CSE544
Data Management

Lectures 13

Datalog

Announcement

- Project Milestone due on Monday, 2/26
- HW3 extended to Thursday, 2/29

Project

- Project meetings w/ Dan: Friday, 3/1
- Printing the poster:
  - Kyle can help on Monday, 3/4, OR ask a colleague with a cse account
- Poster presentations: Wed, 3/6, 10-2pm
  - In the atrium of Allen building
  - Setup: 9:30; poster + demo (optional)
  - Snacks, pizza will be provided

Datalog

Motivation

- RA cannot express iteration/recursion
  SQL can, but clumsy, limited

- Data science needs iteration/recursion

- Datalog: designed for recursion

Datalog

- Proposed in the 80’s as “Prolog for DBs”
- Not adopted by industry, no standards
- A darling of academics, hot topic in DB, PL, Networking, …
- In HW4 we will use Souffle

Outline

- Syntax
- Getting familiar with Datalog
- Semantics

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Schema

Datalog: Facts and Rules

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :-  Movie(x,y,z), z='1940'.
```

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :-  Movie(x,y,z), z='1940'.
```

Find Movies made in 1940

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :-  Movie(x,y,z), z='1940'.
```

```
Q2(f, l) :-  Actor(z,f,l), Casts(z,x),
                 Movie(x,y,'1940').
```

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),
Movie(x,y,'1940').

Find Actors who acted in Movies made in 1940

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

Rules = queries

```
Q1(y) :-  Movie(x,y,z), z='1940'.
```

```
Q2(f, l) :-  Actor(z,f,l), Casts(z,x),
             Movie(x,y,'1940').
```

```
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),
           Casts(z,x2), Movie(x2,y2,1940)
```

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

```
Q1(y) :-  Movie(x,y,z), z='1940'.
```

```
Q2(f, l) :-  Actor(z,f,l), Casts(z,x),
             Movie(x,y,'1940').
```

```
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),
           Casts(z,x2), Movie(x2,y2,1940)
```

Find Actors who acted in a Movie in 1940 and in one in 1910

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)

Datalog: Facts and Rules

Facts = tuples in the database

Rules = queries

```
Actor(344759,'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).
```

```
Q1(y) :-  Movie(x,y,z), z='1940'.
```

```
Q2(f, l) :-  Actor(z,f,l), Casts(z,x),
                 Movie(x,y,'1940').
```

```
Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),
                          Casts(z,x2), Movie(x2,y2,1940)
```

Extensional Database Predicates = EDB = Actor, Casts, Movie

Intensional Database Predicates = IDB = Q1, Q2, Q3

Anatomy of a Rule

head — body

atom, atom, atom (aka predicate)

```
Q2(f, l) :-  Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').
```

f, l = head variables
x,y,z = existential variables

More Datalog Terminology

