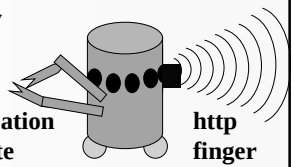
Outline

- Logistics (Project) & Review
- First Order Predicate Calculus
- Relational Algebra
- Datalog
- Information Integration Softbots
- Query Containment
- Rewriting Queries w/ Views

[Etzioni AI Mag93]

Softbot = Software Ro**bot**

- Sensors
- Effectors
- Planning-Based Control
  - **High-Level Goals…**
  - **Increased Autonomy**



cgi invocation
db update

http
finger

The Tuple Extraction Problem

- WWW Sources Formatted for **People**
- **Softbot** wants relational information

<The Rock, 7:20>
<Vertigo, 9:30>
<Star Trek, 7:30>

[Kushmerick 97]

HTML Source

<HTML><TITLE>Showtimes</TITLE>
<BODY><B>Now Showing:</B> <P>
<B>The Rock</B> <I>7:20</I>Great!
<B>Vertigo</B> <I>9:30</I>Classic
<B>Star Trek</B> <I>7:30</I>Beam me up
<HR> Bookmark me now!
<B>Thanks!</B></BODY></HTML>

Note the Movie Names….

<HTML><TITLE>Showtimes</TITLE>
<BODY><B>Now Showing:</B> <P>
<B>The Rock</B> <I>7:20</I>Great!
<B>Vertigo</B> <I>9:30</I>Classic
<B>Star Trek</B> <I>7:30</I>Beam me up
<HR> Bookmark me now!
<B>Thanks!</B></BODY></HTML>

Surrounded by **<B>** and **</B>**

<HTML><TITLE>Showtimes</TITLE>
<BODY><B>Now Showing:</B> <P>
<B>The Rock</B> <I>7:20</I>Great!
<B>Vertigo</B> <I>9:30</I>Classic
<B>Star Trek</B> <I>7:30</I>Beam me up
<HR> Bookmark me now!
<B>Thanks!</B></BODY></HTML>

Similarly, Showtimes by **<I>, </I>**

```
<HTML><TITLE>Showtimes</TITLE>
<BODY><B>Now Showing:</B> <P>
<B>The Rock</B> <I>7:20</I>Great!<BR>
<B>Vertigo</B> <I>9:30</I>Classic<BR>
<B>Star Trek</B> <I>7:30</I>Beam me up<BR>
<HR> Bookmark me now! <BR>
<B>Thanks!</B></BODY></HTML>
```

A Wrapper

```
ExtractMovieTimes
    Tuples := {}
    While P not empty do:
        Skip forward to <B>
        Title := ExtractTextUntilNext( </B> )
        Skip forward to <I>
        Time := ExtractTextUntilNext( </I> )
        Push (Title, Time) onto Tuples
    Return Tuples
```

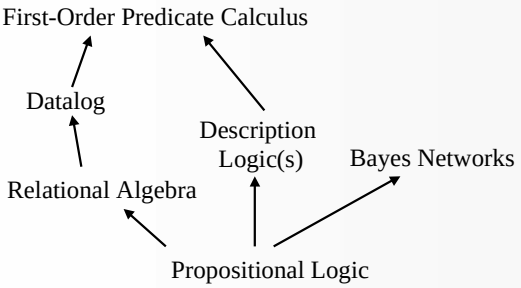
Project

- (5/7) Select Information Sources
  - Movie domain
  - We supply an ontology
  - You provide Datalog source descriptions
- (5/14) Write Wrappers (Class to share)
  - Each one subclasses Java wrapper class
  - Regular expression package
- (6/11) Complete Information Integration Softbot

Course Topics by Week

- Search & Constraint Satisfaction
- Knowledge Representation 1: Propositional Logic
- Autonomous Spacecraft 1: Configuration Mgmt
- Autonomous Spacecraft 2: Reactive Planning
- Information Integration 1: Knowledge Representation
- Information Integration 2: Planning & Execution
- Supervised Learning & Datamining
- Reinforcement Learning
- Bayes Nets: Inference & Learning
- Review & Future Forecast

Knowledge Representation



Reasoning Algorithms

- Tasks
  - Satisfiability
  - Entailment
- Approach
  - Systematic (e.g. DPLL)
  - Stochastic (e.g. GSAT)
- Properties
  - Soundness
  - Completeness
  - Complexity

