



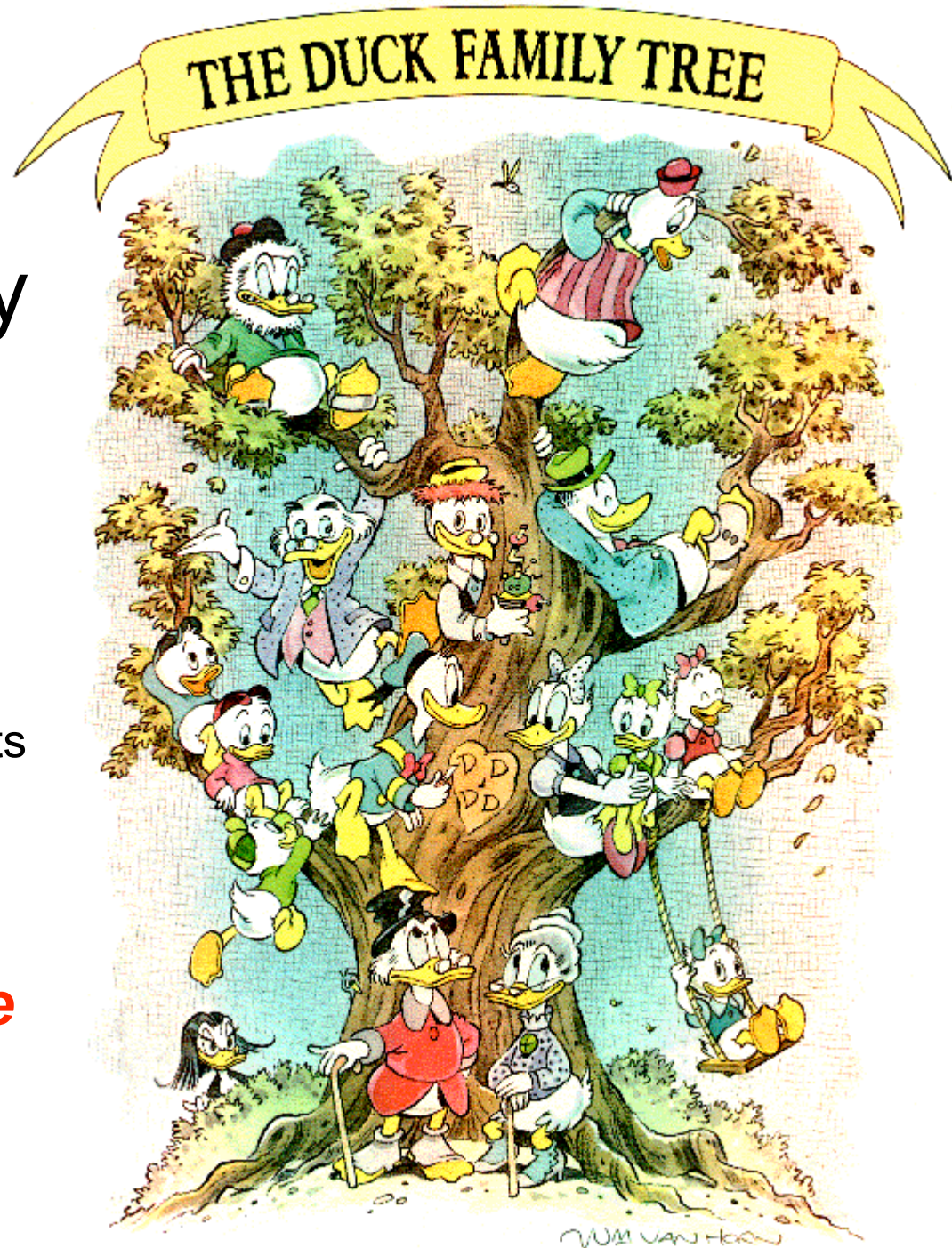
Datalog II

Introduction,
top-down vs. bottom-up evaluation,
magic sets algorithm

Example: The Duck Genealogy

- Is *Scrooge* an ancestor of *Huey*?
- Are *Donald* and *Daisy* related?
- Are *Donald* and *Mickey* related?
- Who are the descendants of *Gladstone*?

→ **Calculus cannot compute transitive closure of graphs!**



- ● ● | Datalog without Negation

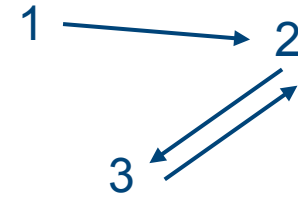
Transitive closure of a graph:

$$T(x, y) :- G(x, y)$$

$$T(x, y) :- G(x, z), T(z, y)$$

● ● ● | Intuition

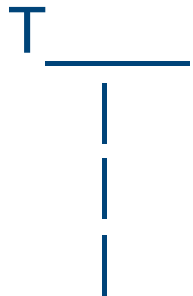
Transitive closure of a graph:



$$T(x, y) :- G(x, y)$$

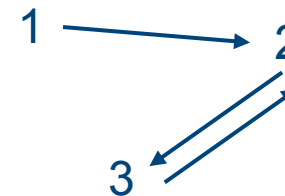
$$T(x, y) :- G(x, z), T(z, y)$$

G	
1	2
2	3
3	2



● ● ● | Intuition

Transitive closure of a graph:



$T(x, y) :- G(1, 2)$

$T(x, y) :- G(x, z), T(z, y)$

G

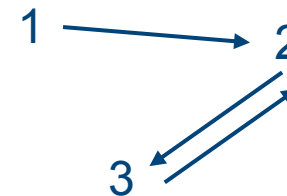
1	2
2	3
3	2

T

(1) Map from instances
over the relations in the rule body

● ● ● | Intuition

Transitive closure of a graph:



$T(1, 2) :- G(1, 2)$

$T(x, y) :- G(x, z), T(z, y)$

G

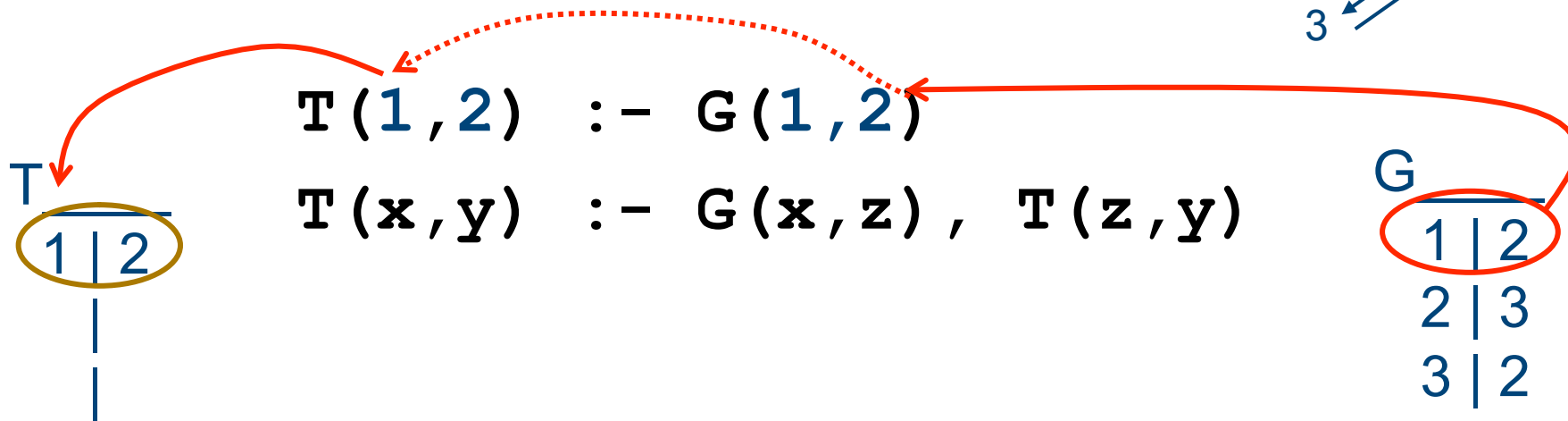
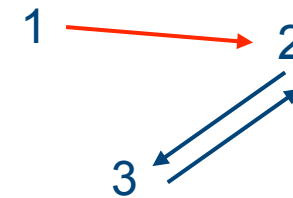
1	2
2	3
3	2

