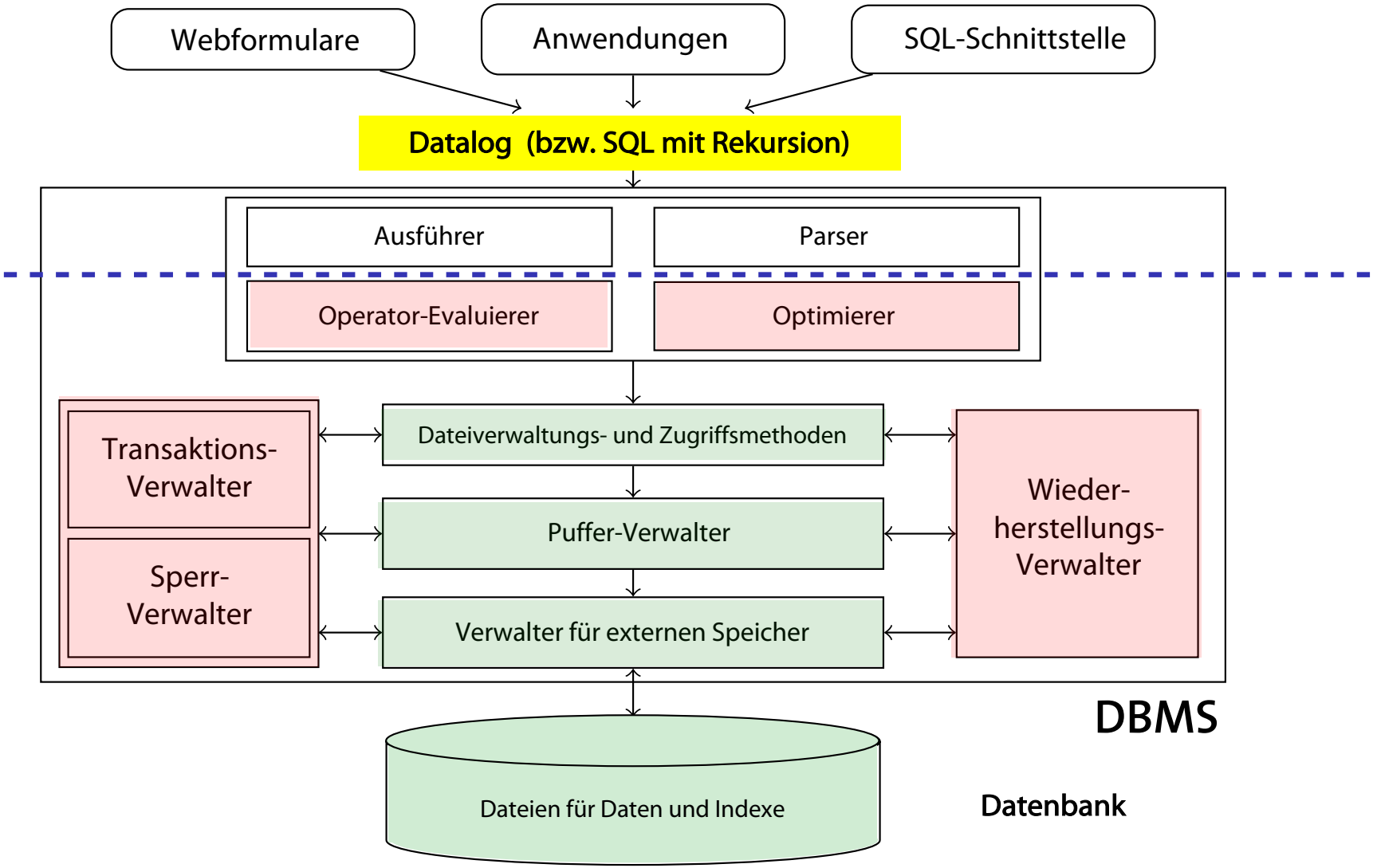
# Datenbanken

## Datalog

Dr. Özgür Özçep
Prof. Dr. Ralf Möller
**Universität zu Lübeck**
**Institut für Informationssysteme**

Felix Kuhr (Übungen)
und studentische Tutoren

# Architektur eines DBMS



# Rekursive Anfragen in SQL

| *Kurz* | *Name* | *Oberabt* |
|---|---|---|
| MFSW | Mainframe SW | LTSW |
| UXSW | Unix SW | LTSW |
| PCSW | PC SW | LTSW |
| LTSW | Leitung SW | **NULL** |
| PERS | Personal | **NULL** |

*Abteilungen*

```
create recursive view Unterabteilungen
  select r.kurz, r.oberabt
  from Abteilungen r
union
  select u.kurz, o.oberabt
  from Abteilungen o,
       Unterabteilungen u
  where o.kurz = u.oberabt
```

PostgreSQL-Syntax

- Beim Start der Rekursion enthält „Unterabteilungen“ nur die Tupel der ersten Teilanfrage.
- Tupel, die sich durch den Join in der zweiten Teilanfrage ergeben, werden der Extension von „Unterabteilungen“ für die nächste Iteration hinzugefügt.
- Abbruch der Rekursion, sobald die zweite Teilanfrage bei Verwendung der Ergebnisse aus der vorigen Iteration keine zusätzlichen Ergebnistupel mehr liefert ⟶ Fixpunkt.

# Rekursive Anfragen in Datalog

| *Kurz* | *Name* | *Oberabt* |
|---|---|---|
| MFSW | Mainframe SW | LTSW |
| UXSW | Unix SW | LTSW |
| PCSW | PC SW | LTSW |
| LTSW | Leitung SW | **NULL** |
| PERS | Personal | **NULL** |

*Abteilungen*

```
create recursive view Unterabteilungen
  select r.kurz, r.oberabt
  from Abteilungen r
union
  select u.kurz, o.oberabt
  from Abteilungen o,
       Unterabteilungen u
  where o.kurz = u.oberabt
```

PostgreSQL-Syntax

$$\forall\, x,y\ (\ \text{Unterabteilung}(x,y) \leftarrow \exists\, w\ \text{Abteilung}(x,w,y)\ )$$

$$\forall\ x,y,z\ (\ \text{Unterabteilung}(x,y) \leftarrow \exists\, w\ \exists\, z(\text{Abteilung}(z,w,y) \wedge \text{Unterabteilung}(x,z)\ )\ )$$

# Datalog: Rekursive Anfragen

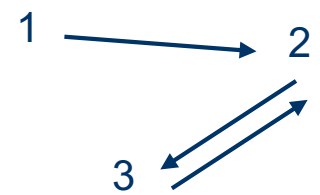
Einige Darstellungen wurden mit Änderungen übernommen aus einer Präsentation von Stephanie Scherzinger Und von Ullman CS345

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

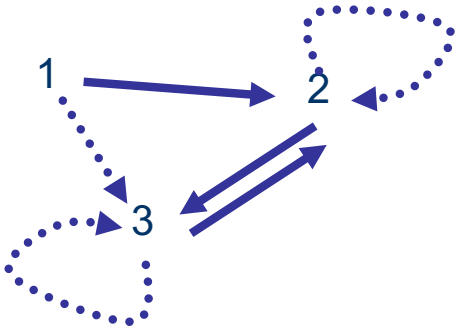
$$\forall\ x,y,z\ (\ T(x,y) \leftarrow (\ G(x,z) \wedge T(z,y)\ )\ )$$

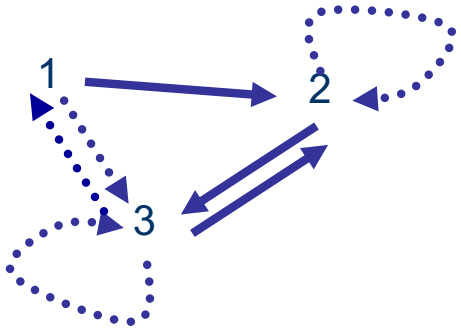
$$G(\ 1,\ 2\ ),\ G(\ 2,\ 3\ ),\ G(\ 3,\ 2\ )$$

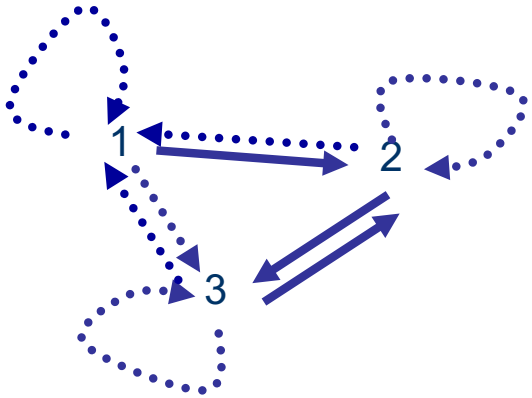
G:




Mögliche Lösungen:




Herve Gallaire und Jack Minker *Logic and Data Bases*. Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse in „Advances in Data Base Theory". Plenum Press, New York **1978**

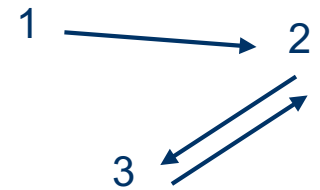
# Minimale-Modell-Semantik

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

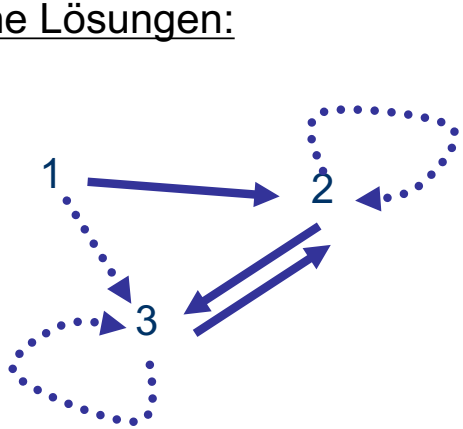
$\forall$ x,y,z ( T(x,y) $\leftarrow$ ( G(x,z) $\wedge$ T(z,y) ) )