Summary: Propositional Logic

- **Syntax**
  - Prop variables: P, Q, …
  - Connectives: and, or, not, =>, =
- **Semantics**
  - Truth Tables
- **Inference**
  - Modus Ponens $\dfrac{\neg P \vee Q, \quad P}{Q}$
  - Resolution $\dfrac{\neg P \vee Q, \quad P \vee R}{Q \vee R}$
- **Complexity**:
  - NPC

13

Propositional. Logic *vs* **First Order**

Ontology	Facts: P, Q	Objects (e.g. Dan) Properties (e.g. mother-of) Relations (e.g. female)
Syntax	Atomic sentences Connectives	Variables & quantification Sentences have structure: terms female(mother-of(X)))
Semantics	Truth Tables	Interpretations (Much more complicated)
Inference	NPC, but SAT algos work well	Undecidable, but theorem proving works *sometimes* Look for tractable subsets

14

Definitions

- ***Constants***: a,b, dog33.
  - Name a specific object.
- ***Variables***: X, Y.
  - Refer to an object without naming it.
- ***Functions***: father-of
  - Mapping from objects to objects.
- ***Terms***: father-of(father-of(dog33))
  - Refer to objects
- ***Atomic Sentences***: in(father-of(dog33), food6)
  - Can be true or false
  - Correspond to propositional symbols P, Q

15

More Definitions

- **Logical connectives**: and, or, not, =>
- **Quantifiers**:
  - For all ∀
  - There exists ∃
- **Examples**
  - Dumbo is grey
  - Elephants are grey
  - There is a grey elephant

16

Interaction of quant + connective

E(x) == "x is an elephant"
G(x) == "x has the color grey

- $\forall x \; E(x) \Rightarrow G(x)$
- $\forall x \; E(x) \wedge G(x)$
- $\exists x \; E(x) \Rightarrow G(x)$
- $\exists x \; E(x) \wedge G(x)$

Nested Quantifiers: Order matters!

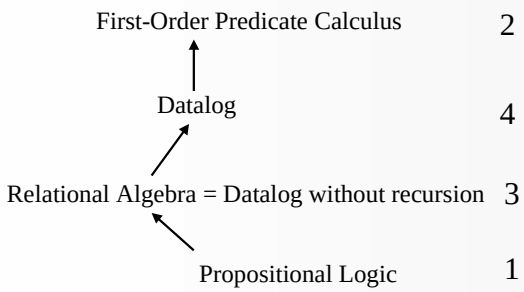
$$\forall x \, \exists y \; P(x,y) \neq \exists y \forall x \, P(x,y)$$

- Examples
  - Every dog has a tail
  - Someone is loved by everyone

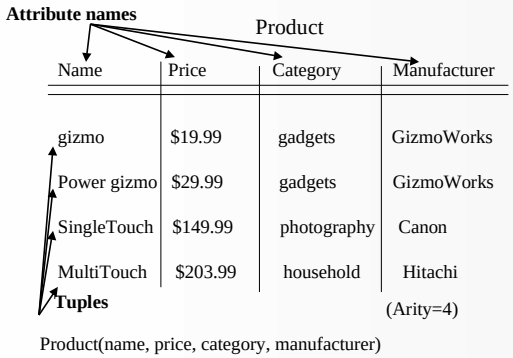
Outline

- Logistics (Project) & Review
- First Order Predicate Calculus
- Relational Algebra
- Datalog
- Information Integration Softbots
- Query Containment
- Rewriting Queries w/ Views

Today's KR Sequence



Terminology



Product(name, price, category, manufacturer)

More Terminology

Every attribute has an atomic type.

Relation Schema: relation name + attribute names + attribute types

Relation instance: a set of tuples. Only one copy of any tuple! (not)

Database Schema: a set of relation schemas.

Database instance: a relation instance for every relation in the schema.

More on Tuples

Formally, a mapping from attribute names to (correctly typed) values:

name → gizmo
price → $19.99
category → gadgets
manufacturer → GizmoWorks

Sometimes we refer to a tuple by itself: (note order of attributes))

(gizmo, $19.99, gadgets, GizmoWorks) or

Product (gizmo, $19.99, gadgets, GizmoWorks).

Integrity Constraints

An important functionality of a DBMS is to enable the specification of integrity constraints and to enforce them.

