Applications of Logic

Applied Logic

Department of Computer Science

Western
UNIVERSITY · CANADA

Relational Databases

- **Databases** store **massive amount** of **persistent** data **efficiently**
- They are **reliable** and provide **safe access** to **multiple users**
- **Database management systems (DBMSs):** Software systems that store and manage databases
- They use **relational data model** for storing data in tables:
  - A database consist of a **schema** and a database **instance**

Introduction to Databases (CS3319) and **Database II (CS4411)**

Database Schemas

- A k-ary **relation schema** $\boldsymbol{R(A_1, \dots, A_k)}$ consists of
  - **Relation name** R
  - **Attributes** $\{A_1, \dots, A_k\}$
- Each attribute A_i has a **domain** $dom(A_i)$ of values
- **Example:** $Account(accoun\_no, customer, branch, balance)$

  $dom(balance) = \mathbb{R}^+ \cup \{0\} \quad dom(account\_no) = \mathbb{Z}^+$

  $$Branch(branch\_no, address)$$
- **Database schema:** A set of relation schemas

  $$Sch = \{Account, Branch, Customer\}$$

Database Instances

- **Relation instance $\boldsymbol{D}$** of a relation schema R:
  $\boldsymbol{D \subseteq dom(A_1) \times \cdots \times dom(A_k)}$
- **Database instance $\boldsymbol{I}$** with schema Sch:
  - Relation instances $D_1, \dots, D_n$ of the relation schemas in Sch
- **Example:** Account and Branch

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
7758	888013	Prince	5,000
0159	863279	Ali	250
5982	789815	Zoya	7,000

Branch

branch_no	address
9205	782 Dovetail Estates
3244	746 Andell Road
2501	3605 Oakridge Lane

Structured Query Language (SQL)

- **SQL** is a language for querying relational data

- **Example:**

```
SELECT account_no, customer
FROM Account
WHERE
        branch = 9205 AND
        balance > 3,000
```

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
7758	888013	Prince	5,000
0159	863279	Ali	250
5982	789815	Zoya	7,000

Integrity Constraints

- **Integrity constraints** prohibit some database instances
- **Example:**
  - account_no is a **primary key** in Account
  - branch is a **foreign key** and refers to the primary key (branch_no) in Branch

There is no pair of accounts with the same account number!

Every account has a valid branch

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
7758	888013	Prince	5,000
0159	863279	Ali	250
5982	789815	Zoya	7,000

Branch

branch_no	address
9205	782 Dovetail Estates
3244	746 Andell Road
2501	3605 Oakridge Lane
…	…

Relational Databases and Predicate Logic

- Two approaches for **representing relational databases in first-order predicate logic** (Raymond Reiter, 1984):
  - **Model theoretic**
  - **Proof theoretic**

Model Theoretic Approach

- A database instance is represented as **a structure (model)** $\mathcal{S}$

- $U_{\mathcal{S}}$ is the union of all attribute domains

- **Example:** $U_{\mathcal{S}} = \{9205, 3244, \ldots,$
  $Ivano, Teresa, \ldots, 500, \ldots\}$

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

Model Theoretic Approach

- Relations correspond to predicates
- **Relation instances** specify **predicate interpretations**
- **Example:**

$$Account_S = \{(9205, 890516, Ivano, 10000),$$
$$(3244, 503947, Teresa, 500), \ldots\}$$

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

Model Theoretic Approach

- **Queries** are **WFFs** in predicate logic
- Free variables represent selected values in the query
- A **lookup** that maps free variables to values in **a query answer**
- **Query answering** is done by **model checking:** $\mathcal{S} \vDash_l Q$
- **Example:**

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

```
SELECT account_no, customer FROM Account
WHERE branch = 9205 AND balance > 3,000
```

$$Q(x,y): \exists z,t\ (Account(z,x,y,t) \wedge z = 9205 \wedge t > 3000)$$

Model Theoretic Approach

- **Boolean queries** are sentences (formulas without free variables)
- A Boolean query returns true if structure $\mathcal{S}$ satisfies the query sentence

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

- **Example:**

$$Q: \exists x, y, z, t\ (Account(z, x, y, t) \wedge t > 10000 \wedge z = 9205)$$

Model Theoretic Approach

- Integrity Constraints are represented as sentences
- **Example:** branch_no is a **primary key**

