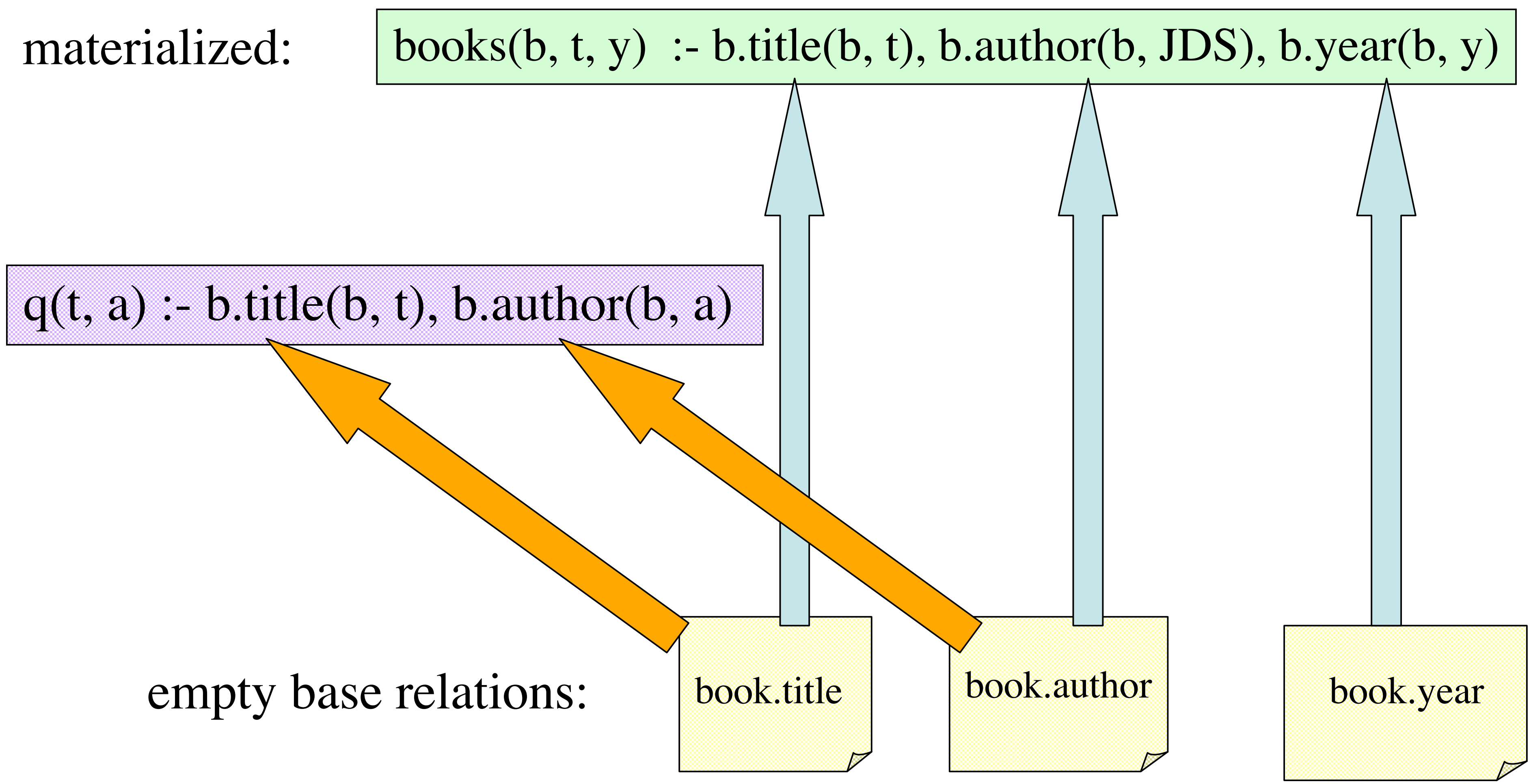
Information Integration Lecture 12

Query Folding

Michael Kassoff

Spring 2003

Answering Queries Using Resources



Answering Queries Using Resources

materialized: books(b, t, y) :- b.title(b, t), b.author(b, JDS), b.year(b, y)

q(t, a) :- b.title(b, t), b.author(b, a)

empty base relations: book.title book.author book.year

Running Example

Pivot Schema

Book.Title
Book.Author
Book.Printing
Printing.Date

Resources

loc(b, t, a) :- b.title(b, t), b.author(b, a)

fan(b, d) :- b.author(b, JDS), b.printing(b, p), p.date(p, d)

