

# CS294-248 Special Topics in Database Theory

## Unit 9: Datalog

Dan Suciu

University of Washington

# Announcement

- Project presentations: Thursday, Nov. 30th, 9:30am, Calvin 146
- By Monday: please add your tentative topic here:  
<https://tinyurl.com/43mdvwzy>
- You can change the topic later, as you wish.

# Outline

- Today: Basic Datalog
- Thursday: Extensions with Negation

# Review

# Motivation

- FO and its fragments cannot express simple, “easy” queries:
  - ▶ Transitive closure
  - ▶ Parity (“Is  $|R|$  even?”)
- Datalog: extends CQs with recursion

# Datalog Syntax

- A program  $P$  = set of rules.
- A rule is a CQ:  
 $H :- A_1 \wedge A_2 \wedge \dots$
- Extensional Database Predicates  
EDBs
- Intensional Database Predicates  
IDBs

$$\begin{aligned} T(X, Y) &:- E(X, Y) \\ T(X, Y) &:- T(X, Z) \wedge E(Z, X) \end{aligned}$$

# Pre-, Post-, and Fixpoints

**Poset** (partially ordered set)  $(P, \leq)$ .

We assume  $P$  has a minimal element  $\perp$ .

$f : P \rightarrow P$  is **monotone** if  $x \leq y \Rightarrow f(x) \leq f(y)$ .

$x$  is a **pre-fixpoint** if  $f(x) \leq x$

$x$  is a **post-fixpoint** if  $f(x) \geq x$

$x$  is a **fixpoint** if  $f(x) = x$ ;

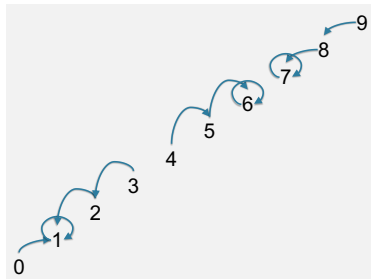
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints:

Post-fixpoints:

Fixpoints:





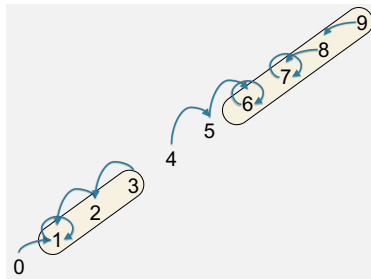
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints:

Fixpoints:



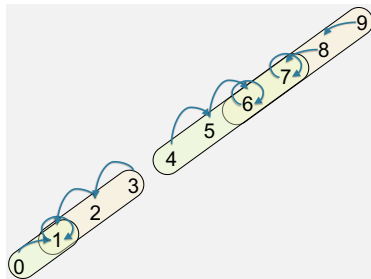
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints:



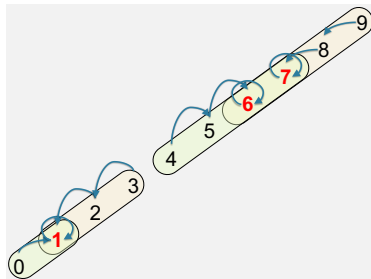
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



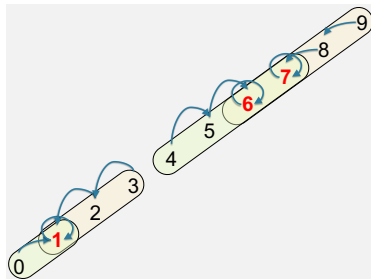
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

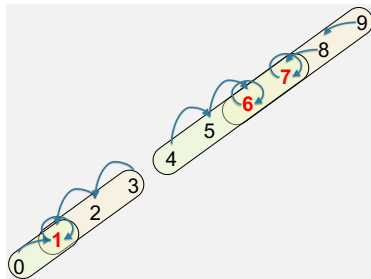
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq), f(x) = x + 1$ .

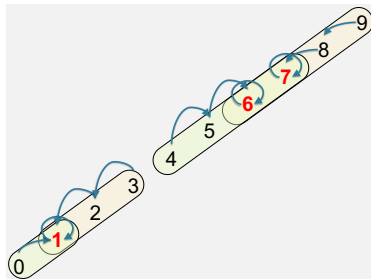
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq)$ ,  $f(x) = x + 1$ .

### Theorem

If the least *pre-fixpoint* exists then the least *fixpoint* exists and they are equal.

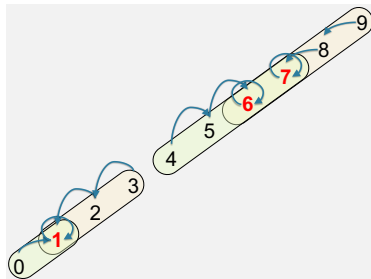
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq)$ ,  $f(x) = x + 1$ .

### Theorem

If the least *pre-fixpoint* exists then the least *fixpoint* exists and they are equal.

**Proof:**  $z \stackrel{\text{def}}{=} \text{least pre-fixpoint.}$

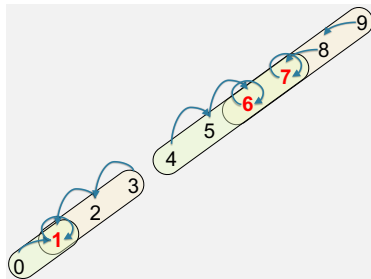
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq)$ ,  $f(x) = x + 1$ .

### Theorem

If the least *pre-fixpoint* exists then the least *fixpoint* exists and they are equal.

**Proof:**  $z \stackrel{\text{def}}{=} \text{least pre-fixpoint.}$

$f(z) \leq z$



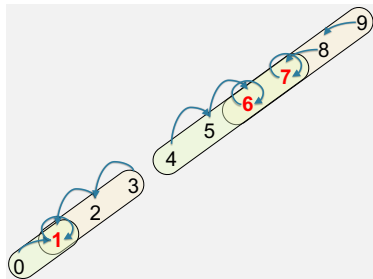
## Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq)$ ,  $f(x) = x + 1$ .

### Theorem

If the least *pre-fixpoint* exists then the least *fixpoint* exists and they are equal.

**Proof:**  $z \stackrel{\text{def}}{=} \text{least pre-fixpoint.}$

$$f(z) \leq z \quad f(f(z)) \leq f(z)$$

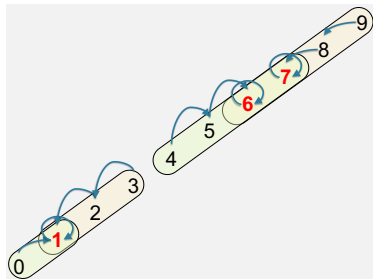
# Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq)$ ,  $f(x) = x + 1$ .

