CSE 544
Principles of Database Management Systems

Fall 2016

Lecture 6 – Datalog (2)

Announcements

Homework 2 posted, due Friday, Nov. 4th

- SimpleDB

References

- Reading:
  Joe Hellerstein, “The Declarative Imperative,”
  SIGMOD Record 2010

- R&G Chapter 24

- Phokion Kolaitis’ tutorial on database theory at Simon’s
  https://simons.berkeley.edu/sites/default/files/docs/5241/simons16-21.pdf

- Daniel Zinn, Todd J. Green, Bertram Ludäscher: Win-move is coordination-free (sometimes). ICDT 2012

Review

- What is datalog?
- What is the naïve evaluation algorithm?
- What is the seminaive algorithm?

Outline

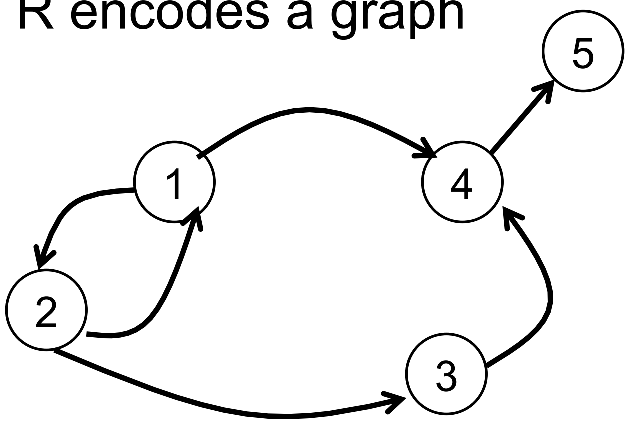
- Magic sets
- Extending datalog with negation and aggregates

Magic Sets

- Problem: datalog programs compute *a lot*, but sometimes we need only *very little*
- Prolog computes top-down and retrieves *very little* datalog computes bottom up retrieves *a lot*
- (Prolog has other issues: left recursive prolog never terminates!)
- Magic sets transform a datalog program P into a new program P', such that bottom-up(P') = top-down(P)

Example 1

R encodes a graph



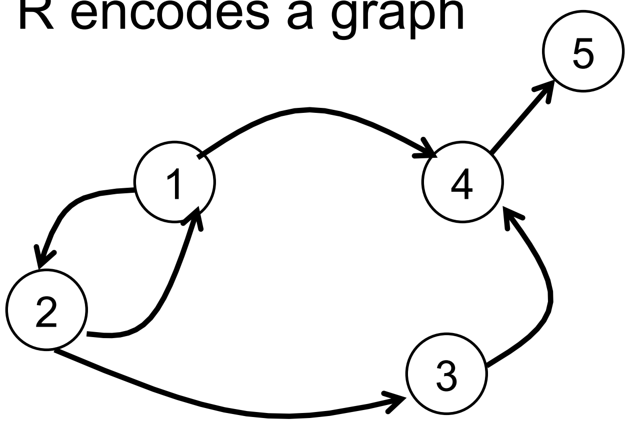
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y) :- T(3,y)

a constant

Bottom-up evaluation
very inefficient

Example 1

R encodes a graph




```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

a constant

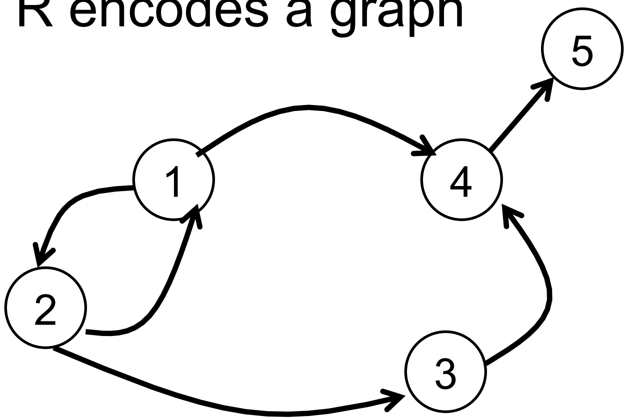
Manual optimization:

```
Q(y) :- E(3,y)
Q(y) :- Q(x),E(x,y)
```

Bottom-up evaluation very inefficient

Example 2

R encodes a graph



Same generation

SG(x,x) :- V(x)
SG(x,y) :- Up(x,u),SG(u,v),Dn(u,y)
Q(y) :- SG(1,y)

If we define
Up(a,b) = E(b,a)
Dn(a,b) = E(a,b)
then SG = "same generation"

Manual optimization???

Magic Set Rewriting (simplified)

- For each IDB predicate create “adorned” versions, with binding patters
- For each adorned IDB P, create a predicate Magic_P
- For each rule, create several rules, one for each possible adornment of the head:
  - Allow information to flow left-to-right (“sideways information passing”), and this defines the required adornments of the IDB’s in the body
  - If there are k IDB’s in the body, create k+1 supplementary relations Supp_i, which guard the set of bound variables passed on to the i’th IDB
- New rules defining Magic_P: one for the query, and one for each Supp_i preceding an occurrence of P in a body

Adorned predicate

- b=bound, f=free
- $T^{bf}(x,y)$ means:
  - The values of x are known
  - The values of y are not known (need to be retrieved)
- Need to create all combinations: T^{bf}, T^{fb}

- Side-ways information passing means that we adorn rules allowing information to flow left-to-right

  - E.g. $T(x,y)$:- $E(x,u),T(u,v),E(v,w),T(w,z),E(z,y)$

  - Adorned: $T^{bf}(x,y)$:- $E(x,u),T^{bf}(u,v),E(v,w),T^{bf}(w,z),E(z,y)$

Supplementary Relations

- Given adornment $T^{bf}(x,y)$, a new predicate Supp(x) contains the (small!) set of values x for which we want to compute $T^{bf}(x,y)$
- E.g. $T^{bf}(x,y)$:- $E(x,u), T^{bf}(u,v), E(v,w), T^{bf}(w,z), E(z,y)$

$Supp_0(x)$

$Supp_1(x,u)$

$Supp_2(x,w)$

$Supp_3(x,y)$

Supp Rules

- E.g. $T^{bf}(x,y)$:- $E(x,u),T^{bf}(u,v),E(v,w),T^{bf}(w,z),E(z,y)$

$Supp_0(x)$

$Supp_1(x,u)$

$Supp_2(x,w)$

$Supp_3(x,y)$

Becomes:

- $Supp_0(x)$:- $Magic_{Tbf}(x)$ /* next slide … */
- $Supp_1(x,u)$:- $Supp_0(x)$, $E(x,u)$
- $Supp_2(x,w)$:- $Supp_1(x,u)$, $T^{bf}(u,v),E(v,w)$
- $Supp_3(x,y)$:- $Supp_2(x,w)$, $T^{bf}(w,z),E(z,y)$
- $T^{bf}(x,y)$:- $Supp_3(x,y)$

$Supp_0$ and $Supp_3$ are redundant

Adding the Magic Predicate

- E.g. $T^{bf}(x,y)$:- $E(x,u), T^{bf}(u,v), E(v,w), T^{bf}(w,z), E(z,y)$

$Supp_0(x)$

$Supp_1(x,u)$

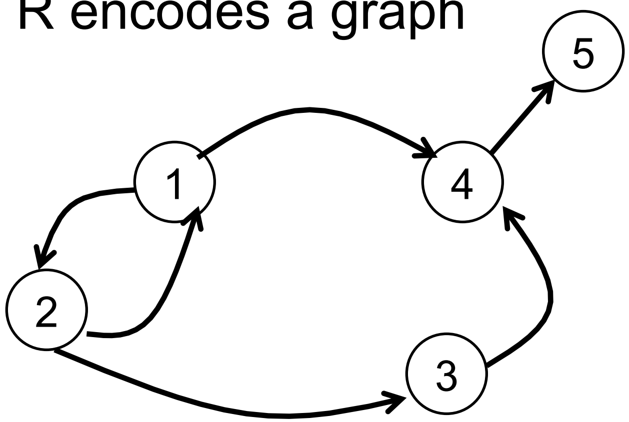
$Supp_2(x,w)$

$Supp_3(x,y)$

- $Magic_{Tbf}(x)$ = the set of bounded values of x for which we need to compute $T^{bf}(x,y)$
- E.g.
  - $Magic_{Tbf}(3)$:- /* if the query is Q(y) :- T(3,y) */
  - $Magic_{Tbf}(u)$:- $Supp_1(x,u)$ /* need to compute $T^{bf}(u,v)$ */
  - $Magic_{Tbf}(w)$:- $Supp_2(x,w)$ /* need to compute $T^{bf}(w,z)$ */

Example 1

R encodes a graph



Magic Sets

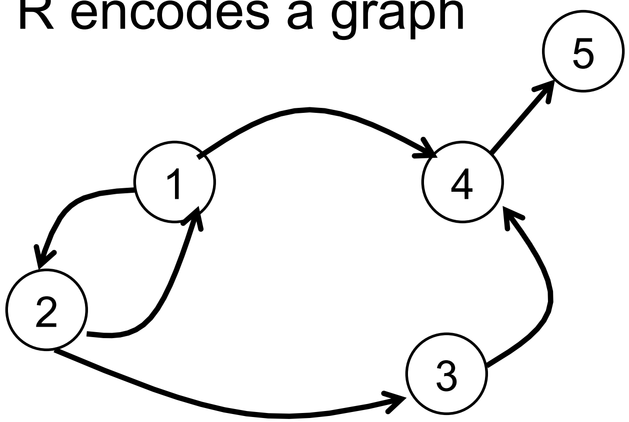
Original:

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

Adorned:

Example 1

R encodes a graph



Magic Sets

Original:

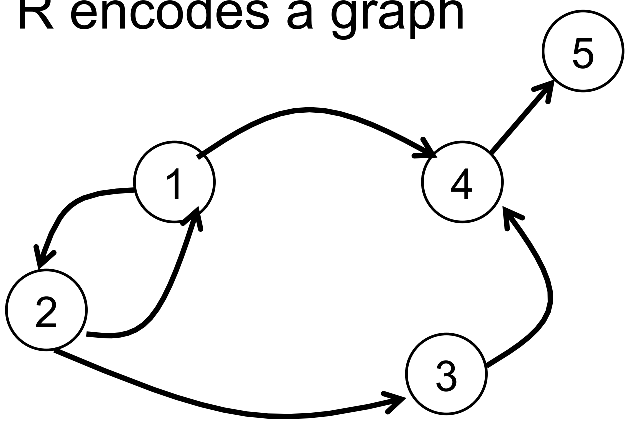
```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