T

(2) ... map to instances
over the relations in the rule head

● ● ● | Intuition

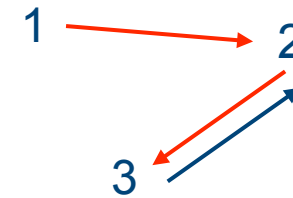
Transitive closure of a graph:



(2) ... map to instances over the relations in the rule head

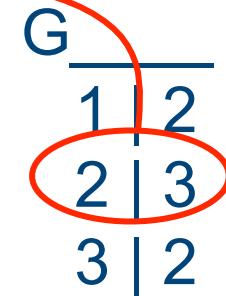
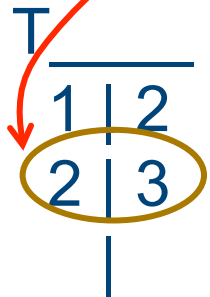
● ● ● | Intuition

Transitive closure of a graph:



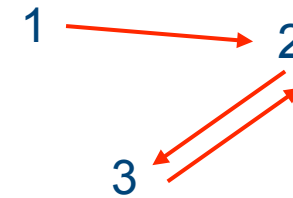
$$T(x, y) :- G(x, y)$$

$$T(x, y) :- G(x, z), T(z, y)$$



● ● ● | Intuition

Transitive closure of a graph:



T

1	2
2	3
3	2

$$T(x, y) :- G(x, y)$$

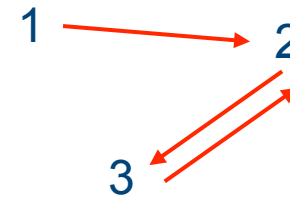
$$T(x, y) :- G(x, z), T(z, y)$$

G

1	2
2	3
3	2

● ● ● | Intuition

Transitive closure of a graph:



$T(x, y) :- G(x, y)$

$T(x, y) :- G(1, 2), T(2, 3)$

T

1	2
2	3
3	2

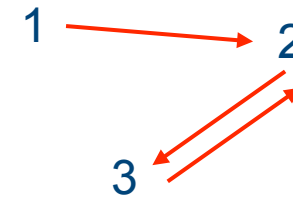
G

1	2
2	3
3	2

(1) Map from instances over the relations in the rule body

● ● ● | Intuition

Transitive closure of a graph:



$$T \frac{}{ \begin{array}{c|c} 1 & 2 \\ \hline 2 & 3 \\ 3 & 2 \end{array} }$$

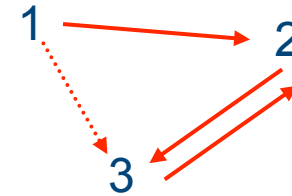
$T(x, y) \text{ :- } G(x, y)$
 $T(1, 3) \text{ :- } G(1, 2), T(2, 3)$

$$G \frac{}{ \begin{array}{c|c} 1 & 2 \\ \hline 2 & 3 \\ 3 & 2 \end{array} }$$

(2) ... map to instances
 over the relations in the rule head

● ● ● | Intuition

Transitive closure of a graph:



$T(x, y) \text{ :- } G(x, y)$

$T(1, 3) \text{ :- } G(1, 2), T(2, 3)$

T

1	2
2	3
3	2
1	3

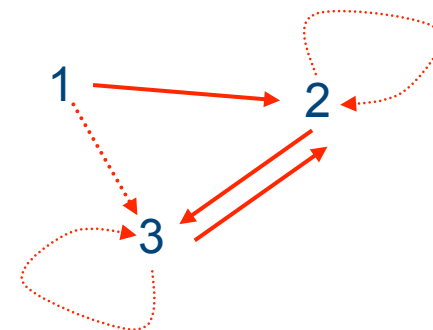
G

1	2
2	3
3	2

(2) ... map to instances over the relations in the rule head

● ● ● | Intuition

Transitive closure of a graph:



$$T \begin{array}{c|c} \hline 1 & 2 \\ 2 & 3 \\ 3 & 2 \\ 1 & 3 \\ 2 & 2 \\ 3 & 3 \\ \hline \end{array}$$

$$T(x, y) :- G(x, y)$$

$$T(x, y) :- G(x, z), T(z, y)$$

$$G \begin{array}{c|c} \hline 1 & 2 \\ 2 & 3 \\ 3 & 2 \\ \hline \end{array}$$

... repeat until fixpoint is reached
(datalog without negation is monotone)



Outline

- Datalog rules
- 3 equivalent semantics
 - Model theoretic
 - Fixpoint
 - Proof theoretic
- Datalog evaluation
 - Naive
 - Semi-naive
 - Magic!



Head :- Body

- Assumption: *Extensional database* (EDB) predicates available
- :- is read *if*
- *Atom* = predicate applied to arguments
- *Head* is an atom
- *Body* is logical AND of zero or more atoms
- Atoms of body are called *subgoals*
- Head predicate is an *intensional database* (IDB) predicate
- Body subgoals may have IDB or EDB predicates
- *Datalog program* = collection of rules.
One IDB predicate is distinguished and represents the result of the program.

```
T(x, y) :- G(x, y)
T(x, y) :- G(x, z), T(z, y)
q(y)      :- T("1", y)
```



Safe Datalog

- o Here: Datalog without negation.
- o Again, notion of *range-restricted* queries.

- o This query is not safe:

`Colored_edges(x, y, col) :- G(x, y)`

→ Every variable in the head of a rule must also appear in the body.



Semantics of Datalog

3 equivalent approaches:

- Model theoretic
- Fixpoint theoretic
- Proof theoretic

● ● ● | Model Theoretic Approach

- View the rules as logical sentences that state a property of the result:

$$\forall x,y(T(x,y) \leftarrow G(x,y))$$

$$\forall x,y,z(T(x,y) \leftarrow (G(x,z) \wedge T(z,y)))$$

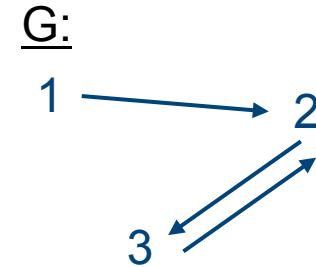
- The result **T** must satisfy these sentences
- But there are many **T**'s that satisfy the sentences...



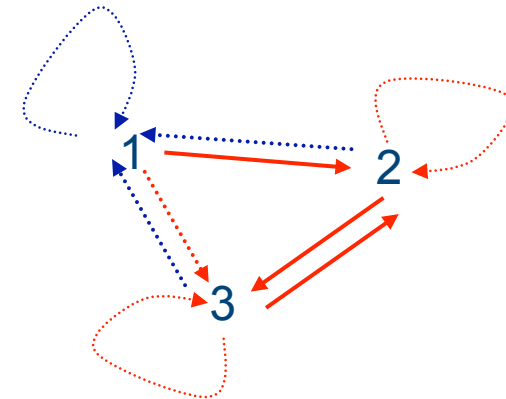
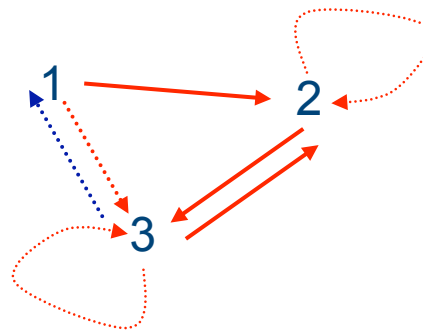
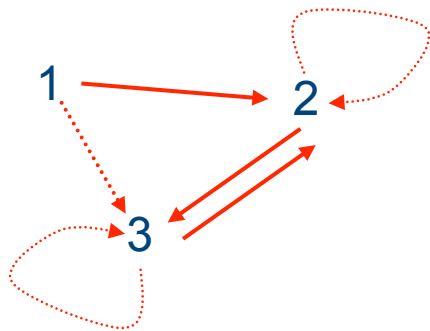
Model Theoretic Approach

$$\forall x,y(T(x,y) \leftarrow G(x,y))$$

$$\forall x,y,z(T(x,y) \leftarrow (G(x,z) \wedge T(z,y)))$$



Possible solutions:

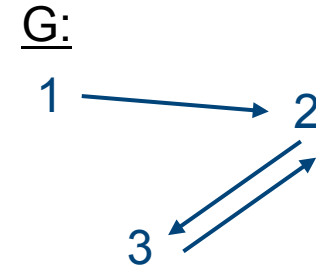




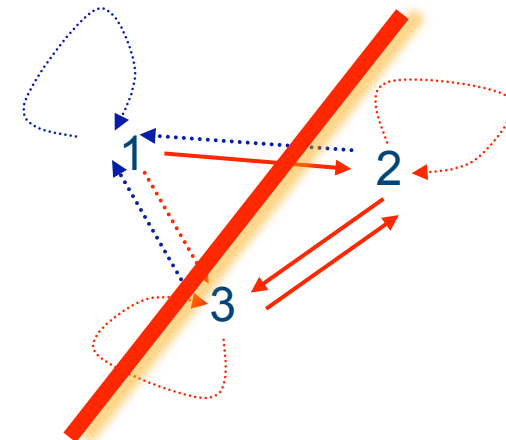
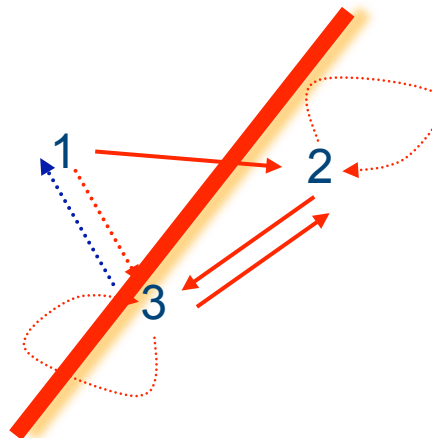
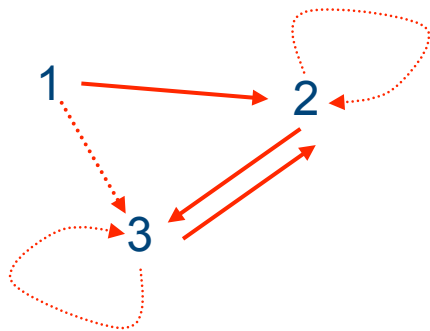
Model Theoretic Approach

$$\forall x,y(T(x,y) \leftarrow G(x,y))$$

$$\forall x,y,z(T(x,y) \leftarrow (G(x,z) \wedge T(z,y)))$$



Possible solutions:



→ Chose the *minimum model* ←
T consists of the smallest set of facts that make the sentences true.

● ● ● | Why the minimum model?

Closed world assumption:

- Databases are incomplete
 - We can't say anything about the data that is not explicitly recorded
- Only those facts are true, that must be true in all worlds modeled by the database.
- Facts that cannot be proven are considered false.

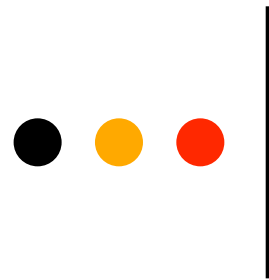
● ● ● | The Fixpoint Approach

- Semantics of program are defined as a particular solution to a fixpoint equation.
 - Query is iterated until a fixpoint is reached.
- Bottom-up evaluation.



Bottom-up Evaluation

- Begin with the facts from database instance
 - Use rules to infer new facts
 - Repeat until no new facts can be inferred
- computes ***all*** facts that can be proven



Bottom-up: Naive Evaluation

Given an EDB:

1. Start with all IDB relations empty
2. Instantiate (with constants) variables of all rules in all possible ways.
If all subgoals become true, then infer that the head is true.
3. Repeat (2) in rounds, as long as new IDB facts can be inferred

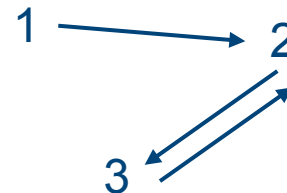


Bottom-up: Naive Evaluation

Given an EDB:

1. Start with all IDB relations empty
2. Instantiate (with constants) variables of all rules in all possible ways.
If all subgoals become true, then infer that the head is true.
3. Repeat (2) in rounds, as long as new IDB facts can be inferred

G:



T:



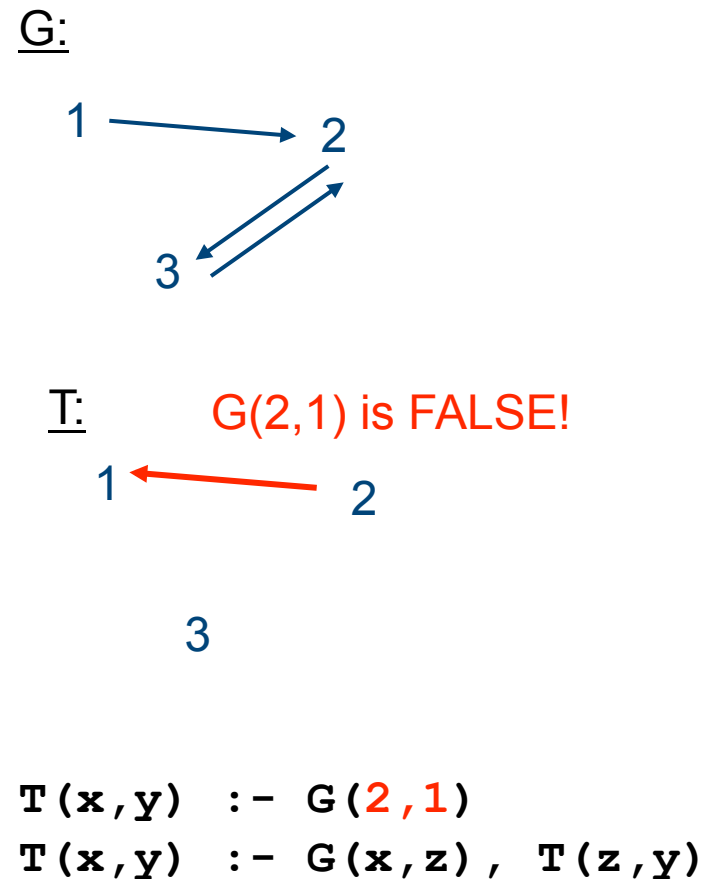
$T(x, y) :- G(x, y)$

$T(x, y) :- G(x, z), T(z, y)$

● ● ● | Bottom-up: Naive Evaluation

Given an EDB:

1. Start with all IDB relations empty
2. Instantiate (with constants) variables of all rules in all possible ways.
If all subgoals become true, then infer that the head is true.
3. Repeat (2) in rounds, as long as new IDB facts can be inferred



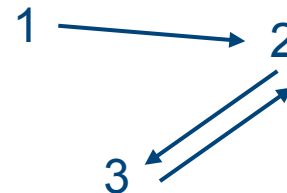


Bottom-up: Naive Evaluation

Given an EDB:

1. Start with all IDB relations empty
2. Instantiate (with constants) variables of all rules in all possible ways.
If all subgoals become true, then infer that the head is true.
3. Repeat (2) in rounds, as long as new IDB facts can be inferred

G:



T:

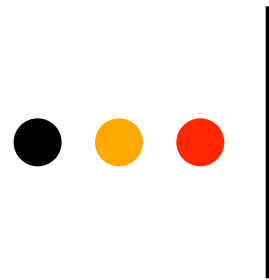
G(1,2) is TRUE!