## Theorem

If the least *pre-fixpoint* exists then the least *fixpoint* exists and they are equal.

**Proof:**  $z \stackrel{\text{def}}{=} \text{least pre-fixpoint.}$

$f(z) \leq z$

$f(f(z)) \leq f(z)$

$f(z)$  pre-fixpoint

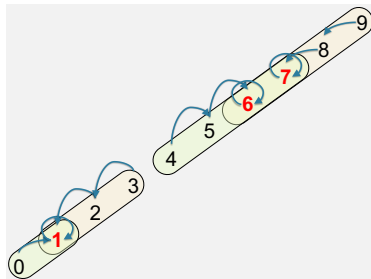
# Pre-, Post-, and Fixpoints

What are the pre-, post-, fixpoints?

Pre-fixpoints: 1,2,3,6,7,8,9

Post-fixpoints: 0,1,4,5,6,7

Fixpoints: 1,6,7



Does every monotone  $f$  have a fixpoint? A pre-fixpoint?

No:  $f : (\mathbb{N}, \leq) \rightarrow (\mathbb{N}, \leq)$ ,  $f(x) = x + 1$ .

## Theorem

If the least *pre-fixpoint* exists then the least *fixpoint* exists and they are equal.

**Proof:**  $z \stackrel{\text{def}}{=} \text{least pre-fixpoint.}$

$$f(z) \leq z$$

$$f(f(z)) \leq f(z)$$

$$f(z) \text{ pre-fixpoint}$$

$$f(z) = z$$

# Kleene's Sequence

$$\boxed{f^{(0)} \stackrel{\text{def}}{=} \perp \quad f^{(t+1)} \stackrel{\text{def}}{=} f(f^{(t)})} \quad f^{(0)} \leq f^{(1)} \leq f^{(2)} \leq \dots$$

## Fact

If  $z$  is any pre-fixpoint, then  $f^{(t)} \leq z$  for all  $t$ .

**Proof** by induction:  $\perp \leq z$  and  $f^{(t+1)} = f(f^{(t)}) \leq f(z) \leq z$ .

# Kleene's Sequence

$$\boxed{f^{(0)} \stackrel{\text{def}}{=} \perp \quad f^{(t+1)} \stackrel{\text{def}}{=} f(f^{(t)})} \quad f^{(0)} \leq f^{(1)} \leq f^{(2)} \leq \dots$$

## Fact

If  $z$  is any pre-fixpoint, then  $f^{(t)} \leq z$  for all  $t$ .

**Proof** by induction:  $\perp \leq z$  and  $f^{(t+1)} = f(f^{(t)}) \leq f(z) \leq z$ .

Is  $\bigvee_{t \geq 0} f^{(t)}$  the least fixpoint?

# Kleene's Sequence

$$\boxed{f^{(0)} \stackrel{\text{def}}{=} \perp \quad f^{(t+1)} \stackrel{\text{def}}{=} f(f^{(t)})} \quad f^{(0)} \leq f^{(1)} \leq f^{(2)} \leq \dots$$

## Fact

If  $z$  is any pre-fixpoint, then  $f^{(t)} \leq z$  for all  $t$ .

**Proof** by induction:  $\perp \leq z$  and  $f^{(t+1)} = f(f^{(t)}) \leq f(z) \leq z$ .

Is  $\bigvee_{t \geq 0} f^{(t)}$  the least fixpoint?

Not always. Two problems:

- $\bigvee_{t \geq 0} f^{(t)}$  may not exist.
- Even if it exists, we may have  $f(\bigvee_{t \geq 0} f^{(t)}) \neq \bigvee_{t \geq 0} f^{(t)}$ .

We will circumvent by requiring **finite rank**

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .



# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .

- What is the rank of  $(\{0, 1\}, \leq)$ ?

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .

- What is the rank of  $(\{0, 1\}, \leq)$ ?

$r = 1$ .

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .

- What is the rank of  $(\{0, 1\}, \leq)$ ?
- What is the rank of  $(\mathcal{P}(A), \subseteq)$ ?

$r = 1$ .

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .

- What is the rank of  $(\{0, 1\}, \leq)$ ?
- What is the rank of  $(\mathcal{P}(A), \subseteq)$ ?

$$r = 1.$$

$$r = |A|.$$

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .

- What is the rank of  $(\{0, 1\}, \leq)$ ?
- What is the rank of  $(\mathcal{P}(A), \subseteq)$ ?
- What is the rank of  $(P_1, \leq_1) \times (P_2, \leq_2)$ ?

$$r = 1.$$

$$r = |A|.$$

# The Rank of a Poset

[Stanley, 1999]

A **chain** of rank  $r$  is a sequence  $x_0 < x_1 < \dots < x_r$ .

The **rank** of a poset  $(P, \leq)$  is the largest  $k$  s.t. there exists a chain of length  $k$  in  $P$ .

Note: rank may be  $\infty$ .

- What is the rank of  $(\{0, 1\}, \leq)$ ?  $r = 1$ .
- What is the rank of  $(\mathcal{P}(A), \subseteq)$ ?  $r = |A|$ .
- What is the rank of  $(P_1, \leq_1) \times (P_2, \leq_2)$ ?  $r = r(P_1) + r(P_2)$ .

# Fixpoints in Posets of Finite Ranks

$$\boxed{f(0) \stackrel{\text{def}}{=} \perp \quad f(t+1) \stackrel{\text{def}}{=} f(f(t))} \quad f(0) < f(1) < f(2) < \dots \leq f(r) = f(r+1)$$

## Theorem

*If  $P$  has finite rank  $r$  then  $\text{lfp}(f) = f(r)$ .*

# Least Fixpoint Semantics of a Datalog Program $P$

$I$  = an EDB instance,  $A \stackrel{\text{def}}{=} \text{ADom}(I)$ .

If  $R$  has arity  $k$ , then an instance is  $R \in \mathcal{P}(A^k)$ .



# Least Fixpoint Semantics of a Datalog Program $P$

$I$  = an EDB instance,  $A \stackrel{\text{def}}{=} \text{ADom}(I)$ .

If  $R$  has arity  $k$ , then an instance is  $R \in \mathcal{P}(A^k)$ .

If IDB predicates have arities  $k_1, k_2, \dots$

then an IDB instance is  $J \in \mathcal{P}(A^{k_1}) \times \mathcal{P}(A^{k_2}) \times \dots$

# Least Fixpoint Semantics of a Datalog Program $P$

$I$  = an EDB instance,  $A \stackrel{\text{def}}{=} \text{ADom}(I)$ .