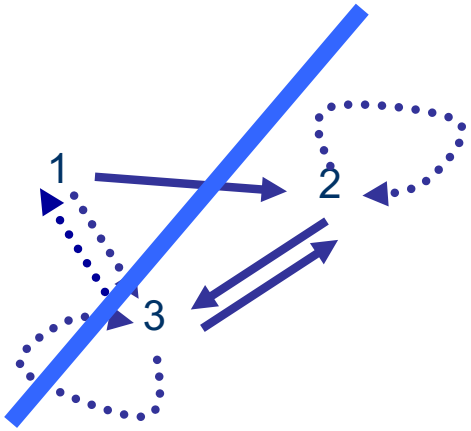
G( 1, 2 ), G( 2, 3 ), G( 3, 2 )

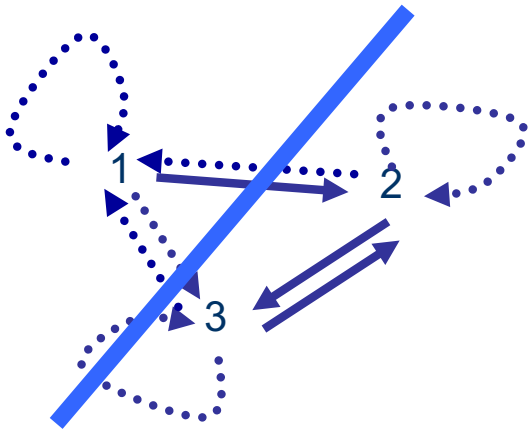
G:




Mögliche Lösungen:




→ ***Wähle minimales Modell*** ←
**T soll kleinste Menge von Fakten enthalten, so dass Regeln wahr sind**

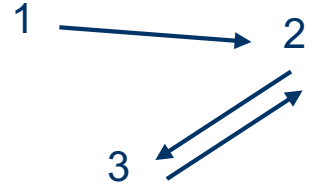
# Minimale-Modell-Semantik

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

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

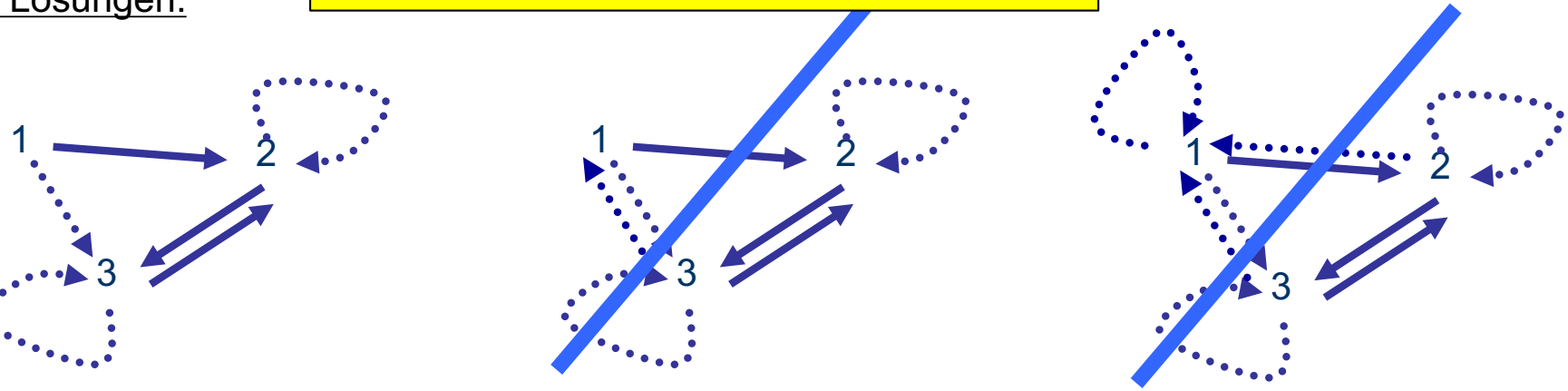
G( 1, 2 ), G( 2, 3 ), G( 3, 2 )

G:




Terminologie:
G is ein EDB-Prädikat (extensionale DB)
T ist ein IDB-Prädikat (intensionale DB)

Mögliche Lösungen:




→ ***Wähle minimales Modell*** ←
**T soll kleinste Menge von Fakten enthalten, so dass Regeln wahr sind**

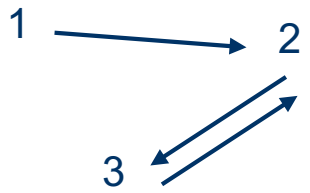
# Datalog$_{(rec, no\text{-}neg)}$

Erreichbarkeit in Graphen (T = transitive Hülle von G):

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

# Intuition

Transitive Hülle:



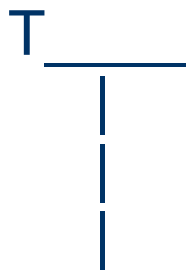

```
T(x,y)  :- G(x,y)
T(x,y)  :- G(x,z), T(z,y)
```

T

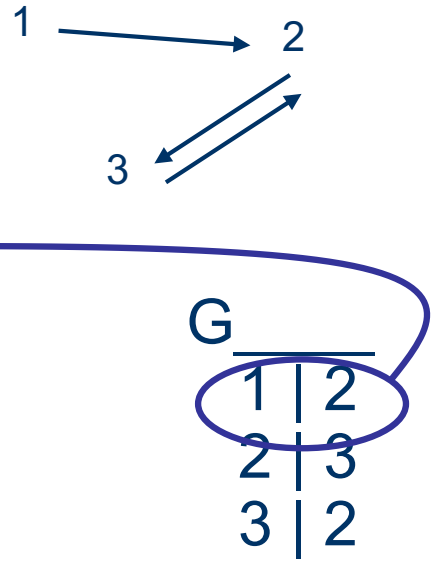
| G | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

# Intuition

Transitive Hülle:



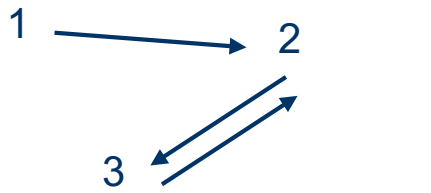

```
T(x,y)  :- G(1,2)
T(x,y)  :- G(x,z), T(z,y)
```




**(1) Bilde ab von den DB-Relationen in den Regelrumpf**

# Intuition

Transitive Hülle:




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

T

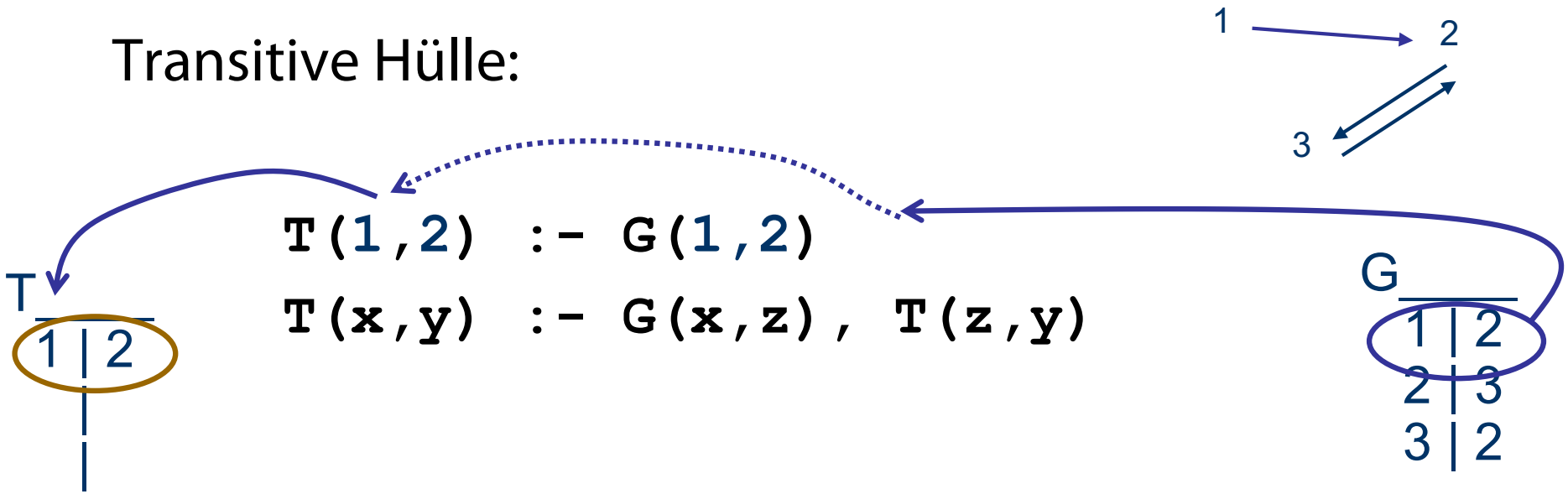
G

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

**(2) Propagiere Bindungen auf Regelkopf**

# Intuition

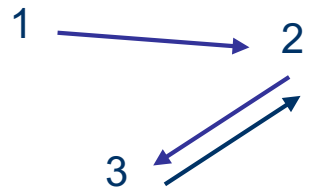
Transitive Hülle:




**(2) Propagiere Bindungen auf Regelkopf**

# Intuition

Transitive Hülle:




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

T

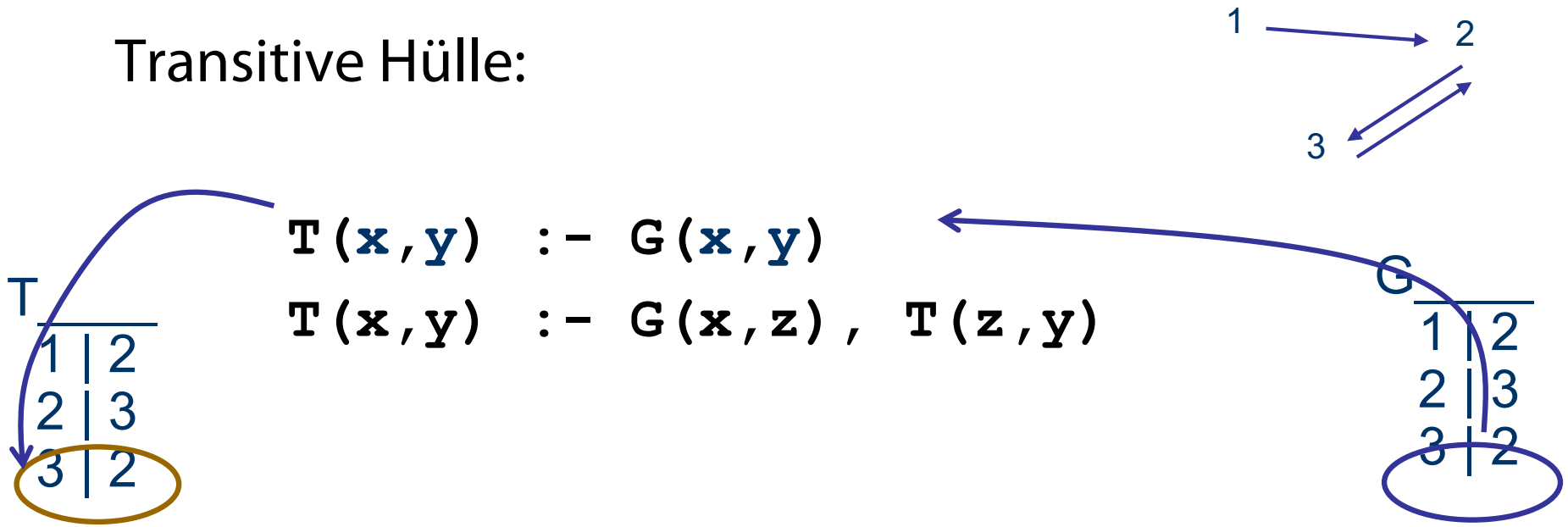
| 1 | 2 |
|---|---|
| 2 | 3 |

G

| 1 | 2 |
|---|---|
| 2 | 3 |
| 3 | 2 |

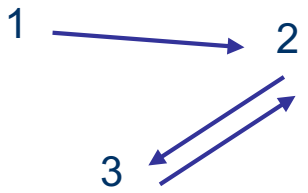
# Intuition

Transitive Hülle:



# Intuition

Transitive Hülle:




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

T

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

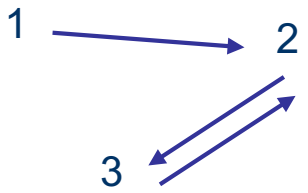
G

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

**(1) Setze Tupel aus den Relationen in den Regelrumpf ein**

# Intuition

Transitive Hülle:




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

T

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

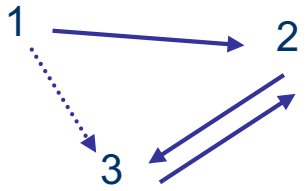
G

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

**(2) Übertrage Bindungen in den Regelkopf**

# Intuition

Transitive Hülle:




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

T

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |
| 1 | 3 |

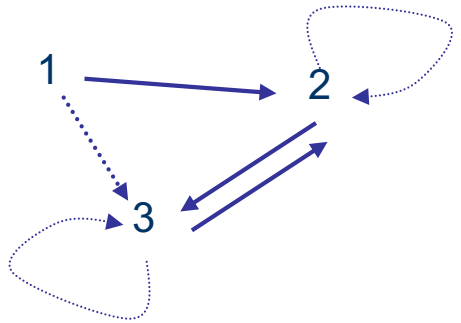
G

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

**(2) Übertrage Bindungen in den Regelkopf**

# Intuition

Transitive Hülle:




T

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |
| 1 | 3 |
| 2 | 2 |
| 3 | 3 |

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

G

| | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |

**... Wiederhole bis Fixpunkt erreicht**

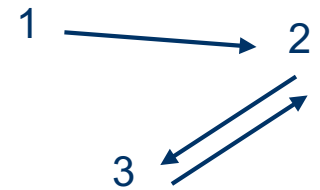
# Korrektheit

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

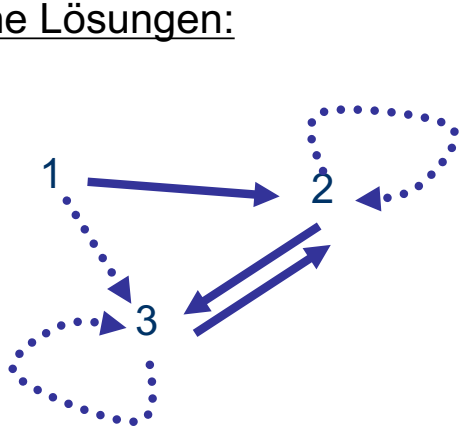
$\forall$ x,y,z ( T(x,y) $\leftarrow$ ( G(x,z) $\wedge$ T(z,y) ) )

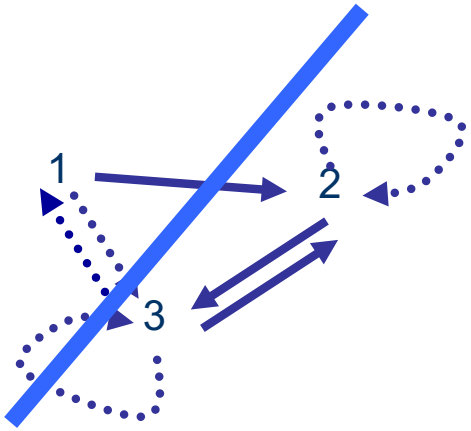
G( 1,2 ), G( 2,3 ), G( 3, 2 )

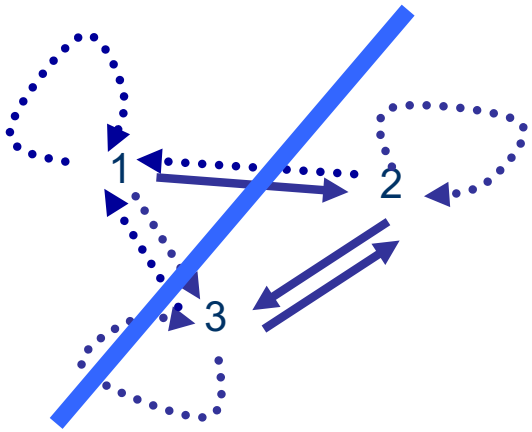
G:




Mögliche Lösungen:




→ ***Wähle minimales Modell*** ←

**T soll kleinste Menge von Fakten enthalten, so dass Regeln wahr sind**

# Pures Datalog: Terminologie

- Annahme: Extensionale Datenbank-Prädikate (EDB) gegeben
- Syntax: **`Kopf :-Rumpf`**
  - Gelesen: Wenn Rumpf, dann Kopf
- Atom = Prädikat angewandt auf Argumente (= Variable oder Konstante)

  (Bsp: T(x,y) oder G(a,x) )
- **`Kopf`** (Head) ist ein Atom
- **`Rumpf`** (Body) ist Konjunktion von Atomen
  - Konjunktion als Komma notiert
- Variablen im Kopf = ausgezeichnete (distinguished) Variablen

# Aufgabe:

Wie sind Regeln mit leerem Rumpf zu lesen?

# Aufgabe:

Wie sind Regeln mit leerem Rumpf zu lesen?

Lösung: Ein leerer Rumpf ist eine Konjunktion von 0 Atomen, das entspricht einer Tautologie (immer wahr). Also entspricht eine Regel mit leerem Rumpf einem Fakt.

T(a,b) :- entspricht T(a,b)

# Pures Datalog: Terminologie (Forts.)

- Atome des Rumpfs auch Teilziele (subgoals) genannt
- Prädikate, die im Kopf erscheinen, heißen intensionale Datenbank Prädikate (IDB)
- Teilziele können EDB- oder IDB-Prädikate enthalten
- Datalog Programm = endliche Menge von Regeln.
  - Ein IDB-Prädikat (meist Q) wird als Anfrage-/Antwortprädikat verstanden, dessen Extension die Antwortmenge präsentiert.

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

# Sichere (safe) Anfragen

- Unendliche Antwortmengen sollen verhindert werden
- Kann durch syntaktische Einschränkungen an Variablen realisiert werden
- Unsichere Anfrage:
  Verboten: `Colored_edges(x,y,col) :- G(x,y)`
  - Antwortmenge besteht aus Tripeln, bei denen das dritte Argument `col` nicht durch die Datenbank gebunden werden kann
  - Ist der Typ von `col` unendlich, bekäme man unendliche Antwortmenge

→Variablen im Kopf müssen im Rumpf vorkommen (Wertebereichsbeschränktheit)

# Sichere Anfragen

- Sicherheit kann zwar durch syntaktische Einschränkungen garantiert werden, aber schränkt zu sehr ein
- Semantisches Kriterium: Domänenunabhängigkeit (domain independence)
  - Antworten sollen nur von Datenbank und Anfrage (und nicht Wertebereichen (Typen)) abhängen
  - Problem: I.A. ist nicht entscheidbar, ob eine gegebene Anfrage domänenunabhängig ist, daher Kompromiss mit Syntaxeinschränkung für Variable

# Aufgabe:

Für die Sicherheit einer Anfrage ist nicht gefordert, dass Variablen im Rumpf auch im Kopf vorkommen müssen. Warum nicht? Welchen SQL-Konstrukten entsprächen solche Variablen?

