Deductive Databases (DDBs)

Dr. Ilias S. Kotsireas, Professor

CP-465: Database II

Wilfrid Laurier University

Note: slides have been modified by Dr. Orestes Appel to fit the structure of COMP-4522 at MRU.

Deductive Databases (DDBs)

- Introduction
- Prolog/Datalog Notation
- Rule Interpretation
- Inference Mechanisms

Introduction

1. DDBs is an area at the intersection of databases, AI, logic
2. a DDB is a database system including capabilities to define **deductive rules**, via which we can deduce or infer additional information from the facts stored in a database
3. theoretical foundation for DDBs is mathematical logic
4. other types of systems (e.g. **expert database systems** or **knowledge-based systems**) also incorporate reasoning and inference capabilities
5. common characteristic: use of AI techniques (production systems)
6. difference: DDBs makes use of data in secondary storage, expert database systems assume that the data needed resides in main memory

Basic Notions of DDBs

In a DDB we specify rules through a **declarative language** (what, not how, e.g. SQL)

An **inference engine** or **deduction mechanism** in the DDB can deduce new facts from the database by interpreting these rules

model used for DDBs is closely related to:

1. Relational Data Model (domain relational calculus)
2. logic programming, **Prolog** language

a variation of Prolog, **Datalog** is used to define rules declaratively.

the existing set of relations is treated as a set of literals in the language

Datalog syntax is similar to Prolog syntax

Datalog semantics[a] ≠ Prolog semantics

---

[a]how programs are executed

a DDB uses two main types of specifications: **facts** and **rules**

1. **Facts** are specified in a manner similar to relations, but not necessarily including attribute names.

   In a DDB the meaning of an attribute is determined by its **position** in a tuple

2. **Rules** are similar to relational views.

   They specify virtual relations that are not actually stored but rather can be deduced from the facts by applying inference mechanisms based on the rule specifications

   Main difference between rules & views is that rules can involve recursion

The evaluation of Prolog programs is based on **backward chaining**, which involves a top-down evaluation of goals

The evaluation of Datalog programs is roughly based on bottom-up evaluation

In Prolog the order of specification of facts and rules and the order of literals are significant, for the evaluation

in Datalog an effort has been made to circumvent these problems

Prolog/Datalog Notation

Principle: provide predicates with unique names

Predicate:
- implicit meaning (via a suggestive name)
- fixed number of **arguments**

If the arguments are all constant values, then the predicate states that a certain fact is true.

If the arguments are variables, then the predicate is considered as a query or as a part of a rule or constraint

`Prolog Conventions adopted here`:

1. **constant values** in a predicate are either numeric or character strings and they are represented by identifiers starting with a `lowercase` letter only
2. **variable names** start with an `uppercase` letter only

Example based on the `company` database

3 predicate names: `supervise`, `superior`, `subordinate`

a) the `supervise` predicate:

defined via a set of facts, each with two arguments: (supervisor name, supervisee name)

`supervise` expresses the idea of direct supervision

facts correspond to actual data stored in the database

facts correspond to tuples in a relation with two attributes and schema:

SUPERVISE(SUPERVISOR, SUPERVISEE)

note the absence of attribute names in facts

attributes are represented by **position**. 1st argument is the supervisor, 2nd argument is the supervisee. `supervise(X,Y)` states the fact that X supervises Y.

b) the `superior`, `subordinate` predicates:

defined via a set of rules (`superior` allows us to express the idea of non-direct supervision)

MAIN IDEA OF DDBs

specify rules and a framework to infer (deduce) new information based on these rules

(a) **Facts**

```
SUPERVISE(franklin, john).
SUPERVISE(franklin, ramesh).
SUPERVISE(franklin, joyce).
SUPERVISE(jennifer, alicia).
SUPERVISE(jennifer, ahmad).
SUPERVISE(james, franklin).
SUPERVISE(james, jennifer).
. . .
```