```
Q(args) :- R1(args), R2(args), ....
```

- $R_i(args_i)$ called an *atom*, or a *relational predicate*
- $R_i(args_i)$ evaluates to true or false
- Can also have arithmetic predicates, e.g. z > 1940

Datalog program

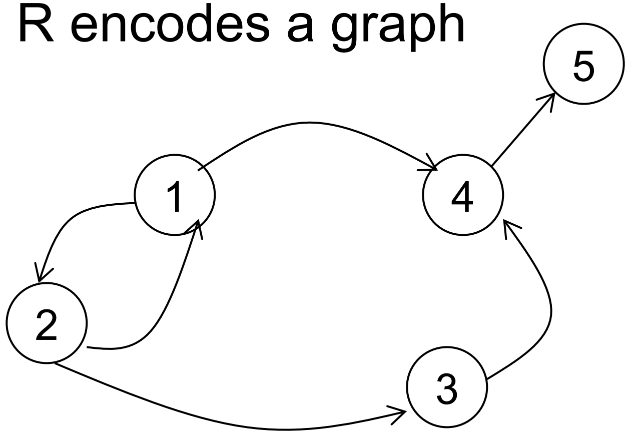
- Datalog program = several rules
- Rules may be recursive!
- Often one IDB is final answer

Outline

- Syntax
- Getting familiar with Datalog
- Semantics

Example

R encodes a graph

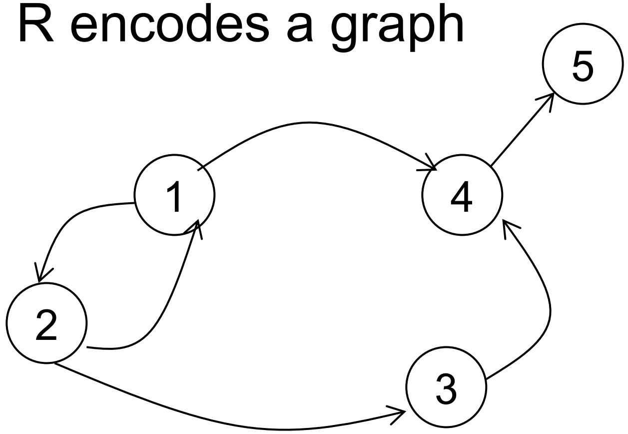


R=

1	2
2	1
2	3
1	4
3	4
4	5

Example

R encodes a graph



1	2
2	1
2	3
1	4
3	4
4	5

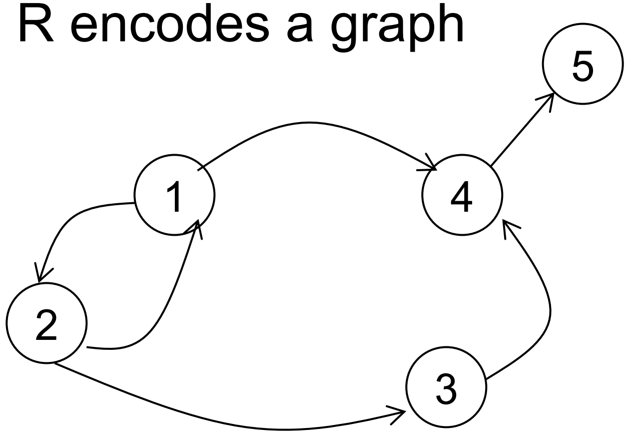
R=

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

What does it compute?

Example

R encodes a graph



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

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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:

T is empty.

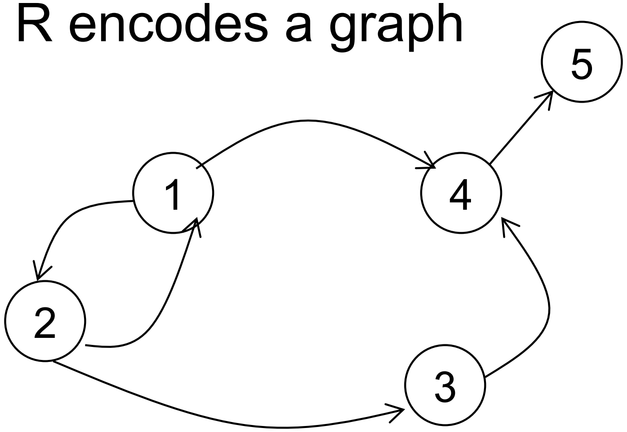
Example

T(x,y) :- R(x,y)

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

What does it compute?

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:

T is empty.

First iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5

First rule generates this

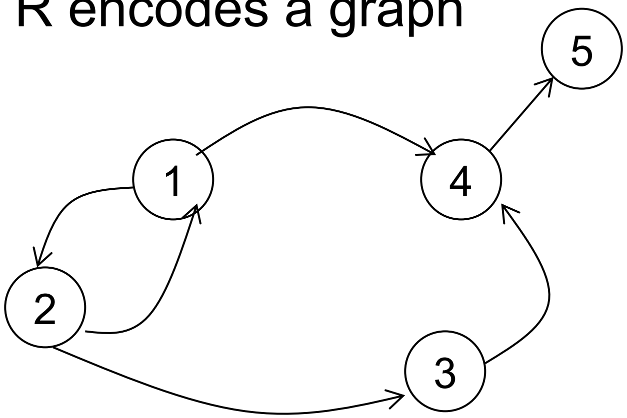
Second rule generates nothing (because T is empty)

Example

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

What does it compute?

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:
T is empty.

First iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:
T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

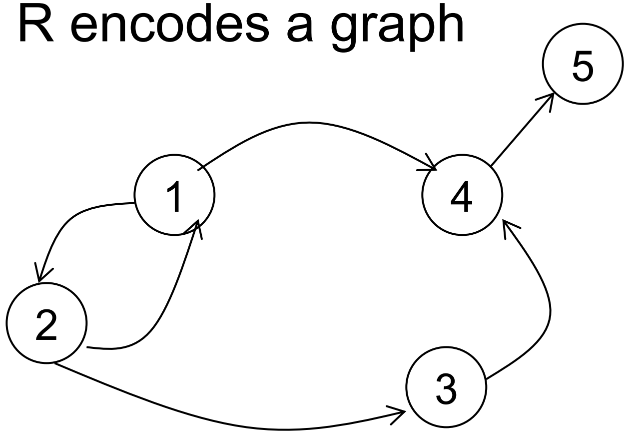
First rule generates this (rows 1–6)

Second rule generates this (rows 7–12)

New facts

Example

R encodes a graph