Query Plans

Query: q(t, a) :- b.title(b, t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

Resources:

loc(b, t, a) :- b.title(b, t), b.author(b, a)

fan(b, d) :- b.author(b, JDS), b.printing(b, p), p.date(p, d)

Query plan: q(t, a) :- loc(b, t, a), fan(b, 1951)

Query Folding

- Treat intensional predicates as being extensional

loc(b, t, a) :- b.title(b, t), b.author(b, a)

q(t, a) :- b.title(b,t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

fan(b, 1951) :- b.author(b, JDS), b.printing(b, p), p.date(p, 1951)

folded query: q(t, a) :- loc(b, t, a), fan(b, 1951)

Query Containment

- A query Q_1 is *contained* in another query Q_2 if $Q_1(D) \subseteq Q_2(D)$ for all databases D
  - Denoted $Q_1 \subseteq Q_2$
- Two queries are *equivalent* if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$
  - Denoted $Q_1 \equiv Q_2$

Maximal Containment

- Can't always find an equivalent query plan
- We'll settle for a maximally-contained plan
- A query plan Q^* is *maximally-contained* in Q if:
  - $Q^* \subseteq Q$
  - There is no rewriting Q' such that $Q^* \subseteq Q' \subseteq Q$ and Q' is not equivalent to Q^*
- Maximal-containment is relative to the query language allowed (i.e., conjunctive, recursive)

Answering Queries Using Resources

We will look at 2 methods:

Bucket Algorithm

Inverse Rules

The Bucket Algorithm

- High level idea: we need to extract tuples from the resources to plug into the subgoals of our query *Q*
- Create a bucket for each subgoal of *Q*
- Fill the bucket with potential sources of tuples for that subgoal
- Try all combinations of items in the buckets, and choose the maximally-contained combination

In More Detail

- Create a bucket *B* for each query subgoal $S = s(t_1,\ldots,t_n)$
- For each resource *v* that contains a subgoal $R = s(u_1,\ldots,u_n)$, test if it is possible to get compatible tuples from *R*
  - Test "compatiblity" using unification
  - If compatible, let □ = mgu(*S*, *R*)
  - Place head(*v*)□ into *B*

Filling buckets

Query: q(t, a) :- b.title(b, t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

Resources:

loc(b, t, a) :- b.title(b, t), b.author(b, a)

fan(b, d) :- b.author(b, JDS), b.printing(b, p), p.date(p, d)

Buckets:

b.title	**b.author**	**b.printing**	**p.date**
loc(b, t, a)	loc(b, t, a) fan(b, d)	fan(b, d)	fan(b, 1951)

Bucket Algorithm (cont'd)

- Consider all query plans built from resource literals, where one literal is taken from each bucket
- Test for containment of each generated query
  - If not contained, add constraints to make it contained if possible
- Choose the maximally-contained query plan

Example (cont'd)

Query: q(t, a) :- b.title(b, t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

Buckets:

b.title	**b.author**	**b.printing**	**p.date**
loc(b, t, a)	loc(b, t, a) fan(b, d)	fan(b, d)	fan(b, 1951)

Candidate Plans:

q(t, a) :- loc(b, t, a), loc(b, t, a), fan(b, d), fan(b, 1951)
q(t, a) :- loc(b, t, a), fan(b, d), fan(b, d), fan(b, 1951)

Simplified plans:

q(t, a) :- loc(b, t, a), fan(b, 1951)
q(t, a) :- loc(b, t, a), fan(b, 1951)