Adorned:

$T^{bf}(x,y)$:- $E(x,y)$
$T^{bf}(x,y)$:- $T^{bf}(x,z),E(z,y)$
$Q(y)$:- $T^{bf}(3,y)$

Example 1

R encodes a graph



Original:

$T(x,y)$:- $E(x,y)$
$T(x,y)$:- $T(x,z),E(z,y)$
$Q(y)$:- $T(3,y)$

Adorned:

$T^{bf}(x,y)$:- $E(x,y)$
$T^{bf}(x,y)$:- $T^{bf}(x,z),E(z,y)$
$Q(y)$:- $T^{bf}(3,y)$

Magic Sets

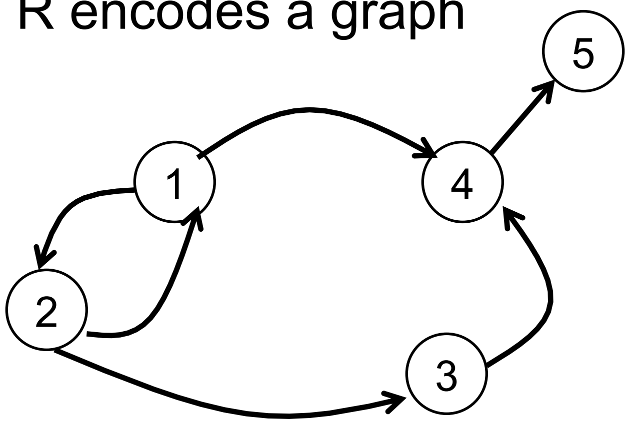
$Supp_0(x)$:- $Magic_{Tbf}(x)$
$Supp_1(x,y)$:- $Supp_0(x),E(x,y)$
$T^{bf}(x,y)$:- $Supp_1(x,y)$

$Supp'_0(x)$:- $Magic_{Tbf}(x)$
$Supp'_1(x,z)$:- $Supp'_0(x), T^{bf}(x,z)$
$Supp'_2(x,y)$:- $Supp'_1(x,z), E(z,y)$
$T^{bf}(x,y)$:- $Supp'_2(x,y)$

$Magic_{Tbf}(3)$:-
$Magic_{Tbf}(x)$:- $Supp'_0(x)$ /* redundant */

Example 1

R encodes a graph



Original:

$T(x,y)$:- $E(x,y)$
$T(x,y)$:- $T(x,z),E(z,y)$
$Q(y)$:- $T(3,y)$

Adorned:

$T^{bf}(x,y)$:- $E(x,y)$
$T^{bf}(x,y)$:- $T^{bf}(x,z),E(z,y)$
$Q(y)$:- $T^{bf}(3,y)$

Magic Sets

$Supp_0(x)$:- $Magic_{Tbf}(x)$
$Supp_1(x,y)$:- $Supp_0(x),E(x,y)$
$T^{bf}(x,y)$:- $Supp_1(x,y)$

$Supp'_0(x)$:- $Magic_{Tbf}(x)$
$Supp'_1(x,z)$:- $Supp'_0(x)$, $T^{bf}(x,z)$
$Supp'_2(x,y)$:- $Supp'_1(x,y)$, $E(z,y)$

$Magic_{Tbf}(3)$:-
$Magic_{Tbf}(x)$:- $Supp'_0(x)$ /* redundant */

Show computation on white board

Adding Negation: $\text{Datalog}^{\neg}$

Adding Negation: Datalog$^{\neg}$

Example: compute the complement of the transitive closure

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

What does this mean??

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

What are the possible outcomes of S and T?

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

What are the possible outcomes of S and T?

J_1 = { } J_2 = {S(a)} J_3 = {T(a)} J_4 = {S(a), T(a) }

Adding Negation: datalog$^{\neg}$

- **Solution 1: Stratified Datalog$^{\neg}$**
  - Rules must be partitioned into strata
  - IDB predicates defined in strata $\leq$ k may be negated in strata $\geq$ k+1
- **Solution 2: Inflationary-fixpoint Datalog$^{\neg}$**
  - Fire rules and always add facts (never retract)
  - Stop when nothing new is added
  - Always terminates (why ?)
- **Solution 3: Partial-fixpoint Datalog$^{\neg,*}$**
  - Fire rules, adding/retracting facts as needed
  - Stop when reaching a fixpoint
  - May not terminate
- **Solution 4: Well-founded semantics**

What semantics does the paper use?