3

T(1,2) :- G(1,2)

T(x,y) :- G(x,z), T(z,y)



Bottom-up: Naive Evaluation

Given an EDB:

1. Start with all IDB relations empty
2. Instantiate (with constants) variables of all rules in all possible ways.
If all subgoals become true, then infer that the head is true.
3. Repeat (2) in rounds, as long as new IDB facts can be inferred

(2) Makes sense and is finite as long as the rules are safe

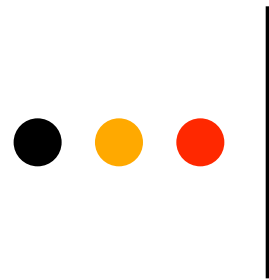


Bottom-up: Naive Evaluation

Given an EDB:

1. Start with all IDB relations empty
2. Instantiate (with constants) variables of all rules in all possible ways.
If all subgoals become true, then infer that the head is true.
3. Repeat (2) in rounds, as long as new IDB facts can be inferred

*Limit of (1) – (3):
Least fixed point
of the rules
and the EDB.*

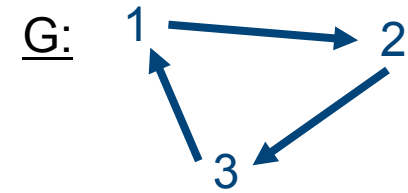


Bottom-up: Seminaive

- More efficient approach to evaluating rules
- Idea: If at round i a fact is inferred, then we must have used a rule in which one or more subgoals were instantiated to facts that were inferred on round $i-1$.
- For each IDB predicate p , keep both the relation P and a relation ΔP ; the latter represents the new facts for p inferred on the most recent round.



r1: $T(x, y) :- G(x, y)$
 r2: $T(x, y) :- G(x, z), T(z, y)$



1. Initialize IDB

T	
1	2
2	3
3	1

2. Initialize Δ IDB

ΔT	
1	2
2	3
3	1

3.a.i $\Delta T(x, y) := G(x, z), \Delta T(z, y)$

ΔT	
1	3
2	1
3	2

3.a.ii $\Delta T := \Delta T \setminus T$

3.a.iii $T := T \cup \Delta T$

T	
1	2
2	3
3	1
1	3
2	1
3	2

3.b.i $\Delta T(x, y) := G(x, z), \Delta T(z, y)$

ΔT	
1	1
2	2
3	3

3.b.ii $\Delta T := \Delta T \setminus T$

3.b.iii $T := T \cup \Delta T$

T	
1	2
2	3
3	1
1	3
2	1
3	2
1	1
2	2
3	3

3.c

...
□



Bottom-up: Seminaive

- Initialize IDB relations by using only those rules without IDB subgoals
- Initialize the Δ -IDB relations to be equal to the corresponding IDB relations
- In one round, for each IDB predicate p :
 - Compute new ΔP by applying each rule for p , but with one subgoal treated as a Δ -IDB relation and the others treated as the correct IDB or EDB relation.
(Do this for all possible choices of the Δ -subgoal).
 - Remove from new ΔP all facts that are already in P .
 - $P := P \cup \Delta P$.
- Repeat (3) until no changes to any IDB relations.

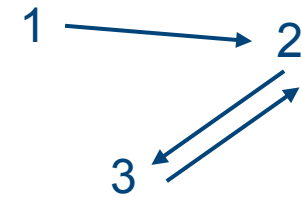
● ● ● | Proof-theoretic Approach

A fact is in the result if there exists a proof for it using the rules and the database facts:

$$(1) \quad T(x, y) \text{ :- } G(x, y)$$

$$(2) \quad T(x, y) \text{ :- } G(x, z), T(z, y)$$

G:



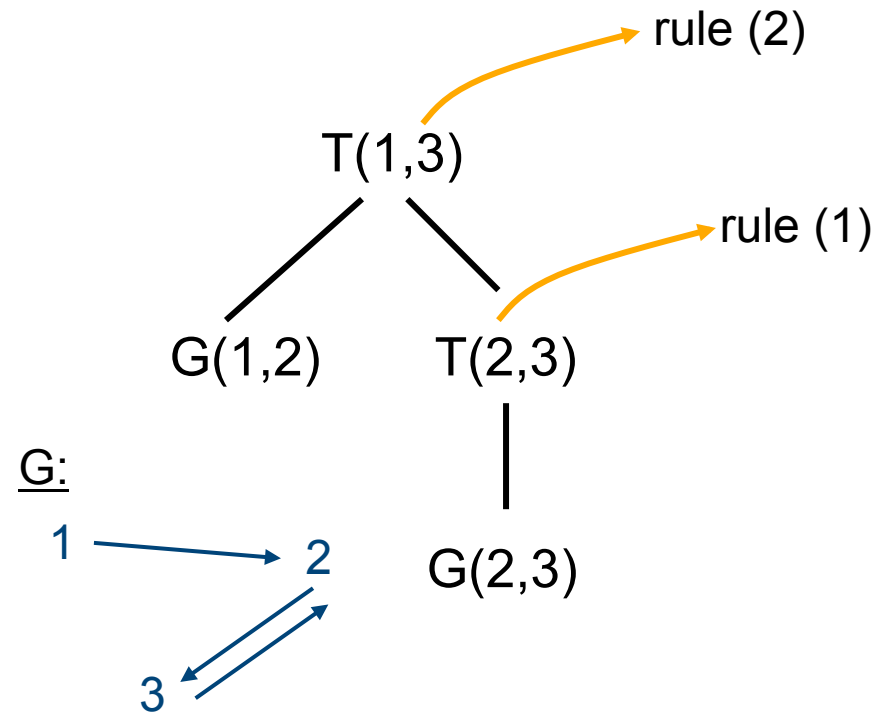
- G(1,2) belongs to the instance
- T(1,2) using (i) and rule (1)
- G(2,3) belongs to the instance
- T(2,3) using (ii) and rule (1)
- T(1,3) using (i), (iv), and rule (2)
-

● ● ● | Proof Trees

Proof tree of a fact **A** from database instance **I** and datalog program **P**:

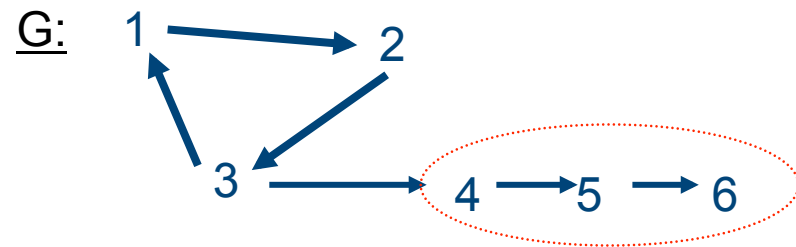
- Each vertex is labeled by a fact
- Each leaf is labeled by a fact from **I**
- The root is labeled by **A**
- Parent-child relationship constructed from rule $A_1 \leftarrow A_2, \dots, A_n$

- (1) $T(x, y) :- G(x, y)$
 (2) $T(x, y) :- G(x, z), T(z, y)$



- ● ● | Top-down Evaluation

- Direct the search for a proof, often one is only interested in particular facts, e.g. $T(4,x)$ for



- Infinite recursive loops possible:

$T(x, y) :- G(x, y)$

$T(x, y) :- \underline{T(x, z)}, G(z, y)$



The Magic Sets Algorithm

Given a datalog query, the magic set approach transforms it into a new query that has two important properties:

- It computes the same answer as the original query.
- When evaluated in a bottom-up technique, it produces only the set of facts produced by top-down approaches.

→ *Selections are pushed from the query into bottom-up computations*

● ● ● | Rule/Goal Graphs (RGGs)

- Needed to assure unique binding patterns for IDB predicates
- Composed of rule and goal nodes.

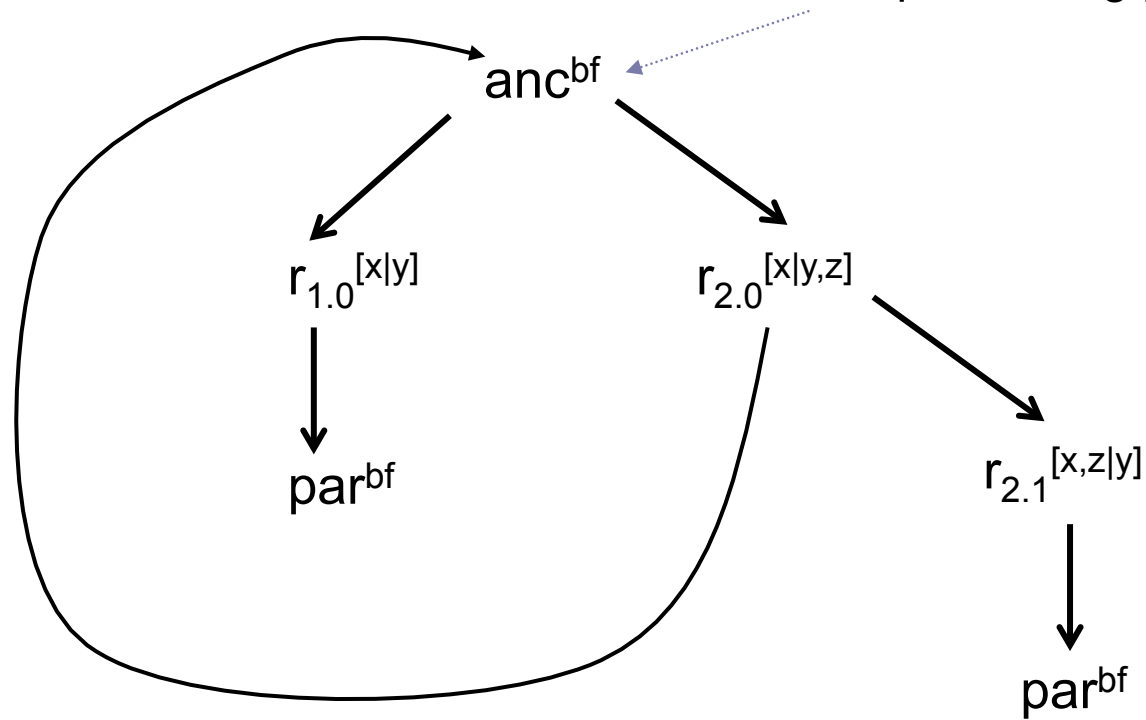
→ *We first look at two examples, then at the definitions.*