Knowledge of integrity constraints is also useful for query planning and optimization.

Examples of constraints:

keys, superkeys
foreign keys
domain constraints, tuple constraints.
Functional dependencies, multivalued dependencies.

Keys

A minimal set of attributes that uniquely identify the tuple (I.e., there is no pair of tuples with the same values for the key attributes):

Person: social security number
name
name + address
name + address + age

Movies?

Perfect keys are often hard to find, but organizations usually invent something anyway.
Superkey: a set of attributes that contains a key.
A relation may have multiple keys, but only one primary key

employee number, social-security number

Foreign Key Constraints

Purchase:

buyer	price	product
Joe	$20	gizmo
Jack	$20	E-gizmo

Product:

name	manufacturer	description
gizmo	G-sym	great stuff
E-gizmo	G-sym	even better

An attribute of a relation R is must refer to a key of a relation S.

Functional Dependencies

Definition:

If two tuples agree on the attributes

$A_1, A_2, \ldots A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots B_m$

Formally: $A_1, A_2, \ldots A_n \longrightarrow B_1, B_2, \ldots B_m$

Key of a relation: all the attributes are either on the left or right.

Relational Algebra

- Operators: tuple sets as input, new set as output
- Basic Binary Set Operators
  - Result is table (set) with same attributes
    - Sets must be compatible!
      - R1(A1,A2,A3) ∩ R2(B1,B2,B3) ∴ Domain(Ai) = Domain(Bi)
  - Union
    - All tuples in either R1 or in R2
  - Intersection
    - All tuples in both R1 and R2
  - Difference
    - All tuples in R1 but not in R2
  - ~~Complement~~ - *what's the universe?*
- Selection, Projection, Cartesian Product, Join

Selection

- Grab a subset of the tuples in a relation that satisfy a given condition
  - Use and, or, not, >, <… to build condition
- Unary operation… returns set with same attributes, but 'selects' rows

Selection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

Select DepartmentID = 2

SSN	Name	DepartmentID	Salary
888888888	Alice	2	45,000

Projection

- Unary operation, selects columns
- Returned schema is different,
  - so returned tuples are not subset of original set
  - Contrast with selection
- Eliminates duplicate tuples

Example: Projection Onto SSN, Name

Employee

SSN	Name	DepartmentID	Salary
			30,000
			32,000
			45,000

Cartesian Product

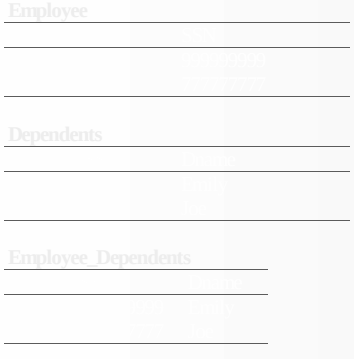
- Binary Operation
- Result is set of tuples combining all elements of R1 with all elements of R2, for R1 × R2
- Schema is union of Schema(R1) & Schema(R2)
- Notice we could do selection on result to get meaningful info!

Cartesian Product Example

Join

- Most often used…
- Combines 2 relations, selecting only related tuples
- Equivalent to a cross product followed by selection
- Resulting schema has all attributes of the two relations, but one copy of join condition attributes

Join Example



Outline

- Logistics (Project) & Review
- First Order Predicate Calculus
- Relational Algebra
- Datalog ⇦
- Information Integration Softbots
- Query Containment
- Rewriting Queries w/ Views

Logic Based Query Languages

- **Datalog:**
  - Subset of First Order Predicate Calculus
    - Function Free
    - Restricted to Horn Clauses
- More Powerful than relational algebra
  - Enables expressing recursive queries
  - More convenient for analysis
- Without recursion (but with negation) it is
  - Equivalent in power to relational algebra

Datalog Concepts

- Atoms
- Datalog rules, datalog programs
- EDB predicates, IDB predicates
- Conjunctive queries
- Recursion
- Built-in predicates
- Negated atoms, stratified programs.
- Semantics: least fixpoint.

Predicates and Atoms

Just like in First-Order Predicate Calculus

- Relations are represented by predicates
- Tuples are represented by atoms.

