TAXIS. Syntactically, a script schema is the same as an inflow schema. Semantically, the ordering is interpreted globally for inflow schemas but at the level of objects for script schemas (details in full paper). It is not hard to see that the above results can basically be translated into this framework. For example, we can show that:

Theorem 5.4: The reachability problem is decidable for SL script schemas and undecidable for CSL$^+$ (CSL) script schemas. □

Acknowledgment

The author thanks Rick Hull for his encouragement and stimulating discussions during the course of this research.

References

[AH87] S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Trans. on Database Systems*, 12(4):525–565, 1987.

[AS83] G.R. Andrews and F.B. Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys*, 15(1):3–44, March 1983.

[AV88] S. Abiteboul and V. Vianu. The connection of static constraints with determinism and boundness of dynamic specifications. In C. Beeri, J.W. Schmidt, and U. Dayal, editors, *Proc. 3rd int. conf. on Data and Knowledge Bases*, pages 324–334, Jerusalem, Israel, June 1988.

[AV89] S. Abiteboul and V. Vianu. A transaction-based approach to relational database specification. *Journal of the ACM*, 36(4):758–789, October 1989.

[BMSW89] A. Borgida, J. Mylopoulos, J.W. Schmidt, and I. Wetzel. Support for data-intensive applications: Conceptual design and software development. In R. Hull, R. Morrison, and D. Stemple, editors, *Proc. 2nd Int. Workshop on Database Programming Languages*. Morgan Kaufmann Publishers, 1989.

[BR84] M.L. Brodie and D. Ridjanovic. On the design and specification of database transactions. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling*, pages 277–306. Springer-Verlag, 1984.

[Bro81] M.L. Brodie. On modelling behavioural semantics of databases. In *Proc. Int. Conf. on Very Large Data Bases*, pages 32–42, 1981.

[CF84] M.A. Casanova and A.L. Furtado. On the description of database transition constraints using temporal constraints. In H. Gallaire, J. Minker, and J.M. Nicolas, editors, *Advances in Data Base Theory*, volume 2, pages 221–236. Plenum Press, New York, 1984.

[CH74] R.H. Campbell and A.N. Habermann. The specification of process synchronization by path expression. In E. Gelenbe and C. Kaiser, editors, *Proc. of Int'l Symp. on Operating Systems*, volume 16 of *Lecture Notes in Computer Science*, pages 89–102. Springer-Verlag, 1974.

[CM84] G. Copeland and D. Maier. Making Smalltalk a database system. In *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, 1984.

[dCCF82] J.M.V. de Castilho, M.A. Casanova, and A.L. Furtado. A temporal framework for specifications. In *Proc. Int. Conf. on Very Large Data Bases*, pages 280–291, 1982.

[DL90] G. Dong and Q. Li. Object migration in object-oriented databases. Manuscript, 1990.

[Har78] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[HJ90] R. Hull and D. Jacobs. On the semantics of rules in database programming languages. Technical report, USC Computer Science Dept., 1990.

[HK87] R. Hull and R. King. Semantic data modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[KM85] R. King and D. McLeod. A database design methodology and tool for information systems. *ACM Trans. on Office Information Systems*, 3(1):2–21, January 1985.

[MBW80] J. Mylopoulos, P.A. Bernstein, and H.K. Wong. A language facility for designing database-intensive applications. *ACM Trans. on Database Systems*, 5(2):185–207, June 1980.

[NCL+87] B.A. Nixon, K.L. Chung, D. Lauzon, A. Borgida, J. Mylopoulos, and M. Stanley. Design of a compiler for a semantic data model. Technical Report CSRI-44, Dept. of Computer Science, Univ. of Toronto, 1987.

[Per90] B. Pernici. Objects with roles. In *Proc. of Conf. on Office Information Systems*, pages 205–215, 1990.

[RS91] J. Richardson and P. Schwarz. Aspects: Extending objects to support multiple, independent roles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, May 29-31 1991.

[Sci89] E. Sciore. Object specialization. *ACM Trans. on Office Information Systems*, 7(2):103–122, April 1989.

[Via87] V. Vianu. Dynamic functional dependencies and database aging. *Journal of the ACM*, 34(1):28–59, January 1987.

[Via88] V. Vianu. Database survivability under dynamic constraints. *Acta Informatica*, 25:55–84, 1988.