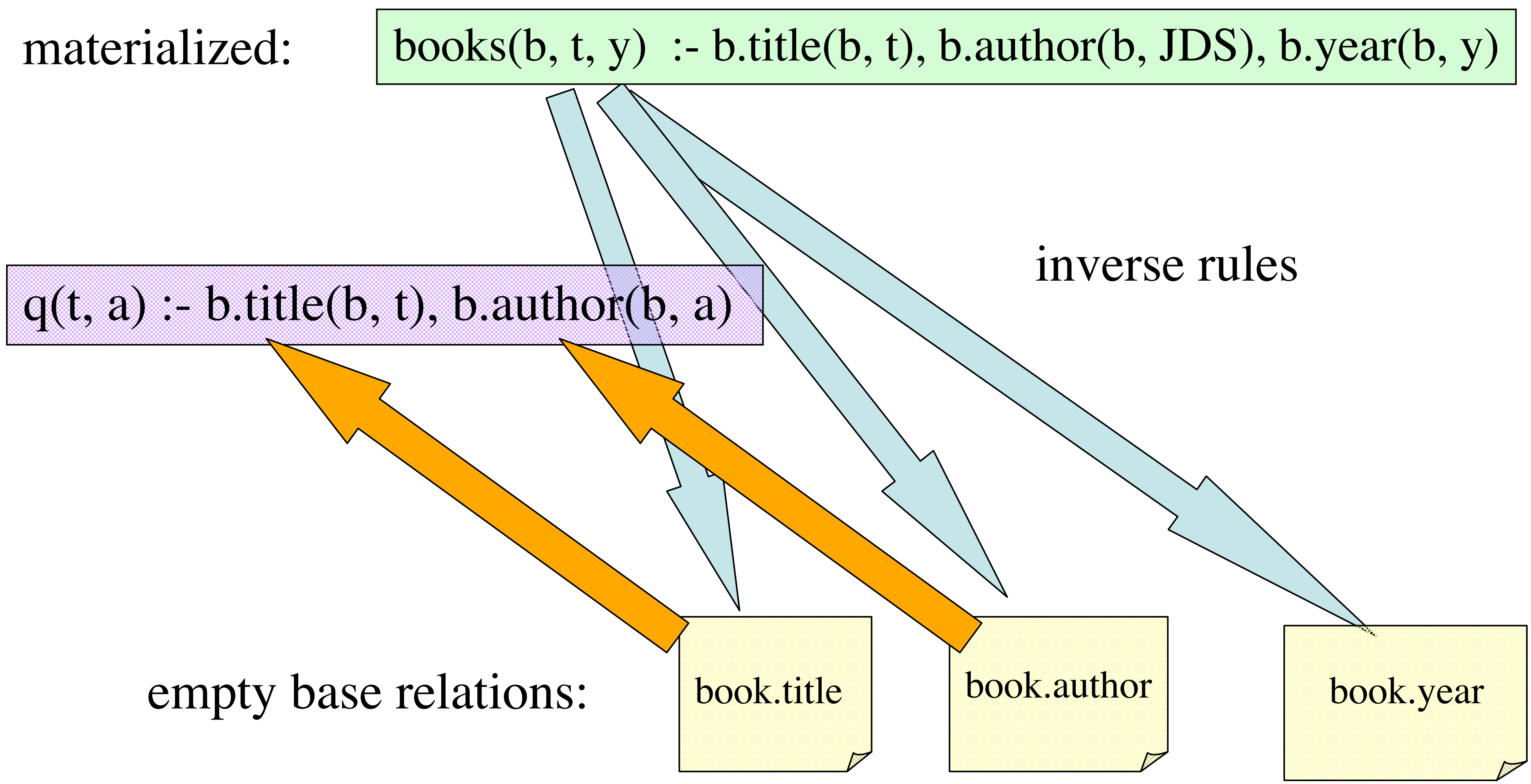
Bottom Line on the Bucket Algorithm

- Simple and intuitive
- Expensive to compute, in large part because containment tests are expensive (NP-complete for CQs, and worse if arithmetic predicates are allowed)
- Must be computed from scratch for each query
- Works only for CQs (with arithmetic predicates)