Branch

branch_no	address
9205	782 Dovetail Estates
3244	746 Andell Road
2501	3605 Oakridge Lane

$$\forall x_1 \forall y_1 \forall x_2 \forall y_2 \; ((Branch(x_1, y_1) \wedge Branch(x_2, y_2) \wedge x_1 = x_2) \\ \rightarrow y_1 = y_2)$$

Model Theoretic Approach

- **Example:** branch in Account is a **foreign key** and refers to branch_no in Branch

$$\forall x, y, z, t \left(Account(x, y, z, t) \rightarrow (\exists u \; Branch(x, u)) \right)$$

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

Branch

branch_no	address
9205	782 Dovetail Estates
3244	746 Andell Road
2501	3605 Oakridge Lane

Proof Theoretic Approach

- A database instance is represented as **a theory T with axioms:**
  - **Equality:** reflexivity, symmetry, transitivity, and substitution (the 4th axiom)
  - **Assertion:** for each tuple in each relation
  - **Completion axioms:** for **closed-world assumption**
  - **Axioms for unique-name assumption**

- Query answering is done by **proving logical consequences of T**: For Boolean query Q, **prove $T \vdash Q$**

Proof Theoretic Approach

- For every tuple, T includes an atomic sentence called **an assertion**

- **Example:**

$$Account(9205, 890516, Ivano, 10000)$$
$$Account(3244, 503947, Teresa, 500)$$
$$\ldots$$

are assertion axioms in T

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

Proof Theoretic Approach

- Databases make the **closed-world assumption (CWA)**
- **CWA:** Any tuple that is not in a database is false in the model of the database
- **Example:** a completion axiom for CWA

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

$$\forall x, y, z, t\ (Account(x, y, z, t) \rightarrow ((x = 9205 \land y = 890516 \land z = Ivano \land t = 10000) \lor$$

$$(x = 3244 \land y = 503947 \land z = Teresa \land t = 500) \lor$$

$$\ldots$$

$$(x = 3244 \land y = 789815 \land z = Zoya \land t = 7000)))$$

Proof Theoretic Approach

- Databases also make the **unique-name assumption (UNA)**
- Constants in assertions are NOT mapped to the same entities in the universe of the database structure

Account

branch	account_no	customer	balance
9205	890516	Ivano	10,000
3244	503947	Teresa	500
9205	888013	Prince	5,000
9205	863279	Ali	250
3244	789815	Zoya	7,000

- **Example:** $Teresa$ and Ali must refer to different individuals
  They must be mapped to different entities in U_S

$$Teresa \neq Ali \wedge Teresa \neq Ivano \wedge Ivano \neq 500 \wedge \cdots$$

Program Verification

- **Software verification:** Checking if a software system does what it is supposed to do
- **Software testing** for software verification
  - Effective in finding certain errors and faults (bugs) in software systems
  - It does not guarantee a software system is error-free

Why Should We Formally Verify Code?

- **Automatic verification** through logical reasoning
- **Documentation**
- **Decreasing time-to-market:** testing and debugging cost and time
- **Refactoring and reuse:** A verified software with a clear formal specification is easier to refactor and reuse
- **Certification audits:** Safety-critical software systems cannot rely on manual testing and debugging

Formal Program Verification

- **Software development** starts with informal **program requirements $\boldsymbol{R}$**
- A framework for formal program verification
  - Convert $\boldsymbol{R}$ to sentences $\boldsymbol{\phi_R}$ **in a formal language**
  - Write **a program $\boldsymbol{P}$** to a realize the requirements
  - Prove the program P satisfies ϕ_R

A Simple Language

- **Integer expressions**
- **Example:**
  - 5
  - x
  - $x - (x * (y - (5 + z)))$
- **Boolean expressions**
- **Example:**
  - $false, true$
  - $x < 3$
  - $!(x + 1 == 3)$
  - $(x < 3 || x > 10) \& (z < y)$

A Simple Language

- **Assignment commands and control structures**
- **Example:**

$x := (z + 1) * y$
If $(x == y)$ {
    $z := y$
} else {
    $x := 2 * y$
}

while $(x == y)$ {
    $y := z * 2$
    $x := 2 + y$
    …
}

- **Program:** A sequence of assignments and control structures

Hoare Logic: Syntax

- **A sentence is a Hoare triple $(\phi)P(\psi)$**
  - ϕ and ψ are sentences in some formal language, e.g., First-Order Predicate Logic
  - **P is a program in the simple language**
- **ϕ and ψ are pre-conditions and post-conditions**

