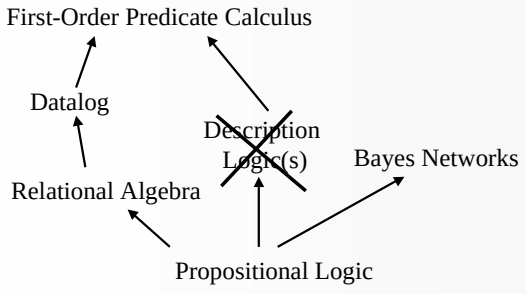
Outline

- Logistics
- Review
- Rule Evaluation
- Robust & Efficient Execution [paper 6.2]
  - Representing Local Completenes
  - Computing informational alternatives
  - Generating contingent plans
- Machine Learning

Logistics

- Wrappers due Monday
- Don't delay work on rest of project!

Knowledge Representation



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

4

Datalog Rules, Programs & Queries

A pure **datalog rule** = first-order horn clause with a positive literal

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

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.

IIIIIS Representation III **[Rajaraman95]**

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

    Source may be incomplete: $\Rightarrow$ (not $\Leftrightarrow$)
- **For Example**

  IMDBActor($Actor, M)$\Rightarrow$ actor-in(M, Part, Actor)

  Spot($M, Rev, Y) $\Rightarrow$ review-of(M, Rev) & year-of(M, Y)

  Sidewalk($C, M, Th) $\Rightarrow$ shows-in(M, C, Th)

Query Containment

Let q1, q2 be datalog rules
E.g.

q1(X) :- p(X) & r(X)

- Containment
  - q1 ⊇ q2 iff q1(D) ⊇ q2(D) for ***every*** database instance, D
- Equivalence
  - q1 ≡ q2 iff q1 ⊇ q2 and q2 ⊇ q1

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 ⊇ q2 is equivalent to saying the next sentence is valid:

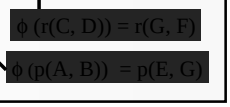
∀A, D (∃ B p(A, B) ∧ r(B, D)) =>
(∃ B,C p(A, B) ∧ r(C, D))

I.e. body(q2) => body(q1)

Containment Mappings

[Chandra & Merlin 77]

- q1 contains q2 iff ∃ ϕ: vars(q1) -> vars(q2) s.t.
  - ∀ literals L ∈ body(q1), ϕ(L) ∈ 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



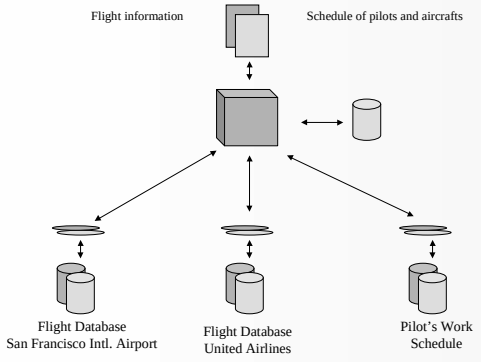
Computing Containment

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

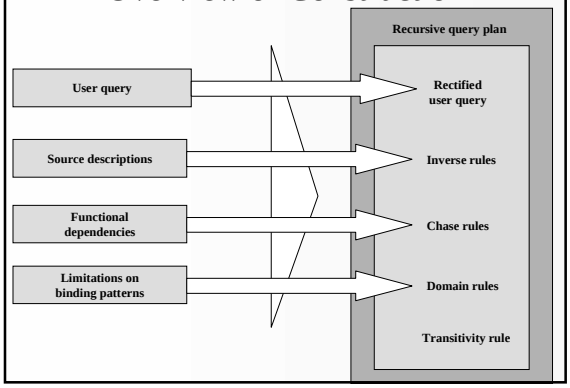
Query Planning

- Given
  - Data source definitions (e.g. in datalog)
  - Query (written in datalog)
- Produce
  - Plan to gather information
    - I.e. either a conjunctive query
      - Equivalent to a join of several information sources
    - Or a recursive datalog program
      - Necessary to account for functional dependencies,
      - Binding pattern restrictions
      - Maximality

Example



Overview of Construction



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

Inverse Rules

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

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 airport names*
- Can use SFO (and United!) sources to "prime" the pump for the United source

Priming the Pump

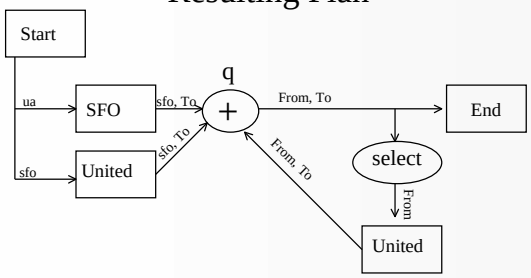
- Instead of
  - q(From, To) => United(FlightNum, $From, To)
- We'll write
  - q(From, To) => **AllPossibleAirports(From)** & United(FlightNum, $From, To)
  - **AllPossibleAirports(Name)** => …
- Must generate these domain rules automatically
  - Paper generates one domain predicate
  - You should generate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

Resulting Plan



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

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.