T(x,y) :- R(x,y)

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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:

T is empty.

First iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:

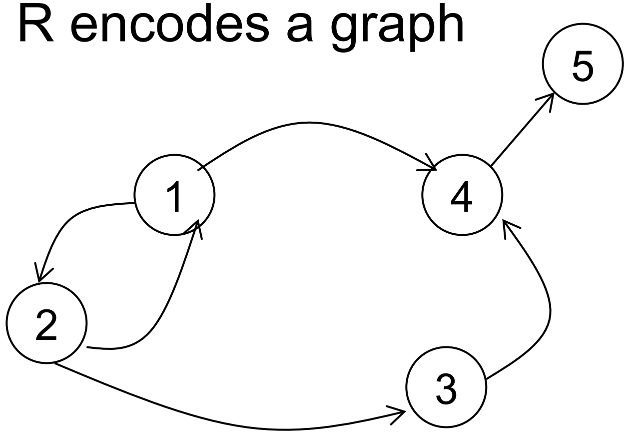
T =

1	2	Both rules
2	1	Both rules
2	3	First rule
1	4	First rule
3	4	First rule
4	5	First rule
1	1	
2	2	Second rule
1	3	Second rule
2	4	Second rule
1	5	Second rule
3	5	Second rule
2	5	Second rule (New fact)

New fact

Example

R encodes a graph



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

What does it compute?

R=

1	2
2	1
2	3
1	4
3	4
4	5

Initially:

T is empty.

First iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5

Second iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5

Third iteration:

T =

1	2
2	1
2	3
1	4
3	4
4	5
1	1
2	2
1	3
2	4
1	5
3	5
2	5

Fourth iteration

T =

(same)

No new facts. DONE

Iteration k computes pairs (x,y) connected by path of length ≤ k

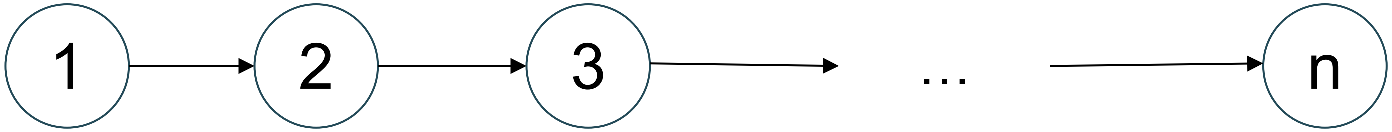
Discussion

- Datalog evaluation is iterative
- It adds new facts at each iteration, stops when nothing new to add
- It always terminates, because the set of possible facts is finite

Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

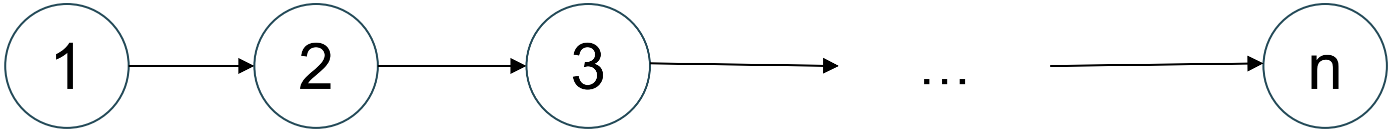
How many iterations until termination?



Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

How many iterations until termination?

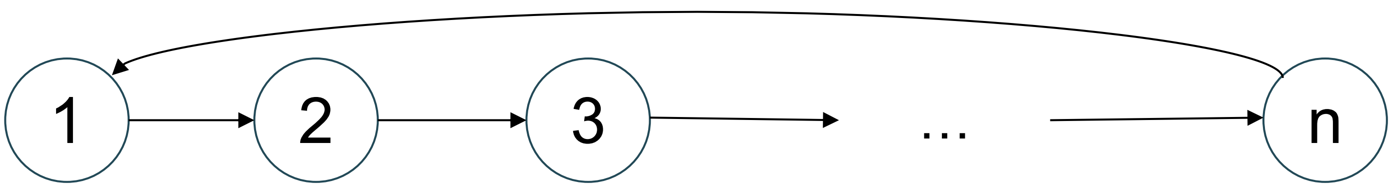


n iterations

Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

How many iterations until termination?

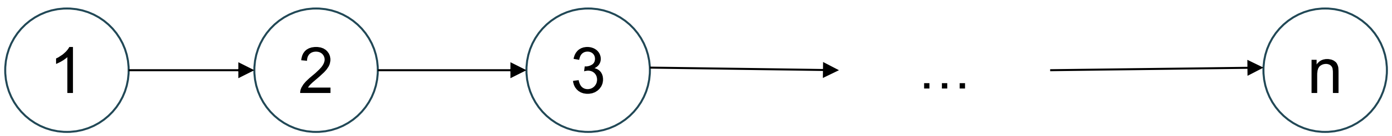


n iterations

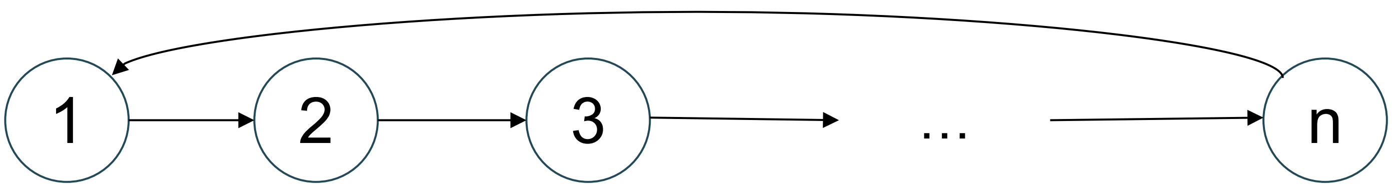
Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