If  $R$  has arity  $k$ , then an instance is  $R \in \mathcal{P}(A^k)$ .

If IDB predicates have arities  $k_1, k_2, \dots$

then an IDB instance is  $J \in \mathcal{P}(A^{k_1}) \times \mathcal{P}(A^{k_2}) \times \dots$

Immediate Consequence Operator:

$$T_P : \mathcal{P}(A^{k_1}) \times \mathcal{P}(A^{k_2}) \times \dots \rightarrow \mathcal{P}(A^{k_1}) \times \mathcal{P}(A^{k_2}) \times \dots$$

The semantics of the datalog program  $P$  is  $\text{lfp}(T_P)$ .

## Naive Evaluation Algorithm

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $J^{(t+1)} := T_P(J^{(t)})$   
  if  $J^{(t+1)} = J^{(t)}$  break
```

Notice:  $J^{(0)} \subseteq J^{(1)} \subseteq \dots$  is Kleene's sequence.

### Theorem

*The Naive Algorithm takes  $O(\text{ADom}(I)^k)$  iterations, where  $I$  is the EDB instance and  $k$  is the largest arity of any IDB.*

Data complexity is in PTIME.

# Examples in Datalog

# Overview

- We have seen **Transitive Closure**. Can we write something different?
- Regular expressions, CFGs.
- Same generation.
- AND/OR reachability.

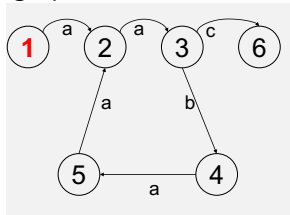
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

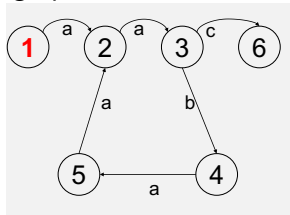
EDB graph:



## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



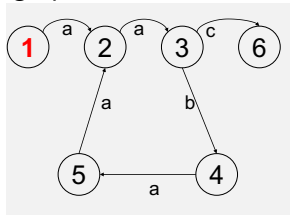
Answer: **1**, 4, 6.



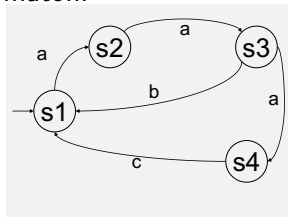
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:

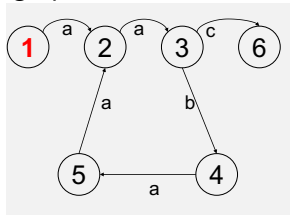


Answer: **1**, 4, 6.

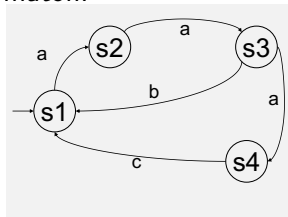
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:



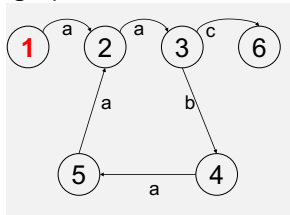
Answer: **1**, 4, 6.

Q1(**1**) :-

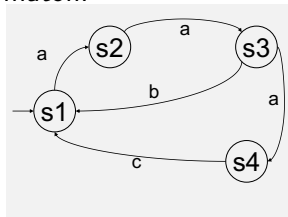
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:



Answer: **1**, 4, 6.

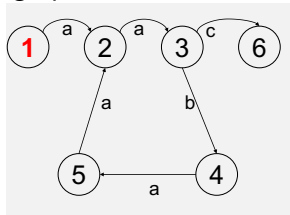
$Q1(\mathbf{1}) :-$

$Q2(Y) :- Q1(X) \wedge E(X, Y, 'a')$

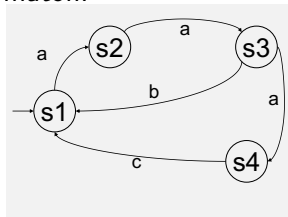
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:



Answer: **1**, 4, 6.

$Q1(\mathbf{1}) :-$

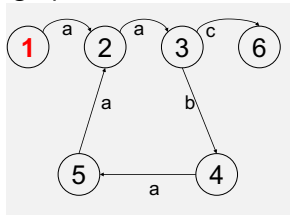
$Q3(Y) :- Q2(X) \wedge E(X, Y, 'a')$

$Q2(Y) :- Q1(X) \wedge E(X, Y, 'a')$

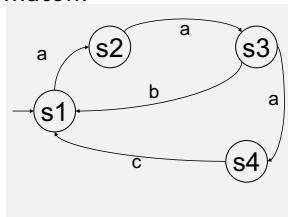
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:



Answer: **1**, 4, 6.

$Q1(\mathbf{1}) :-$

$Q2(Y) :- Q1(X) \wedge E(X, Y, 'a')$

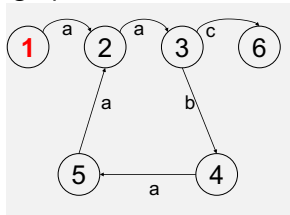
$Q3(Y) :- Q2(X) \wedge E(X, Y, 'a')$

$Q1(Y) :- Q3(X) \wedge E(X, Y, 'b')$

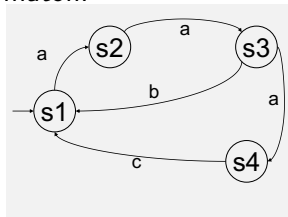
## Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:



Answer: **1**, 4, 6.

$Q1(\mathbf{1}) :-$

$Q3(Y) :- Q2(X) \wedge E(X, Y, 'a')$

$Q4(Y) :- Q3(X) \wedge E(X, Y, 'a')$

$Q2(Y) :- Q1(X) \wedge E(X, Y, 'a')$

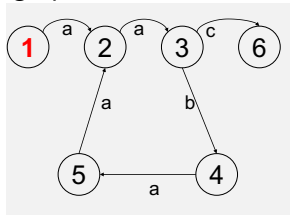
$Q1(Y) :- Q3(X) \wedge E(X, Y, 'b')$

$Q1(Y) :- Q4(X) \wedge E(X, Y, 'c')$

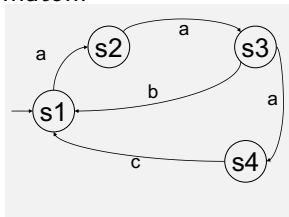
# Regular Expressions

Find nodes reachable from **1** with a path labeled with  $(aab|aaac)^*$

EDB graph:



Automaton:



Answer: **1**, 4, 6.

$Q1(\mathbf{1}) :-$

$Q3(Y) :- Q2(X) \wedge E(X, Y, 'a')$

$Q4(Y) :- Q3(X) \wedge E(X, Y, 'a')$

$Q2(Y) :- Q1(X) \wedge E(X, Y, 'a')$

$Q1(Y) :- Q3(X) \wedge E(X, Y, 'b')$

$Q1(Y) :- Q4(X) \wedge E(X, Y, 'c')$

Answer(X) :- Q1(X)

# Discussion

- Automaton need not be deterministic.



## Discussion

- Automaton need not be deterministic.
- Also CFG. E.g language of parentheses:  $S \rightarrow \epsilon | SS | aSb$ . **How?**

## Discussion

- Automaton need not be deterministic.
- Also CFG. E.g language of parentheses:  $S \rightarrow \epsilon | SS | aSb$ . **How?**

$S(X, X) :- \text{Node}(X)$

$S(X, Y) :- S(X, Z) \wedge S(Z, Y)$

$S(X, Y) :- E(X, U, 'a') \wedge S(U, V) \wedge E(V, Y, 'b')$

## Discussion

- Automaton need not be deterministic.
- Also CFG. E.g language of parentheses:  $S \rightarrow \epsilon | SS | aSb$ . **How?**

$$S(X, X) :- \text{Node}(X)$$

$$S(X, Y) :- S(X, Z) \wedge S(Z, Y)$$

$$S(X, Y) :- E(X, U, 'a') \wedge S(U, V) \wedge E(V, Y, 'b')$$

- Exercise<sup>\*\*</sup>: non-CFG, e.g. the language  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ .

## Discussion

- Automaton need not be deterministic.
- Also CFG. E.g language of parentheses:  $S \rightarrow \epsilon | SS | aSb$ . **How?**

$$S(X, X) :- \text{Node}(X)$$

$$S(X, Y) :- S(X, Z) \wedge S(Z, Y)$$

$$S(X, Y) :- E(X, U, 'a') \wedge S(U, V) \wedge E(V, Y, 'b')$$

- Exercise<sup>\*\*</sup>: non-CFG, e.g. the language  $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ .  
(won't discuss in class)

