Outline

- Logistics
- Review
- Recursive Plans
  - Datalog programs *vs.* conjunctive queries
  - Recursive Programs
  - Maximality
    - Effect of Functional dependencies
    - Effect of Binding pattern restrictions
- Construction of maximal recursive plans

Logistics

- Site Descriptions due today
  - Effects of omitting (redundant) world relations in site description?
- Wrappers due +1 wk
  - What about sites that return redundant garbage?
  - Should wrappers filter?
- Project writeups due 6/11

Course Topics by Week

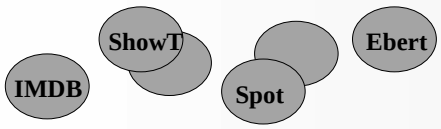
- Search & Constraint Satisfaction
- Knowledge Representation 1: Propositional Logic
- Autonomous Spacecraft 1: Configuration Mgmt
- Autonomous Spacecraft 2: Reactive Planning
- Information Integration 1: Knowledge Representation
- Information Integration 2: Planning & Execution
- Supervised Learning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

Demos

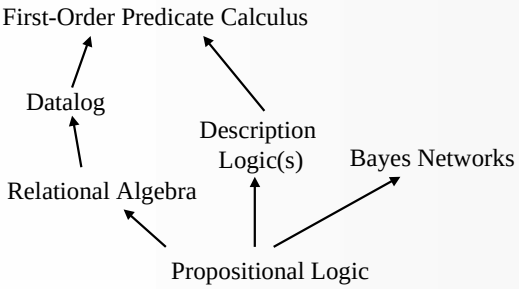
- Parallel Aggregation Engine
  - www.metacrawler.com
  - Fixed set of wrappers
  - One step plans
- Search Broker
  - sb.cs.arizona.edu/sb
  - Uses first word to select source; routes query
  - No aggregation
  - One step plans
- Parallel Shopping Agent
  - jango.excite.com
  - Matches queries to best wrappers from set of 500
  - One step plans

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



Knowledge Representation



Relational Algebra

- Union
- Intersection
- Subtraction
- Selection
- Projection
- Cartesian Product
- Join

Example: Empl JOIN Dependents

Empl

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Dave
777777777	Joe

Result of Join

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Dave
Tony	777777777	Joe

where join condition
is Empl.SSN=Dependents.EmployeeSSN

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

9

Datalog Rules, Programs & 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.

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.

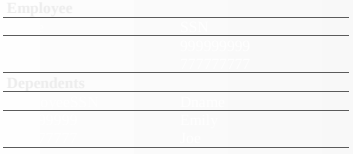
If each of the atoms in the body is made true by the assignment,

then

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

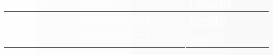
Correspondence: Datalog ~ Relational Algebra

Given: EDBs



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

Define: IDB



IIIIS 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

IIIIS 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*)

IIIIIS Representation III [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)

Unsafe Rules

✗ Sidewalk($C, M, Th, Time) ⇒ shows-in(M, C, Th)

- All variables on left-hand-side ***must*** appear on right-hand-side
  - Otherwise rule is "unsafe"
- Converse not necessary
  - RHS var (e.g. Part) is existentially quantified

✓ IMDBActor($Actor, M) ⇒ actor-in(M, Part, Actor)

Two Questions

- How find a valid solution plans?
  - Search...
  - Search-free synthesis of maximal recursive plan
- How verify a 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
     - $P \equiv Q$ iff $P \subseteq Q$ and $Q \subseteq P$
  2. Verifying binding pattern constraints

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)

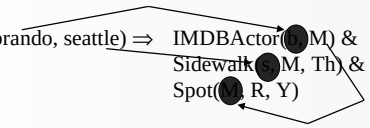
ϕ: M -> M
Part -> P
b -> A
s -> C
R -> R

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
- Review
- Recursive Plans
  - Datalog programs *vs.* conjunctive queries
  - Recursive Programs
  - Maximality
    - Effect of Functional dependencies
    - Effect of Binding pattern restrictions
- Construction of maximal recursive plans

Problem: Want ***All*** the Reviews

plan1(M, R, brando, seattle) ⇒ IMDBActor(b, M) &
Sidewalk(s, M, Th) &
Spot(M, R, Y)

- Many review sites
- None is complete
- Might wish to go to several review sources….

plan2(M, R, brando, seattle) ⇒ IMDBActor(b, M) &
Sidewalk(s, M, Th) &
RevOf(M, R)

RevOf(M, R) => Spot(M, R, Y)
RevOf(M, R) => Ebert(M, R, Y)
RevOf(M, R) => ...

Maximality

- Ideal: Plan ≡ Query
  - I.e. Plan ⊆ Query ∧ Query ⊆ Plan
  - With most real info sources, one can never guarentee completeness
- Soundness: Plan ⊆ Query
- Plan P is Maximal if...