How many iterations until termination?



n iterations

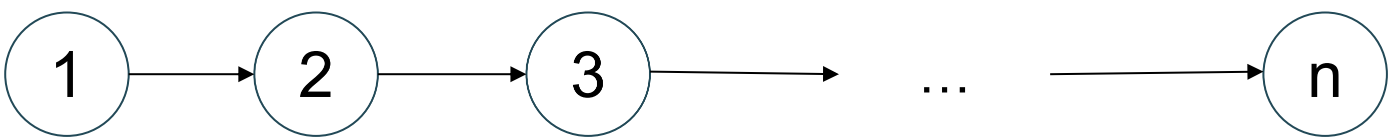


n iterations

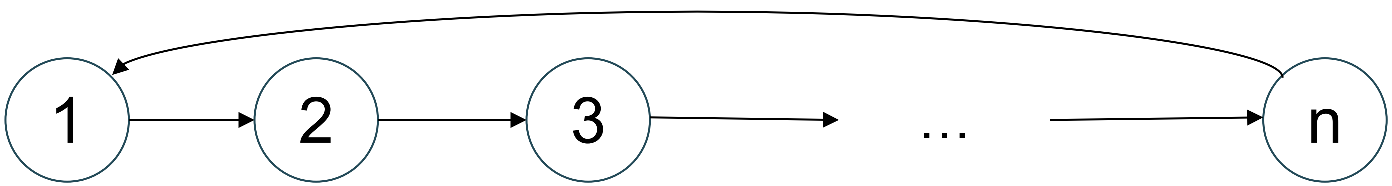
Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

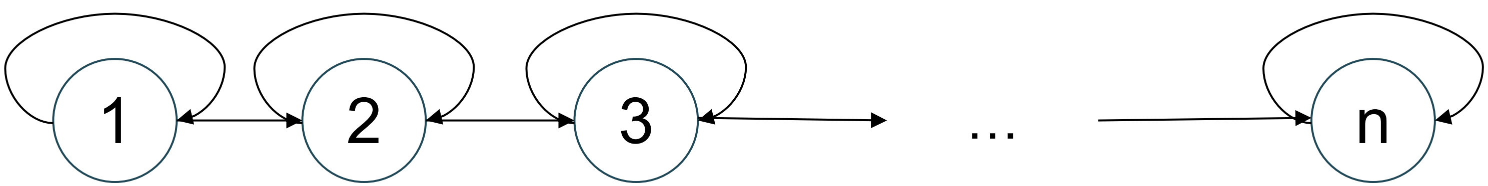
How many iterations until termination?



n iterations



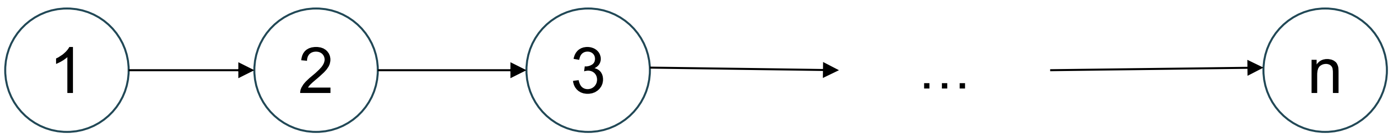
n iterations



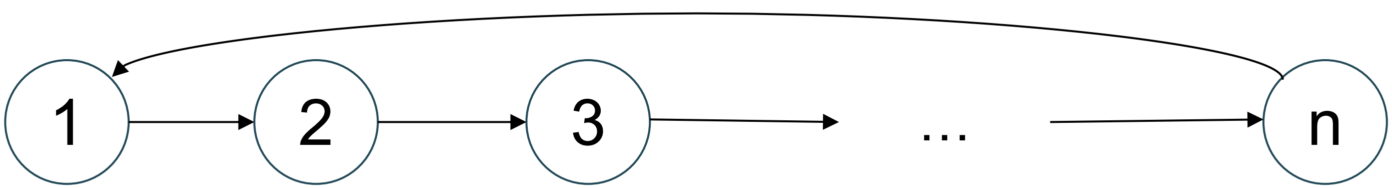
Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

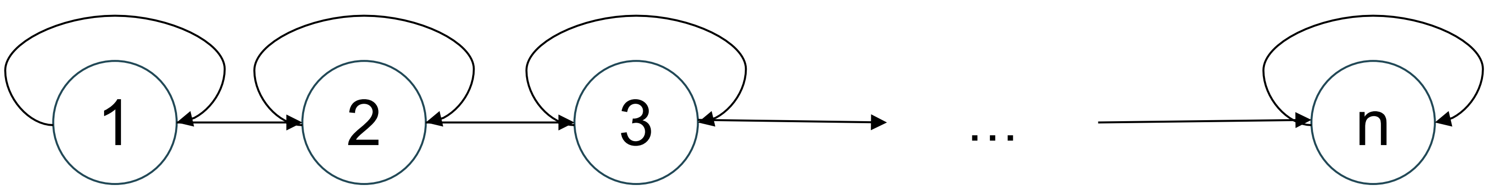
How many iterations until termination?



n iterations



n iterations

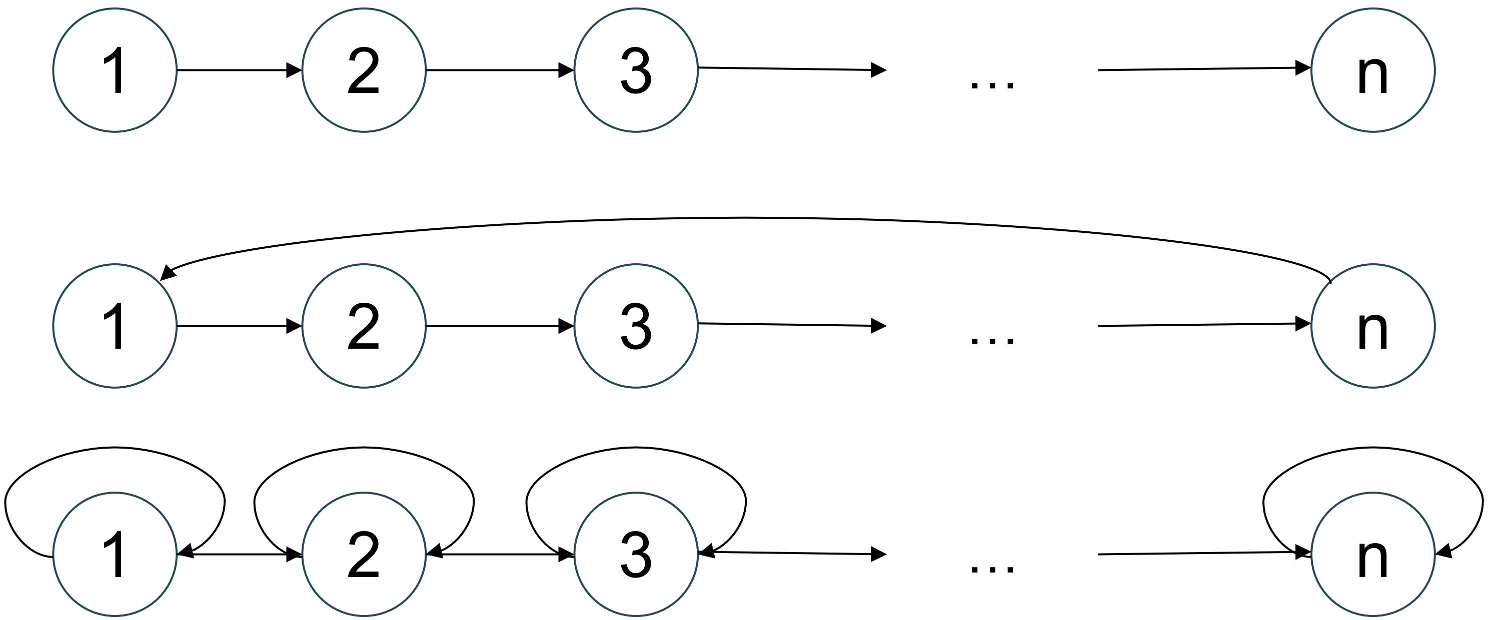


n iterations

Example

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

How many iterations until termination?



n iterations

n iterations

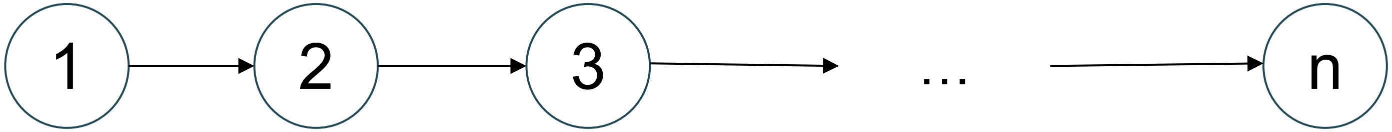
n iterations

How many iterations on an arbitrary graph G?

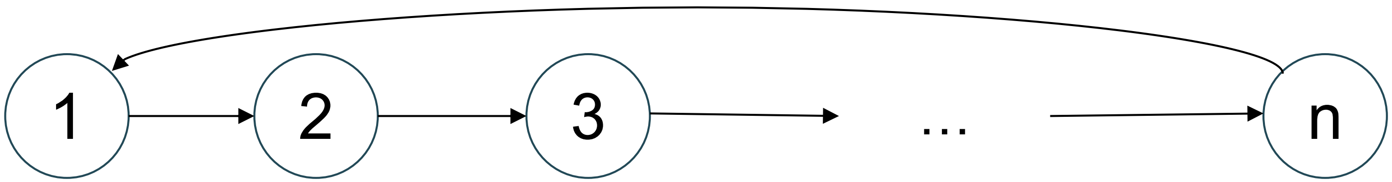
Example

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

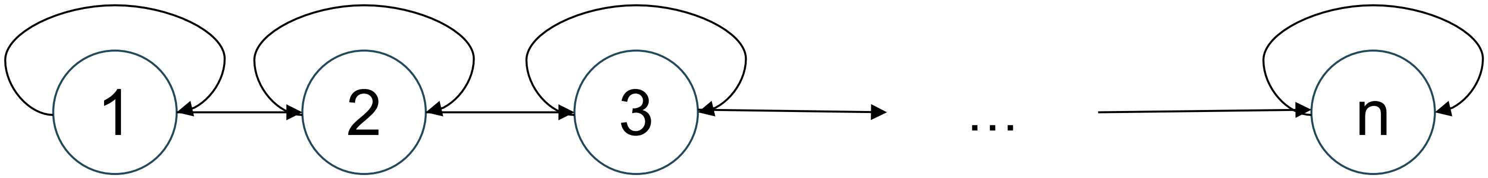
How many iterations until termination?



n iterations



n iterations

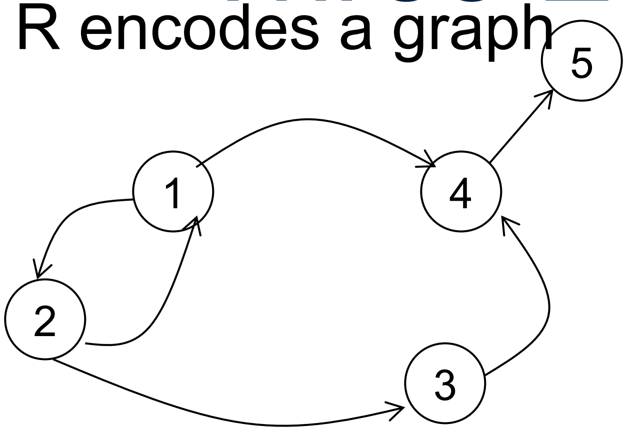


n iterations

How many iterations on an arbitrary graph G? Diameter(G)

Three Equivalent Programs

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

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

Right linear

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

Left linear

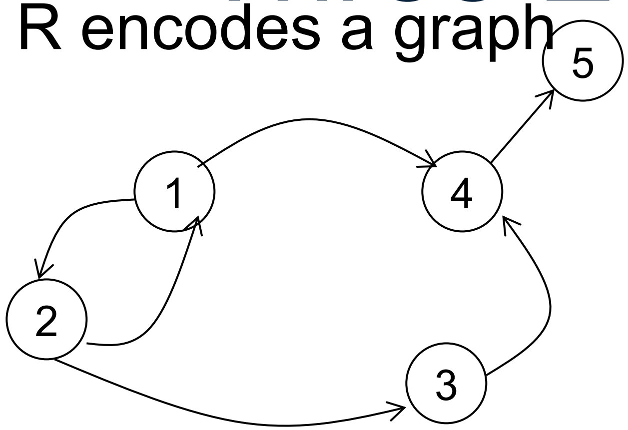
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), T(z,y)

Non-linear

How many iterations on an arbitrary graph G?

Three Equivalent Programs

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

Right linear

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
```