r1: anc(x, y) :- par(x, y)
r2: anc(x, y) :- anc(x, z), par(z, y)
query: query(u) :- anc("a", u)

a is a constant

unique binding pattern **bf**





Magic



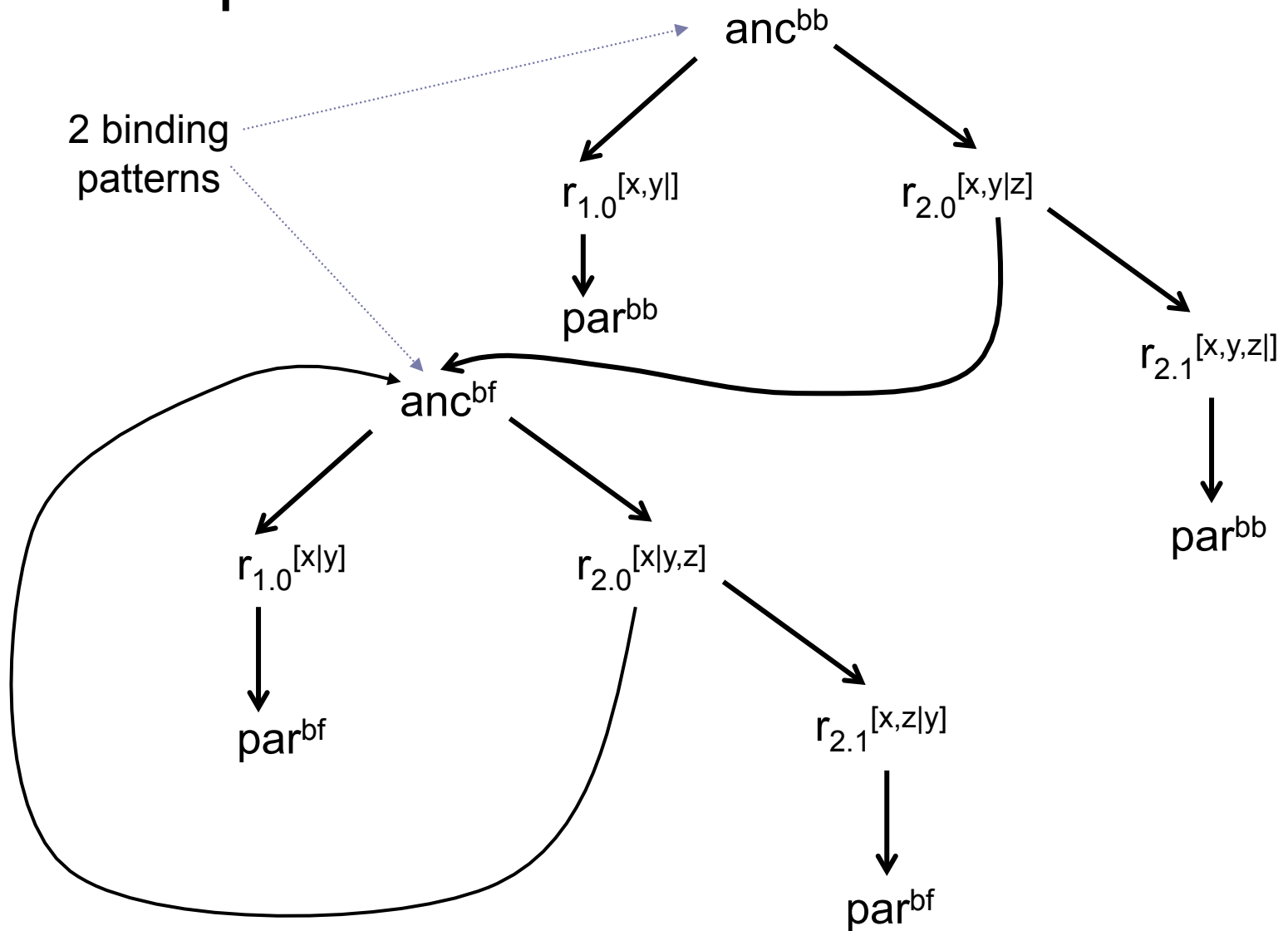
`anc(x,y) :- m_anc(x), par(x,y)`

`anc(x,y) :- m_anc(x), anc(x,z), par(z,y)`

`m_anc("a") :-`



r1: anc(x,y) :- par(x,y)
r2: anc(x,y) :- anc(x,z), par(z,y)
query: query() :- anc("a", "b")





Rule and Goal Nodes

Goal Nodes:

- Predicate + *adornment*
- *Adornment* =
list of **b**'s and **f**'s
indicating which
arguments are bound
and which are free.

Rule Nodes:

- $r_i^{[B|F]}$ represents the
point in rule **r** after
seeing **i** subgoals, with
the variables in **B**
bound, and the
variables in **F** free.

● ● ● | Children of Goal Nodes

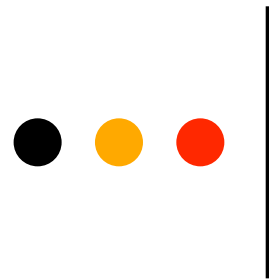
Children of goal node p^α are those rule nodes $r_0^{[B|F]}$ such that...

- Rule r has head predicate p .
- B is the set of variables that appear in those arguments of the head that α says are bound.
- F represents the other variables of r .

● ● ● | Children of Rule Nodes

Children of the rule node $r_j^{[B|F]}$ are:

- The goal node of the $(j+1)$ st subgoal of r , with adornment that binds those arguments whose only variables are in B .
- The rule node $r_{j+1}^{[B'|F']}$, where $B' = B +$ the variables appearing in the $(j+1)$ st subgoal; F' are the other variables.
- Exceptions:
 - No r_{j+1} rule node if r has only $j+1$ subgoals.
 - No goal child if $j=0$ and r has no subgoals.



Constructing the RGG

- Start with goal node whose adornment matches the bindings of the query
- Add nodes by constructing children
- Reordering of subgoals of a rule is allowed
→this helps maximize the bound arguments

● ● ● | Magic Sets Transformation

Start with a datalog program and a binding pattern for a query

○ Split predicates to get unique binding patterns

○ Rectify subgoals

○ Introduce magic and supplementary predicates as follows...



Magic Predicates

For each IDB predicate p ,
introduce predicate m_p .