Purchase(“joe”, “bob”, “Nike Town”, “Nike Air”, 2/2/98)

- arithmetic: *built-in relations*:

X < 100, X+Y+5 > Z/2

- negated atoms:

NOT Product(“Brooklyn Bridge”, $100, “Microsoft”)

Datalog Rules and Queries

A *pure* datalog rule (e.g. first-order horn clause with a positive literal) has the following form:

head :- atom1, atom2, …., atom,…

where all the atoms are non-negated and relational.

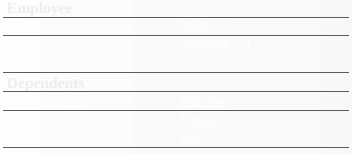
BritishProduct(X) :- Product(X,Y,P) & Company(P, “UK”, SP)

A datalog program is a set of datalog rules.
A program with a single rule is a conjunctive query.

We distinguish EDB predicates and IDB predicates
- EDB’s are stored in the database, appear only in the bodies
- IDB’s are intensionally defined, appear in both bodies and heads.

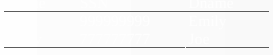
Correspondence: Datalog ~ Relational Algebra

Given: EDBs



ED(Name, SSN, Dname) :-
Employee(Name, SSN) & Dependents(SSN, Dname)

Define: IDB



The Meaning of Datalog Rules

Start with the facts in the EDB and iteratively derive facts for IDBs.

Repeat the following until you cannot derive any new facts:

Consider every assignment from the variables in the body to the constants in the database.

If each of the atoms in the body is made true by the assignment,

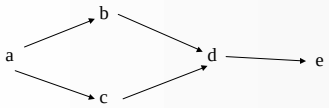
then

add the tuple for the head into the relation of the head.

Transitive Closure

Suppose we are representing a graph by a relation *Edge(X,Y):*

Edge(a,b), Edge (a,c), Edge(b,d), Edge(c,d), Edge(d,e)



I want to express the query:

Find all nodes reachable from a.

Recursion in Datalog

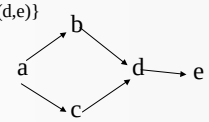
Path(X, Y) :- Edge(X, Y)
Path(X, Y) :- Path(X, Z), Path(Z, Y).

Semantics: evaluate the rules until a fixedpoint:

Iteration #0: Edge: {(a,b), (a,c), (b,d), (c,d), (d,e)}
Path: {}

Iteration #1: Path: {(a,b), (a,c), (b,d), (c,d), (d,e)}

Iteration #2: Path gets the new tuples:
(a,d), (b,e), (c,e)



Iteration #3: Path gets the new tuple:
(a,e)

Iteration #4: Nothing changes -> We stop.

Note: number of iterations depends on the data. Cannot be anticipated by only looking at the query!

Built in Predicates

Rules may include atoms with built-in predicates:

ExpensiveProduct(X) :- Product(X,Y,P) & P > $100

But: we need to restrict the use of built-in atoms in rules.

P(X) :- R(X) & X<Y

What does this mean?

Hence, we require that every variable that appears in a built-in atom also appears in a relational atom.

Negated Subgoals

Rules may include negated subgoals, but in restricted forms:

Ok:
P(X,Y) :- Between(X,Y,Z) & NOT Direct(X,Z)

Bad:
Q(X, Y) :- R(X) & NOT S(Y)

Bad but salvagable:
T(X) :- R(X) & NOT S(X,Y)

We'll rewrite as:
S'(X) :- S(X,Y)
T(X) :- R(X) & NOT S'(X)

Stratified Negation is Ok

A predicate P depends on a predicate Q if:
Q appears negated in a rule defining P.

If there is a cycle in the dependency graph, the datalog program is not stratified.

Example:

p(X) :- r(X) & NOT q(X)
q(X) :- r(X) & NOT p(X)

Suppose r has the tuple {1}
What is the fixed point?

Subtleties with Stratified Rules

Example:

p(X) :- r(X)
q(X) :- s(X) & NOT p(X).

Suppose: r = {1}, and s = {1,2}

One solution: p = {1} and q = {2}

Another solution: p={1,2} and q={}.

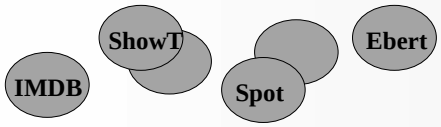
Perfect model semantics: apply the rules stratum after stratum.