Left linear

#iterations = diameter

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), T(z,y)
```

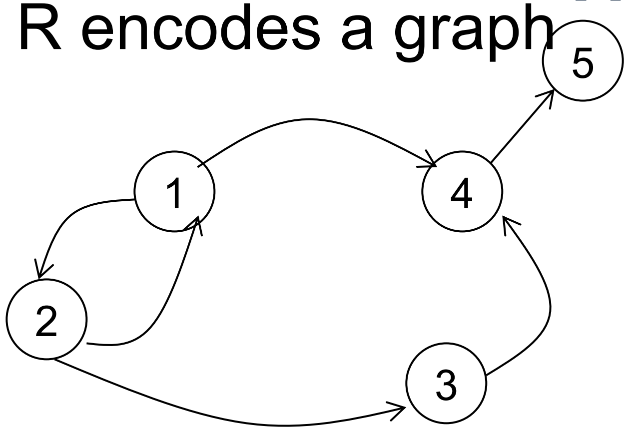
Non-linear

#iterations = log(diameter)

How many iterations on an arbitrary graph G?

Multiple IDBs

R encodes a graph



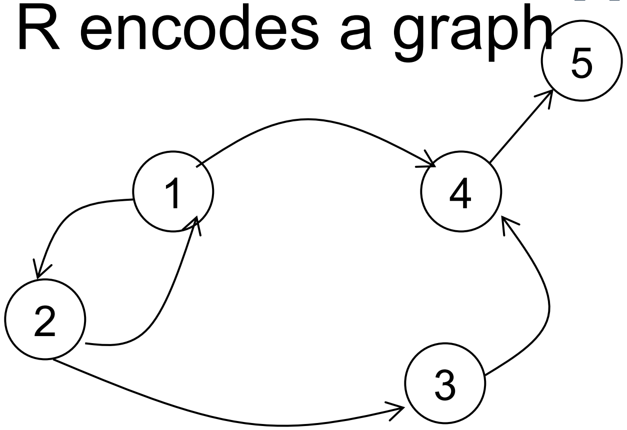
Find pairs of nodes (x,y) connected by a path of *even* length

R=

1	2
2	1
2	3
1	4
3	4
4	5

Multiple IDBs

R encodes a graph



Find pairs of nodes (x,y) connected by a path of *even* length

R=

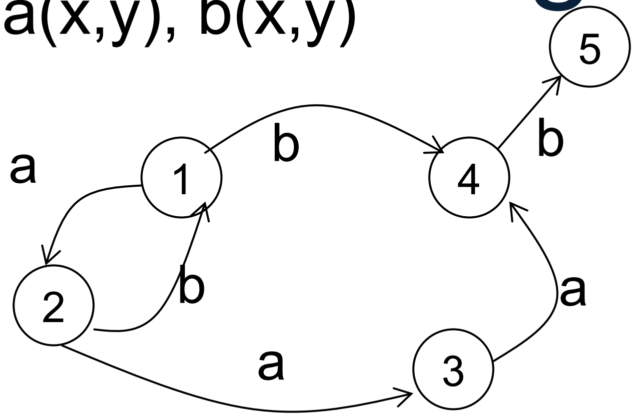
1	2
2	1
2	3
1	4
3	4
4	5

Odd(x,y) :- R(x,y)
Even(x,y) :- Odd(x,z), R(z,y)
Odd(x,y) :- Even(x,z), R(z,y)

Two IDBs: Odd(x,y) and Even(x,y)

Regular Expressions

Labeled Graph:
a(x,y), b(x,y)

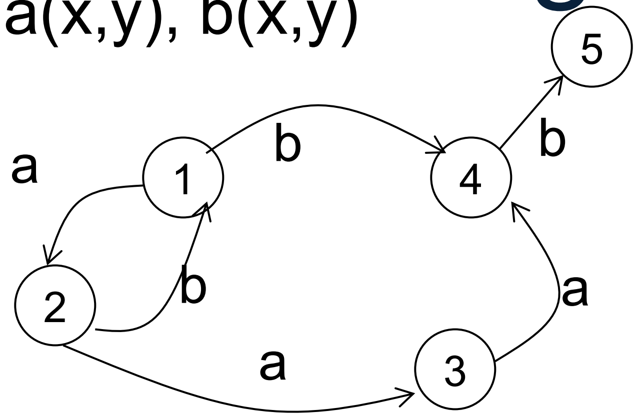


Find pairs of nodes connected by a path whos labels match

(a.a.b*)*.a

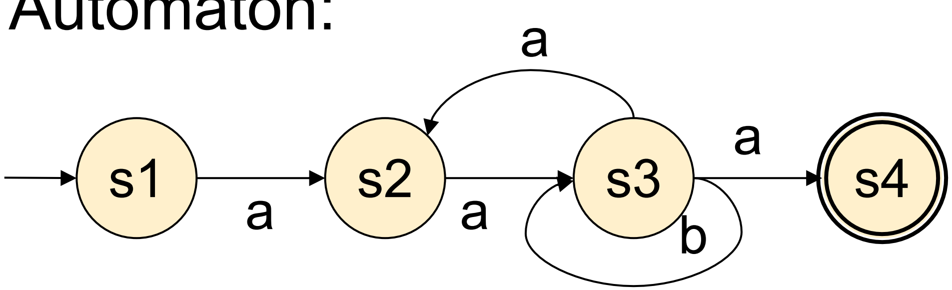
Regular Expressions

Labeled Graph:
a(x,y), b(x,y)



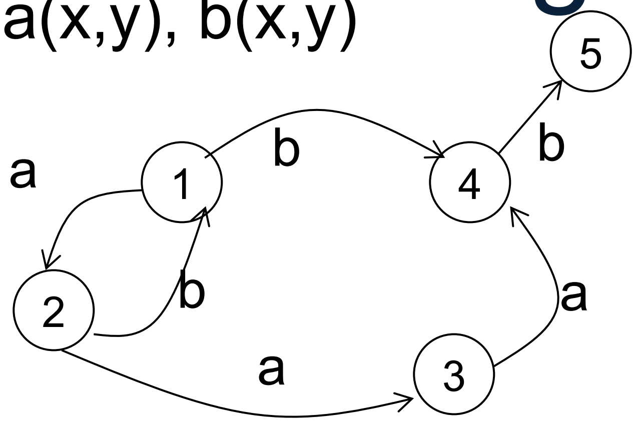
Find pairs of nodes connected by a path whos labels match
(a.a.b*)*.a

Automaton:



Regular Expressions

Labeled Graph:
a(x,y), b(x,y)

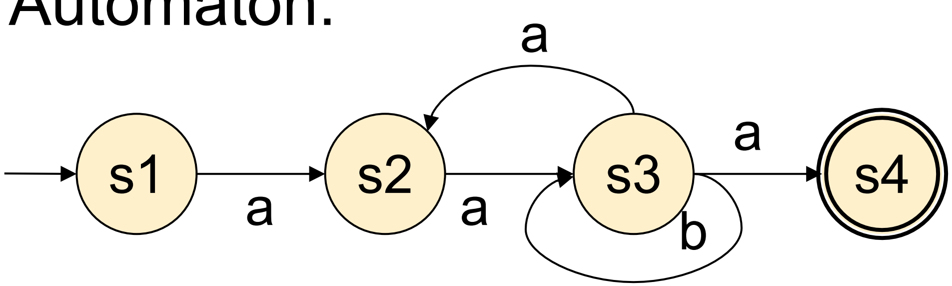


$T_i(x,y)$ = pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is s_i.

Find pairs of nodes connected by a path whos labels match

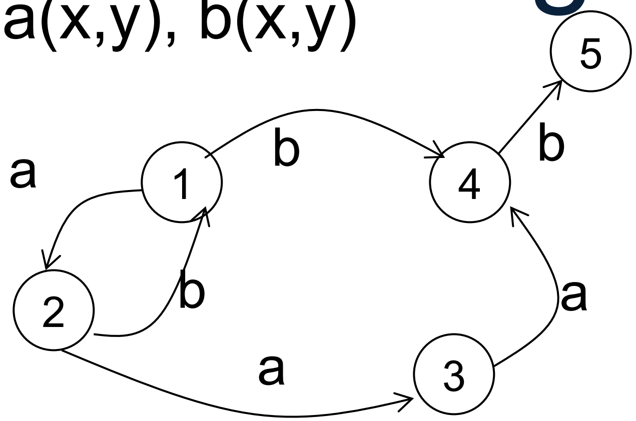
(a.a.b*)*.a

Automaton:



Regular Expressions

Labeled Graph:
a(x,y), b(x,y)

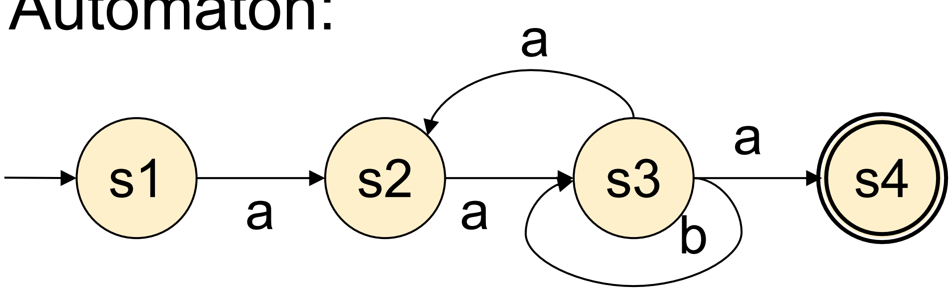


$T_i(x,y)$ = pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is s_i.

Find pairs of nodes connected by a path whos labels match

(a.a.b*)*.a

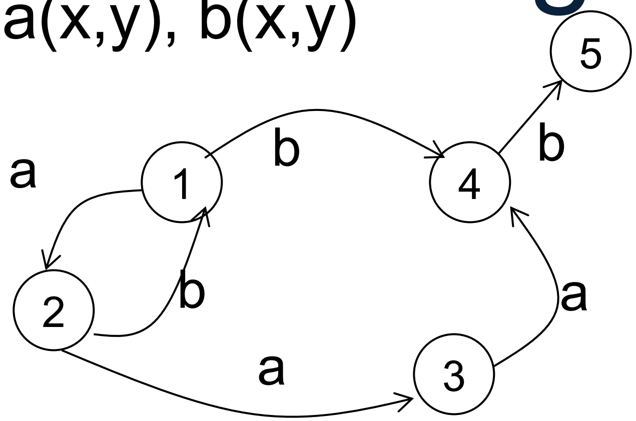
Automaton:



T2(x,y) :- a(x,y)

Regular Expressions

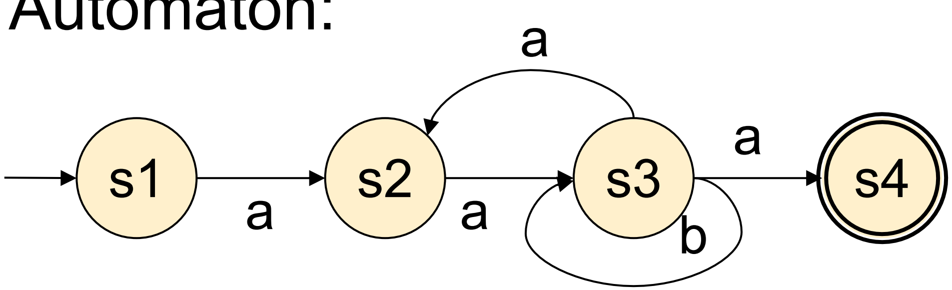