# Aufgabe:

Für die Sicherheit einer Anfrage ist nicht gefordert, dass Variablen im Rumpf auch im Kopf vorkommen müssen. Warum nicht? Welchen SQL-Konstrukten entsprächen solche Variablen?

Lösung: Variablen im Rumpf, die nicht auch im Kopf vorkommen, sind von einem Existenzquantor gebunden. (Das Reinziehen vom Allquantor solcher Variablen macht sie zu einem Existenzquantor). Die dahinter steckende "harmlose" Operation ist die der Projektion: Diese Variablen entsprechen gerade den weg-projizierten Attributen. Wenn sie mehrfach im Rumpf auftreten, dann haben sie die Rolle von Verbundattributen, die weg-projiziert werden

# Semantik von purem Datalog

- 3 verschiedene Ansätze
  - Modelltheoretisch
  - Fixpunkttheoretisch
  - Beweistheoretisch

# Modelltheoretisch

- Regeln werden als logische Sätze verstanden, die die Menge möglicher Modelle einschränken
- Anfangsbeispiel zur transitiven Hülle
  - Q = T = Antwortprädikat

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

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

- Allerdings gibt es meistens mehrere Modelle
  - In Datalog betrachtet man das minimal Modell
  - (Minimalität bezieht sich auf Teilmengenbeziehung bzgl. der Extensionen der IDB-Prädikate)

# Warum minimales Modell?

- Hängt mit „**Closed-World-Assumption**“ zusammen
- Betrachte diejenigen Fakten, die nicht zwingend als „wahr“ beweisbar sind, als falsch
- Warnung: Die Wahl des minimalen Modells ist nicht selbstverständlich.

# Der Fixpunktansatz

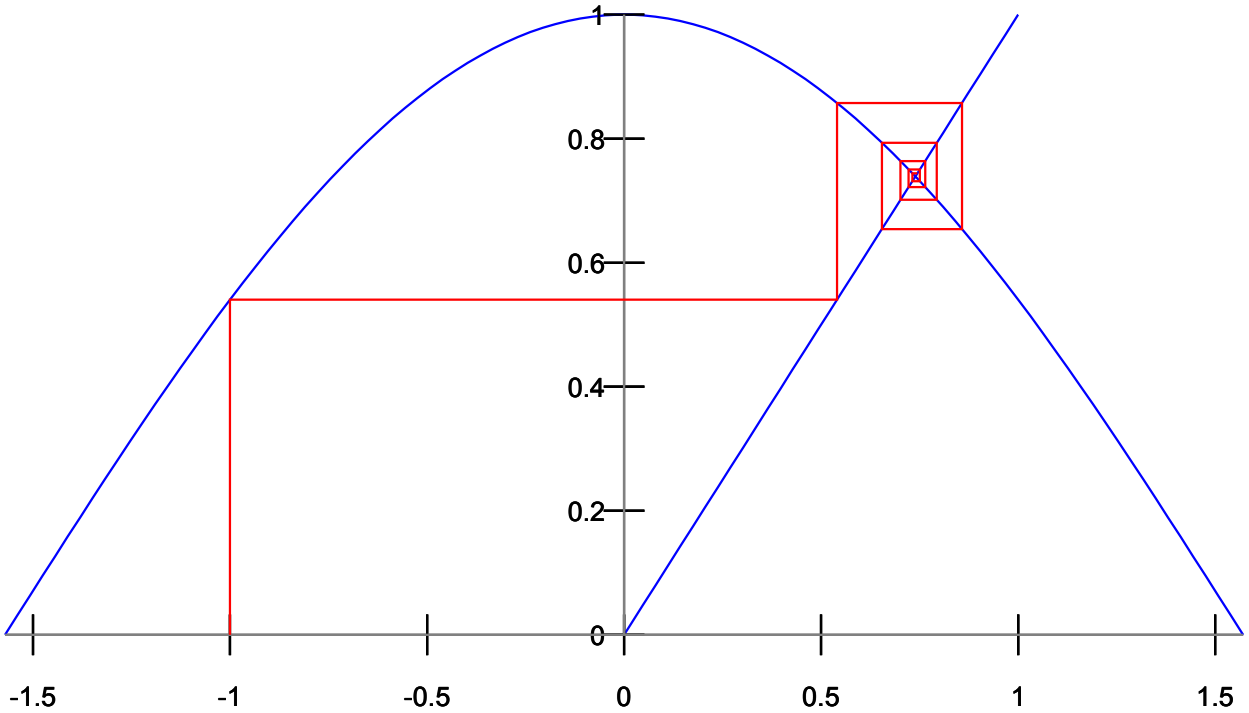
- Modelltheoretischer Ansatz ist „statisch“, gibt keine Möglichkeit zu Konstruktion von Lösungen

- Betrachte Semantik durch Auswahl eines Fixpunktes
  - (Siehe Graphbeispiel)

- Anfrage wird iteriert bis Fixpunkt erreicht $F(P) = P$ für IDB-Prädikat P

→Bottom-up Evaluation.

# Fixpunkt als Attraktor




"Cosine fixed point". Licensed under CC BY-SA 3.0 via Wikimedia Commons – https://commons.wikimedia.org/wiki/File:Cosine_fixed_point.svg\#/media/File:Cosine_fixed_point.svg

# Bottom-up Evaluation

- Starte mit den Fakten der Datenbankinstanz (EDB-Fakten)
- Verwende Regeln von rechts nach links, um neue Fakten abzuleiten
- Wiederhole bis keine Fakten mehr abgeleitet werden können

→Produziert alle (!) Fakten, die beweisbar sind.

# Bottom-up: Naive Evaluation

Gegeben sind EDB Fakten:

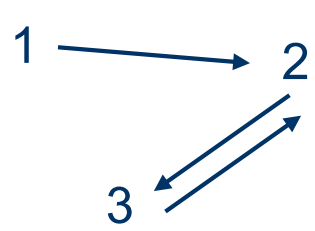
1. Initialisiere alle IDB Prädikate jeweils mit leerer Menge
2. Instanziiere mit Konstanten alle Variablen in allen Regeln auf alle (konsistenten) Weisen.
   Falls alle Teilziele (einer Regel) wahr werden, dann leite auch (instanziierten) Kopf der Regel ab.
3. Wiederhole (2) solange in Runden wie neue IDB-Fakten abgeleitet werden können

# Bottom-up: Naive Evaluation

Gegeben sind EDB Fakten:

1. Initialisiere alle IDB Prädikate jeweils mit leerer Menge
2. Instanziiere mit Konstanten alle Variablen in allen Regeln auf alle (konsistenten) Weisen.
   Falls alle Teilziele (einer Regel) wahr werden, dann leite auch (instanziierten) Kopf der Regel ab.
3. Wiederhole (2) in Runden solange wie neue IDB-Fakten abgeleitet werden können

G:




T:

1

2

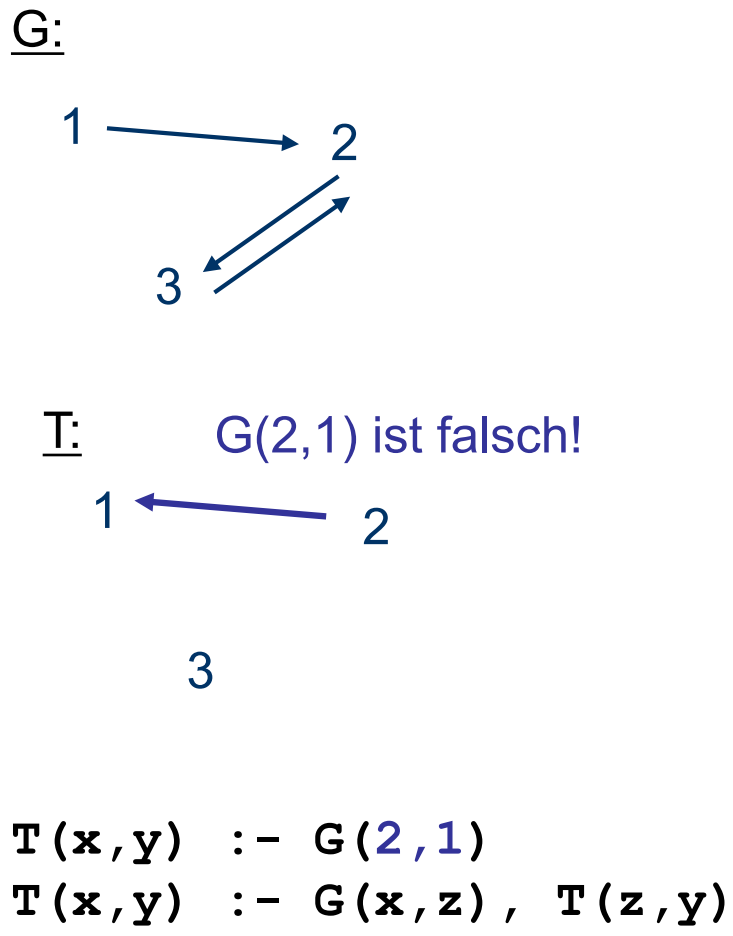
3

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

# Bottom-up: Naive Evaluation

Gegeben sind EDB Fakten:

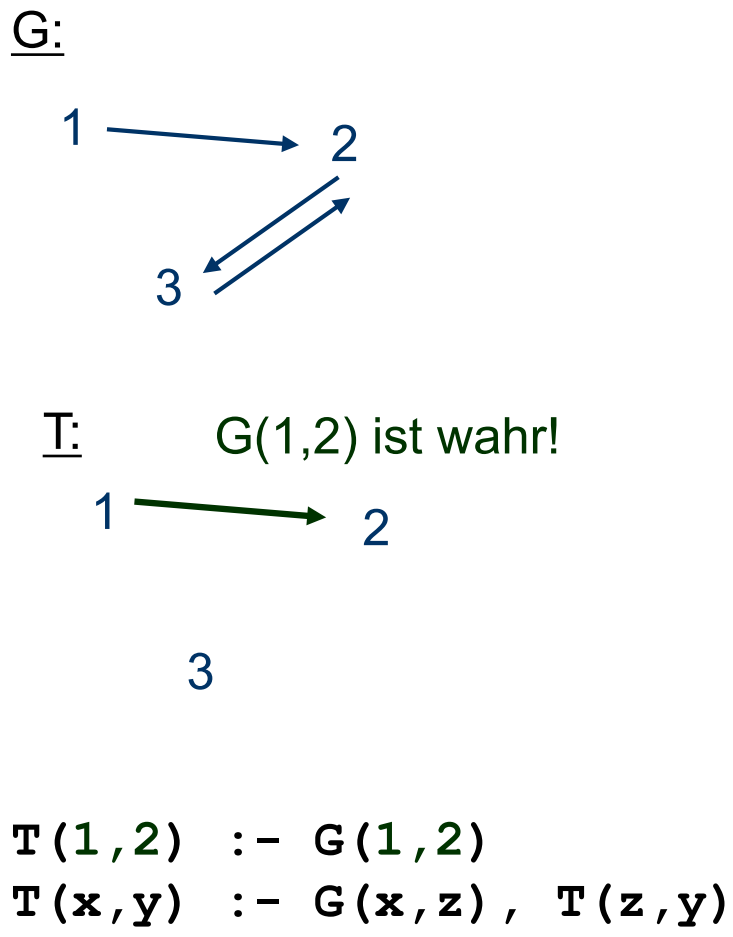
1. Initialisiere alle IDB Prädikate jeweils mit leerer Menge
2. Instanziiere mit Konstanten alle Variablen in allen Regeln auf alle (konsistenten) Weisen.
   Falls alle Teilziele (einer Regel) wahr werden, dann leite auch (instanziierten) Kopf der Regel ab.
3. Wiederhole (2) in Runden solange wie neue IDB-Fakten abgeleitet werden können



# Bottom-up: Naive Evaluation

Gegeben sind EDB Fakten:

1. Initialisiere alle IDB Prädikate jeweils mit leerer Menge
2. Instanziiere mit Konstanten alle Variablen in allen Regeln auf alle (konsistenten) Weisen.
   Falls alle Teilziele (einer Regel) wahr werden, dann leite auch (instanziierten) Kopf der Regel ab.
3. Wiederhole (2) in Runden solange wie neue IDB-Fakten abgeleitet werden können



# Bottom-up: Naive Evaluation

Gegeben sind EDB Fakten:

1. Initialisiere alle IDB Prädikate jeweils mit leerer Menge
2. Instanziiere mit Konstanten alle Variablen in allen Regeln auf alle (konsistenten) Weisen.
   Falls alle Teilziele (einer Regel) wahr werden, dann leite auch (instanziierten) Kopf der Regel ab.
3. Wiederhole (2) in Runden solange wie neue IDB-Fakten abgeleitet werden können

*(2) Ist sinnvoll (und gibt endlich viele Instanziierung)*
*solange Regeln sicher*

# Bottom-up: Naive Evaluation

Gegeben sind EDB Fakten:

1. Initialisiere alle IDB Prädikate jeweils mit leerer Menge
2. Instanziiere mit Konstanten alle Variablen in allen Regeln auf alle (konsistenten) Weisen.
   Falls alle Teilziele (einer Regel) wahr werden, dann leite auch (instanziierten) Kopf der Regel ab.
3. Wiederhole (2) in Runden solange wie neue IDB-Fakten abgeleitet werden können

*Algorithmus ergibt kleinsten (!) Fixpunkt der Regeln + EDB-Fakten.*

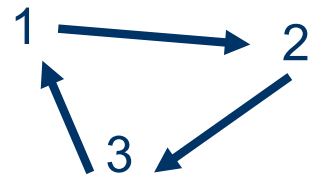
*Existenz garantiert wegen Monotonie: Extension der rekursiven IDB-Prädikate wächst! (Satz von Tarski-Knaster)*

# Bottom-up: Seminaive

- Effizienterer Ansatz zur Evaluierung der Regeln
  - Naive Evaluation reproduziert bereits abgeleitete Fakten
- Idee: Falls in Runde i ein neuer Fakt abgeleitet wurde, dann muss eine Regel genutzt worden sein, in der mindestens ein Teilziel auf solche Fakten angewandt wurde, die in Runde i-1 abgeleitet wurden.
- Für jedes IDB Prädikat **P** führe Buch über die in der letzten Runde abgeleiteten P-Fakten $\Delta\mathbf{P}$ aber auch über die aktuelle Extension **P** (da Ableitung auch auf „ältere“ Fakten zugreifen kann)

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

G:




1. Initialisiere IDB

| T | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |

2. Init. ΔIDB

| ΔT | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |

3.a.i ΔT(x,y) := G(x,z), ΔT(z,y)

| ΔT | |
|---|---|
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

3.a.ii ΔT := ΔT \ T

3.a.iii T := T ∪ ΔT

| T | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |

3.b.i ΔT(x,y) := G(x,z), ΔT(z,y)

| ΔT | |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

3.b.ii ΔT := ΔT \ T

3.b.iii T := T ∪ ΔT

| T | |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |
| 1 | 3 |
| 2 | 1 |
| 3 | 2 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

3.c

...

□

# Bottom-up: Seminaive (inkrementell)

1. Initialisiere IDB Prädikate unter Nutzung derjenigen Regeln, die keine IDB Teilziele enthalten
2. Initialisiere die $\Delta$-IDB Prädikate mit den korrespondierenden IDB-Prädikaten
3. In jeder Runde führe für jedes IDB-Prädikat P Folgendes aus:
   - Berechne neues $\Delta P$ durch Anwendung jeder Regel für P, indem ein Teilziel als $\Delta$-IDB-Prädikat gesehen wird und die anderen als IDB- oder EDB-Prädikat. (Man muss ja eventuell auf ältere Fakten zurückgreifen) Mache das für jede mögliche Wahl des $\Delta$–Teilziels.
   - Entferne vom neuen $\Delta P$ alle Fakten, die schon in $P$ sind.
   - $P := P \cup \Delta P$.
4. Wiederhole 3. bis keines der IDB-Prädikate sich ändert.

# Datalog mit Negation

- Erweiterung von Datalog mit Negation möglich
  1. Definition einer „sicheren Regel“ ist zu erweitern
  2. Negation hierarchisch auswertbar gestalten (Stratifikation)

- Ad 1: Jede Variable eines negativen Atomen muss in einem (eventuell jeweils anderen) positiven Atom des Regelrumpfs vorkommen

- D.h., Negation nur als „Filter“
  - Verboten: `T(x,y) :- ¬ H(x,y);`
  - Erlaubt: `T(x,y) :- ¬ K(x,y,w),G(x,w), R(w,y)`

# Ad 2: Stratifikation

- Negiertes Atom nicht in Schleife eines IDB-Prädikats
  - `P(x) :- Q(x), ¬ P(x)` Verboten
  - `Q(1), Q(2)`

- Problem: Semantik von solchen IDB-Prädikaten P nicht als kleinster Fixpunkt berechenbar (da nicht existent)

- In obigen Beispiel „eiert“ die aktuelle Extension von P hin und her zwischen zwei Extension (keine Monotonie; kein Fixpunkt)
  - Initial: P = {}
  - Runde 1 : P = {1,2}
  - Runde 2: P = {}
  - Runde 3: P = {1,2} etc.

# Ad 2: Stratifikation

- Negiertes Atom nicht in Schleife eines IDB-Prädikats
  - `R(x) :- Q(x), ¬ S(x)`
  - `S(x) :- Q(x), ¬ R(x)`
  - `Q(1), Q(2)`

Verboten

- Hier liegen zwei minimale Fixpunkte vor
  - Fixpunkt 1: $S_1 = \{\}$ , $R_1 = \{1,2\}$
  - Fixpunkt 2 : $S_2 = \{1,2\}$ , $R_2 = \{\}$
  - (S1,R1) und (S2,R2) nicht vergleichbar: Es gilt
    - weder $S1 \subseteq S2$ und $R1 \subseteq R2$
    - noch $S2 \subseteq S1$ und $R2 \subseteq R1$

# Stratum

- Stratum eines Prädikats P = maximale Anzahl von Negationen, die man auf ein IDB-Prädikat anwenden muss, um P zu evaluieren.

- Stratifiziertes Datalog: alle Prädikate haben endliches Stratum

- In obigem Beispiel

  ```
  P(x)   :- Q(x), ¬ P(x)
  ```

  würde man die Extension von P unendlich oft negieren

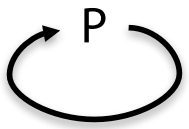
# Stratum-Graph

- Knoten = IDB-Prädikate.
- Kante *A* ->*B* falls Prädikat *A* abhängt von Prädikat *B* (B kommt in Rumpf eine Regel vor, deren Kopf A enthält).
- Kante ist mit "–" (für Negation) markiert, falls *B* negiertes Teilziel.