Outline

- Logistics (Project) & Review
- First Order Predicate Calculus
- Relational Algebra
- Datalog
- Information Integration Softbots ⇐
- Query Containment
- Rewriting Queries w/ Views

Motivation: Info Integration

- Want agent such that
  - User says ***what*** she wants
  - Softbot determines ***how*** & ***when*** to achieve it
- Example:
  - Show me all reviews of movies starring Marlon Brando that are currently playing in Seattle



Problems

User must know ***which*** sites have relevant info

User must ***go to each one*** in turn

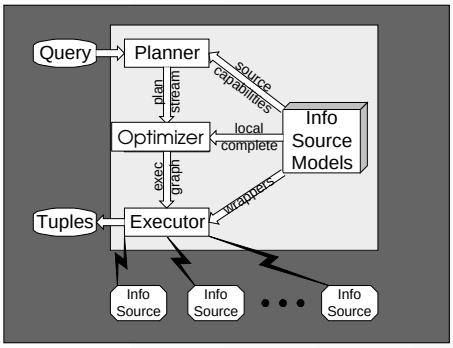
Slow: Sequential access takes time

Confusing: Each site has a different interface

User must ***manually integrate*** information

Before your softbot can solve these problems it must be able to perceive WWW content...

Information Integration



Representation I

- **World Ontology**
  - Defines predicates of relational schemata
  - E.g.,
    - **actor-in (Movie, Part, Name),**
    - **review-of (Movie, Part)**
    - **year-of (Movie, Year)**
    - **shows-in** (**Movie, City, Theatre**)
  - User uses this language to specify queries
  - ***You*** use language to specify content of info sites

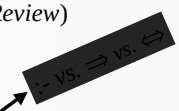
Representation II:

- **Queries**

  Find-all (*M, Review*, brando, seattle)
  Such That **actor-in**(*M, Part*, brando) &
  shows-in(*M*, seattle, *T*) &
  review-of(*M, Review*)

- **Writen in Datalog:**

query(*M, R*, Brando, Seattle) :- **actor-in**(*M, Part*, brando) &
shows-in(*M*, seattle, *T*) &
review-of(*M, R*)



Representation II [Rajaraman95]

- **Information Source Functionality**
  - Info Required? $ Binding Patterns
  - Info Returned?
  - Mapping to World Ontology

  Source may be incomplete: ⇒ (not ⇔)

- **For Example**

IMDBActor($Actor, M) ⇒ actor-in(M, Part, Actor)
Spot($M, Rev, Y) ⇒ review-of(M, Rev) & year-of(M, Y)
Sidewalk($C, M, Th) ⇒ shows-in(M, C, Th)

A Plan to Solve the Query

query(*M, R*, Brando, Seattle) ⇒ **actor-in**(*M, Part*, brando) &
shows-in(*M*, seattle, *T*) &
review-of(*M, R*)

IMDBActor($Actor, M) ⇒ actor-in(M, Part, Actor)
Spot($M, Rev, Y) ⇒ review-of(M, Rev) & year-of(M, Y)
Sidewalk($C, M, Th) ⇒ shows-in(M, C, Th)

plan(M, R, Brando, Seattle) ⇒ IMDBActor(brando, M) &
Sidewalk(seattle, M, Th) &
Spot(M, Rev, Y)

- How verify plan answers query?
- How ***find*** this solution?

Two Questions

- How verify this plan answers query?
  1. Verify information content of plan
     - Same as DB problem of rewriting queries using views
     - Show expansion of plan equivalent to query
     - Technique of query containment
  2. Verifying binding pattern constraints
- How find a valid solution plans?
  - Search...
  - Search-free synthesis of maximal recursive plan

Outline

- Logistics (Project) & Review
- First Order Predicate Calculus
- Relational Algebra
- Datalog
- Information Integration Softbots
- Query Containment ⇦
- Rewriting Queries w/ Views

Query Containment

Let q1, q2 be datalog rules
E.g. q1(X) :- p(X) & r(X)

- Containment
  - $q1 \supseteq q2$ iff $q1(D) \supseteq q2(D)$ for ***every*** database instance, D
- Equivalence
  - $q1 \equiv q2$ iff $q1 \supseteq q2$ and $q2 \supseteq q1$
- Satisfiability
  - q is satisfiable if $\exists$ D such that $q(D) \neq \emptyset$

Motivation

- Removing redundant subgoals
- Detecting independence of queries from update
- Knowledge Base verification
- Semantic caching
- Reusing views (results of previous queries)
  - Internet Information Integration Softbots