Rules

```
SUPERIOR(X, Y) :– SUPERVISE(X, Y).
SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).
SUBORDINATE(X, Y) :– SUPERIOR(Y, X).
```

Queries

```
SUPERIOR(james, Y)?
SUPERIOR(james, joyce)?
```

(b)

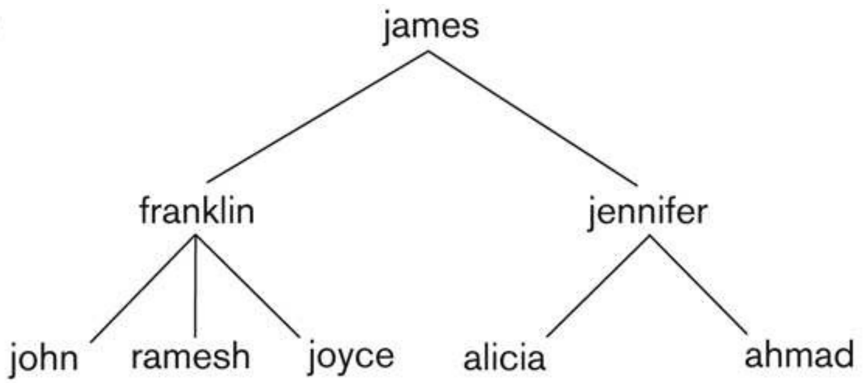

Figure 24.11
(a) Prolog notation. (b) The supervisory tree.

Rule Syntax

$$\boxed{\text{head :- body}}$$

the symbol :- is interpreted as IFF (if and only if)

the head is also called LHS or conclusion. it is (usually) made up of a single predicate

the body is also called RHS or premise(s). its made up of one or more predicates separated by commas (implicitly connected by AND)

predicates with constant arguments are called **ground predicates** or **instantiated predicates**

the arguments of a predicate in a rule are (typically) variables

Query Syntax

A query typically involves a predicate symbol with some variable arguments.

The answer to a query is to deduce all different combinations of constant values, that (when assigned to the variables) make the predicate true.

Datalog Notation

- A Datalog program is built from combining **atomic formulas**.
- Atomic formulas are **literals** of the form $p(a_1, \ldots, a_n)$ where p is the predicate name and n is the # of its arguments.
- n is called the **arity** or the **degree** of p
- The arguments can be constant values (lowercase) or variable names (uppercase).
- **built-in predicates**: (syntax: `less(X,3)`, `X<3`)

  d binary comparison $<$, ≤, $>$, ≥ over ordered domains

  d binary comparison =, / = over ordered/unordered domains

- Attention to infinite ranges: `greater(X,3)`
- A **literal** is an atomic formula as defined earlier (positive literal) or an atomic formula precede by **not** (negative literal)
- Only formulas specified in a restricted **clausal form** called **Horn clauses**, can be used in Datalog.

Clausal Form, Horn clauses

A formula can contain predicates called **atoms** & quantifiers (universal ∀, existential ∃)

In **clausal form**, a formula must be transformed into another formula with the following characteristics:

- **all variables in the formula are universally quantified.** (since there is no need to include ∀ explicitly, it is removed and all variables are implicitly quantified by ∀)
- the formula is made up of a number of clauses, **each clause is a disjunction of literals** (the literals are connected by OR)
- **a formula is a conjunction of clauses** (the clauses are connected by AND)

It can be shown that any formula can be converted into clausal form.