- Das Stratum eines Knoten (Prädikats) ist die maximale Zahl von – Kanten auf einem Pfad von diesem Knoten.
- Ein Datalog-Programm ist stratifiziert, falls alle IDB-Prädikate endliche Strata haben, sprich im Stratum-graph kein Zykel mit Negation vorkommt.

# Beispiel

```
P(x) <- Q(x) , NOT P(x)
```



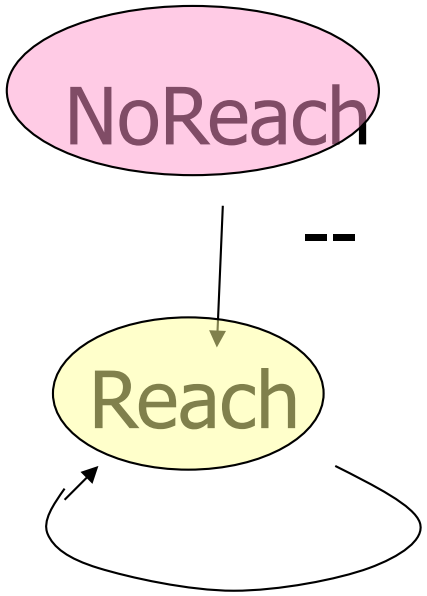

--

Man bekommt unendlichen Pfad: P -> P ->P -> P -> P -> ...
mit negierten markierten Labels

# Weiteres Beispiel

- EDB = **`Source(x), Target(x), Arc(x,y).`**
- Regeln die Ziele (**`Target(x)`**) beschreiben, die von keiner Source (**`Source(x)`**) erreichbar sind:
  - **`Reach(x) <- Source(x)`** Erlaubt
  - **`Reach(x) <- Reach(y), Arc(y,x)`**
  - **`NoReach(x) <- Target(x), NOT Reach(x)`**

Stratum 1:
<= 1 Negations-Kante auf jedem Pfad von NoReach.




Stratum 0:
Keine Negations-Kante von allen Pfaden ausgehend von Reach.

# Auswertung stratifizierter Programme

- Werte nacheinander Stratum-i-Prädikate aus, indem alle EDB-Prädikate und Stratum-(j)-Prädikate der unteren Strata $j < i$ als EDB-Prädikate gesehen werden
- $i = 0,1,...,m$

# Semantik von reinem Datalog

- 3 verschiedene Ansätze
  - Modelltheoretisch
  - Fixpunkttheoretisch
  - Beweistheoretisch

# Beweistheoretisch

Ein Fakt ist im Resultat, wenn es einen Beweis (eine Ableitung) aus den Fakten der EDB und der Regeln gibt.

(1) `T(x,y) :- G(x,y)`

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

i. G(1,2) in EDB

ii. T(1,2) gemäß (i) und Regel (1)

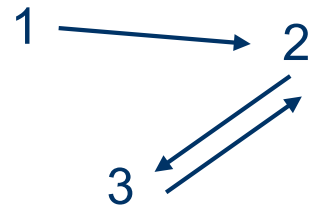
iii. G(2,3) in EDB

iv. T(2,3) gemäß (ii) und Regel (1)

v. T(1,3) gemäß (i), (iv) und Regel(2)

vi. ....

G:

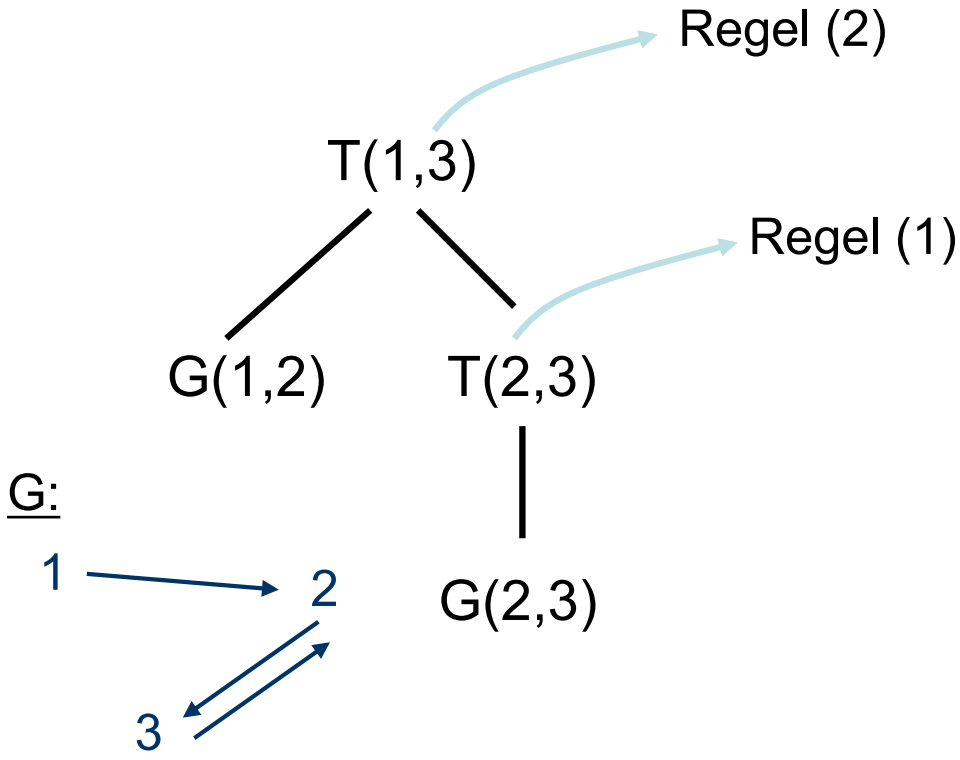


# Beweisbäume

Beweisbaum für ein Faktum **A** von EDB **I** und Datalog Program **P**:

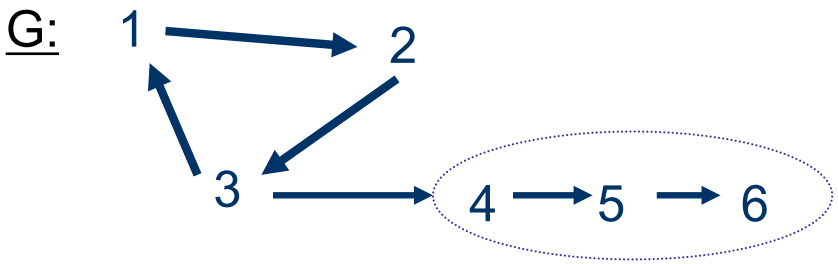
- Jeder Knoten ist ein Fakt
- Jedes Blatt ist Fakt aus **I**
- Wurzel ist **A**
- Vater-Kind-Kante gemäß einer Regel (Vater <- Kind1, Kind2, ..., Kind)

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



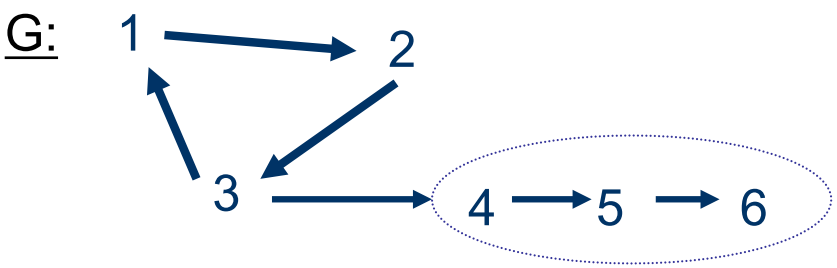
# Top-down Evaluation

- Materialisiere nicht alle möglichen Fakten, sondern gehe von konkreter Zielanfrage aus. Diese mag sehr viel konkreter sein, z.B. teilinstanziiertes T(4,x) statt T(x,y)



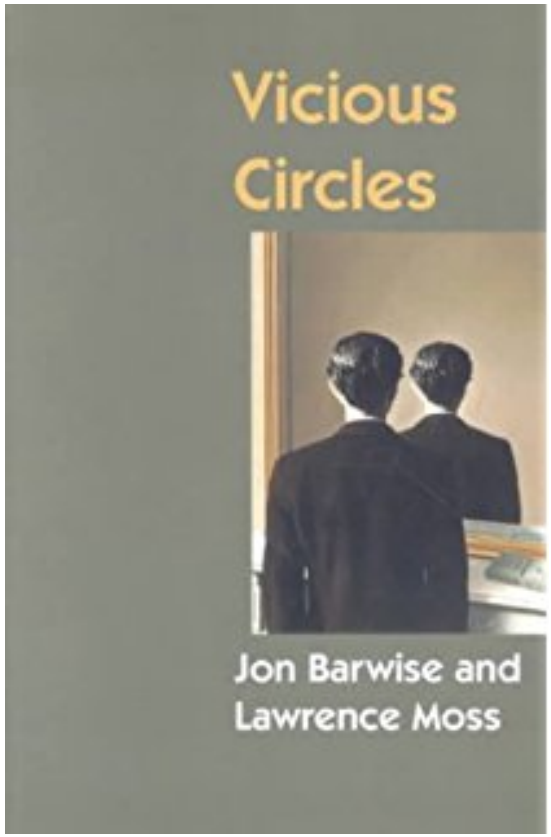
# Top-down Evaluation

- Materialisiere nicht alle möglichen Fakten, sondern gehe von konkreter Zielanfrage aus. Diese mag sehr viel konkreter sein, z.B. teilinstanziiertes T(4,x) statt T(x,y)




- Problem: unendliche Zyklen möglich bei Linksrekursion

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



# ... Magie reicht: Magic Set Algorithmus

Der Magic-Set-Algorithmus überführt eine Datalog-Anfrage Q in eine neue Anfrage Q‘ mit folgenden Eigenschaften:

- Q‘ gibt dieselbe Menge an Antworten wie Q
- Wenn Q‘ bottom-up ausgewertet wird, produziert es nur solche Fakten, die in einem Top-Down-Verfahren produziert würden.