$$T(X, X, Y, Y, Z, Z) :- \text{Node}(X) \wedge \text{Node}(Y) \wedge \text{Node}(Z)$$

$$T(X_1, X_2, Y_1, Y_2, Z_1, Z_2) :- T(X_1, X_3, Y_1, Y_3, Z_1, Z_3) \wedge E(X_3, X_2, 'a') \\ \wedge E(Y_3, Y_2, 'b') \wedge E(Z_3, Z_2, 'c')$$

$$\text{Answer}(X, Y) :- T(X, U, V, U, V, Y)$$

## Same Generation

$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

## Same Generation

$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

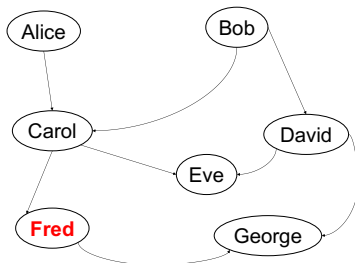
Find people at the same generation  
with **Fred**.

## Same Generation

$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

Find people at the same generation  
with **Fred**.

EDB graph:

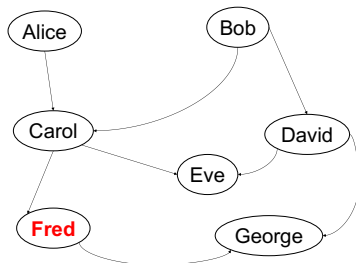


## Same Generation

$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

Find people at the same generation  
with **Fred**.

EDB graph:



Answer: **Fred**, Eve, George



## Same Generation

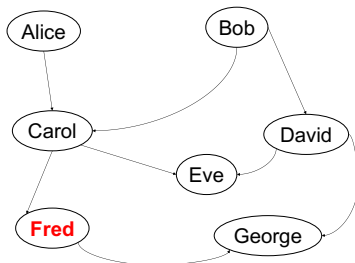
$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

Find people at the same generation  
with **Fred**.

$SG(X, X) :- \text{Person}(X)$

$SG(X, Y) :-$

EDB graph:



Answer: **Fred**, Eve, George

## Same Generation

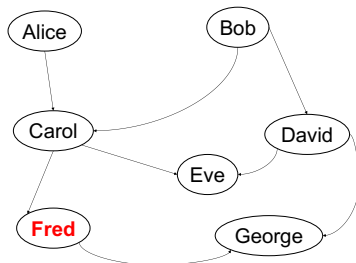
$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

Find people at the same generation  
with **Fred**.

$SG(X, X) :- \text{Person}(X)$

$SG(X, Y) :- \text{????}$

EDB graph:



Answer: **Fred**, Eve, George

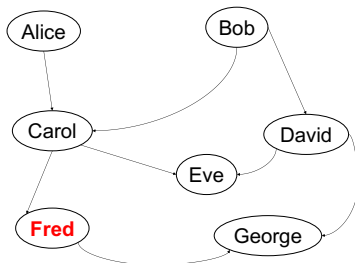
## Same Generation

$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

Find people at the same generation  
with **Fred**.

$$SG(X, X) \text{ :- Person}(X)$$
$$SG(X, Y) \text{ :- } SG(U, V) \wedge E(U, X) \wedge E(V, Y)$$

EDB graph:



Answer: **Fred**, Eve, George

## Same Generation

$x, y$  are at the same generation  
if they have a common ancestor  $z$   
at the same distance.

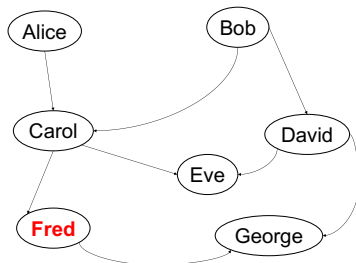
Find people at the same generation  
with **Fred**.

$$SG(X, X) :- \text{Person}(X)$$

$$SG(X, Y) :- SG(U, V) \wedge E(U, X) \wedge E(V, Y)$$

$$\text{Answer}(X) :- SG('Fred', X)$$

EDB graph:



Answer: **Fred**, Eve, George

# Discussion

- The examples so far are still just transitive at their essence! **why?**
- Recall that transitive closure is in  $NLOGSPACE$ . The next example goes beyond  $NLOGSPACE$ .

# AND/OR-Graph Accessibility

OR-nodes: unlimited AND-children

AND-nodes: two OR-children

# AND/OR-Graph Accessibility

OR-nodes: unlimited AND-children

AND-nodes: two OR-children

$T(X, Y, Z)$ :

OR-node  $X$  has AND-child with children  $Y, Z$ .

# AND/OR-Graph Accessibility

OR-nodes: unlimited AND-children

AND-nodes: two OR-children

$T(X, Y, Z)$ :

OR-node  $X$  has AND-child with children  $Y, Z$ .

Find all accessible nodes from  $a$



# AND/OR-Graph Accessibility

OR-nodes: unlimited AND-children

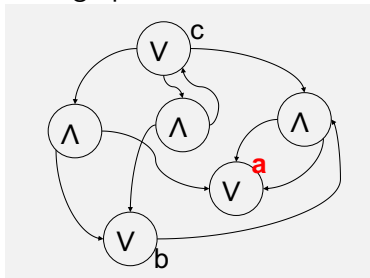
AND-nodes: two OR-children

$T(X, Y, Z)$ :

OR-node  $X$  has AND-child with children  $Y, Z$ .

Find all accessible nodes from  $a$

EDB graph:



$T$

$X$	$Y$	$Z$
$c$	$a$	$b$
$c$	$b$	$c$
$c$	$a$	$a$
$b$	$a$	$a$

## AND/OR-Graph Accessibility

OR-nodes: unlimited AND-children

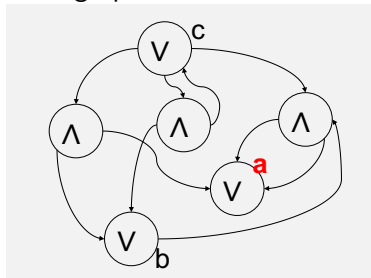
AND-nodes: two OR-children

$T(X, Y, Z)$ :

OR-node  $X$  has AND-child with children  $Y, Z$ .

Find all accessible nodes from  $a$

EDB graph:



$T$

$X$	$Y$	$Z$
$c$	$a$	$b$
$c$	$b$	$c$
$c$	$a$	$a$
$b$	$a$	$a$

Answer:  $a, b, c$ .

# AND/OR-Graph Accessibility

OR-nodes: unlimited AND-children

AND-nodes: two OR-children

$T(X, Y, Z)$ :

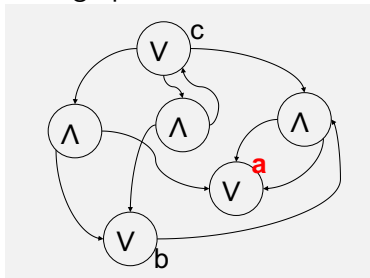
OR-node  $X$  has AND-child with children  $Y, Z$ .

Find all accessible nodes from  $a$

$A(a) :-$

$A(X) :- T(X, Y, Z) \wedge A(Y) \wedge A(Z)$

EDB graph:



$T$

$X$	$Y$	$Z$
$c$	$a$	$b$
$c$	$b$	$c$
$c$	$a$	$a$
$b$	$a$	$a$

Answer:  $a, b, c$ .

# Discussion

- AGAP is PTIME-complete. Recall:  $\text{NLOGSPACE} \subseteq \text{PTIME}$  and inclusion is conjecture to be strict.
- It follows that datalog can express strictly more than transitive closure.
- The data complexity of datalog is in PTIME.
- Limitation of “pure” datalog: **monotone queries only**.
- Montone queries have huge potential for optimizations (next).

# Optimizing Monotone Datalog

# Outline

- Semi-naive evaluation.
- Asynchronous execution: also discuss [grounding](#).
- Will not discuss: Magic Set optimization

# Naive, and Semi-naive

## Naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $J^{(t+1)} := T_P(J^{(t)})$   
  if  $J^{(t+1)} = J^{(t)}$   
    break
```