- ϕ and ψ only bound variables that do not appear in P (variables in P appear as free variables in ϕ and ψ)

Hoare Logic: Syntax

- **Example:** A program P that calculates a number whose square is less than x:
  $$(x > 0)\ P\ (y \times y < x)$$
- **Example:**
  $$(\top)\ y := x + 1\ (y = x + 1)$$
- **Example:**
  $$(x \geq 0)\ P'\ (y = x!)$$

P

```
y := 0;
while (y × y < x) {
        y := y + 1;
}
y := y − 1;
```

P'

```
z := x;
y := 1;
while(z != 0) {
        y := y × z;
        z := z − 1;
}
```

Hoare Logic: Semantics

- **State:** A variable look up that assigns a real number to each variable in the program and the free variables in ϕ and ψ
- A state l **satisfies** ϕ if $\mathbb{R} \vDash_l \boldsymbol{\phi}$
- $\boldsymbol{(\phi)P(\psi)}$ **is valid,** denoted by $\vDash (\boldsymbol{\phi})\boldsymbol{P}(\boldsymbol{\psi})$, if for all states that satisfy ϕ, the state resulting from P's execution satisfies ψ
- Software validity reduces to checking if a Hoar sentence is valid

Hoare Logic: Semantics

- If P always terminates, we say the triple is **totally correct**, denoted by $\vDash_{tot} (\boldsymbol{\phi})\boldsymbol{P}(\boldsymbol{\psi})$

- If P does not always terminate, $(\phi)P(\psi)$ is **partially correct**, denoted by $\vDash_{par} (\boldsymbol{\phi})\boldsymbol{P}(\boldsymbol{\psi})$

Hoare Logic: Semantics

- **Example:**

$$(x = x_0 \wedge x \geq 0) Fac (y = x_0!)$$
$$(x \geq 0) Fac (y = x!)$$

- **Example:**

$$(x = 3) Sum (z = 6)$$
$$(x = 8) Sum (z = 36)$$
$$(x = x_0 \wedge x \geq 0) Sum (z = x_0 \times (x_0 + 1)/2)$$

Fac

```
y := 1;
while(x != 0) {
        y := y × x;
        x := x − 1;
}
```

Sum

```
z := 0;
while(x > 0) {
        z := z + x;
        x := x − 1;
}
```

Hoare Logic: Proof System

- **Proof of correctness:** $\vdash (\phi)P(\psi)$ $(\vdash_{tot}$ **or** $\vdash_{par})$

- **Two proof rules (among other rules):**

$$\frac{(\phi)C_1(\eta) \quad (\eta)C_2(\psi)}{(\phi)C_1;C_2(\psi)} \text{ Composition}$$

$$\frac{(\phi \wedge B)C_1(\psi) \;\; (\phi \wedge \neg B)C_2(\psi)}{(\phi)\ \text{If}(B)\{C_1\}\ \text{else}\ \{C_2\}(\psi)} \text{ If-statement}$$

Hoare Logic: Proof System

- **Example:**

$$\frac{(\phi)C_1(\eta) \quad (\eta)C_2(\psi)}{(\phi)C_1;C_2(\psi)}\ \text{Composition}$$

$$\frac{\begin{array}{c}(x=0)\ x := x+1;\ (x=1) \\ (x=1)\ x := x*2;\ (x=2)\end{array}}{(x=0)\ x := x+1;\ x := x*2;\ (x=2)}\ \text{Composition}$$

Hoare Logic: Proof System

- **Example:**

$$\frac{(\phi \wedge B)C_1(\psi) \quad (\phi \wedge \neg B)C_2(\psi)}{(\phi)\ \text{If}(B)\{C_1\}\ \text{else}\ \{C_2\}(\psi)}\ \text{If-statement}$$

$$\frac{\begin{array}{c}(x > 0)\ y \coloneqq x;\ (y = |x|) \\ (x \le 0)\ y \coloneqq -x;\ (y = |x|)\end{array}}{(\top)\ \text{if}\ (x > 0)\ \{\ y \coloneqq x;\}\ \text{else}\ \{y \coloneqq -x;\}\ (y = |x|)}\ \text{If-statement}$$

1878

Western

UNIVERSITY · CANADA