  $\forall P_i$ if $P_i \subseteq$ Query then $P_i \subseteq P$

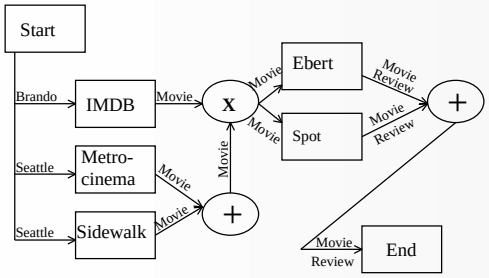
Datalog programs

- New predicates ≈ subroutines

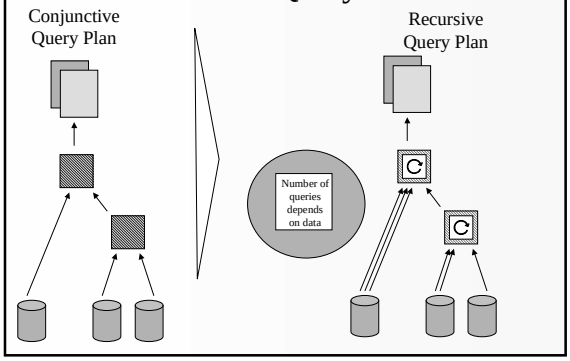
plan(M, R, brando, seattle) ⇒ IMDBActor(brando, M) & ShowsInCity(M, seattle) & RevOf(M, R)

ShowsInCity(M, A, C) => Sidewalk(s, M, Th)
ShowsInCity(M, A, C) => MetroCinema(s, M, Th)
ShowsInCity(M, A, C) => ...
RevOf(M, R) => Spot(M, R, Y)
RevOf(M, R) => Ebert(M, R, Y)
RevOf(M, R) => ...

Datalog Program ≈ Dataflow Diagram



Recursive Query Plans



Solution to Two Open Problems

6.1 [Duschka & Levy]

Two open query planning problems can be solved using recursive query plans:

1. Functional dependencies in domain model

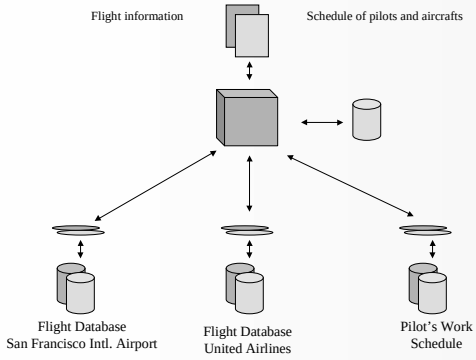
White_pages(Name,Address,Phone)
Phone → Address
Every phone number corresponds to a single address.

2. Limitations on binding patterns of sources

address_server ($Name,Address,Phone)

Name must be provided to query for Address and Phone

Example



World Relations & Source Relations

World Relations:

Flight Information
flight(Airline,Flight_no,From,To)

Schedule of pilots and aircrafts
schedule(Airline,Flight_no,Date,Pilot,Aircraft)

Source Relations:

Flight database San Francisco International Airport
sfo(Airline,Flight_no,To)

Flight database United Airlines
united(Flight_no,From,To)

Pilot's work schedule
ws(Date,From,To,Pilot,Aircraft)

Source Descriptions

Flight database San Francisco International Airport
sfo(Airline,Flight_no,To)
=> flight(Airline,Flight_no,sfo,To)

Flight database United Airlines
united(Flight_no,From,To)
=> flight(ua,Flight_no,From,To)

Pilot's work schedule
ws(Date,From,To,Pilot,Aircraft)
=> flight(Airline,Flight_no,From,To) &
schedule(Airline,Flight_no,Date,Pilot,Aircraft)

Functional Dependencies

flight(Airline,Flight_no,From,To

Airlines assign flight numbers to specific routes:
Airline, Flight_no ⟶ From
Airline, Flight_no ⟶ To

schedule(Airline,Flight_no,Date,Pilot,Aircraft)

Pilots work for only one airline:
Pilot ⟶ Airline

Different airlines don't own same aircraft:
Aircraft ⟶ Airline

Example Query

Query:
Which pilots work for the same airline as Mike?

q(Pilot) <=>
schedule(Airline,Flight_no1,Date1,Pilot,Aircraft1) &
schedule(Airline,Flight_no2,Date2,mike,Aircraft2)

Answers can be retrieved from Pilot's Work Schedule:

ws(Date,From,To,Pilot,Aircraft)
=> flight(Airline,Flight_no,From,To) &
schedule(Airline,Flight_no,Date,Pilot,Aircraft)

More Answers because of FDs

ws(Date,From,To,Pilot,Aircraft)
=> flight(Airline,Flight_no,From,To) & schedule(Airline,Flight_no,Date,Pilot,Aircraft)

Functional dependencies: Aircraft ⟶ Airline
Pilot ⟶ Airline

ws

Date	From	To	Pilot	Aircraft	
08/28	sfo	nrt	mike	#111	**same aircraft, therefore same airline**
08/29	nrt	sfo	ann	#111	**same aircraft, therefore same airline**
09/03	sfo	fra	ann	#222	**same aircraft, therefore same airline**
09/04	fra	sfo	john	#222	
…	…	…	…	…	

Therefore, all these pilots work for the same airline

Data Dependency

Query:
Which pilots work for the same airline as Mike?