Labeled Graph:
a(x,y), b(x,y)



$T_i(x,y)$ = pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is s_i.

Find pairs of nodes connected by a path whos labels match
(a.a.b*)*.a

Automaton:

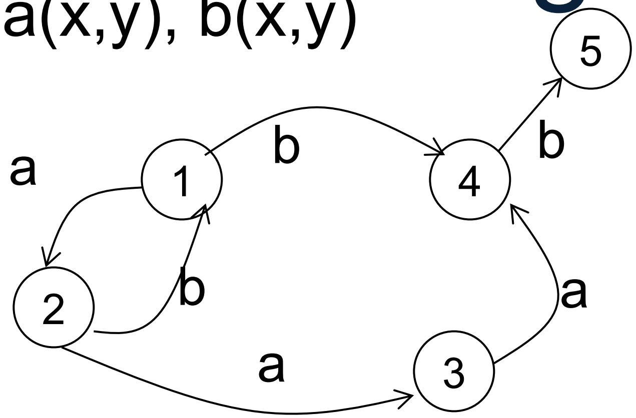


T2(x,y) :- a(x,y)

T2(x,y) :- T3(x,z),a(z,y)

Regular Expressions

Labeled Graph:
a(x,y), b(x,y)

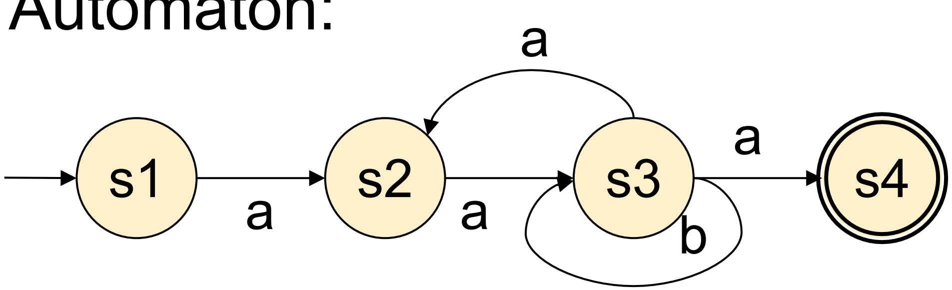


$T_i(x,y)$ = pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is s_i.

Find pairs of nodes connected by a path whos labels match

(a.a.b*)*.a

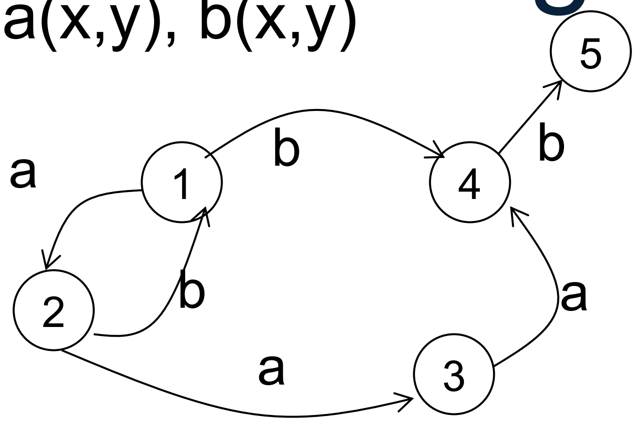
Automaton:



T2(x,y) :- a(x,y)
T2(x,y) :- T3(x,z),a(z,y)
T3(x,y) :- T2(x,z),a(z,y)
T3(x,y) :- T3(x,z),b(z,y)

Regular Expressions

Labeled Graph:
a(x,y), b(x,y)

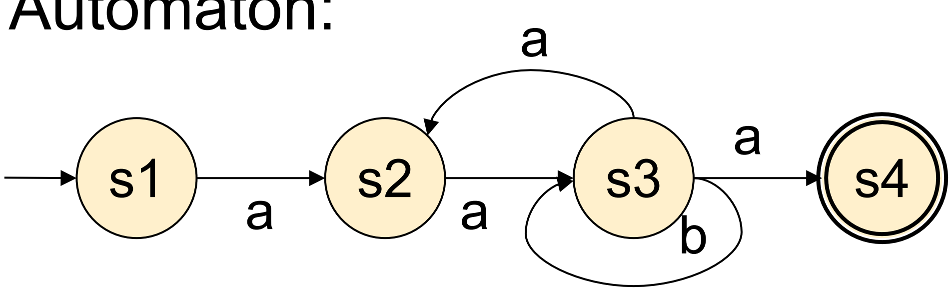


$T_i(x,y)$ = pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is s_i.

Find pairs of nodes connected by a path whos labels match

(a.a.b*)*.a

Automaton:



T2(x,y) :- a(x,y)

T2(x,y) :- T3(x,z),a(z,y)

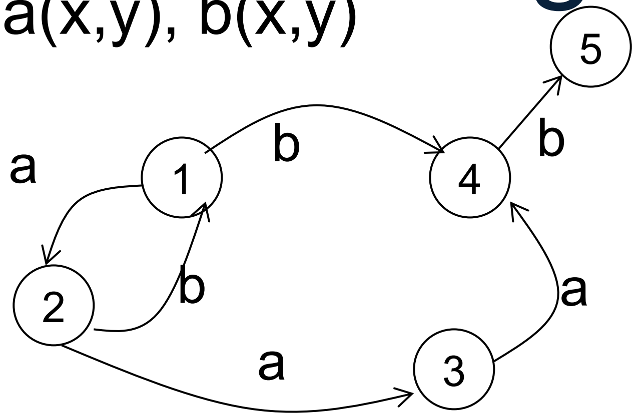
T3(x,y) :- T2(x,z),a(z,y)

T3(x,y) :- T3(x,z),b(z,y)

T4(x,y) :- T3(x,z),a(z,y)

Regular Expressions

Labeled Graph:
a(x,y), b(x,y)

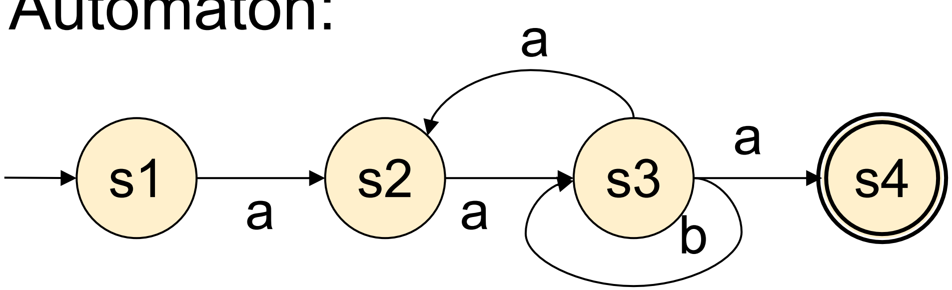


$T_i(x,y)$ = pairs of nodes connected by a paths whose labels match the language accepted by the automaton when the terminal state is s_i.

Find pairs of nodes connected by a path whos labels match

(a.a.b*)*.a

Automaton:



T2(x,y) :- a(x,y)

T2(x,y) :- T3(x,z),a(z,y)

T3(x,y) :- T2(x,z),a(z,y)

T3(x,y) :- T3(x,z),b(z,y)

T4(x,y) :- T3(x,z),a(z,y)

Answ(x,y) :- T4(x,y)

Recursion in SQL

- SQL supports a limited form of recursion by using Common Table Expression (CTE)

Recursion in SQL

T is called a CTE

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