Outline

- Logistics
- Review
- Rule Evaluation ⇦
- Robust & Efficient Execution [paper 6.2]
  - Representing Local Completenes
  - Computing informational alternatives
  - Generating contingent plans
- Machine Learning

Pre-Process Rules

- Note: 3 kinds of relations in rules:
  - IDBs, EDBs, comparators like = and <
- All rules must be SAFE
- Massage rules to obey 2 further constraints:
  1. The head has no constants.
     - Change p(X, "a") <= expr(X).
     - To p(X,Y) <= expr(X), Y= "a".
     - where Y is a new variable not used elsewhere in the rule.
  2. The head has no repeats.
     - Change p(X,X) <= expr(X).
     - To p(X,Y) <= expr(X), Y=X.
     - where Y is a new variable not used elsewhere in the rule.

Bottom-up Evaluation

- Find all the tuples in all relations simultaneously.
- Suppose we have predicates r1 ... Rn associated with relations R1 ... Rn.
- For each of them we maintain two tables Ti and Ti'.
  - Ti = all the tuples of relation Ri, so far.
  - Ti' contains the tuples for the next
- Iterate until Ti' stops growing

Evaluation

BOTTOM-UP-EVALUATE(r_1...r_n)

```
While some Ri' - Ri is not empty
    Forall i
        Ri = UNION(Ri,Ri')
    Foreach rule
        say the rule is rj(X) <= expr(X,Y).
        Rj' =UNION(Rj', PROJECT(X,
            EVAL(emptyTable, expr(X,Y))))
```

- EVAL takes a table and an expression; returns a table.
- The empty table has no attributes, and **ONE** tuple with zero elements.

Defining EVAL

- EVAL(inTable,X="a")
  - tempTable = table with one attribute X and one tuple <"a">
  - return NJOIN(tempTable,inTable)
- EVAL(inTable,X=Y)
  - return SELECT(X=Y,inTable)
- EVAL(inTable,X<Y)
  - return SELECT(X<Y,inTable)
- EVAL(inTable,X<"5")
  - return SELECT(X<"5",inTable)
- EVAL(inTable,ri(X,Y))
  - If ri is IDB, return NJOIN(inTable,Ti).
  - If ri is EDB, with input attributes (X). Fire the wrapper
  - associated with ri on PROJECT(X,inTable). Return result.
- EVAL(inTable, (ri(X,Y) , expr(Y,Z)))
- EVAL(inTable, (ri(X,Y) ; expr(Y,Z)))
  - return NJOIN(EVAL(inTable,ri),EVAL(inTable,expr(Y,Z)))

Limitations

- Treats unordered join the same as an ordered join
- Evaluates relations that may not contribute to query
- Does redundant work when only a few tuples are desired.
- Hard to multithread.

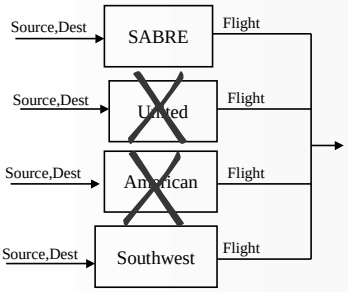
Outline

- Logistics
- Review
- Rule Evaluation
- Robust & Efficient Execution ⇐ [paper 6.2]
  - Representing Local Completenes
  - Computing informational alternatives
  - Generating contingent plans
- Machine Learning

Suppose

- Site descriptions for each airline's flights
  - United($Source, $Dest, FN, Time) => …
  - Alaska($Source, $Dest, FN, Time) => ...
  - Southwest($Source, $Dest, FN, Time) => ...
  - SABRE($Source, $Dest, FN, Time) => ...
  - ...
- Query: find all flights...

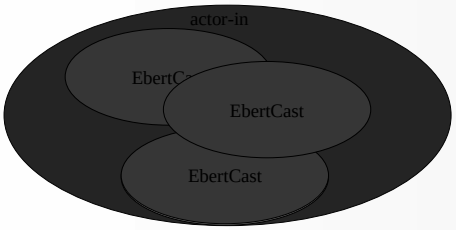
Efficient & Robust Execution



29

Source Descriptions

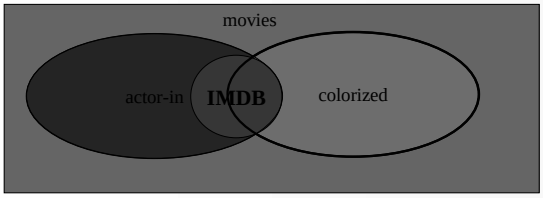
- IMDB($A,M) ⇒ actor-in(A,M).
- EbertCast($A,M) ⇒ actor-in(A,M).