Query plan (not maximal):
q(Pilot) <=> ws(Date1,From1,To1,mike,Aircraft1) &
ws(Date2,From2,To2,Pilot2,Aircraft1) &
ws(Date3,From3,To3,Pilot2,Aircraft2) &
ws(Date4,From4,To4,Pilot,Aircraft2)

Depending on the data in the sources, query plans with 1, 2, 4, 6, 8, … *subgoals* are needed to extract all available information

Maximal Query Plan

Recursive query plan (maximal):

```
q(mike)     => ws(Date,From,To,mike,Aircraft)
a(Aircraft) => ws(Date,From,To,mike,Aircraft)
q(Pilot)    => ws(Date,From,To,Pilot,Aircraft) &
               a(Aircraft)
a(Aircraft) => ws(Date,From,To,Pilot,Aircraft) &
               q(Pilot)
```

Is it possible to generate maximal recursive query plans automatically?

Limitations on Binding Patterns

Flight database San Francisco Intl. Airport

sfo ($Airline, Flight_no, To)

Given airline, source returns flight number and destination airports.

Flight database United Airline

united (Flight_no, $From, To)

Given airport of departure, source returns flight numbers and destination airports.

Data Dependency

Query:
Which connections are served by United Airlines?
q(From,To) <=> flight(ua,Flight_no,From,To)

Query plan (not maximal):

```
q(City3,City4) <=>  sfo  ($ua,N1,City1) &
                    united (N2,$City1,City2) &
                    united (N3,$City2,City3) &
                    united (N4,$City3,City4)
```

Depending on the data in the sources, query plans with 1, 2, 3, … subgoals are needed to extract all available information

Maximal Query Plan

Recursive query plan (maximal):

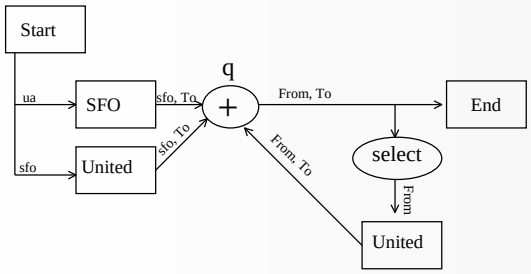
q(sfo,To) => sfo ($ua,N,To)

q(sfo,To) => united (N,$sfo,To)

q(From,To) => q(A,From) & united (N,$From,To)

Is it possible to generate maximal recursive query plans automatically?

Dataflow View

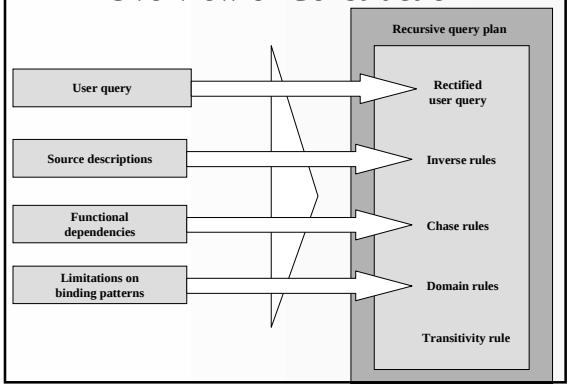


q(sfo,To) <=> sfo ($ua,N,To)
q(sfo,To) <=> united (N,$sfo,To)
q(From,To) <=> q(A,From) & united (N,$From,To)

Outline

- Logistics
- Review
- Recursive Plans
  - Datalog programs *vs.* conjunctive queries
  - Recursive Programs
  - Maximality
    - Effect of Functional dependencies
    - Effect of Binding pattern restrictions
- Construction of maximal recursive plans

Overview of Construction



Skolem Functions

- First-order logic is great because ∃ and ∀ right?
- Can eliminate existenially quantified vars!
  - Replace with a skolem function
- ∃ X president_of(usa, X)
  - president_of(usa, bill_clinton)
  - president_of(usa, f())
- ∀D ∃T dog(D) ⇒ has-tail(D, T)
  - ∀D dog(D) ⇒ has-tail(D, g(D))
  - g is the "tail-choosing" function; it maps dogs to their tails
  - Skolem function takes all (only!) ***preceding*** ∀ vars as args