In Datalog, rules are expressed as a restricted form of clauses, called **Horn clauses**: they can contain at most one positive literal:

d $NOT(P_1)\ OR\ NOT(P_2)\ OR\ \cdots\ OR\ NOT(P_N)\ OR\ Q$

d $NOT(P_1)\ OR\ NOT(P_2)\ OR\ \cdots\ OR\ NOT(P_N)$

(1)

$NOT(P_1)\ OR\ NOT(P_2)\ OR\ \cdots\ OR\ NOT(P_N)\ OR\ Q$

is equivalent to $P_1\ AND\ P_2\ AND\ \cdots\ AND\ P_N \Rightarrow Q$

which is written in Datalog as the rule: $Q :- P_1, P_2, \ldots, P_n$

this rule says that if the predicates P_i are all true, for a particular binding of their variables, then Q is also true and can be deduced

(2)

$NOT(P_1)\ OR\ NOT(P_2)\ OR\ \cdots\ OR\ NOT(P_N)$

is equivalent to $P_1\ AND\ P_2\ AND\ \cdots\ AND\ P_N \Rightarrow$

which is written in Datalog as the rule: $P_1, P_2, \ldots, P_n$

this rule can be considered as an integrity constraint

Rule Semantics

A rule means that if a binding of constant values to the variables in the body (RHS) of a rule makes all its predicates TRUE, then the **same** binding makes the head (LHS) TRUE

A rule provides a way to generate new facts, instantiations of the head of the rule. These new facts are based on already existing facts, corresponding to bindings of predicates in the body of the rule

Note that listing multiple comma-separated predicates in the body or a rule, implicitly means that they are connected by **AND**

Example the `superior` predicate of the `company` database (direct/indirect supervision)

```
superior(X,Y):-supervise(X,Y)

superior(X,Y):-supervise(X,Z),superior(Z,Y)
```

NOTE 1: a rule may be defined recursively

NOTE 2: when two (or more) rules have the same (LHS) head predicate, the head predicate is true if the 1st body is TRUE **OR** if the 2nd body is TRUE

Rule Interpretation

2 main ways of interpreting the theoretical meaning of the rules:

1. proof-theoretic interpretation
2. model-theoretic interpretation

In practice, in specific systems, the **inference mechanism** defines the exact interpretation and provides a computational interpretation of the meaning of the rules.

Proof-theoretic interpretation

Facts and rules are true statements or **axioms**. **Ground axioms** contain no variables. Facts are ground axioms that are taken to be true. Rules are called **deductive axioms** because they can be used to deduce new facts.

The deductive axioms are used to `derive new facts from existing facts`

Provides a procedural approach to compute the answer to a Datalog query.

Figure 24.12
Proving a new fact.

1. SUPERIOR(*X*, *Y*) :– SUPERVISE(*X*, *Y*).	(rule 1)
2. SUPERIOR(*X*, *Y*) :– SUPERVISE(*X*, *Z*), SUPERIOR(*Z*, *Y*).	(rule 2)
3. SUPERVISE(jennifer, ahmad).	(ground axiom, given)
4. SUPERVISE(james, jennifer).	(ground axiom, given)
5. SUPERIOR(jennifer, ahmad).	(apply rule 1 on 3)
6. SUPERIOR(james, ahmad).	(apply rule 2 on 4 and 5)

Model-theoretic interpretation

Specify a (usually) finite domain of constant values.

Assign to a predicate every possible combination of values as arguments

Determine if the predicate is true or false

In practice, we specify the combinations of argument values that make the predicate true and state that **all other** combinations make the predicate false. When this is done for all predicates, this is called an **interpretation** of this particular set of predicates.

`an interpretation for the 2 predicates` **supervise, superior**

An interpretation assigns a truth value (T, F) to every possible combination of argument values (taken from a finite domain) for a set of predicates.

Figure 24.13
An interpretation that is a minimal model.

Rules
SUPERIOR(*X*, *Y*) :– SUPERVISE(*X*, *Y*).
SUPERIOR(*X*, *Y*) :– SUPERVISE(*X*, *Z*), SUPERIOR(*Z*, *Y*).