30

Local Completeness 1 [Etzioni, 94 & Levy, '96]

– IMDB(A,M) ⇐ actor-in(A,M) & colorized(M).



31

Levy's Local Completeness

- Defined local completeness.
- Only allow ontology relations on RHS.
  + Problem reduces to computing independence of queries from updates.
  - You can't compare sources.

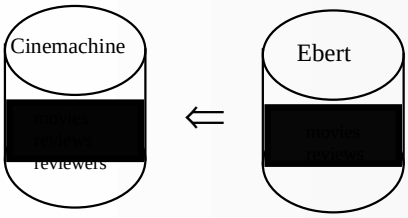
32

We'd Like to Express

- Mirror sites
- Different search forms on the same database
  – IMDB Title
  – IMDB Actor
- Metasources like Cinemachine, SABRE
- "Functional" completeness within a source

33

Local Completeness 2

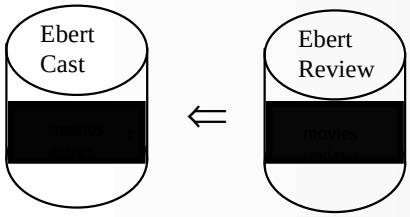


Cinemachine(M,R,Ebert) ⇐ Ebert(M,R).

34

Local Completeness 3

If Ebert review's a movie, it lists the entire cast



EbertCast(M,A) ⇐ Ebert(M,R) & actor-in(A,M).

35

Local Completeness 4

If IMDB lists any movie,
it lists its entire cast.

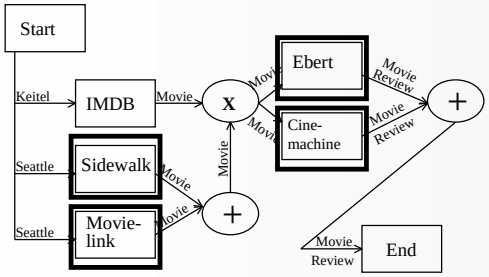
IMDB(movie, actor) ⇐ IMDB(movie, actor2) &
actor-in(actor, movie)

36

Outline

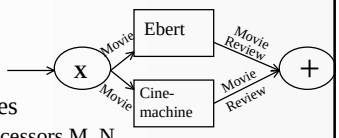
- Logistics
- Review
- Rule Evaluation
- Robust & Efficient Execution [paper 6.2]
  - Representing Local Completenes
  - Computing informational alternatives
  - Generating contingent plans
- Machine Learning

Plans & Alternatives



38

Finding Alternatives



- Consider union nodes
  - with at least 2 predecessors M, N
- Gm & Gn are alternatives when
  - M is sink node of Gm
    - N is sink node of Gn
  - All nodes is Gm are connected to M by a path in Gm
    - Same for Gn
  - Predecessors (Gm) = Predecessors(Gn)

Subsumption of Alternatives

Given local completeness declaration:

– Cinemachine(M,R) ⇐ review-of(M,R).

Inference problem:
Does one alternative subsume another?

40

Subsumption Proof

– Step:

– **Ebert**(M,R)
– ⇒ Ebert(M,R) & movie-review(M,R)

– …

– ⇒ movie-review(M,R)
– ⇒ **Cinemachine**(M,R)

Reason:

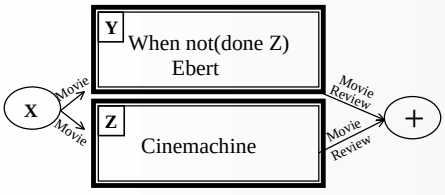
Expand source desc.

Chain of containments

Loc. complete.

41

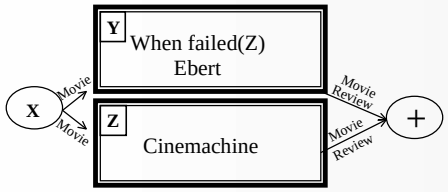
Aggressive Execution



Cinemachine subsumes Ebert

42

Frugal Execution



Cinemachine subsumes Ebert

43

Summary

- Local completeness lets one compare sites.
- **Alternatives** are the decision points in execution.
- We compute **subsumption** over alternatives.
- Subsumption enables efficient and robust execution.