Discussion in Class

The *Declarative Imperative* paper:

- What are the extensions to datalog in Dedalus?
- What is the main usage of Dedalus described in the paper?
- What limitations of datalog does the paper describe?

Semantics of a Datalog Program

Three different, equivalent semantics:

- Minimal model semantics
- Least fixpoint semantics
- Proof-theoretic semantics (will not discuss)

Minimal Model Semantics

To each rule r: $\boxed{P(x_1 \ldots x_k) \text{ :- } R_1(\ldots), R_2(\ldots), \ldots}$

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n . [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Minimal Model Semantics

To each rule r: $\boxed{P(x_1 \ldots x_k) \text{ :- } R_1(\ldots), R_2(\ldots), \ldots}$

All variables in the rule

Associate the logical sentence Σ_r: $\boxed{\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]}$

Same as: $\forall x_1 \ldots \forall x_k. [\exists y_1 \ldots \exists y_m.(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Head variables

Existential variables

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k) \text{ :- } R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Same as: $\forall x_1 \ldots \forall x_k. [\exists y_1 \ldots \exists y_m.(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Head variables

Existential variables

Definition. If **P** is a datalog program,
$\Sigma_{\mathbf{P}}$ is the set of all logical sentences associated to its rules.

Minimal Model Semantics

To each rule r: $P(x_1 \ldots x_k)$:- $R_1(\ldots), R_2(\ldots), \ldots$

All variables in the rule

Associate the logical sentence Σ_r: $\forall z_1 \ldots \forall z_n. [(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Same as: $\forall x_1 \ldots \forall x_k. [\exists y_1 \ldots \exists y_m.(R_1(\ldots) \wedge R_2(\ldots) \wedge \ldots) \rightarrow P(\ldots)]$

Head variables

Existential variables

<u>Definition</u>. If **P** is a datalog program,
$\Sigma_\mathbf{P}$ is the set of all logical sentences associated to its rules.

Example. Rule: T(x,y) :- R(x,z), T(z,y)

Sentence: $\forall x. \forall y. \forall z.(R(x,z) \wedge T(z,y) \rightarrow T(x,y))$
$\equiv \forall x. \forall y.(\exists z.R(x,z) \wedge T(z,y) \rightarrow T(x,y))$

Minimal Model Semantics

Definition. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if $(I,J) \models \Sigma_{\mathbf{P}}$

Definition. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. $(I,J) \models \Sigma_{\mathbf{P}}$

Theorem. The minimal model always exists, and is unique.

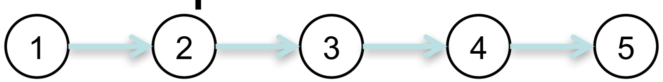
Minimal Model Semantics

<u>Definition</u>. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if (I,J) $\models \Sigma_P$

<u>Definition</u>. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. (I,J) $\models \Sigma_P$

<u>Theorem</u>. The minimal model always exists, and is unique.

Example:




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

Σ_P: $\forall x\ \forall y\ (R(x,y) \rightarrow T(x,y))$
$\forall x\ \forall y\ (R(x,z) \wedge T(z,y) \rightarrow T(x,y))$

Which of these IDBs are *<u>models</u>*?
Which are *<u>minimal models</u>*?

R=

1	2
2	3
3	4
4	5

T=

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

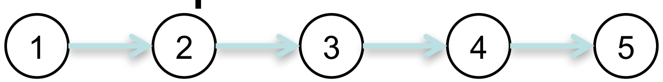
Minimal Model Semantics

Definition. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if $(I,J) \models \Sigma_P$

Definition. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. $(I,J) \models \Sigma_P$

Theorem. The minimal model always exists, and is unique.

Example:



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

Σ_P: $\forall x \, \forall y \, (R(x,y) \rightarrow T(x,y))$
$\forall x \, \forall y \, (R(x,z) \wedge T(z,y) \rightarrow T(x,y))$

Which of these IDBs are *models*?
Which are *minimal models*?

R=

1	2
2	3
3	4
4	5

T=

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

T=

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

Minimal Model Semantics

Definition. A pair (I,J) where I is an EDB and J is an IDB is a *model* for P, if $(I,J) \vDash \Sigma_P$

Definition. Given an EDB database instance I and a datalog program **P**, the minimal model, denoted J = **P**(I) is a minimal database instance J s.t. $(I,J) \vDash \Sigma_P$

Theorem. The minimal model always exists, and is unique.

Example:

$1 \to 2 \to 3 \to 4 \to 5$

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

Σ_P: $\forall x\ \forall y\ (R(x,y) \to T(x,y))$
$\forall x\ \forall y\ (R(x,z) \wedge T(z,y) \to T(x,y))$

Which of these IDBs are *models*?
Which are *minimal models*?

R=

1	2
2	3
3	4
4	5

T=

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

T=

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

T=

...	...
i	j

$\forall\, i<j$

Grounding

- A *grounding* of an atom is obtained by substituting its variables with constants from the active domain
- Examples:
  - T(5,2) is a grounding of T(x,y)
  - T(5,5) is a grounding of T(x,y)
  - T(5,5) is a grounding of T(x,x)
  - T(5,2) is *not* a grounding of T(x,x)
- A *grounding* of a rule is obtained by substituting its variables with constants from the active domain
- Examples:
  - (T(5,2) ← R(5,7),T(7,2)) is a grounding of (T(x,y) :- R(x,z),T(z,y))

Minimal Fixpoint Semantics

Definition. Fix an EDB I, and a datalog program **P**.
The *immediate consequence* operator T_P is defined as follows.
For any IDB J:

$T_P(J)$ = all IDB facts that are immediate consequences from I and J:

$$= \{H \mid (H \leftarrow B_1, \ldots, B_m) \in \text{ground}(P),\ J \models B_1, \ldots, B_m\}$$

Fact. For any datalog program P, the immediate consequence operator is monotone. In other words, if $J_1 \subseteq J_2$ then $T_P(J_1) \subseteq T_P(J_2)$.

Minimal Fixpoint Semantics

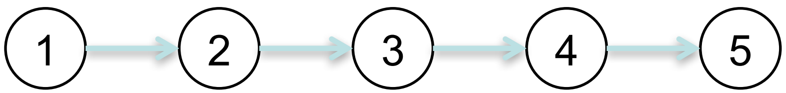
Definition. Fix an EDB I, and a datalog program **P**.
The *immediate consequence* operator T_P is defined as follows.
For any IDB J:
$T_P(J)$ = all IDB facts that are immediate consequences from I and J:
$= \{H \mid (H \leftarrow B_1, \ldots, B_m) \in \text{ground}(P), J \vDash B_1, \ldots, B_m\}$

Fact. For any datalog program P, the immediate consequence operator is monotone. In other words, if $J_1 \subseteq J_2$ then $T_P(J_1) \subseteq T_P(J_2)$.

Theorem. The immediate consequence operator has a unique, minimal fixpoint J: $\text{fix}(T_P) = J$, where J is the minimal instance with the property $T_P(J) = J$.