```
with recursive T as
        (select * from R
         union
         select distinct R.x as x, T.y as y from R, T
         where R.y=T.x)
select * from T;
```

Recursion in SQL

R(X, Y)

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

```
with recursive T as
        (select * from R
         union
         select distinct R.x as x, T.y as y from R, T
         where R.y=T.x)
select * from T;
```

If you forgot ‘distinct’, then it diverges

Recursion in SQL

Clumsy, restricted, inefficient:

- Only a single IDB
- Only linear query
- Only this structure:
  - (non-recursive) union (recursive)
- Set or bag semantics (which diverges)

Outline

- Syntax
- Getting familiar with Datalog
- Semantics

Semantids of Datalog

Datalog has three equivalent ways to define its semantics. We consider two:

- Least fixpoint semantics
- Minimal model semantics

Immediate Consequence Operator

- The Immediate Consequence Operator (ICO) is a query that takes all EDBs, all IDBs, and computes a new state of the IDBs, by applying all rules

Immediate Consequence Operator

- The Immediate Consequence Operator (ICO) is a query that takes all EDBs, all IDBs, and computes a new state of the IDBs, by applying all rules

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

ICO

$$R(x, y) \cup \Pi_{xy}(R(x, z) \bowtie T(z, y))$$

Immediate Consequence Operator

- A function f is monotone if:

$$R_1 \subseteq R_1', R_2 \subseteq R_2', \ldots :$$
$$f(R_1, R_2, \ldots) \subseteq f(R_1', R_2', \ldots)$$

Immediate Consequence Operator

- A function f is monotone if:

$$R_1 \subseteq R'_1, R_2 \subseteq R'_2, \ldots :$$
$$f(R_1, R_2, \ldots) \subseteq f(R'_1, R'_2, \ldots)$$

- The ICO is a monotone function, because it uses only $\bowtie, \Pi, \sigma, \cup$
- The only non-monotone operator is -

1. Fixpoint Semantics

- x is a fixpoint of a function f if $f(x)=x$

1. Fixpoint Semantics

- x is a fixpoint of a function f if f(x)=x
- x is the least fixpoint if for any other fixpoint y, it holds that $x \subseteq y$

1. Fixpoint Semantics

- x is a fixpoint of a function f if f(x)=x
- x is the least fixpoint if for any other fixpoint y, it holds that $x \subseteq y$

- **Definition**. The semantics of a datalog program is the least fixpoint of the ICO

1. Fixpoint Semantics

- x is a fixpoint of a function f if f(x)=x
- x is the least fixpoint if for any other fixpoint y, it holds that $x \subseteq y$

- **Definition**. The semantics of a datalog program is the least fixpoint of the ICO

- Next: we prove that it exists.

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; t = 0
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    t = t+1
Until $IDB_t = IDB_{t-1}$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
  $IDB_{t+1} = ICO(EDB, IDB_t)$
  $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \dots$

Proof by induction. $\emptyset = IDB_0 \subseteq IDB_1$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
$IDB_{t+1} = ICO(EDB, IDB_t)$
$t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Proof by induction. $\emptyset = IDB_0 \subseteq IDB_1$

If $\mathrm{IDB}_{t-1} \subseteq \mathrm{IDB}_t$
then $\mathrm{IDB}_t = \mathrm{ICO}(\mathrm{IDB}_{t-1}) \subseteq \mathrm{ICO}(\mathrm{IDB}_t) = \mathrm{IDB}_{t+1}$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
$IDB_{t+1} = ICO(EDB, IDB_t)$
$t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $\mathrm{IDB}_{t_0} = \mathrm{IDB}_{t_0+1}$ Fixpoint!

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $IDB_{t_0} = IDB_{t_0+1}$ Fixpoint!

Proof. Because the number of possible tuples from EDBs is finite.

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
  $IDB_{t+1} = ICO(EDB, IDB_t)$
  $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $IDB_{t_0} = IDB_{t_0+1}$ Fixpoint!

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $IDB_{t_0} = IDB_{t_0+1}$ Fixpoint!

Fact: if IDB is any fixpoint, then $\forall t, IDB_t \subseteq IDB$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $\mathrm{IDB}_{t_0} = \mathrm{IDB}_{t_0+1}$ Fixpoint!

Fact: if IDB is any fixpoint, then $\forall t, \mathrm{IDB}_t \subseteq \mathrm{IDB}$

Proof. Induction on t. $\emptyset = \mathrm{IDB}_0 \subseteq \mathrm{IDB}$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $\mathrm{IDB}_{t_0} = \mathrm{IDB}_{t_0+1}$ Fixpoint!

Fact: if IDB is any fixpoint, then $\forall t, \mathrm{IDB}_t \subseteq \mathrm{IDB}$

Proof. Induction on t. $\emptyset = \mathrm{IDB}_0 \subseteq \mathrm{IDB}$

If $\mathrm{IDB}_t \subseteq \mathrm{IDB}$ then $\mathrm{IDB}_{t+1} = \mathrm{ICO}(\mathrm{IDB}_t) \subseteq \mathrm{ICO}(\mathrm{IDB}) = \mathrm{IDB}$

1. Fixpoint Semantics

Naïve evaluation algorithm

Start: $IDB_0 = \emptyset$; $t = 0$
Repeat:
    $IDB_{t+1} = ICO(EDB, IDB_t)$
    $t = t+1$
Until $IDB_t = IDB_{t-1}$

Fact: $\emptyset = IDB_0 \subseteq IDB_1 \subseteq IDB_2 \subseteq \ldots$

Fact: There exists t_0 such that $IDB_{t_0} = IDB_{t_0+1}$ Fixpoint!

Fact: if IDB is any fixpoint, then $\forall t, IDB_t \subseteq IDB$

Corollary. The Least Fixpoint of the ICO exists, and is computed by the Naïve Algorithm

Datalog and Logic

We need:

- A Quick review of Boolean Logic, FO
- Datalog as logical sentences

Boolean Logic

- Propositional symbols: p, q, r, …
- Boolean connectives: $\vee, \wedge, \neg, \Rightarrow$
- $(p \vee q) \wedge (q \vee \neg r) \wedge \neg(p \wedge q \vee r)$

Boolean Logic

- Propositional symbols: p, q, r, …
- Boolean connectives: $\vee, \wedge, \neg, \Rightarrow$
- $(p \vee q) \wedge (q \vee \neg r) \wedge \neg(p \wedge q \vee r)$
- Things to know:
  - De Morgan: $\neg(p \vee q) = \neg p \wedge \neg q$ and dual
  - Implications: $p \Rightarrow q \equiv \neg p \vee q$
  - Therefore: $\neg(p \Rightarrow q) \equiv p \wedge \neg q$

First Order Logic

- Relation symbols, variables, ops $\vee, \wedge, \neg, \Rightarrow, \forall, \exists$
- A sentence is a formula w/o free vars
- A model is a database that makes the formula true

First Order Logic

- Relation symbols, variables, ops $\vee, \wedge, \neg, \Rightarrow, \forall, \exists$
- A sentence is a formula w/o free vars
- A model is a database that makes the formula true
- What are the models of:
  - $\exists x \exists y \exists z (R(x, y) \wedge R(y, z))$
  - $\exists x \forall y \big(R(x, y)\big)$

First Order Logic

- Relation symbols, variables, ops $\vee, \wedge, \neg, \Rightarrow, \forall, \exists$
- A sentence is a formula w/o free vars
- A model is a database that makes the formula true
- What are the models of:
  - $\exists x \exists y \exists z (R(x, y) \wedge R(y, z))$
  - $\exists x \forall y \big(R(x, y)\big)$
- Things to know:
  - De Morgan $\neg \forall x(\ldots) \equiv \exists x \neg(\ldots)$

First Order Logic

- Relation symbols, variables, ops $\vee, \wedge, \neg, \Rightarrow, \forall, \exists$
- A sentence is a formula w/o free vars
- A model is a database that makes the formula true
- What are the models of:
  - $\exists x \exists y \exists z (R(x, y) \wedge R(y, z))$
  - $\exists x \forall y \big(R(x, y)\big)$
- Things to know:
  - De Morgan $\neg \forall x(\ldots) \equiv \exists x \neg(\ldots)$
  - $\forall x \forall y \big(R(x, y) \Rightarrow T(x)\big) \equiv \forall x \big(\exists y R(x, y) \Rightarrow T(x)\big)$

First Order Logic

- Relation symbols, variables, ops $\vee, \wedge, \neg, \Rightarrow, \forall, \exists$
- A sentence is a formula w/o free vars
- A model is a database that makes the formula true
- What are the models of:
  - $\exists x \exists y \exists z (R(x,y) \wedge R(y,z))$
  - $\exists x \forall y \big(R(x,y)\big)$
- Things to know:
  - De Morgan $\neg \forall x(\ldots) \equiv \exists x \neg(\ldots)$
  - $\forall x \forall y \big(R(x,y) \Rightarrow T(x)\big) \equiv \forall x \big(\exists y R(x,y) \Rightarrow T(x)\big)$
    Because $\forall x \forall y \big(\neg R(x,y) \vee T(x)\big) \equiv \forall x((\forall y \neg R(x,y)) \vee T(x))$

A datalog rule is a Sentence

Q1(y) :- Movie(x,y,z), z='1940'.

This is why
a non-head
variable is called
"existential"
variable

$$\forall x \forall y \forall z\ [(\text{Movie}(x,y,z) \text{ and } z=\text{'1940'}) \Rightarrow \text{Q1}(y)]$$

$$\forall y\ [(\exists x \exists z\ \text{Movie}(x,y,z) \text{ and } z=\text{'1940'}) \Rightarrow \text{Q1}(y)]$$

2. Minimal Model Semantics:

- Let Φ_P be the sentence that is the conjunction of all rules of the datalog program P
- A model of P is an IDB instance that is a model of Φ_P
- The minimal model of P is a model that is contained in all other models

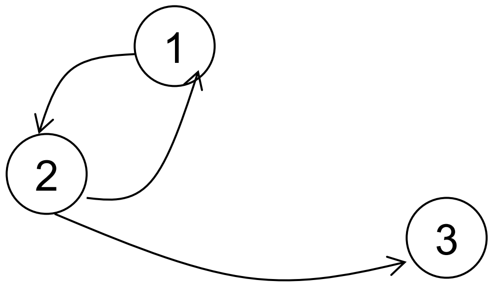
2. Minimal Model Semantics:

- **Definition**. The minimal model semantics of a program P is the minimal model of P

- **Theorem**. The minimal model exists and coincides with the least fixpoint of P

Example

R encodes a graph



R=

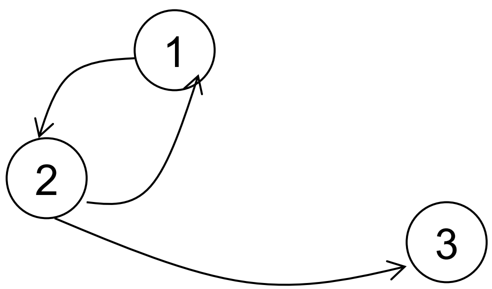
1	2
2	1
2	3

T(x,y) :- R(x,y)

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

Example

R encodes a graph



R=

1	2
2	1
2	3

T(x,y) :- R(x,y)

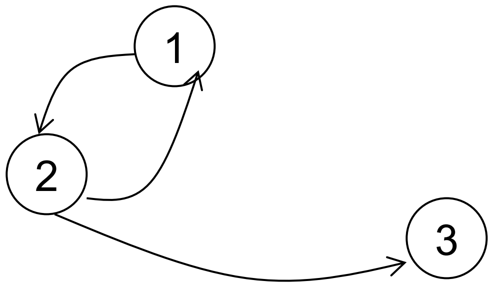
T(x,y) :- R(x,z), T(z,y)

1. Least fixpoint semantics:

Repeat $T_{t+1}(x,y) := R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$

Example

R encodes a graph



R=

1	2
2	1
2	3