- Arguments of m_p correspond to bound arguments of p in its unique binding pattern
- Intuition: m_p is true of exactly those tuples that are members of queries to some p -node in the top-down expansion

```
anc(x, y) :- par(x, y)
anc(x, y) :- anc(x, z),
               par(z, y)
```

```
query(u) :- anc("a", u)
```



```
m_anc(x) :- ...
```



Supplementary Predicates

- For each rule r of n subgoals, introduce supplementary predicates $\mathbf{sup}_{r,j}$ for $0 \leq j < n$.
- Arguments are the *bound* and *active* variables before the j +1st subgoal of r .
 - A variable is *active* iff it appears either in the head or a subgoal from $j+1$ on.
 - Intuition: True for a tuple iff that tuple represents a possible binding for the bound, active variables at that point.

```
r1: anc(x, y) :- par(x, y)
r2: anc(x, y) :- anc(x, z),
                    par(z, y)
```

```
query(u) :- anc("a", u)
```



```
anc(x, y) :- sup1.0(x),
              par(x, y)
anc(x, y) :- sup2.0(x),
              anc(x, z),
              sup2.1(x, z),
```



5 Groups of Rules for Magic Construction

Let r be a rule

$H :- G_1, \dots, G_n$

```
r1: anc(x,y) :- par(x,y)
r2: anc(x,y) :- anc(x,z),
                               par(z,y)
query(u) :- anc("a",u)
```

● ● ● | 1) Supplementary → Magic

Let r be a rule

$H :- G_1, \dots, G_n$

If G_i has IDB predicate p :

$m_p(\text{bound args of } G_i) :- \text{sup}_{r.i-1}(\text{variables})$

$r_1: \text{anc}(x, y) :- \text{par}(x, y)$
 $r_2: \text{anc}(x, y) :- \text{anc}(x, z), \text{par}(z, y)$

$\text{query}(u) :- \text{anc}(\text{"a"}, u)$



$m_{\text{anc}}(x) :- \text{sup}_{2.0}(x)$

● ● ● | 2) Magic \rightarrow 0th supplementary

Let r be a rule

$H :- G_1, \dots, G_n$

If Head has predicate p :

$sup_{r.0}(variables) :-$
 $m_p(bound arguments)$

$r_1: anc(x, y) :- par(x, y)$
 $r_2: anc(x, y) :- anc(x, z),$
 $par(z, y)$

query(u) :- anc("a", u)



$sup_{1.0}(x) :- m_anc(x)$

$sup_{2.0}(x) :- m_anc(x)$

- ● ● | 3) $i-1^{\text{st}}$ supplementary + G_i
 → i^{th} supplementary

Let r be a rule

$H :- G_1, \dots, G_n$

$\text{sup}_{r.i}(\text{variables}) :-$
 $\text{sup}_{r.i-1}(\text{variables}), G_i$

$r_1: \text{anc}(x, y) :- \text{par}(x, y)$
 $r_2: \text{anc}(x, y) :- \text{anc}(x, z), \text{par}(z, y)$

$\text{query}(u) :- \text{anc}(\text{"a"}, u)$



$\text{sup}_{2.1}(x, z) :- \text{sup}_{2.0}(x), \text{anc}(x, z)$

- ● ● | 4) last supplementary
+ last subgoal → head

Let r be a rule

$H :- G_1, \dots, G_n$

$H :-$
 $\text{sup}_{r.n-1}(\text{variables}), G_n$

$r_1: \text{anc}(x, y) :- \text{par}(x, y)$
 $r_2: \text{anc}(x, y) :- \text{anc}(x, z), \text{par}(z, y)$
 $\text{query}(u) :- \text{anc}(\text{"a"}, u)$



$\text{anc}(x, y) :- \text{sup}_{1.0}(x), \text{par}(x, y)$
 $\text{anc}(x, y) :- \text{sup}_{2.1}(x, z), \text{par}(z, y)$



5) Initialize

If query has predicate **p** then
we have the bodyless rule

`m_p (bound args from query) :-`

```
r1: anc(x,y) :- par(x,y)
r2: anc(x,y) :- anc(x,z),
                        par(z,y)
query(u) :- anc("a",u)
```



```
m_anc("a") :-
```



The computed rules

(1) $\text{m_anc}(\mathbf{x}) \text{ :- sup}_{2.0}(\mathbf{x})$

(2) $\text{sup}_{1.0}(\mathbf{x}) \text{ :- m_anc}(\mathbf{x})$

$\text{sup}_{2.0}(\mathbf{x}) \text{ :- m_anc}(\mathbf{x})$

(3) $\text{sup}_{2.1}(\mathbf{x}, \mathbf{z}) \text{ :- sup}_{2.0}(\mathbf{x}), \text{anc}(\mathbf{x}, \mathbf{z})$

(4) $\text{anc}(\mathbf{x}, \mathbf{y}) \text{ :- sup}_{1.0}(\mathbf{x}), \text{par}(\mathbf{x}, \mathbf{y})$

$\text{anc}(\mathbf{x}, \mathbf{y}) \text{ :- sup}_{2.1}(\mathbf{x}, \mathbf{z}), \text{par}(\mathbf{z}, \mathbf{y})$

(5) $\text{m_anc}(\text{"a"}) \text{ :-}$

● ● ● | Simplification of Rules

- Use Group-2 rules to replace $\text{sup}_{r.0}$ by magic predicates.
- If a supplementary predicate comes before an EDB subgoal, it is used only once and may be eliminated.
- However, if it is before an IDB subgoal, it is used twice and should not be eliminated.



Simplify

(1) $m_anc(x) :- sup_{2.0}(x)$

(2) $sup_{1.0}(x) :- m_anc(x)$

$sup_{2.0}(x) :- m_anc(x)$

(3) $sup_{2.1}(x, z) :- sup_{2.0}(x), anc(x, z)$

(4) $anc(x, y) :- sup_{1.0}(x), par(x, y)$

$anc(x, y) :- sup_{2.1}(x, z), par(z, y)$

(5) $m_anc("a") :-$



Simplify...

$$(1) \quad m_anc(x) \quad :- \quad \cancel{sup_{2.0}(x)} \quad m_anc(x)$$

Use Group-2 rules to replace $sup_{r.0}$ by magic predicates.

$$(2) \quad \cancel{sup_{1.0}(x)} \quad :- \quad m_anc(x)$$
$$\cancel{sup_{2.0}(x)} \quad :- \quad m_anc(x)$$

$$(3) \quad sup_{2.1}(x, z) \quad :- \quad \cancel{sup_{2.0}(x)} \quad m_anc(x), \quad anc(x, z)$$

$$(4) \quad anc(x, y) \quad :- \quad \cancel{sup_{1.0}(x)} \quad m_anc(x), \quad par(x, y)$$
$$anc(x, y) \quad :- \quad sup_{2.1}(x, z), \quad par(z, y)$$

$$(5) \quad m_anc("a") \quad :-$$



Simplify...

(1) ~~$m_anc(x) :- m_anc(x)$~~

(3) $sup_{2.1}(x, z) :- m_anc(x), anc(x, z)$

(4) $anc(x, y) :- m_anc(x), par(x, y)$
 $anc(x, y) :- sup_{2.1}(x, z), par(z, y)$

(5) $m_anc("a") :-$



Simplify...

`sup2.1(x, z) :- m_anc(x), anc(x, z)`

`anc(x, y) :- m_anc(x), par(x, y)`

`anc(x, y) :- sup2.1(x, z), par(z, y)`

`m_anc("a") :-`



Simplify...

If a supplementary predicate comes before an EDB subgoal, it is used only once and may be eliminated.

~~$\text{sup}_{2.1}(x, z) :- \text{m_anc}(x), \text{anc}(x, z)$~~

$\text{anc}(x, y) :- \text{m_anc}(x), \text{par}(x, y)$

$\text{anc}(x, y) :- \text{sup}_{2.1}(x, z), \text{par}(z, y)$

$\text{m_anc}(\text{"a"}) :-$

● ● ● | Done.