Proof: using Knaster-Tarski's theorem for monotone functions.
The fixpoint is given by:
$\text{fix}(T_P) = J_0 \cup J_1 \cup J_2 \cup \ldots$ where $J_0 = \varnothing$, $J_{k+1} = T_P(J_k)$

Minimal Fixpoint Semantics



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

R=

1	2
2	3
3	4
4	5

$J_0 = \varnothing$

T =

$J_1 = T_{\mathbf{P}}(J_0)$

1	2
2	3
3	4
4	5

$J_2 = T_{\mathbf{P}}(J_1)$

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

$J_3 = T_{\mathbf{P}}(J_2)$

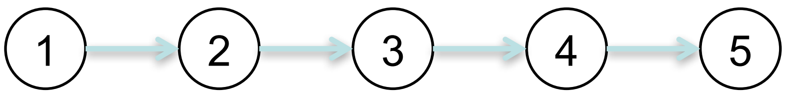
1	2
2	3
3	4
4	5
1	3
2	4
3	5
1	4
2	5

$J_4 = T_{\mathbf{P}}(J_3)$

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

Proof Theoretic Semantics

Every fact in the IDB has a *derivation tree*, or *proof tree* justifying its existence.

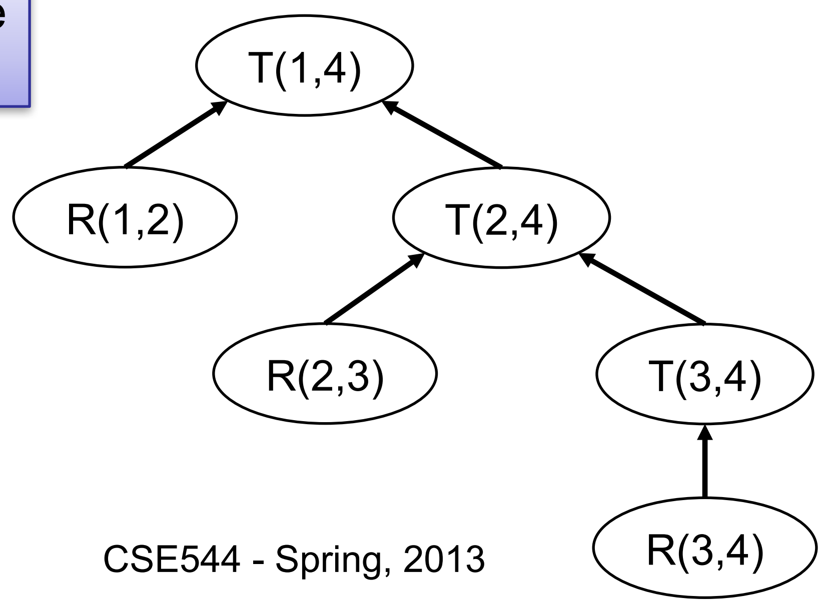


T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

R=

1	2
2	3
3	4
4	5

Derivation tree of T(1,4)



Adding Negation: Datalog$^\neg$

Example: compute the complement of the transitive closure

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

What does this mean??

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

$J_1 = \{ \}$ $J_2 = \{S(a)\}$ $J_3 = \{T(a)\}$ $J_4 = \{S(a), T(a)\}$

Recursion and Negation Don't Like Each Other

EDB: $I = \{ R(a) \}$

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

$J_1 = \{ \}$ — No: both rules fail

$J_2 = \{S(a)\}$ — Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

$J_3 = \{T(a)\}$ — Yes

$J_4 = \{S(a), T(a) \}$ — Yes

There is no *minimal* model!

Recursion and Negation Don't Like Each Other

EDB: I = { R(a) }

S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)

Which IDBs are models of **P**?

$J_1 = \{ \}$ $J_2 = \{S(a)\}$ $J_3 = \{T(a)\}$ $J_4 = \{S(a), T(a) \}$

No: both rules fail

Yes: the facts in J_2 are R(a), S(a), ¬T(a) and both rules are *true*.

Yes

Yes

There is no <u>*minimal*</u> model!

There is no minimal fixpoint! (Why does Knaster-Tarski's theorem fail?)

Adding Negation: datalog$^\neg$

- **Solution 1: Stratified Datalog$^\neg$**
  - Rules must be partitioned into strata
  - IDB predicates defined in strata ≤ k may be negated in strata ≥ k+1
- **Solution 2: Inflationary-fixpoint Datalog$^\neg$**
  - Fire rules and always add facts (never retract)
  - Stop when nothing new is added
  - Always terminates (why ?)
- **Solution 3: Partial-fixpoint Datalog$^{\neg,*}$**
  - Fire rules, adding/retracting facts as needed
  - Stop when reaching a fixpoint
  - May not terminate
- **Solution 4: Well-founded semantics**

Stratified datalog¬

A datalog¬ program is *stratified* if its rules can be partitioned into k strata, such that:

- If an IDB predicate P appears negated in a rule in stratum i, then it can only appear in the head of a rule in strata 1, 2, …, i-1

P:

P_1 :- body_1
P_2 :- body_2
….

….

P_j :- body_j

….

….

….
P_n :- body_n

Stratum

Note: a datalog¬ program either is stratified or it ain't!

Which programs are stratified?

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

```
S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)
```

Stratified datalog¬

- Evaluation algorithm for stratified datalog¬:

- For each stratum i = 1, 2, …, do:
  - Treat all IDB's defined in prior strata as EBS
  - Evaluate the IDB's defined in stratum i, using either the naïve or the semi-naïve algorithm

Does this compute a minimal model?

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
------------------------------------------
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

Stratified datalog¬

- Evaluation algorithm for stratified datalog¬:

- For each stratum i = 1, 2, …, do:
  - Treat all IDB's defined in prior strata as EBS
  - Evaluate the IDB's defined in stratum i, using either the naïve or the semi-naïve algorithm

Does this compute a minimal model?

NO:
J_1 = { T = transitive closure, CT = its complement}
J_2 = { T = all pairs of nodes, CT = empty}

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

CT(x,y) :- Node(x), Node(y), not T(x,y)

Inflationary-fixpoint datalog¬

Let **P** be any datalog¬ program, and I an EDB.
Let $T_P(J)$ be the *immediate consequence* operator.
Let $F(J) = J \cup T_P(J)$ be the *inflationary immediate consequence* operator.

Define the sequence: $J_0 = \varnothing$, $J_{n+1} = F(J_n)$, for $n \geq 0$.

Definition. The inflationary fixpoint semantics of **P** is $J = J_n$ where n is such that $J_{n+1} = J_n$

Why does there always exists an n such that $J_n = F(J_n)$?