Inverse Rules

Source description

ws(Date,From,To,Pilot,Aircraft)
=> flight(Airline,Flight_no,From,To) &
schedule(Airline,Flight_no,Date,Pilot,Aircraft)

Inverse rules

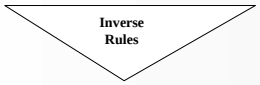
flight(f(D,F,T,P,A),g(D,F,T,P,A),F,T) <= ws(D,F,T,P,A)
schedule(f(D,F,T,P,A),g(D,F,T,P,A),D,P,A) <= ws(D,F,T,P,A)

variable Airline is replaced by a ***function term*** whose arguments are the variables in the source relation

Example

ws

Date	From	To	Pilot	Aircraft
08/28	sfo	nrt	mike	#111
08/29	nrt	sfo	ann	#111
09/03	sfo	fra	ann	#222
09/04	fra	sfo	john	#222



flight

Airline	Flight_no	From	To
?1	?2	sfo	nrt
?3	?4	nrt	sfo
?5	?6	sfo	fra
?7	?8	fra	sfo

schedule

Airline	Flight_no	Date	Pilot	Aircraft
?1	?2	08/28	mike	#111
?3	?4	08/29	ann	#111
?5	?6	09/03	ann	#222
?7	?8	09/04	john	#222

Handling Skolem Functions

- As a first cut can discard all inverse rules with skolem functions
- Only useful if doing functional dependencies

The Chase

Airline	Flight_no	Date	Pilot	Aircraft
?1	?2	08/28	mike	#111
?3	?4	08/29	ann	#111
?5	?6	09/03	ann	#222
?7	?8	09/04	john	#222

Aircraft ⟶ Airline
Therefore: ?3 = ?1

Airline	Flight_no	Date	Pilot	Aircraft
?1	?2	08/28	mike	#111
?1	?4	08/29	ann	#111
?5	?6	09/03	ann	#222
?7	?8	09/04	john	#222

Pilot ⟶ Airline
Therefore: ?5 = ?1

Airline	Flight_no	Date	Pilot	Aircraft
?1	?2	08/28	mike	#111
?1	?4	08/29	ann	#111
?1	?6	09/03	ann	#222
?7	?8	09/04	john	#222

Aircraft ⟶ Airline
Therefore: ?7 = ?1

Chase Rules

Chase rule for functional dependency "Aircraft ⟶ Airline":

e(Airline1,Airline2) <= schedule(Airline1,N1,D1,P1,Aircraft1) &
schedule(Airline2,N2,D2,P2,Aircraft2) &
e(Aircraft1,Aircraft2)

Chase rule for functional dependency "Pilot ⟶ Airline":

e(Airline1,Airline2) <= schedule(Airline1,N1,D1,Pilot1,A1) &
schedule(Airline2,N2,D2,Pilot2,A2) &
e(Pilot1,Pilot2)

Domain Rules

Flight database San Francisco Intl. Airport
sfo ($Airline, Flight_no, To)
Given airline, source returns flight number and destination airports.

Flight database United Airline
United (Flight_no, $From, To)
Given airport of departure, source returns flight numbers and destination airports.

- Can't use United source unless know originating arirport names
- Can use SFO (and United!) sources to "prime" the pump for the United source

Priming the Pump

- Instead of
  - q(From, To) => United(FlightNum, $From, To)
- We'll write
  - q(From, To) => AllPossibleAirports(From) &
    United(FlightNum, $From, To)
  - AllPossibleAirports(Name) => …
- Must generate these domain rules automatically
  - Paper generates one domain predicate
  - You should generate one per type

Generating Domain Rules

Given:

Source1($A, $B, $C, X, Y, Z) => ….

Where

A has is of TypeA, B is of TypeB …

Generate the following rules

TypeX(X) <= TypeA(A) & TypeB(B) & TypeC(C) & Source1(A, B, C, X, Y, Z)

TypeY(Y) <= …

...

Maximality of Constructed Plan

Theorem:

Given a user query, source descriptions, functional dependencies, and limitations,

(i) rectified user query,
(ii) inverse rules,
(iii) chase rules,
(iv) domain rules, and the
(v) transitivity rule

is a maximal query plan.

Arithmetic Predicates <

- Optional
- Must ensure that all variables are bound before < is called
- Must add arithmetic to your engine
  - (along with join…)
- Can ignore < if your engine doesn't support them

Conclusions

- Conjunctive query plans are insufficient to handle functional dependencies and limitations on binding patterns.
- Recursive query plans can produce maximal sets of answers, even in the presence of functional dependencies and limitations on binding patterns.
- Recursive query plans can be easily constructed in polynomial time.