→ Selektionen (spezifische Instanzen) werden von der Anfrage in die bottom-up-Berechnungen geschoben (mittels Magic-Prädikaten)

# Rule/Goal Graphs (RGGs)

- Nötig, um eindeutige Instanziierungsmuster für die IDB Prädikate zu gewährleisten
- Besteht aus Regel- und Ziel-Knoten.

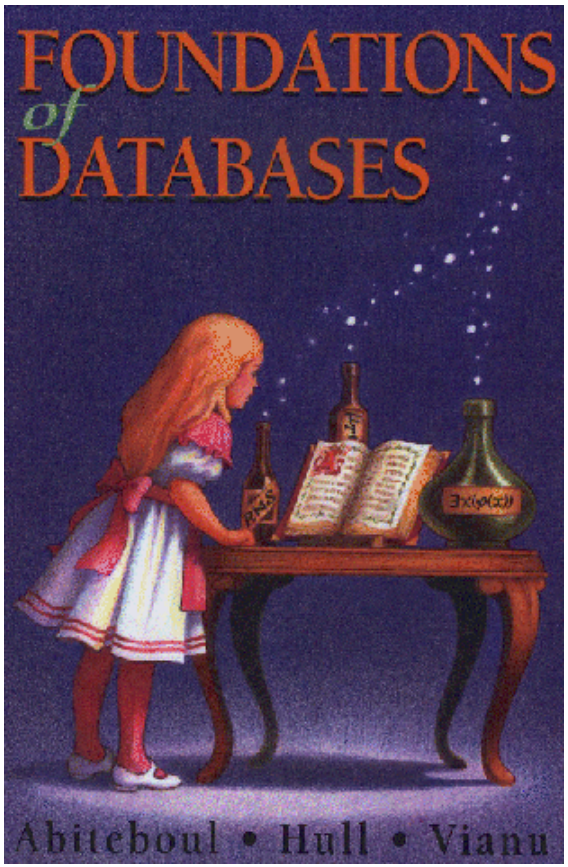
→Wir betrachten ein Beispiel;

für eine ausführliche Darstellung siehe

Abiteboul, Hull, Vianu:

Foundations of Databases,

Abschnitt 13.3 betitelt mit : „Magic“



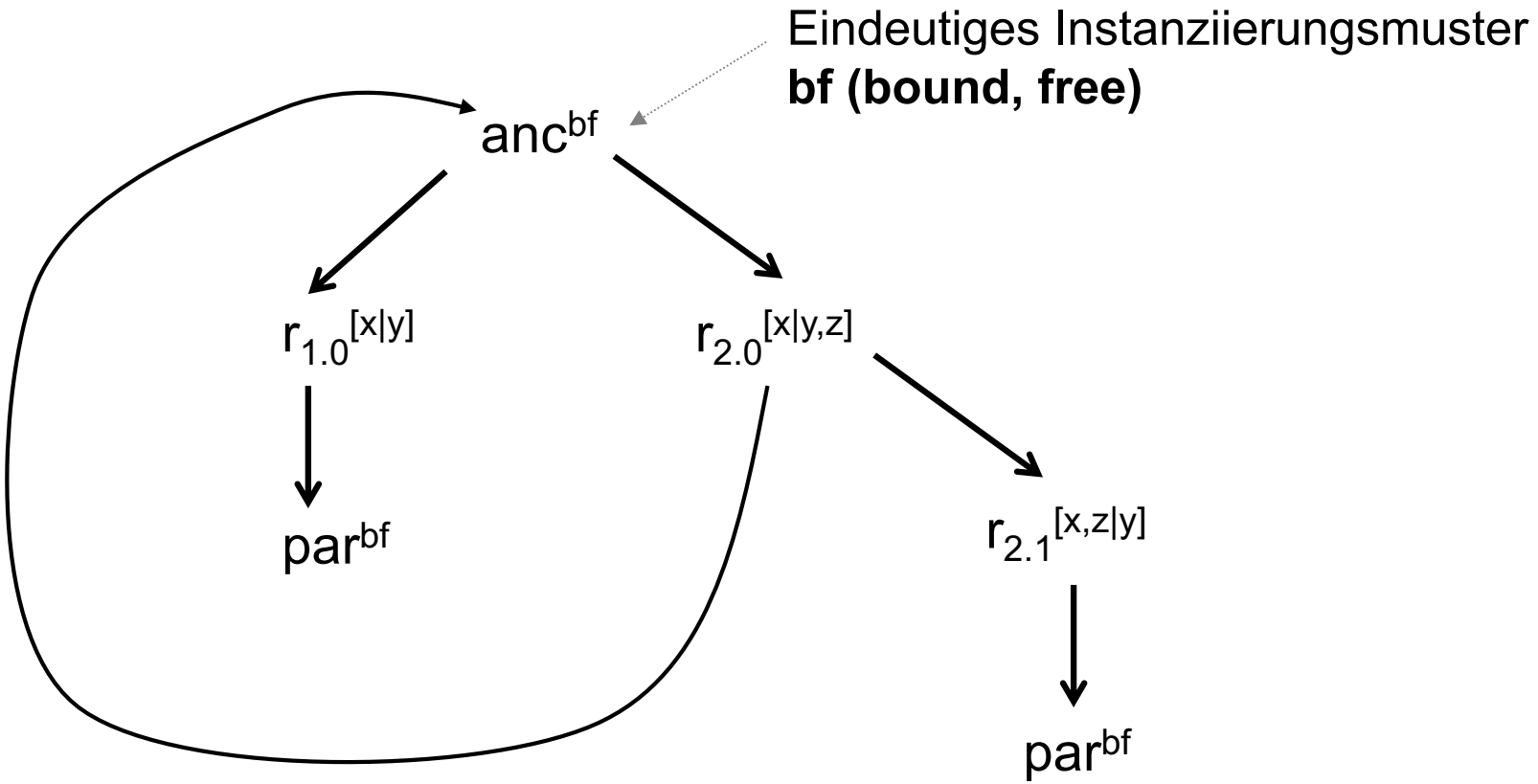
# Magic-Set-Transformation

Beginnend mit einem Datalog-Programm und einem Instanziierungsmuster aus der Anfrage führe folgende Schritte aus:

1. Splitte Prädikate, um eindeutige Instanziierungsmuster zu bekommen
2. Rektifiziere Teilziele (behebe mehrfache Vorkommnisse einer Variablen in einem IDB-Prädikat)
3. Führe Magie-Prädikate (und Hilfsprädikate) ein

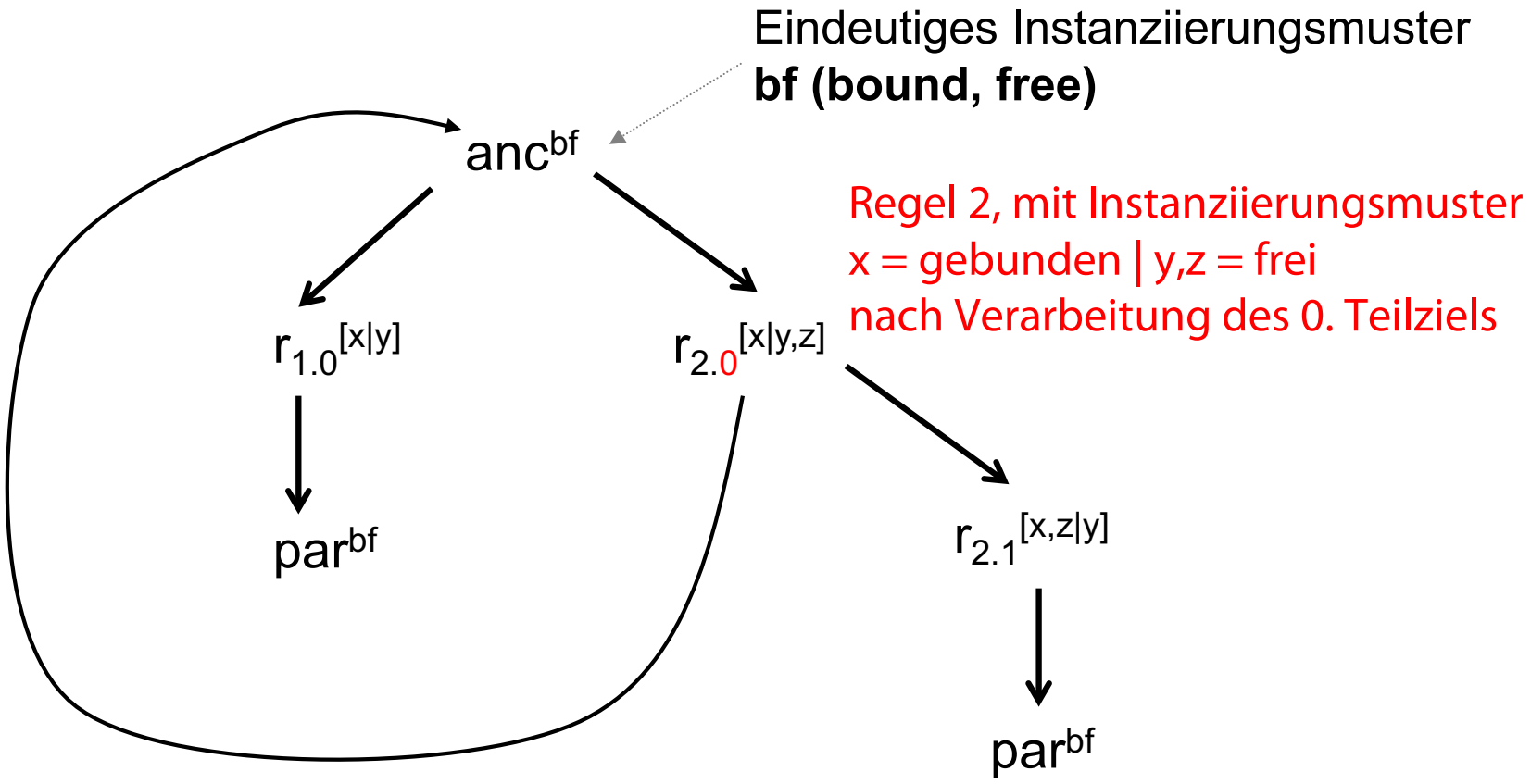
```
r1:          anc(x,y) :- par(x,y)
r2:          anc(x,y) :- anc(x,z), par(z,y)
Anfrage:     query(u) :- anc("a",u)
```