```
T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)
```

1. Least fixpoint semantics:

Repeat $T_{t+1}(x,y) := R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$

2. Minimal model semantics

which one is a model? A minimal model?

2	1
2	3

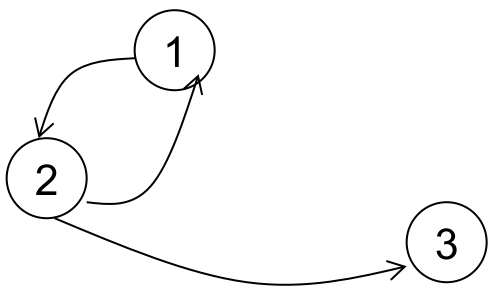
1	2
2	1
2	3

1	2
2	1
2	3
1	1
2	2
1	3
2	3

1	1
1	2
1	3
…	…
…	…
3	1
3	2
3	3

Example

R encodes a graph



R=

1	2
2	1
2	3

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

1. Least fixpoint semantics:

Repeat $T_{t+1}(x,y) \; := R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y))$

2. Minimal model semantics

which one is a model? A minimal model?

2	1
2	3

1	2
2	1
2	3

1	2
2	1
2	3
1	1
2	2
1	3
2	3

This is the minimal model

1	1
1	2
1	3
…	…
…	…
3	1
3	2
3	3

Datalog Semantics

- The fixpoint semantics tells us how to compute a datalog query
- The minimal model semantics is more declarative: only says what we get
- Analogous to SQL and RA

Next week: aggregates, negation, semi-naïve evaluation