# Naive, and Semi-naive

## Naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $J^{(t+1)} := T_P(J^{(t)})$ 
  if  $J^{(t+1)} = J^{(t)}$ 
    break
  
```

## Semi-naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break
  
```



# Naive, and Semi-naive

## Naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $J^{(t+1)} := T_P(J^{(t)})$   
  if  $J^{(t+1)} = J^{(t)}$   
    break
```

## Semi-naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$   
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$   
  if  $\Delta^{(t)} = \emptyset$  break
```

## w/ incremental computation

```
 $J^{(0)} := \emptyset, \Delta^{(0)} := T_P(\emptyset)$   
for  $t = 1, \infty$   
   $\Delta^{(t)} := T_P(J^{(t-1)} \cup \Delta^{(t-1)}) - J^{(t)}$   
            $= \Delta T_P(J^{(t-1)}, \Delta^{(t-1)}) - J^{(t)}$   
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$   
  if  $\Delta^{(t)} = \emptyset$  break
```

# Naive, and Semi-naive

## Naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $J^{(t+1)} := T_P(J^{(t)})$   
  if  $J^{(t+1)} = J^{(t)}$   
    break
```

## Semi-naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$   
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$   
  if  $\Delta^{(t)} = \emptyset$  break
```

## w/ incremental computation

```
 $J^{(0)} := \emptyset, \Delta^{(0)} := T_P(\emptyset)$   
for  $t = 1, \infty$   
   $\Delta^{(t)} := T_P(J^{(t-1)} \cup \Delta^{(t-1)}) - J^{(t)}$   
            $= \Delta T_P(J^{(t-1)}, \Delta^{(t-1)}) - J^{(t)}$   
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$   
  if  $\Delta^{(t)} = \emptyset$  break
```

Transitive Closure:

$T(X, Y) :- E(X, Y)$

$T(X, Y) :- T(X, Z) \wedge E(Z, Y)$

# Naive, and Semi-naive

## Naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $J^{(t+1)} := T_P(J^{(t)})$   
  if  $J^{(t+1)} = J^{(t)}$   
    break
```

## Semi-naive

```
 $J^{(0)} := \emptyset$   
for  $t = 0, \infty$   
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$   
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$   
  if  $\Delta^{(t)} = \emptyset$  break
```

## w/ incremental computation

```
 $J^{(0)} := \emptyset, \Delta^{(0)} := T_P(\emptyset)$   
for  $t = 1, \infty$   
   $\Delta^{(t)} := T_P(J^{(t-1)} \cup \Delta^{(t-1)}) - J^{(t)}$   
            $= \Delta T_P(J^{(t-1)}, \Delta^{(t-1)}) - J^{(t)}$   
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$   
  if  $\Delta^{(t)} = \emptyset$  break
```

Transitive Closure:

$$T^{(0)}(X, Y) := \text{false}, \Delta^{(0)}(X, Y) := E(X, Y)$$

$$T(X, Y) :- E(X, Y)$$

$$T(X, Y) :- T(X, Z) \wedge E(Z, Y)$$

# Naive, and Semi-naive

## Naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $J^{(t+1)} := T_P(J^{(t)})$ 
  if  $J^{(t+1)} = J^{(t)}$ 
    break