Interpretation

Known Facts:
SUPERVISE(franklin, john) is **true**.
SUPERVISE(franklin, ramesh) is **true**.
SUPERVISE(franklin, joyce) is **true**.
SUPERVISE(jennifer, alicia) is **true**.
SUPERVISE(jennifer, ahmad) is **true**.
SUPERVISE(james, franklin) is **true**.
SUPERVISE(james, jennifer) is **true**.
SUPERVISE(*X*, *Y*) is **false** for all other possible (*X*, *Y*) combinations

Derived Facts:
SUPERIOR(franklin, john) is **true**.
SUPERIOR(franklin, ramesh) is **true**.
SUPERIOR(franklin, joyce) is **true**.
SUPERIOR(jennifer, alicia) is **true**.
SUPERIOR(jennifer, ahmad) is **true**.
SUPERIOR(james, franklin) is **true**.
SUPERIOR(james, jennifer) is **true**.
SUPERIOR(james, john) is **true**.
SUPERIOR(james, ramesh) is **true**.
SUPERIOR(james, joyce) is **true**.
SUPERIOR(james, alicia) is **true**.
SUPERIOR(james, ahmad) is **true**.
SUPERIOR(*X*, *Y*) is **false** for all other possible (*X*, *Y*) combinations

An interpretation is called a **model** for a specific set of rules, when these rules are **always true** in this interpretation. (e.g. for any values assigned in the variables in the rules, the head is true, when the body is true, rules are not violated)

`this interpretation is a model for` **superior**

Q when is a rule violated?

A particular bindings of the arguments of the predicates in the rule body make it true, and simultaneously make the rule head predicate false.

$\boxed{\text{Example}}$ I denotes a certain interpretation

`supervise(a,b)` is TRUE in I

`superior(b,c)` is TRUE in I

`superior(a,c)` is FALSE in I

then, I can’t be a model for the (recursive) rule

```
superior(X,Y) :- supervise(X,Z),superior(Z,Y)
```

In the model-theoretic approach, the meaning of the rules is established by providing a model for these rules.

A model is called a **minimal model** for a set of rules, if we cannot change any fact from T to F and still have a model for these rules.

add `superior(james,bob)` to the true predicates, we still have a model
this is a non-minimal model, because changing `superior(james,bob)` to false, still provides us with a model

Comparison of proof-theoretic & model-theoretic interpretations

For rules with simple structure (i.e. no negation involved) the minimal model of the model-theoretic interpretation is the same as the facts generated by the proof-theoretic interpretation

The use of negations destroys the uniqueness property of minimal models and therefore the correspondence between the two types of interpretations

Datalog Programs

There are two main methods to define the truth values of predicates in Datalog programs:

- Fact-defined predicates (or relations)
  correspond to relations in RDBMSs
  defined by listing all the combinations of values that make the predicate true
  `EMPLOYEE, MALE, FEMALE, DEPARTMENT, SUPERVISE, PROJECT, WORKS_ON`
- Rule-defined predicates (or views)
  defined by LHS of one or more Datalog rules

A program or a rule is called **safe** if it generates a finite set of facts.