The Inverse Rules Algorithm

- At a high level:
  - *Invert* the resource definitions, and then use these inverted rules to answer the original query

Inverse Rules



Predicate Completion

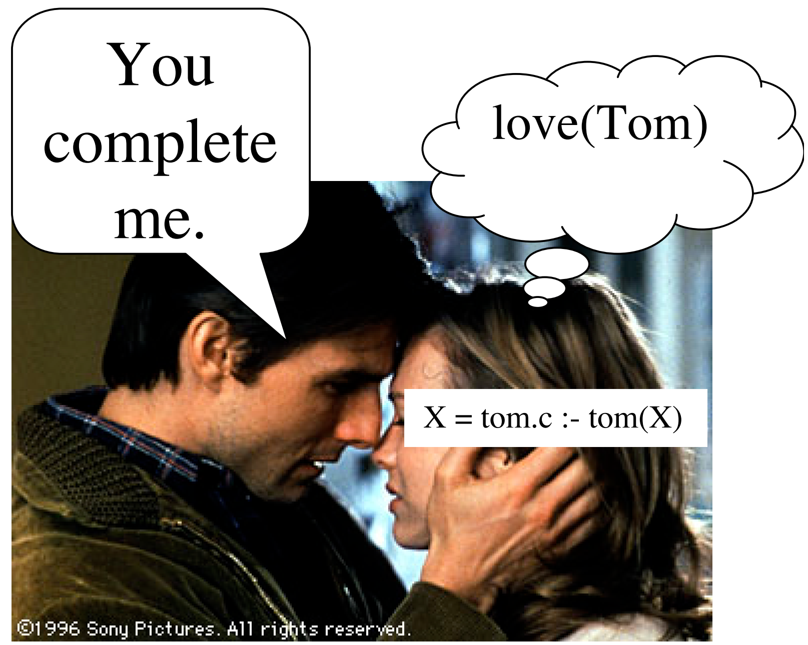
The completion of a predicate says “that’s all there is.”

Say we have a resource flies(X) with the following definition:

flies(X) :- bird(X)
flies(X) :- plane(X)

Then the completion of flies(X) is:

bird(X)   plane(X) :- flies(X)



Inverse Rules

The completion of a resource definition puts the resource predicate on the right and the base predicates on the left!

Definition: amazon(t, a) :- b.title(b, t), b.author(b, a)

Completion: b.title(f(t,a), t), b.author(f(t,a), a) :- amazon(t, a)

Inverse rules:

b.title(f(t,a), t) :- amazon(t, a)
b.author(f(t,a), a) :- amazon(t, a)

Application of Inverse Rules

Inverse rules:

b.title(f(t,a), t) :- amazon(t, a)
b.author(f(t,a), a) :- amazon(t, a)

Resource: {amazon("MD", HM), amazon("CITR", JDS)}

Application:

{b.title(f("MD", HM), "MD"), b.author(f("MD", HM), HM),
b.title(f("CITR", JDS), "CITR"), b.author(f("CITR", JDS), JDS)}

Inverse Rules Algorithm

- If resource definitions are conjunctive, we can simply:

1) In a preprocessing step, compute the inverse rules of our resource definitions
2) Given a query Q on the pivot schema, the query plan is simply Q together with the inverse rules
   - Q can even be a recursive query

Inverse Rules Algorithm (step 1)

Resources:

loc(b, t, a) :- b.title(b, t), b.author(b, a)

fan(b, d) :- b.author(b, JDS), b.printing(b, p), p.date(p, d)

Inverse rules:

b.title(b, t) :- loc(b, t, a)
b.author(b, a) :- loc(b, t, a)
b.author(b, JDS) :- fan(b, d)
b.printing(b, f(b,d)) :- fan(b, d)
p.date(f(b,d), d) :- fan(b, d)

Inverse Rules Algorithm (step 2)

Query: q(t, a) :- b.title(b, t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