```

## Semi-naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break

```

## w/ incremental computation

```

 $J^{(0)} := \emptyset, \Delta^{(0)} := T_P(\emptyset)$ 
for  $t = 1, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t-1)} \cup \Delta^{(t-1)}) - J^{(t)}$ 
   $\quad = \Delta T_P(J^{(t-1)}, \Delta^{(t-1)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break

```

Transitive Closure:

$$T^{(0)}(X, Y) := \text{false}, \Delta^{(0)}(X, Y) := E(X, Y)$$

for  $t = 1, \infty$

$$T(X, Y) :- E(X, Y)$$

$$\Delta^{(t)}(X, Y) :=$$

$$T(X, Y) :- T(X, Z) \wedge E(Z, Y)$$

# Naive, and Semi-naive

## Naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $J^{(t+1)} := T_P(J^{(t)})$ 
  if  $J^{(t+1)} = J^{(t)}$ 
    break
  
```

## Semi-naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break
  
```

## w/ incremental computation

```

 $J^{(0)} := \emptyset, \Delta^{(0)} := T_P(\emptyset)$ 
for  $t = 1, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t-1)} \cup \Delta^{(t-1)}) - J^{(t)}$ 
   $\quad = \Delta T_P(J^{(t-1)}, \Delta^{(t-1)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break
  
```

Transitive Closure:

$$T^{(0)}(X, Y) := \text{false}, \Delta^{(0)}(X, Y) := E(X, Y)$$

for  $t = 1, \infty$

$$T(X, Y) :- E(X, Y)$$

$$\Delta^{(t)}(X, Y) := \Delta^{(t-1)}(X, Z) \wedge E(Z, Y) \wedge \neg T^{(t)}(X, Y)$$

$$T(X, Y) :- T(X, Z) \wedge E(Z, Y)$$

# Naive, and Semi-naive

## Naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $J^{(t+1)} := T_P(J^{(t)})$ 
  if  $J^{(t+1)} = J^{(t)}$ 
    break

```

## Semi-naive

```

 $J^{(0)} := \emptyset$ 
for  $t = 0, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break

```

## w/ incremental computation

```

 $J^{(0)} := \emptyset, \Delta^{(0)} := T_P(\emptyset)$ 
for  $t = 1, \infty$ 
   $\Delta^{(t)} := T_P(J^{(t-1)} \cup \Delta^{(t-1)}) - J^{(t)}$ 
   $\quad = \Delta T_P(J^{(t-1)}, \Delta^{(t-1)}) - J^{(t)}$ 
   $J^{(t+1)} := J^{(t)} \cup \Delta^{(t)}$ 
  if  $\Delta^{(t)} = \emptyset$  break

```

Transitive Closure:

```

 $T^{(0)}(X, Y) := \text{false}, \Delta^{(0)}(X, Y) := E(X, Y)$ 
for  $t = 1, \infty$ 
   $\Delta^{(t)}(X, Y) := \Delta^{(t-1)}(X, Z) \wedge E(Z, Y) \wedge \neg T^{(t)}(X, Y)$ 
   $T^{(t+1)}(X, Y) := T^{(t)}(X, Y) \vee \Delta^{(t)}(X, Y)$ 
  if  $\Delta^{(t)} = \emptyset$  break

```

$T(X, Y) :- E(X, Y)$

$T(X, Y) :- T(X, Z) \wedge E(Z, Y)$

## Discussion

Semi-naive is implemented by virtually all datalog systems.

Non-linear datalog rules have more complex delta-queries:

- Exponential number of queries:

$$(A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie (C \cup \Delta C)$$

## Discussion

Semi-naive is implemented by virtually all datalog systems.

Non-linear datalog rules have more complex delta-queries:

- Exponential number of queries:

$$(A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie (C \cup \Delta C) = (A \bowtie B \bowtie C) \cup \\ \cup (\Delta A \bowtie B \bowtie C) \cup \dots \cup (\Delta A \bowtie \Delta B \bowtie C) \cup \dots \cup (\Delta A \bowtie \Delta B \bowtie \Delta C)$$



## Discussion

Semi-naive is implemented by virtually all datalog systems.

Non-linear datalog rules have more complex delta-queries:

- Exponential number of queries:

$$(A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie (C \cup \Delta C) = (A \bowtie B \bowtie C) \cup \\ \cup (\Delta A \bowtie B \bowtie C) \cup \dots \cup (\Delta A \bowtie \Delta B \bowtie C) \cup \dots \cup (\Delta A \bowtie \Delta B \bowtie \Delta C)$$

- Mix of old/new tables (issue: new tables are bigger):

$$(A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie (C \cup \Delta C)$$

# Discussion

Semi-naive is implemented by virtually all datalog systems.

Non-linear datalog rules have more complex delta-queries:

- Exponential number of queries:

$$(A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie (C \cup \Delta C) = (A \bowtie B \bowtie C) \cup \\ \cup (\Delta A \bowtie B \bowtie C) \cup \dots \cup (\Delta A \bowtie \Delta B \bowtie C) \cup \dots \cup (\Delta A \bowtie \Delta B \bowtie \Delta C)$$

- Mix of old/new tables (issue: new tables are bigger):

$$(A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie (C \cup \Delta C) = (A \bowtie B \bowtie C) \cup \\ \cup (\Delta A \bowtie B \bowtie C) \cup ((A \cup \Delta A) \bowtie \Delta B \bowtie C) \cup ((A \cup \Delta A) \bowtie (B \cup \Delta B) \bowtie \Delta C)$$

# Asynchronous Execution

- (Semi-) naive is synchronous:      apply **all** rules to **all** tuples.
- Asynchronous execution:      apply **some** rules to **some** tuples.
- Simple principle: fair computation of a fixpoint.

# Asynchronous Sequence

Posets  $(P_1, \leq), (P_2, \leq)$ , finite ranks,  $f : P_1 \times P_2 \rightarrow P_1, g : P_1 \times P_2 \rightarrow P_2$ .