Find the inflationary semantics for:

```
T(x,y) :- R(x,y)
T(x,y) :- T(x,z), R(z,y)
CT(x,y) :- Node(x), Node(y), not T(x,y)
```

```
S(x) :- R(x), not T(x)
T(x) :- R(x), not S(x)
```

Inflationary-fixpoint datalog¬

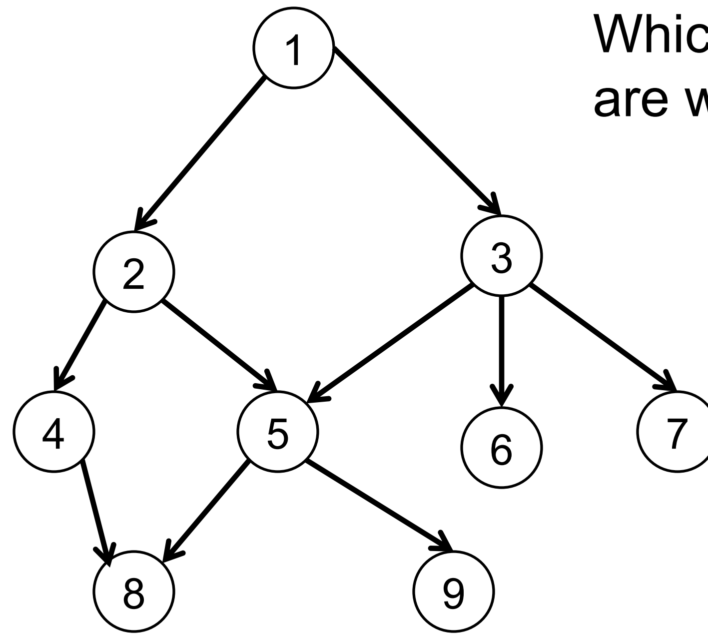
- Evaluation for Inflationary-fixpoint datalog¬
- Use the naïve, of the semi-naïve algorithm
- Inhibit any optimization that rely on monotonicity (e.g. out of order execution)

Well-Founded Semantics

- The lecture follows:
  Daniel Zinn, Todd J. Green, Bertram Ludäscher: Win-move is coordination-free (sometimes). ICDT 2012

Example: Win-Move Game

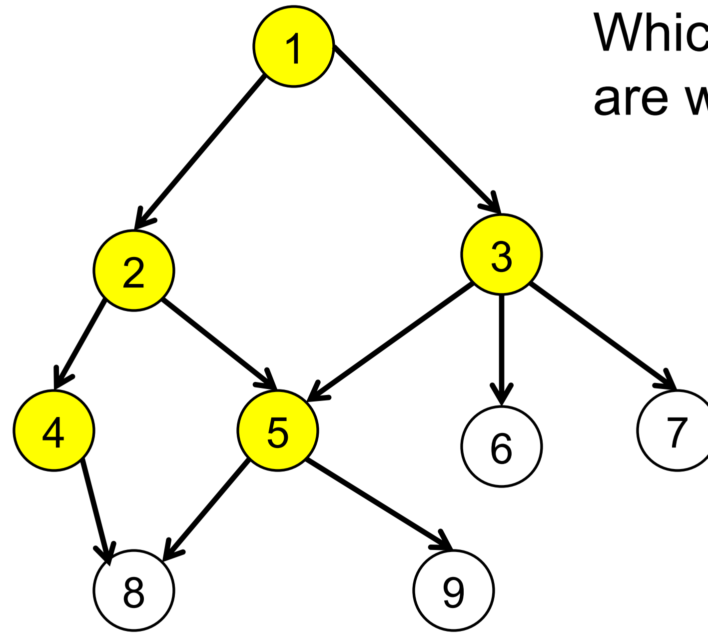
Win(X) :- Move(X,Y), ¬Win(Y)



Which nodes are winning?

Example: Win-Move Game

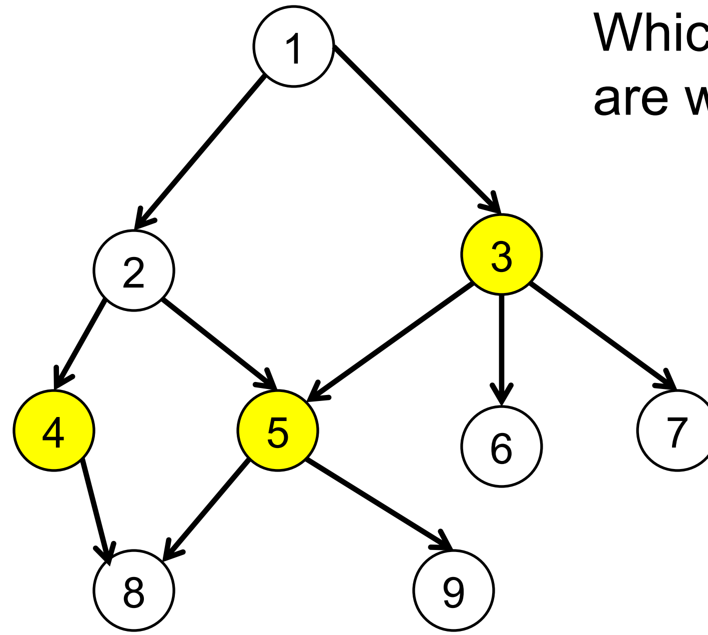
Win(X) :- Move(X,Y), ¬Win(Y)



Which nodes are winning?

Example: Win-Move Game

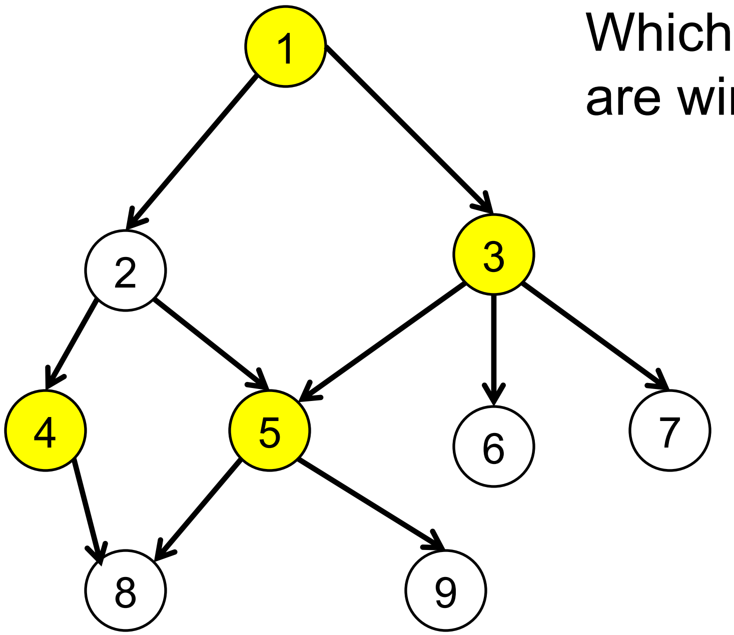
Win(X) :- Move(X,Y), ¬Win(Y)



Which nodes are winning?

Example: Win-Move Game

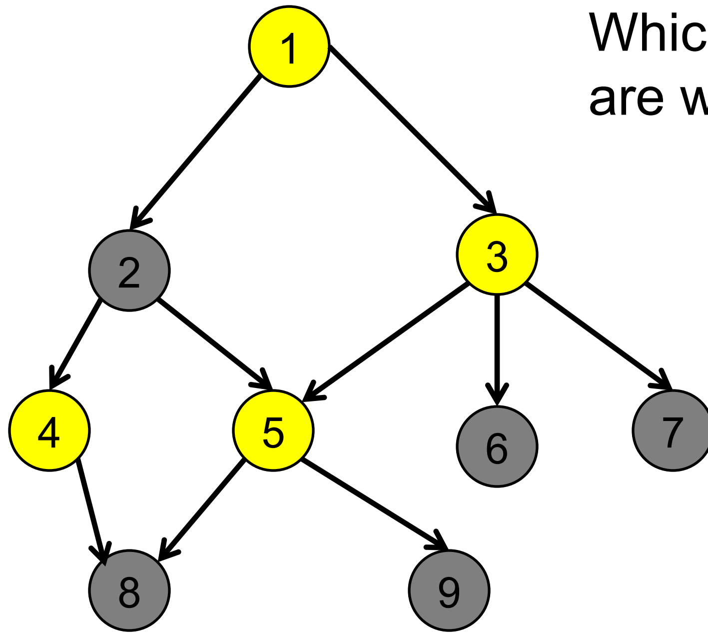
Win(X) :- Move(X,Y), ¬Win(Y)



Which nodes are winning?