Perspective from Logic

- Containment a special form of validity

Given

q1(A, D) :- p(A, B) & r(***C***, D)
q2(A, D) :- p(A, B) & r(***B***, D)

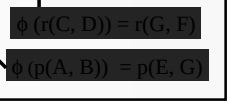
q1 $\supseteq$ q2 is equivalent to saying the next sentence is valid:

$\forall$A, D ($\exists$ B p(A, B) $\wedge$ r(B, D)) =>
($\exists$ B,C p(A, B) $\wedge$ r(C, D))

Containment Mappings

[Chandra & Merlin 77]

- q1 contains q2 iff $\exists$ ϕ: vars(q1) -> vars(q2) s.t.
  - $\forall$ literals L $\in$ body(q1), ϕ(L) $\in$ body(q2)
  - ϕ(head(q1)) = head(q2)
- For example
  - Q1: q(A, D) :- p(A, B) & r(C, D)
  - Q2: q(E, F) :- p(E, G) & r(G, F) & s(E, F)
  - ϕ: A -> E, D -> F, B -> G, C -> G



Computing Containment

- To show q1 contains q2
- Search ...
  - Space of possible containment mappings
  - Incrementally verify: ∀ literals L ∈ body(q1),
    - ∃ literal L' ∈ body(q2) such that ϕ(L)=L'
- NP-complete for pure conjunctive queries
- "Works" for unions of conjunctive queries

Reusing Materialized Views

q (A, E) :- r(A, B) & r(B, C) & s(C, D) & s(D, E)

Suppose ***all we have*** are results of previous queries:

v(F, G) :- r(F, H) & r(H, G) & s(G, I)

u(J, K) :- r(M, J) & s(J, N) & s(N, K)

Can we still answer q?

Yes! q'(X, Y) :- v(X, Z) & u(Z, Y)

Let q" denote expansion of q'

q"(X, Y) :- r(X, H) & r(H, Y) & s(Y, I) &
r(M, Z) & s(Z, N) & s(N, Y)

Equivalence chain: q ≡ q" ≡ q'

I.e. prove q ⊇ q' ⊇ q" ⊇ q

q ⊇ q"

q (A, E) :-
r(A, B) & r(B, C) & s(C, D) & s(D, E)

q"(X, Y) :-
r(X, H) & r(H, Y) & s(Y, I) & r(M, Z) & s(Z, N) & s(I, Y)

ϕ: A -> X
B -> H
C -> Y
D -> I
E -> Y

Back to Information Integration

- How verify this plan answers query?
  1. Verify information content of plan
     - Same as DB problem of rewriting queries using views
     - Show expansion of plan equivalent to query
     - Technique of query containment
  2. Verifying binding pattern constraints
- How find a valid solution plans?
  - Search...
  - Search-free synthesis of maximal recursive plan

A Plan to Solve the Query

query(*M*, *R*, b, s) ⇒ **actor-in**(*M*, *Part*, b) &
shows-in(*M*, s, *T*) &
review-of(*M*, *R*)

plan'(M, R, b, s) ⇒ actor-in(M, P, A) &
review-of(M, R) & year-of(M, Y) &
shows-in(M, C, T)

ϕ: M -> M
Part -> P
b -> A
s -> C
R -> R

How verify this plan answers query?

1. Verify information content of plan ✓
2. Verifying binding pattern constraints

IMDBActor(**$Actor**, M)	⇒	actor-in(M, Part, Actor)
Spot(**$M**, Rev, Y)	⇒	review-of(M, Rev) & year-of(M, Y)
Sidewalk(**$C**, M, Th)	⇒	shows-in(M, C, Th)

plan(M, R, brando, seattle) ⇒ IMDBActor(b, M) &
Sidewalk(s, M, Th) &
Spot(M, R, Y)

Outline

- Logistics (Project) & Review
- First Order Predicate Calculus
- Relational Algebra
- Datalog
- Information Integration Softbots
- Query Containment
- Rewriting Queries w/ Views

Summary

- How Represent Contents of Information Sources?
  - Datalog
- How pose a query?
  - Datalog
- How verify a plan answers query?
  1. Verify information content of plan
     - Check containment of query and plan expansion
  2. Verifying binding pattern constraints
- How find a valid solution plans?
  - Search through the space of...
  - **Search-free synthesis of maximal recursive plan Paper 6.1**