**Goal** compute  $\text{lfp}(f, g)$ :

$$(f(x, y), g(x, y)) = (x, y)$$

# Asynchronous Sequence

Posets  $(P_1, \leq), (P_2, \leq)$ , finite ranks,  $f : P_1 \times P_2 \rightarrow P_1, g : P_1 \times P_2 \rightarrow P_2$ .

**Goal** compute  $\text{lfp}(f, g)$ :

$$(f(x, y), g(x, y)) = (x, y)$$

**Kleene's sequence:**

$$(x^{(0)}, y^{(0)}) \stackrel{\text{def}}{=} (\perp, \perp)$$

$$(x^{(t+1)}, y^{(t+1)}) \stackrel{\text{def}}{=} (f(x^{(t)}, y^{(t)}), g(x^{(t)}, y^{(t)}))$$

Every step is an **fg-step**.

# Asynchronous Sequence

Posets  $(P_1, \leq), (P_2, \leq)$ , finite ranks,  $f : P_1 \times P_2 \rightarrow P_1$ ,  $g : P_1 \times P_2 \rightarrow P_2$ .

**Goal** compute  $\text{lfp}(f, g)$ :

$$(f(x, y), g(x, y)) = (x, y)$$

**Kleene's sequence:**

$$\begin{aligned} (x^{(0)}, y^{(0)}) &\stackrel{\text{def}}{=} (\perp, \perp) \\ (x^{(t+1)}, y^{(t+1)}) &\stackrel{\text{def}}{=} \\ &\quad (f(x^{(t)}, y^{(t)}), g(x^{(t)}, y^{(t)})) \end{aligned}$$

Every step is an **fg-step**.

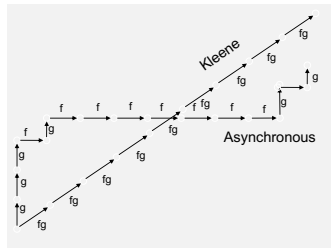
**Asynchronous sequence:**

$$\begin{aligned} (u^{(0)}, v^{(0)}) &\stackrel{\text{def}}{=} (\perp, \perp) \\ (u^{(k+1)}, v^{(k+1)}) &\stackrel{\text{def}}{=} \\ &\quad \begin{cases} (f(u^{(k)}, v^{(k)}), g(u^{(k)}, v^{(k)})) & \text{or} \\ (f(u^{(k)}, v^{(k)}), v^{(k)}) & \text{or} \\ (u^{(k)}, g(u^{(k)}, v^{(k)})) \end{cases} \end{aligned}$$

**fg-step**, or **f-step**, or **g-step**.

# Fair Computation of a Fixpoint

**Fact 1:** for any pre-fixpoint  $(x, y)$  of  $(f, g)$ ,  
 $(u^{(k)}, v^{(k)}) \leq (x, y)$ .

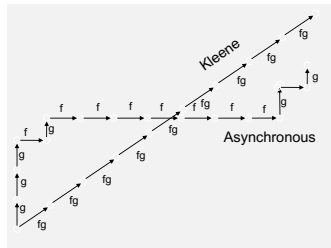


## Fair Computation of a Fixpoint

**Fact 1:** for any pre-fixpoint  $(x, y)$  of  $(f, g)$ ,  
 $(u^{(k)}, v^{(k)}) \leq (x, y)$ .

Sequence is **fair** if:  $\forall k \exists m > k \exists n > k$  s.t:

- $m$  is an  $f$ -step or  $fg$ -step, and
- $n$  is a  $g$ -step or  $fg$ -step.



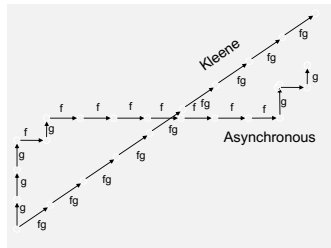


## Fair Computation of a Fixpoint

**Fact 1:** for any pre-fixpoint  $(x, y)$  of  $(f, g)$ ,  
 $(u^{(k)}, v^{(k)}) \leq (x, y)$ .

Sequence is **fair** if:  $\forall k \exists m > k \exists n > k$  s.t:

- $m$  is an  $f$ -step or  $fg$ -step, and
- $n$  is a  $g$ -step or  $fg$ -step.



**Fact 2:** If the sequence is fair, then  $\exists k$  s.t.  $(u^{(k)}, v^{(k)}) = \text{lfp}(f, g)$ .

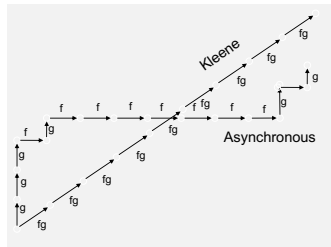


## Fair Computation of a Fixpoint

**Fact 1:** for any pre-fixpoint  $(x, y)$  of  $(f, g)$ ,  
 $(u^{(k)}, v^{(k)}) \leq (x, y)$ .

Sequence is **fair** if:  $\forall k \exists m > k \exists n > k$  s.t:

- $m$  is an  $f$ -step or  $fg$ -step, and
- $n$  is a  $g$ -step or  $fg$ -step.



**Fact 2:** If the sequence is fair, then  $\exists k$  s.t.  $(u^{(k)}, v^{(k)}) = \text{lfp}(f, g)$ .

**Proof** suffices to prove:  $\forall t \exists k, (x^{(t)}, y^{(t)}) \leq (u^{(k)}, v^{(k)})$ .

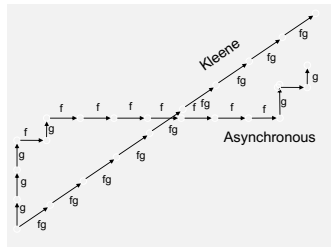
$$(x^{(t+1)}, y^{(t+1)}) = (f(x^{(t)}, y^{(t)}), g(x^{(t)}, y^{(t)})) \leq (f(u^{(k)}, v^{(k)}), g(u^{(k)}, v^{(k)}))$$

## Fair Computation of a Fixpoint

**Fact 1:** for any pre-fixpoint  $(x, y)$  of  $(f, g)$ ,  
 $(u^{(k)}, v^{(k)}) \leq (x, y)$ .

Sequence is **fair** if:  $\forall k \exists m > k \exists n > k$  s.t:

- $m$  is an  $f$ -step or  $fg$ -step, and
- $n$  is a  $g$ -step or  $fg$ -step.