Theoretically, determining whether a set of rules is safe is an undecidable problem.

```
EMPIOYEE(john).
EMPLOYEE(franklin).
EMPLOYEE(alicia).
EMPLOYEE(jennifer).
EMPLOYEE(ramesh).
EMPLOYEE(joyce).
EMPLOYEE(ahmad).
EMPLOYEE(james).

SALARY(john, 30000).
SALARY(franklin, 40000).
SALARY(alicia, 25000).
SALARY(jennifer, 43000).
SALARY(ramesh, 38000).
SALARY(joyce, 25000).
SALARY(ahmad, 25000).
SALARY(james, 55000).

DEPARTMENT(john, research).
DEPARTMENT(franklin, research).
DEPARTMENT(alicia, administration).
DEPARTMENT(jennifer, administration).
DEPARTMENT(ramesh, research).
DEPARTMENT(joyce, research).
DEPARTMENT(ahmad, administration).
DEPARTMENT(james, headquarters).

SUPERVISE(franklln, john).
SUPERVISE(franklln, ramesh)
SUPERVISE(frankin , joyce).
SUPERVISE(jennifer, alicia).
SUPERVISE(jennifer, ahmad).
SUPERVISE(james, franklin).
SUPERVISE(james, jennifer).

MALE(john).
MALE(franklin).
MALE(ramesh).
MALE(ahmad).
MALE(james).

FEMALE(alicia).
FEMALE(jennifer).
FEMALE(joyce).

PROJECT(productx).
PROJECT(producty).
PROJECT(productz).
PROJECT(computerization).
PROJECT(reorganization).
PROJECT(newbenefits).

WORKS_ON(john, productx, 32).
WORKS_ON(john, producty, 8).
WORKS_ON(ramesh, productz, 40).
WORKS_ON(joyce, productx, 20).
WORKS_ON(joyce, producty, 20).
WORKS_ON(franklin, producty, 10).
WORKS_ON(franklin, productz, 10).
WORKS_ON(franklin, computerization, 10).
WORKS_ON(franklin, reorganization, 10).
WORKS_ON(alicia, newbenefits, 30).
WORKS_ON(alicia, computerization, 10).
WORKS_ON(ahmad, computerization, 35).
WORKS_ON(ahmad, newbenefits, 5).
WORKS_ON(jennifer, newbenefits, 20).
WORKS_ON(jennifer, reorganization, 15).
WORKS_ON(james, reorganization, 10).
```

Figure 24.14
Fact predicates for part of the database from Figure 5.6.

```
SUPERIOR(X, Y) :– SUPERVISE(X, Y).
SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).

SUBORDINATE(X, Y) :– SUPERIOR(Y, X).

SUPERVISOR(X) :– EMPLOYEE(X), SUPERVISE(X, Y).
```

Figure 24.15
Rule-defined predicates.

bottom-up inference mechanisms, forward chaining

the inference engine starts with the facts and applies the rules to generate new facts

the generated facts are checked against the query predicate goal for a match

the term `forward chaining` indicates that the inference moves forward from the facts toward the goal.

$\boxed{\text{Example: query}}$ `superior(james,Y)?`

does any of the existing facts match the query? NO

apply the first (non-recursive) rule to existing facts to generate new facts ⇒ facts for the `superior` predicate are generated

each generated fact is tested for a match against the query. the first match is `superior(james, franklin)` ⇒ Y=franklin. the next match is `superior(james, jennifer)` ⇒ Y=jennifer.

apply the second (recursive) rule to existing (initial & generated) facts

matches each `supervise` fact with each `superior` fact, to satisfy both the RHS predicates `supervise(X,Z)` & `superior(Z,Y)` ⇒ additional answers: Y=john, Y=ramesh, Y=joyce, Y=alicia, Y=ahmad

top-down inference mechanisms, backward chaining

the inference engine starts with the query predicate goal and attempts to find matches to the variables that lead to valid facts in the database

the term `backward chaining` indicates that the inference moves backward from the goal to determine facts that would satisfy the goal

facts are NOT explicitly generated (as in forward chaining)

$\boxed{\text{Example: query}}$ `superior(james,Y)?`

search for any facts of the `superior` predicate whose first argument matched james. no such facts found.

locate the first rule whose head has the same predicate name as the query ⇒
`superior(X,Y):-supervise(X,Y)`

bind the variable X to james, leading to the rule
`superior(james,Y):-supervise(james,Y)`

search (in the order of listing in the program) for facts that match `supervise(james,Y)`
⇒ Y=franklin, Y=jennifer

locate the next rule whose head has the same predicate name as the query ⇒

```
superior(X,Y):-supervise(X,Z),superior(Z,Y)
```

bind the variable X to james, leading to the rule

```
superior(james,Y):-supervise(james,Z),superior(Z,Y)
```

search for facts that match both subgoals (depth-first) `supervises(james,franklin)`

bind Z to franklin