`anc(x,y) :- m_anc(x), par(x,y)`

`anc(x,y) :- m_anc(x), anc(x,z), par(z,y)`

`m_anc("a") :-`

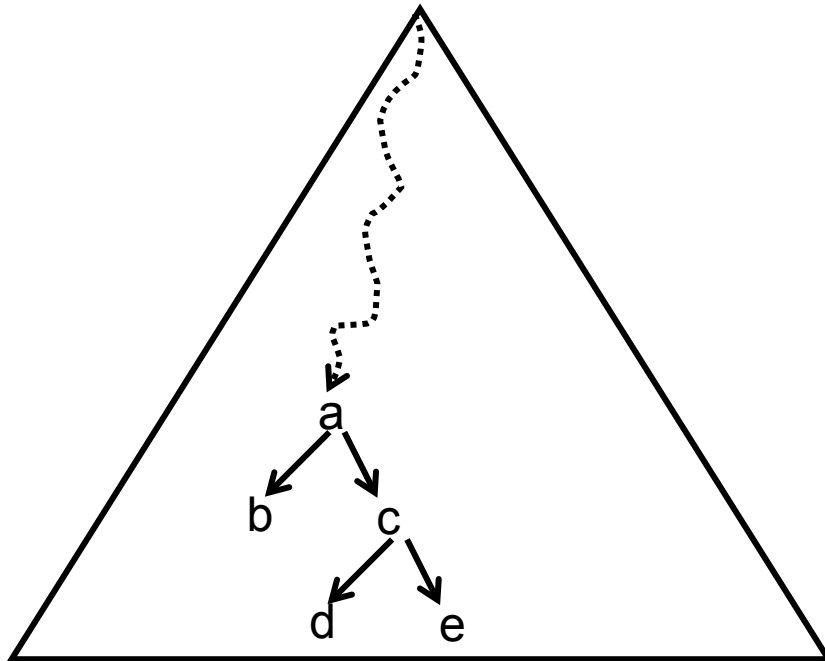


```
anc(x,y) :- m_anc(x), par(x,y)
anc(x,y) :- m_anc(x), anc(x,z), par(z,y)
m_anc("a") :-
```

parent:

m_anc___

anc___



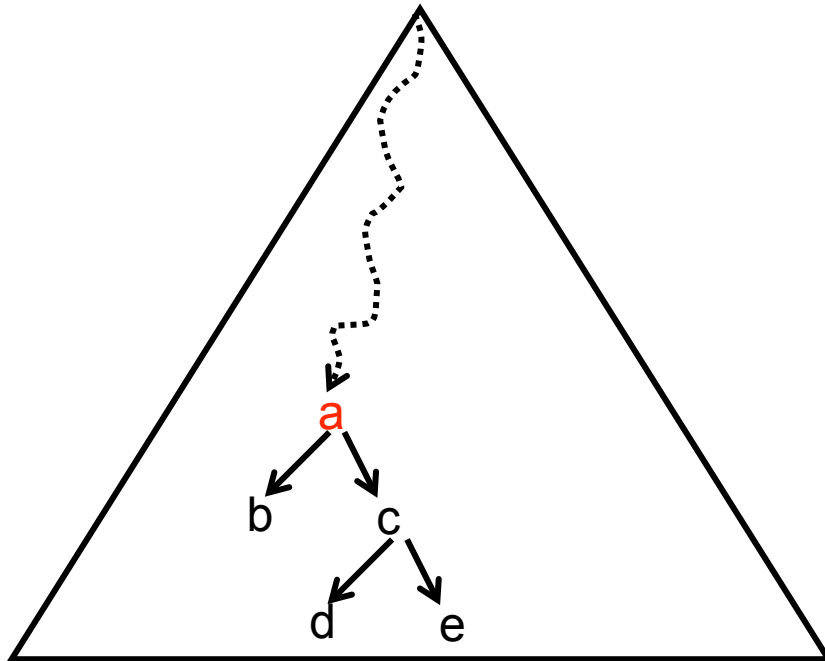


```
anc(x,y) :- m_anc(x), par(x,y)
anc(x,y) :- m_anc(x), anc(x,z), par(z,y)
m_anc("a") :-
```

parent:

m_anc__
a

anc__



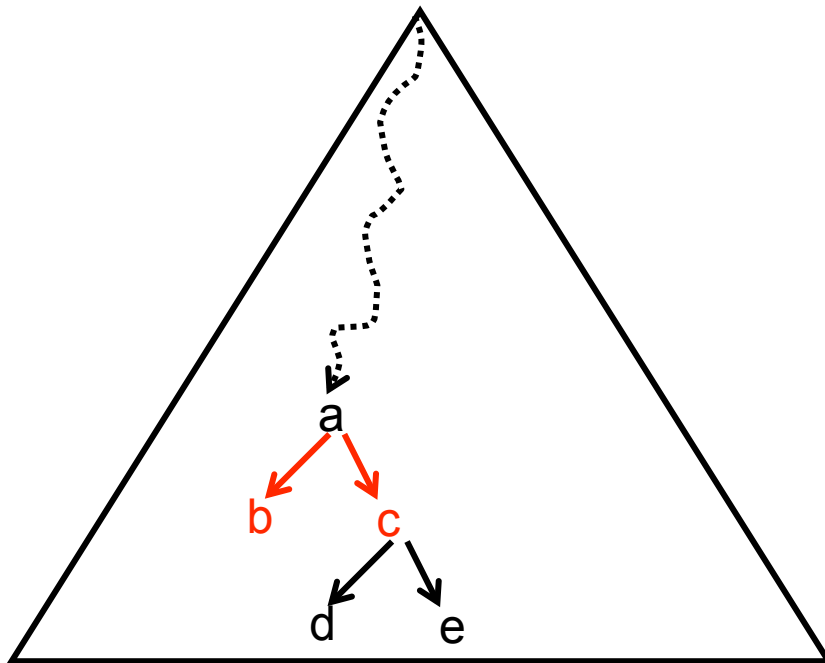


```
anc(x,y) :- m_anc(x), par(x,y)
anc(x,y) :- m_anc(x), anc(x,z), par(z,y)
m_anc("a") :-
```

parent:

m_anc
a

anc
a | b
a | c





`anc(x,y) :- m_anc(x), par(x,y)`

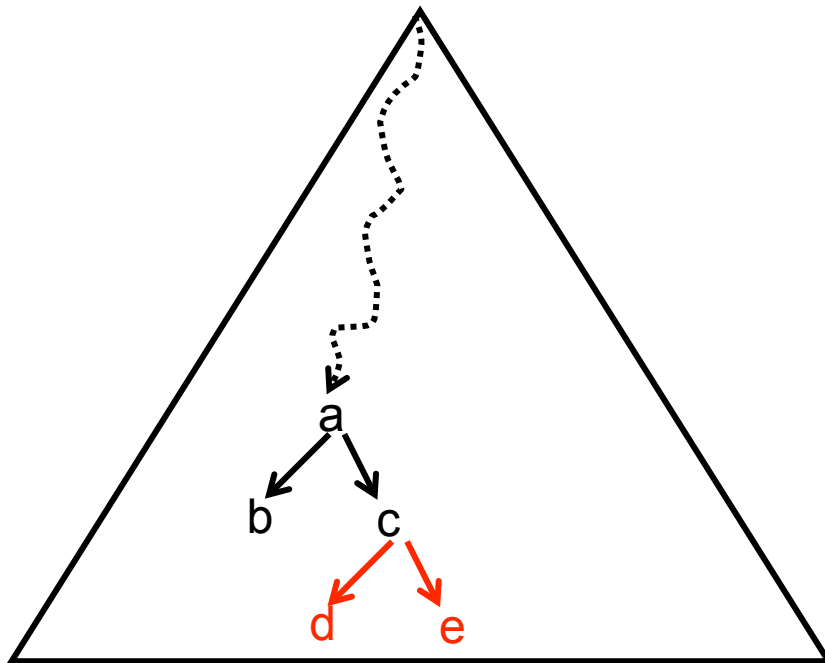
`anc(x,y) :- m_anc(x), anc(x,z), par(z,y)`

`m_anc("a") :-`

parent:

m_anc _____
a

anc _____
a | b
a | c
a | d
a | e





`anc(x,y) :- m_anc(x), par(x,y)`

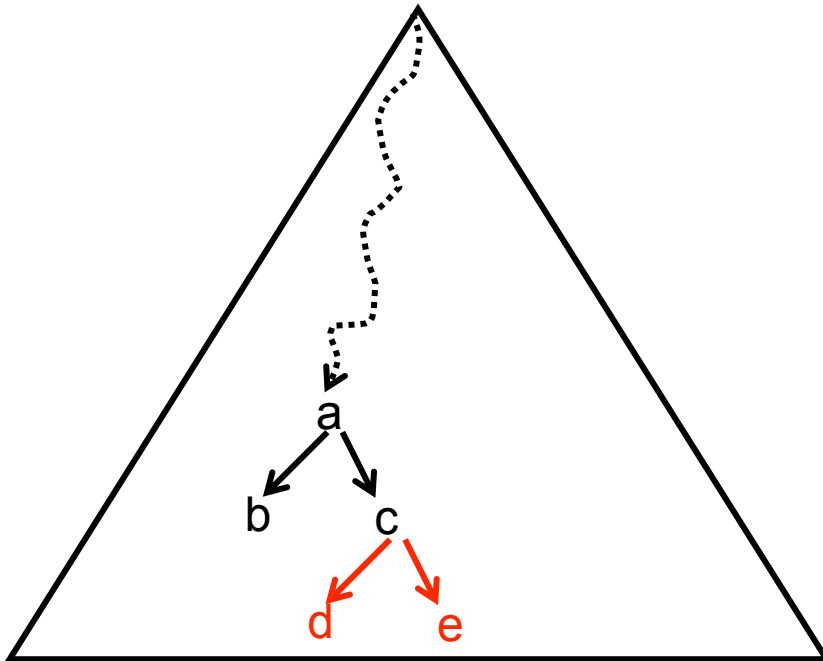
`anc(x,y) :- m_anc(x), anc(x,z), par(z,y)`