**Fact 2:** If the sequence is fair, then  $\exists k$  s.t.  $(u^{(k)}, v^{(k)}) = \text{lfp}(f, g)$ .

**Proof** suffices to prove:  $\forall t \exists k, (x^{(t)}, y^{(t)}) \leq (u^{(k)}, v^{(k)})$ .

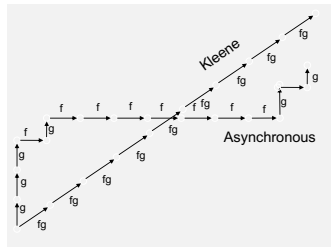
$$\begin{aligned} (x^{(t+1)}, y^{(t+1)}) &= (f(x^{(t)}, y^{(t)}), g(x^{(t)}, y^{(t)})) \leq (f(u^{(k)}, v^{(k)}), g(u^{(k)}, v^{(k)})) \\ &\leq (f(u^{(m)}, v^{(m)}), g(u^{(n)}, v^{(n)})) \end{aligned}$$

## Fair Computation of a Fixpoint

**Fact 1:** for any pre-fixpoint  $(x, y)$  of  $(f, g)$ ,  
 $(u^{(k)}, v^{(k)}) \leq (x, y)$ .

Sequence is **fair** if:  $\forall k \exists m > k \exists n > k$  s.t:

- $m$  is an  $f$ -step or  $fg$ -step, and
- $n$  is a  $g$ -step or  $fg$ -step.



**Fact 2:** If the sequence is fair, then  $\exists k$  s.t.  $(u^{(k)}, v^{(k)}) = \text{lfp}(f, g)$ .

**Proof** suffices to prove:  $\forall t \exists k, (x^{(t)}, y^{(t)}) \leq (u^{(k)}, v^{(k)})$ .

$$\begin{aligned}
 (x^{(t+1)}, y^{(t+1)}) &= (f(x^{(t)}, y^{(t)}), g(x^{(t)}, y^{(t)})) \leq (f(u^{(k)}, v^{(k)}), g(u^{(k)}, v^{(k)})) \\
 &\leq (f(u^{(m)}, v^{(m)}), g(u^{(n)}, v^{(n)})) = (u^{(m+1)}, v^{(n+1)}) \leq (u^{(p)}, v^{(p)}) \\
 &\qquad\qquad\qquad \text{where } p = \max(m, n) + 1.
 \end{aligned}$$

## Discussion

- Kleene's sequence has rank  $\text{rank}(P_1) + \text{rank}(P_2)$ ; the asynchronous sequence could be as long as  $\text{rank}(P_1) \times \text{rank}(P_2)$

- Application: nested recursion

$$\begin{aligned} \text{lfp}(f, g) = & \text{let } u = \text{lfp}(\lambda x. \text{let } v = \text{lfp}(\lambda y. g(x, y)) \\ & \text{in } (f(x, v), v)) \\ & \text{in } (u, \text{lfp}(\lambda y. g(u, y))) \end{aligned}$$

RHS is asynchronous sequence with steps  $ggg \cdots fggg \cdots fggg \cdots$

- Immediate generalization to  $n$  posets  $(P_1, \leq) \times \cdots (P_n, \leq)$ .

# Grounding of a Datalog Program

What are the posets  $(P_1, \leq), (P_2, \leq), \dots$  for a datalog program?

- Option 1:  $P_i$  is  $(\text{ADom}^{k_i}, \subseteq)$  represents an IDB predicate.
- Option 2 (better):  $P_i$  is  $(\{0, 1\}, \leq)$  represents an IDB tuple.

## Example

$$R(X) \text{ :- } E(a, X)$$
$$R(X) \text{ :- } R(Z) \wedge E(Z, X)$$



## Example

$$R(X) \text{ :- } E(\textcolor{red}{a}, X)$$
$$R(X) \text{ :- } R(Z) \wedge E(Z, X)$$

EDB input graph:



## Example

$$R(X) \text{ :- } E(\textcolor{red}{a}, X)$$
$$R(X) \text{ :- } R(Z) \wedge E(Z, X)$$

Grounded program:

$$R(a) \text{ :- } E(a, a)$$
$$R(a) \text{ :- } R(a) \wedge E(a, a)$$
$$R(a) \text{ :- } R(b) \wedge E(b, a)$$
$$R(b) \text{ :- } E(a, b)$$
$$R(b) \text{ :- } R(a) \wedge E(a, b)$$
$$R(b) \text{ :- } R(b) \wedge E(b, b)$$

EDB input graph:



## Example

$$R(X) \text{ :- } E(\textcolor{red}{a}, X)$$
$$R(X) \text{ :- } R(Z) \wedge E(Z, X)$$

Grounded program:

$$R(a) \text{ :- } E(a, a)$$
$$R(a) \text{ :- } R(a) \wedge E(a, a)$$
$$R(a) \text{ :- } R(b) \wedge E(b, a)$$
$$R(b) \text{ :- } E(a, b)$$
$$R(b) \text{ :- } R(a) \wedge E(a, b)$$
$$R(b) \text{ :- } R(b) \wedge E(b, b)$$
$$R(a) \text{ :- } E(a, a) \vee R(a) \wedge E(a, a) \vee R(b) \wedge E(b, a),$$
$$R(b) \text{ :- } E(a, b) \vee R(a) \wedge E(a, b) \vee R(b) \wedge E(b, b)$$

EDB input graph:



## Example

$$R(X) \text{ :- } E(\textcolor{red}{a}, X)$$
$$R(X) \text{ :- } R(Z) \wedge E(Z, X)$$

Grounded program:

$$R(a) \text{ :- } E(a, a)$$
$$R(a) \text{ :- } R(a) \wedge E(a, a)$$
$$R(a) \text{ :- } R(b) \wedge E(b, a)$$
$$R(b) \text{ :- } E(a, b)$$
$$R(b) \text{ :- } R(a) \wedge E(a, b)$$
$$R(b) \text{ :- } R(b) \wedge E(b, b)$$
$$R(a) \text{ :- } E(a, a) \vee R(a) \wedge E(a, a) \vee R(b) \wedge E(b, a),$$
$$R(b) \text{ :- } E(a, b) \vee R(a) \wedge E(a, b) \vee R(b) \wedge E(b, b)$$

The grounded program allows more fine-grained asynchronous execution.

EDB input graph:



# Summary

- Main purpose of datalog is to add recursion.
- Least-fixpoint semantics; Kleene's sequence; Naive algorithm.
- Cool optimizations: semi-naive, magic-sets (difficult!), asynchronous evaluation.
- Can express PTIME-complete problems (AGAP).
- But limited to **monotone** queries.

Next lecture: adding negation to datalog.



Stanley, R. P. (1999).

*Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*.

Cambridge University Press, Cambridge.

With a foreword by Gian-Carlo Rota and appendix 1 by Sergey Fomin.