Example: Win-Move Game

$$Win(X) \text{ :- } Move(X,Y), \neg Win(Y)$$

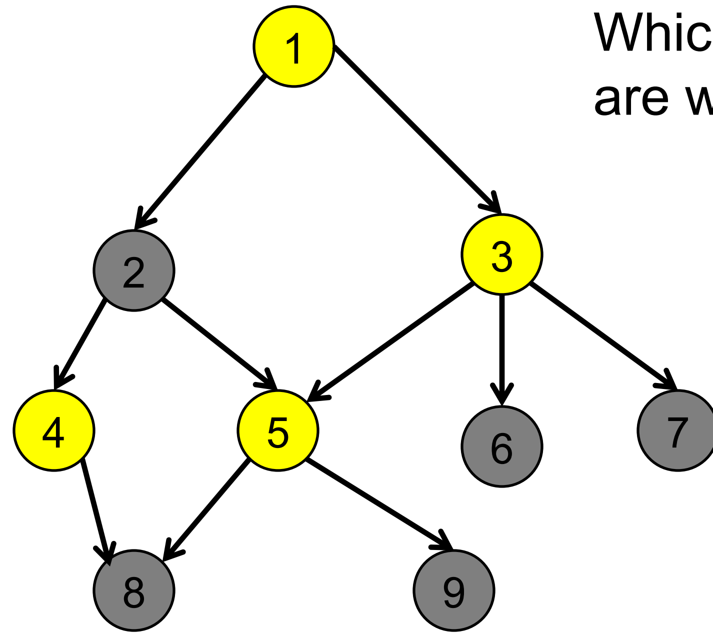


Which nodes are winning?

Win(1), Win(3), Win(4), Win(5)
¬Win(2), ¬Win(6), ¬Win(7),¬Win(8),¬Win(9)

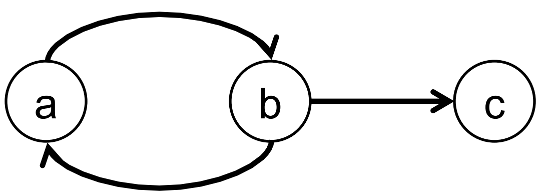
Example: Win-Move Game

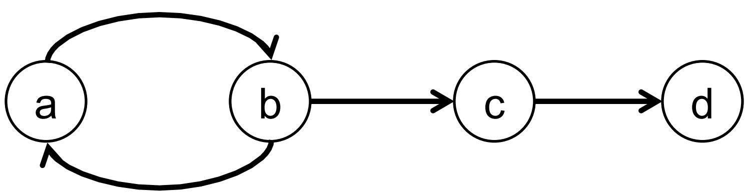
Win(X) :- Move(X,Y), ¬Win(Y)



Which nodes are winning?

What about these?

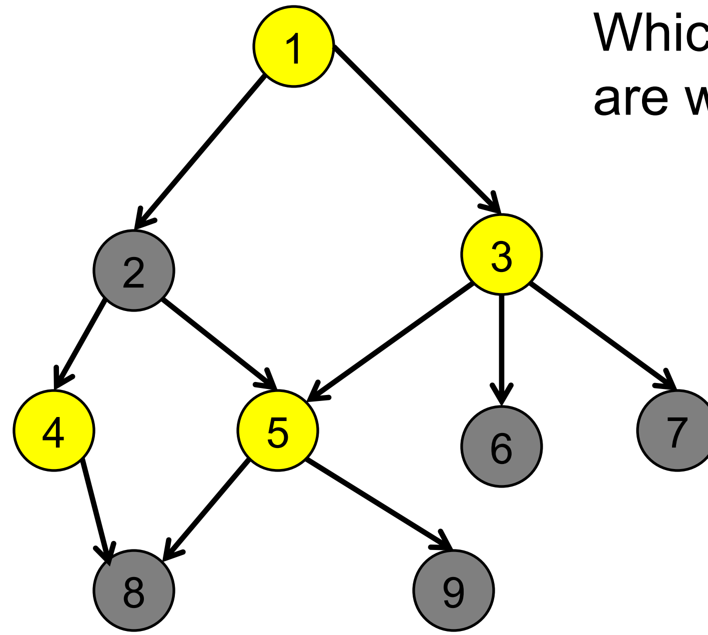




Win(1), Win(3), Win(4), Win(5)
¬Win(2), ¬Win(6), ¬Win(7),¬Win(8),¬Win(9)

Example: Win-Move Game

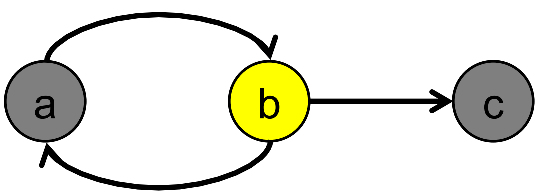
Win(X) :- Move(X,Y), ¬Win(Y)

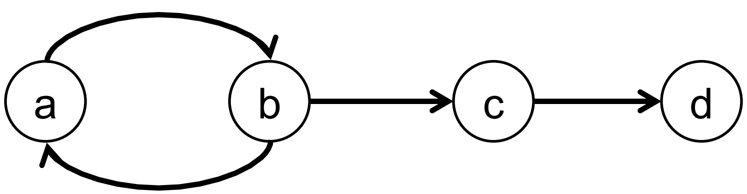


Which nodes are winning?

Win(1), Win(3), Win(4), Win(5)
¬Win(2), ¬Win(6), ¬Win(7),¬Win(8),¬Win(9)

What about these?

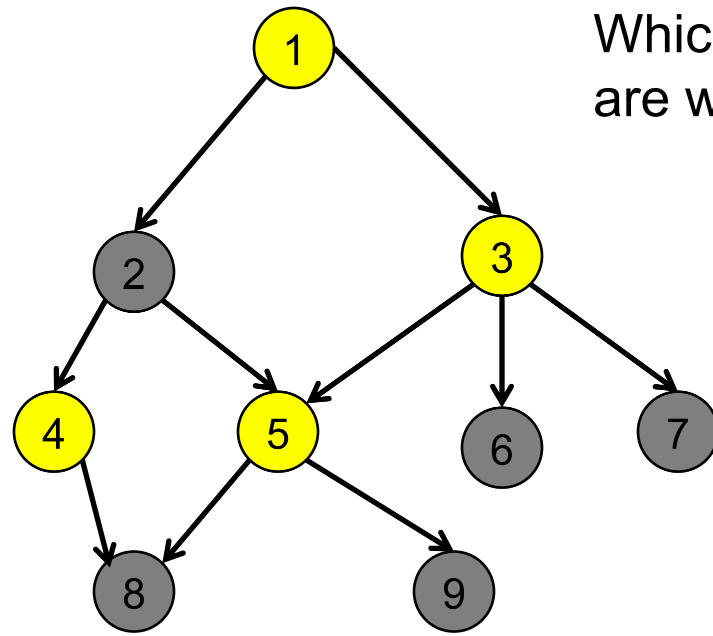




Example: Win-Move Game

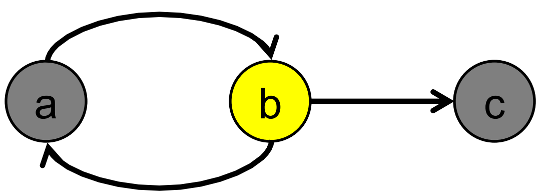
Win(X) :- Move(X,Y), ¬Win(Y)

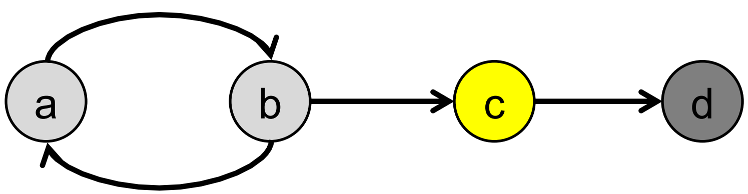
Which nodes are winning?