`m_anc("a") :-`

parent:

m_anc _____
a

anc _____
a | b
a | c
a | d
a | e

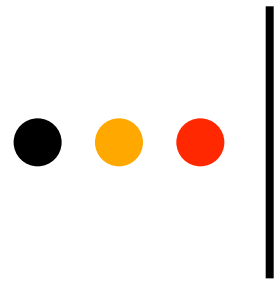


→ *Magic rules + seminaive evaluation only infers facts that any top-down implementation would infer*

● ● ● | Magic Sets Transformation

Start with a datalog program and a binding pattern for a query

- Split predicates to get unique binding patterns
- Rectify subgoals
- Introduce magic and supplementary predicates as follows...

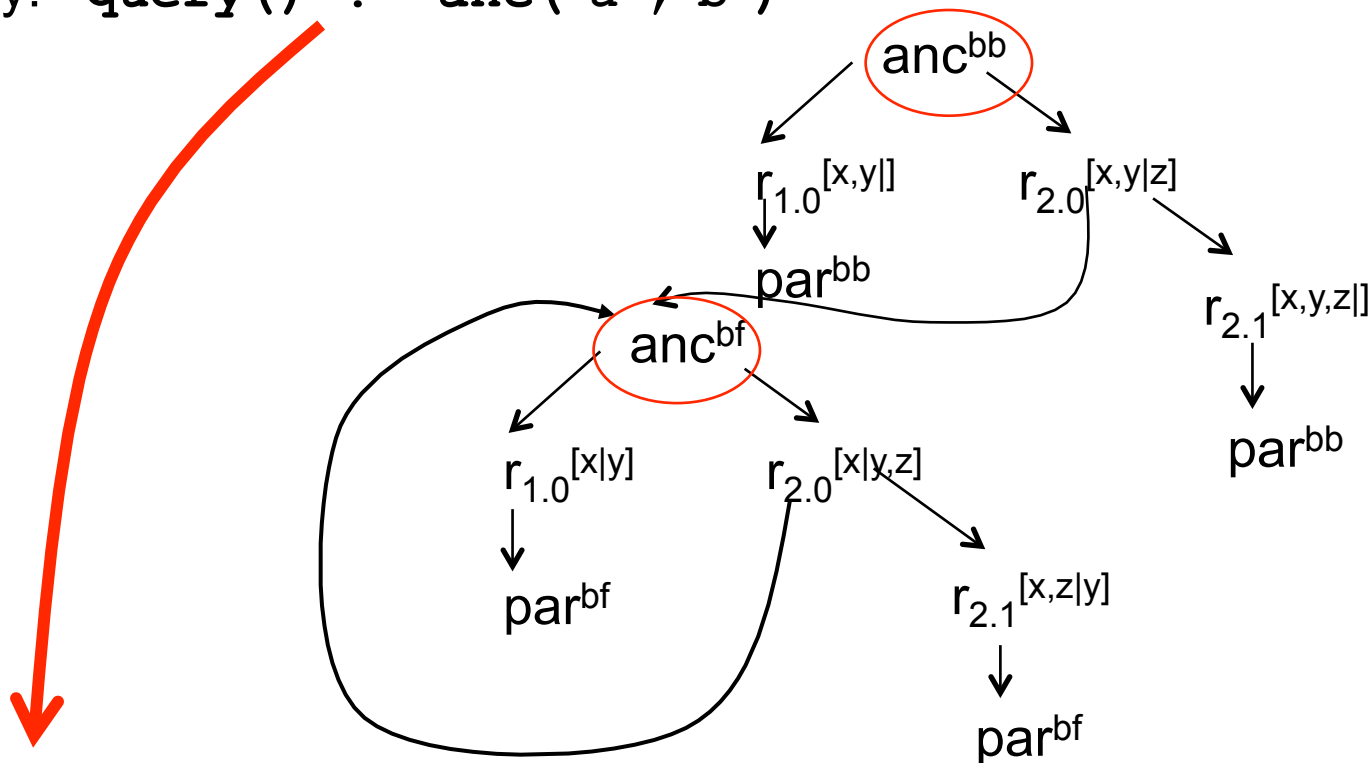


(1) Unique Binding Patterns

- For magic sets to work, there must be a unique binding pattern associated with each IDB predicate
- No constraint on EDB predicates
- RGG helps us figure out the needed binding patterns.



r1: anc(x,y) :- par(x,y)
r2: anc(x,y) :- anc(x,z), par(z,y)
query: query() :- anc("a","b")



r1a: **bb_anc**(x,y) :- par(x,y)
r2a: **bb_anc**(x,y) :- **bf_anc**(x,z), par(z,y)
r1b: **bf_anc**(x,y) :- par(x,y)
r2b: **bf_anc**(x,y) :- **bf_anc**(x,z), par(z,y)
query: query() :- **bb_anc**("a","b")



Unique Binding Patterns

Key idea:

- For each adornment α such that \mathbf{p}^α appears in the RGG, make a new predicate α_p .
- Rules for α_p are the same as for p , but predicates of IDB subgoals are the version with the correct binding pattern.

● ● ● | Magic Sets Transformation

Start with a datalog program and a binding pattern for a query

- Split predicates to get unique binding patterns
- Rectify subgoals
- Introduce magic and supplementary predicates as follows...

- ● ● | (2) Rectifying Subgoals

- All IDB subgoals must have arguments that are distinct variables.

Violation:

r1: $p(x, y) :- a(x, y)$

r2: $p(x, y) :- b(x, z), p(z, z), b(z, y)$



r1: $p(x, y) :- a(x, y)$

r2: $p(x, y) :- b(x, z), p(z, z), b(z, y)$

- o $p(z, z)$ is unrectified. Create $q(z) :- p(z, z)$.
- o Unify heads of rules with $p(z, z)$, then r1 becomes
 $q(z) :- a(z, z)$
and r2 becomes
 $q(z) :- b(z, w), q(w), b(w, z)$
- o In the original r2, replace subgoal $p(z, z)$ by $q(z)$.

- o The resulting rules:

r1: $p(x, y) :- a(x, y)$

r2': $p(x, y) :- b(x, z), q(z), b(z, y)$

r3: $q(z) :- a(z, z)$

r4: $q(z) :- b(z, w), q(w), b(w, z)$



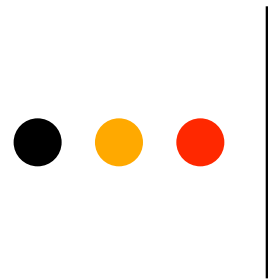
Rectifying Subgoals

- Replace an IDB subgoal **G** with variables appearing in more than one argument and/or constant arguments by a new predicate whose arguments are single copies of the variables appearing in **G**.
- Create rules for the new predicate by unifying **G** with heads of rules for **G**'s predicate.
- Repetition may be needed because the new rules may have unrectified subgoals.

● ● ● | Magic Sets Transformation

Start with a datalog program and a binding pattern for a query

- Split predicates to get unique binding patterns
- Rectify subgoals
- Introduce magic and supplementary predicates according to rules (1) – (5).



Final Words

- o Datalog by itself is interesting as a query language
- o Recursive datalog is a basis for recursive SQL
- o Magic sets is also an optimization method for nonrecursive SQL

Reading material:

Abiteboul, Hull, Vianu: *Foundations of Databases*.
Ullman: *Database and Knowledge-Base Systems*, Vol. II.
Website www-db.stanford.edu/~ullman/cs345-notes.html