Query plan:

q(t, a) :- b.title(b, t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

b.title(b, t) :- loc(b, t, a)
b.author(b, a) :- loc(b, t, a)
b.author(b, JDS) :- fan(b, d)
b.printing(b, f(b,d)) :- fan(b, d)
p.date(f(b,d), d) :- fan(b, d)

Inverse Rules Algorithm (step 3)

Query plan:

q(t, a) :- b.title(b, t), b.author(b, a), b.printing(b, p), p.date(p, 1951)

b.title(b, t) :- loc(b, t, a)
b.author(b, a) :- loc(b, t, a)
b.author(b, JDS) :- fan(b, d)
b.printing(b, f(b,d)) :- fan(b, d)
p.date(f(b,d), d) :- fan(b, d)

Resources:

{loc(523-3, "CITR", JDS), loc(322-8, "MD", HM)}
{fan(523-3, 1951), fan(523-3, 1979)}

Answer:

{q("CITR", JDS)}

Nice properties

- Despite the inclusion of function constants, the application of the inverse rules + query will always terminate. (*Why?*)

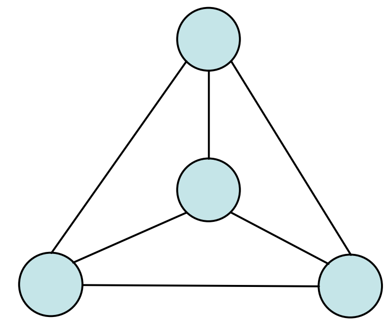
- Inverse rules always produces a maximally-contained rewriting

3-Colorability Example

Resources:

```
rgb(X) :- color(X, red)
rgb(X) :- color(X, green)
rgb(X) :- color(X, blue)

e(X, Y) :- edge(X, Y)
```



Query:

```
q('yes') :- edge(X, Y), color(X, Z), color(Y, Z)
```

"Are there two adjacent nodes with the same color?"

(Returns 'yes' if the graph is *not* 3-colorable)

Plan Using Disjunction

Resources:

```
rgb(X) :- color(X, red)
rgb(X) :- color(X, green)
rgb(X) :- color(X, blue)

e(X, Y) :- edge(X, Y)
```

Query plan:

```
q('yes') :- edge(X, Y), color(X, Z), color(Y, Z)

color(X, red)    color(X, greeen)    color(X, blue) :- rgb(X)
                                       edge(X, Y) :- e(X, Y)
```

Need for Recursive Query Plan

- If our sources are defined using union, sometimes the maximally contained query plan is recursive, *even if the original query wasn't recursive*

- In this case, we need to also include some contrapositives of rules

Recursive Rewritings: Example

Resources:

s1(X,Y) :- virgin(X, Y), major(X), major(Y)

s2(X,Y) :- united(X, Y), major(X), major(Y)

s3(X,Y) :- virgin(X, Y)
s3(X,Y) :- united(X, Y)

Query:

query() :- virgin(X, Y), united(Y, Z)

Example (cont’d)

Query plan:

```
query() :- virgin(X, Y), united(Y, Z)

□virgin(X, Y) :- □query(), united(Y, Z)
□united(Y, Z) :- □query(), virgin(X, Y)

virgin(X, Y) :- s1(X, Y)
virgin(X, Y) :- s3(X, Y), □united(X, Y)

united(X, Y) :- s2(X, Y)
united(X, Y) :- s3(X, Y), □virgin(X, Y)
```

Example (cont'd)

Query plan (simplified):

```
query() :- virgin(X, Y), united(Y, Z)

virgin(X,Y) :- s1(X, Y)
virgin(X,Y) :- virgin(X', X), s3(X, Y)

united(X,Y) :- s2(X, Y)
united(X,Y) :- s3(X, Y), united(Y, Y')
```

The plan is recursive!

+s of Inverse Rules Algorithm

- Demonstrates the power of Logic
  - What could be simpler? Just invert the rules and drop in any query you like
  - Works even for recursive queries and for resources defined using union, which the Bucket Method does not handle
  - In conjunctive case, once the inverse rules are computed, we can use them to make a query plan in constant time!