Win(1), Win(3), Win(4), Win(5)
¬Win(2), ¬Win(6), ¬Win(7),¬Win(8),¬Win(9)

What about these?





Win(c), ¬Win(d)
a and b are neither winning nor losing

Well-founded Semantics

Let **P** be any datalog¬ program
Let I be instances of both EDB and IDB (note: was only EDB before)
Let $T_{\mathbf{P,I}}(J)$ be the *immediate consequence* operator defined as follows:

$$T_{\mathbf{P,I}}(J) = \{H \mid (H \text{ :- } B_1, \ldots, B_m, \neg C_1, \ldots, \neg C_n) \in \text{ground}(P), \; J \vDash B_1, \ldots, B_m, \; I \vDash \neg C_1, \ldots, \neg C_n \}$$

Note that $T_{\mathbf{P,I}}(J)$ is monotone in J, hence has a Least Fix Point (lfp).

Let $\Gamma_P(I) = \text{lfp}(T_{\mathbf{P,I}})$

Note that $\Gamma_P(I)$ is antimonotone in the IDB's: $I \subseteq I'$ implies $\Gamma_P(I) \supseteq \Gamma_P(I')$

$\Gamma_P^2(I)$ ($:= \Gamma_P(\Gamma_P(I))$) is monotone: has Least Fix Point (lfp), Greatest Fix Point (gfp)

Definition. The well-founded semantics is defined on ground facts A as:

$W_P(A)$ = true if $A \in \text{lfp}(\Gamma_P^2)$
$W_P(A)$ = false if $A \notin \text{gfp}(\Gamma_P^2)$
$W_P(A)$ = undefined if $A \in \text{gfp}(\Gamma_P^2) - \text{lfp}(\Gamma_P^2)$

Well-founded Semantics

Note how we compute the lfp and gfp of $\Gamma_P^2(I)$.
Apply Γ_P repeatedly:

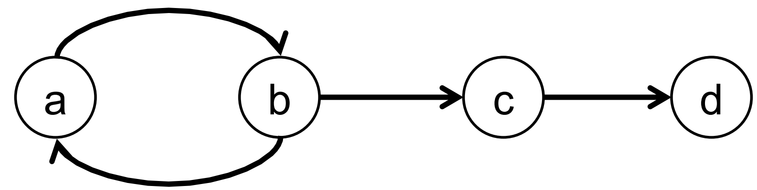
- Odd iterations increase → towards lfp
- Even iterations decrease → towards gfp

Denoting $I_k = T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\ldots\ T_{\mathbf{P,I}}(\varnothing)\ \ldots)))$ (k times)

$\varnothing \subseteq I_2 \subseteq I_4 \subseteq I_6 \subseteq \ldots \subseteq \mathrm{lfp}(\Gamma_P^2(I)) \subseteq \mathrm{gfp}(\Gamma_P^2(I)) \subseteq \ldots \subseteq I_5 \subseteq I_3 \subseteq I_1 \subseteq \mathrm{Domain}^k$

Example: Win-Move Game

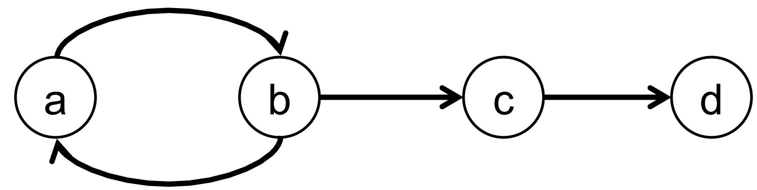
$Win(X) \text{ :- } Move(X,Y), \neg Win(Y)$



$T_{\mathbf{P,I}}(J)$ says: "fix ¬Win according to I, and Win according to J"

Example: Win-Move Game

Win(X) :- Move(X,Y), ¬Win(Y)



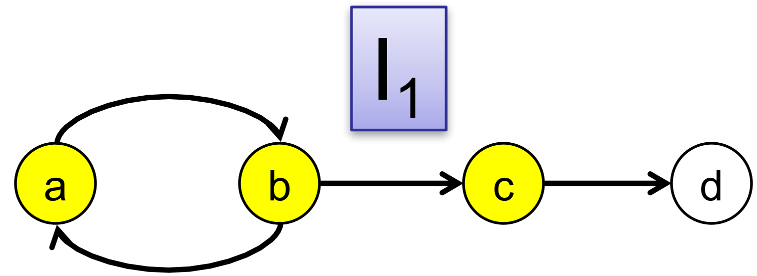
$T_{\mathbf{P,I}}$(J) says: "fix ¬Win according to I, and Win according to J"

Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

Example: Win-Move Game

Win(X) :- Move(X,Y), ¬Win(Y)



$T_{\mathbf{P,I}}$(J) says: "fix ¬Win according to I, and Win according to J"

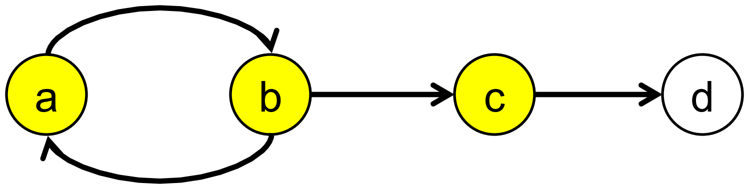
Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

I_1 = {Win(a), Win(b), Win(c), ¬Win(d)}

Example: Win-Move Game

Win(X) :- Move(X,Y), ¬Win(Y)



$T_{\mathbf{P,I}}$(J) says: "fix ¬Win according to I, and Win according to J"

Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

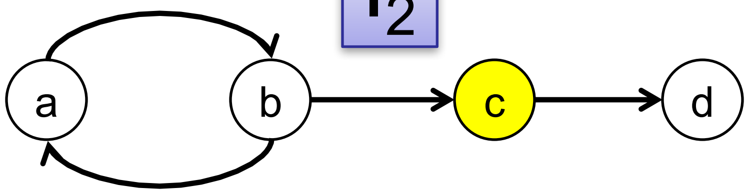
$I_1 =$ {Win(a), Win(b), Win(c), ¬Win(d)}

$I_2 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

Example: Win-Move Game

$Win(X) \text{ :- } Move(X,Y), \neg Win(Y)$

I_2



$T_{\mathbf{P,I}}(J)$ says: "fix ¬Win according to I, and Win according to J"

Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

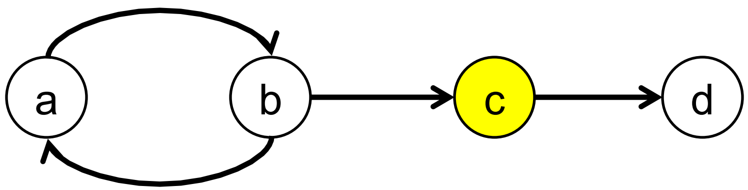
$I_1 = \{Win(a), Win(b), Win(c), \neg Win(d)\}$

$I_2 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

$I_2 = \{\neg Win(a), \neg Win(b), Win(c), \neg Win(d)\}$

Example: Win-Move Game

Win(X) :- Move(X,Y), ¬Win(Y)



$T_{\mathbf{P,I}}(J)$ says: “fix ¬Win according to I, and Win according to J”

Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

I_1 = {Win(a), Win(b), Win(c), ¬Win(d)}

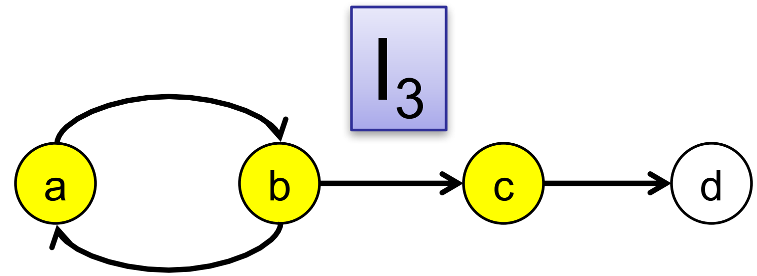
$I_2 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

I_2 = {¬Win(a), ¬Win(b), Win(c), ¬Win(d)}

$I_3 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

Example: Win-Move Game

Win(X) :- Move(X,Y), ¬Win(Y)



$T_{P,I}(J)$ says: "fix ¬Win according to I, and Win according to J"

Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{P,I}) = \varnothing \cup T_{P,I}(\varnothing) \cup T_{P,I}(T_{P,I}(\varnothing)) \cup T_{P,I}(T_{P,I}(T_{P,I}(\varnothing))) \cup \ldots$

I_1 = {Win(a), Win(b), Win(c), ¬Win(d)}

$I_2 = \Gamma_P(I) = \mathrm{lfp}(T_{P,I}) = \varnothing \cup T_{P,I}(\varnothing) \cup T_{P,I}(T_{P,I}(\varnothing)) \cup T_{P,I}(T_{P,I}(T_{P,I}(\varnothing))) \cup \ldots$

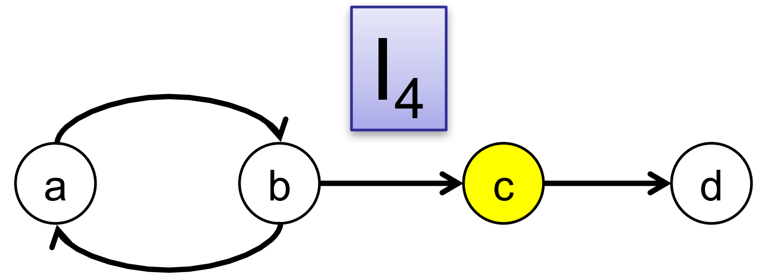
I_2 = {¬Win(a), ¬Win(b), Win(c), ¬Win(d)}

$I_3 = \Gamma_P(I) = \mathrm{lfp}(T_{P,I}) = \varnothing \cup T_{P,I}(\varnothing) \cup T_{P,I}(T_{P,I}(\varnothing)) \cup T_{P,I}(T_{P,I}(T_{P,I}(\varnothing))) \cup \ldots$

$I_3 = I_1$ = have reached the gfp

Example: Win-Move Game

$Win(X)$:- $Move(X,Y)$, $\neg Win(Y)$



$T_{P,I}(J)$ says: "fix ¬Win according to I, and Win according to J"

Start with $I_0 = \varnothing$ $\quad (= \{\neg Win(a), \neg Win(b), \neg Win(c), \neg Win(d)\}$
$I_1 = \Gamma_P(I) = lfp(T_{P,I}) = \varnothing \cup T_{P,I}(\varnothing) \cup T_{P,I}(T_{P,I}(\varnothing)) \cup T_{P,I}(T_{P,I}(T_{P,I}(\varnothing))) \cup \ldots$
$I_1 = \{Win(a), Win(b), Win(c), \neg Win(d)\}$

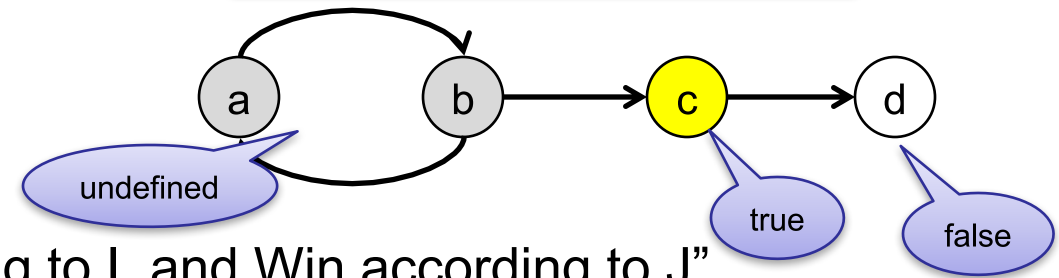
$I_2 = \Gamma_P(I) = lfp(T_{P,I}) = \varnothing \cup T_{P,I}(\varnothing) \cup T_{P,I}(T_{P,I}(\varnothing)) \cup T_{P,I}(T_{P,I}(T_{P,I}(\varnothing))) \cup \ldots$
$I_2 = \{\neg Win(a), \neg Win(b), Win(c), \neg Win(d)\}$

$I_3 = \Gamma_P(I) = lfp(T_{P,I}) = \varnothing \cup T_{P,I}(\varnothing) \cup T_{P,I}(T_{P,I}(\varnothing)) \cup T_{P,I}(T_{P,I}(T_{P,I}(\varnothing))) \cup \ldots$
$I_3 = I_1$ = have reached the gfp
$I_4 = I_2$ = have reached the lfp

Example: Win-Move Game

Well founded semantics:

Win(X) :- Move(X,Y), ¬Win(Y)



$T_{\mathbf{P,I}}$(J) says: "fix ¬Win according to I, and Win according to J"

Start with $I_0 = \varnothing$ (= {¬Win(a), ¬Win(b), ¬Win(c), ¬Win(d)}

$I_1 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

I_1 = {Win(a), Win(b), Win(c), ¬Win(d)}

$I_2 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

I_2 = {¬Win(a), ¬Win(b), Win(c), ¬Win(d)}

$I_3 = \Gamma_P(I) = \mathrm{lfp}(T_{\mathbf{P,I}}) = \varnothing \cup T_{\mathbf{P,I}}(\varnothing) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing)) \cup T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(T_{\mathbf{P,I}}(\varnothing))) \cup \ldots$

$I_3 = I_1$ = have reached the gfp

$I_4 = I_2$ = have reached the lfp

Discussion

- Which semantics does Daedalus adopt?

Discussion

Comparing datalog¬

- Compute the complement of the transitive closure in inflationary datalog¬
- Compare the expressive power of:
  - Stratified datalog¬
  - Inflationary fixpoint datalog¬
  - Partial fixpoint datalog¬