44

Future work: Decision-theoretic execution

Trade off:

-- Number of tuples
-- Expected Time
-- Payments for sources
-- "Netiquette"
-- Possibility for remote computation
-- Bandwidth
-- Expectations on join ordering

See [Etzioni, Karp… FOCS '96]

45

Outline

- Logistics
- Review
- Rule Evaluation
- Robust & Efficient Execution [paper 6.2]
  - Representing Local Completenes
  - Computing informational alternatives
  - Generating contingent plans
- Machine Learning ⇦

Inductive Learning of Rules

Mushroom			***Edible?***
Spores	Spots	Color	
Y	N	Brown	N
Y	Y	Grey	Y
N	Y	Black	Y
N	N	Brown	N
Y	N	White	N
Y	Y	Brown	Y
Y	N	Brown	
N	N	Red	

spots (X) => edible(X)

Don't try this at home...

Types of Learning

- What is learning?
  - Improved performance over time/experience
  - Increased knowledge
- Speedup learning
  - No change to ***set*** of theoretically inferable facts
  - Change to ***speed*** with which agent can infer them
- Inductive learning
  - ***More*** facts can be inferred

Mature Technology

- Many Applications
  - Detect fraudulent credit card transactions
  - Information filtering systems that learn user preferences
  - Autonomous vehicles that drive public highways (ALVINN)
  - Decision trees for diagnosing heart attacks
  - Speech synthesis (correct pronunciation) (NETtalk)
- Datamining: huge datasets, scaling issues

Defining a Learning Problem

A program is said to **learn** from experience ***E*** with respect to task ***T*** and performance measure ***P***, if it's performance at tasks in T, as measured by P, improves with experience E.

- Experience:
- Task:
- Performance Measure:

Example: Checkers

- Task T:
  - ***Playing checkers***
- Performance Measure P:
  - ***Percent of games won against opponents***
- Experience E:
  - ***Playing practice games against itself***

Example: Handwriting Recognition

- Task T:
  - ***Recognizing and classifying handwritten words within images***
- Performance Measure P:
  - ***Percent of words correctly classified***
- Experience E:
  - ***Database of handwritten words with given classifications***

Example: Robot Driving

- Task T:

  Driving on a public four-lane highway using vision sensors
- Performance Measure P:

  Average distance traveled before an error (as judged by a human overseer)
- Experience E:

  A sequence of images and steering commands recorded while observing a human driver

Example: Speech Recognition

- Task T:

  Identification of a word sequence from audio recorded from arbitrary speakers ... noise
- Performance Measure P:

  Percent correct … sample phrase distribution … speaker distribution
- Experience E:

  Corpus of prelabeled signals

Issues

- What feedback (experience) is available?
- What kind of knowledge is being increased?
- How is that knowledge represented?
- What prior information is available?
- What is the right learning algorithm?

Choosing the Training Experience

- Credit assignment problem:
  - **Direct** training examples:
    - E.g. individual checker boards + correct move for each
  - **Indirect** training examples :
    - E.g. complete sequence of moves and final result
- Which examples:
  - Random, teacher chooses, learner chooses

Supervised learning
Reinforcement learning
Unsupervised learning

Choosing the Target Function

- What type of knowledge will be learned?
- How will the knowledge be used by the performance program?
- E.g. checkers program
  - Assume it knows legal moves
  - Needs to choose best move
  - So learn function: F: Boards -> Moves
    - hard to learn
  - Alternative: F: Boards -> R

The Ideal Evaluation Function

- V(b) = **100** if b is a final, won board
- V(b) = **-100** if b is a final, lost board
- V(b) = **0** if b is a final, drawn board
- Otherwise, if b is not final
  V(b) = **V(s)** where s is best, reachable final board

Nonoperational…
Want operational approximation of V: $\hat{V}$

Choosing Repr. of Target Function

- x1 = number of black pieces on the board
- x2 = number of red pieces on the board
- x3 = number of black kings on the board
- x4 = number of red kings on the board
- x5 = number of black pieces threatened by red
- x6 = number of red pieces threatened by black

$$\hat{V}(b) = a + bx1 + cx2 + dx3 + ex4 + fx5 + gx6$$

Now just need to learn 7 numbers!

Example: Checkers

- Task T:
  - ***Playing checkers***
- Performance Measure P:
  - ***Percent of games won against opponents***
- Experience E:
  - ***Playing practice games against itself***
- Target Function
  - ***V: board -> R***
- Target Function representation
  $\hat{V}(b) = a + bx1 + cx2 + dx3 + ex4 + fx5 + gx6$

Representation

- **Decision Trees**
  - Restricted Representation (optimized for learning)
- **Decision Lists**
  - Order of rules matters
- **Datalog Programs**
- **Version Spaces**
  - More general representation (inefficient)
- **Neural Networks**
  - Arbitrary nonlinear numerical functions