*a ist eine Konstante*



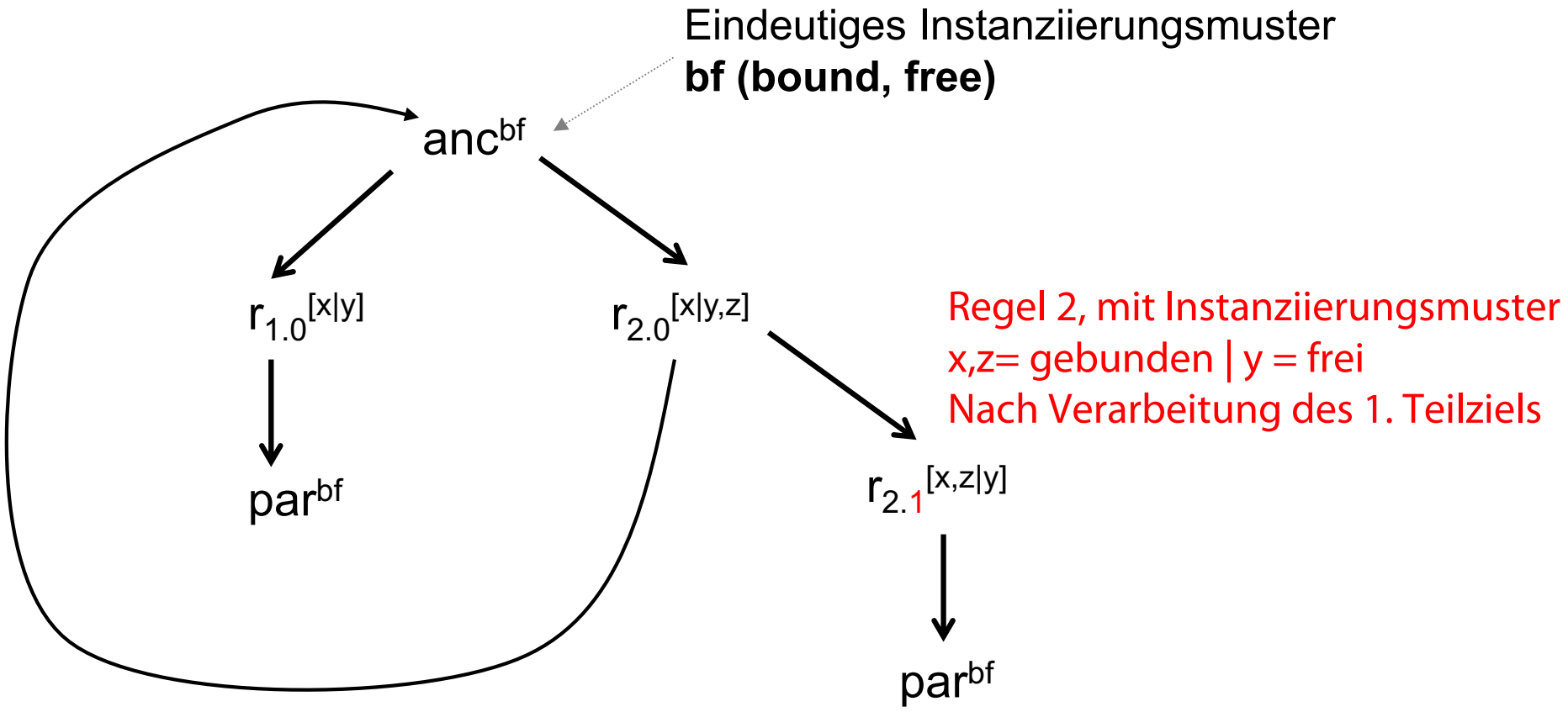
```
r1:          anc(x,y) :- par(x,y)
r2:          anc(x,y) :- anc(x,z), par(z,y)
Anfrage:     query(u) :- anc("a",u)
```

*a ist eine Konstante*



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

*a ist eine Konstante*



# Magie mit magischen Prädikaten

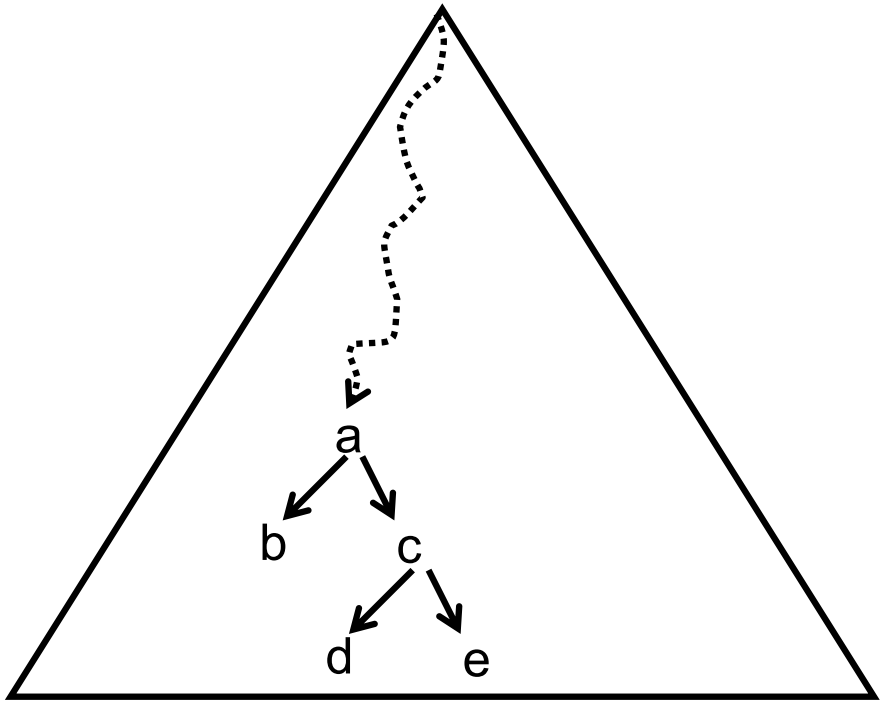
```
anc(x,y)  :- m_anc(x), par(x,y)
anc(x,y)  :- m_anc(x), anc(x,z), par(z,y)

m_anc("a")  :-
```

Dieses Programm lässt sich semi-naiv auswerten und produziert nur die Fakten, die für die Anfrage (mit der spezifischen Instanz „a“) nötig sind

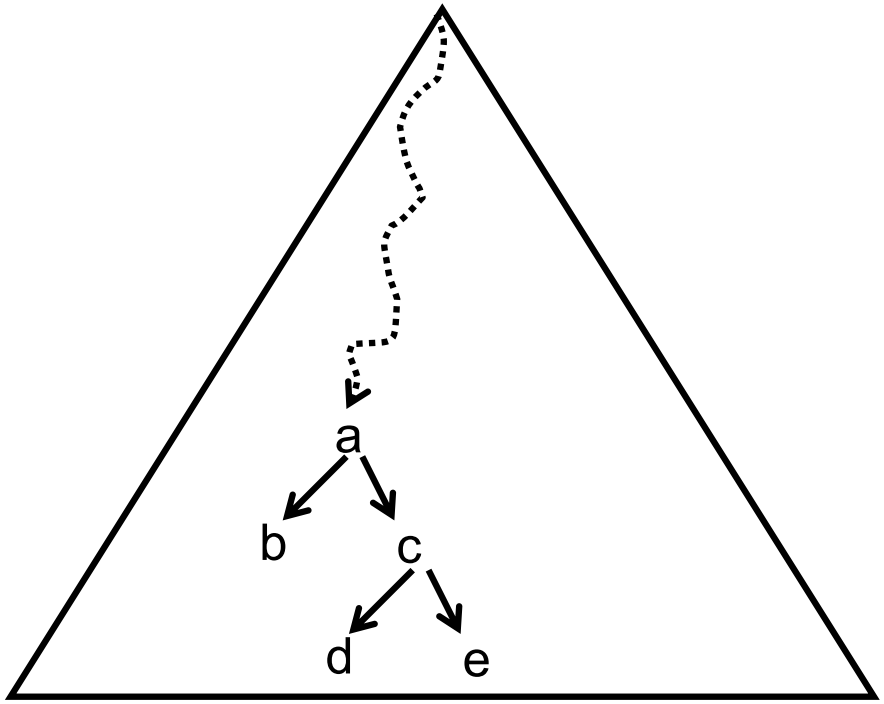
```
anc(x,y) :- m_anc(x), par(x,y)
anc(x,y) :- m_anc(x), anc(x,z), par(z,y)
m_anc("a") :-
```

par: m_anc___ anc___



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

par: m_anc___ anc___



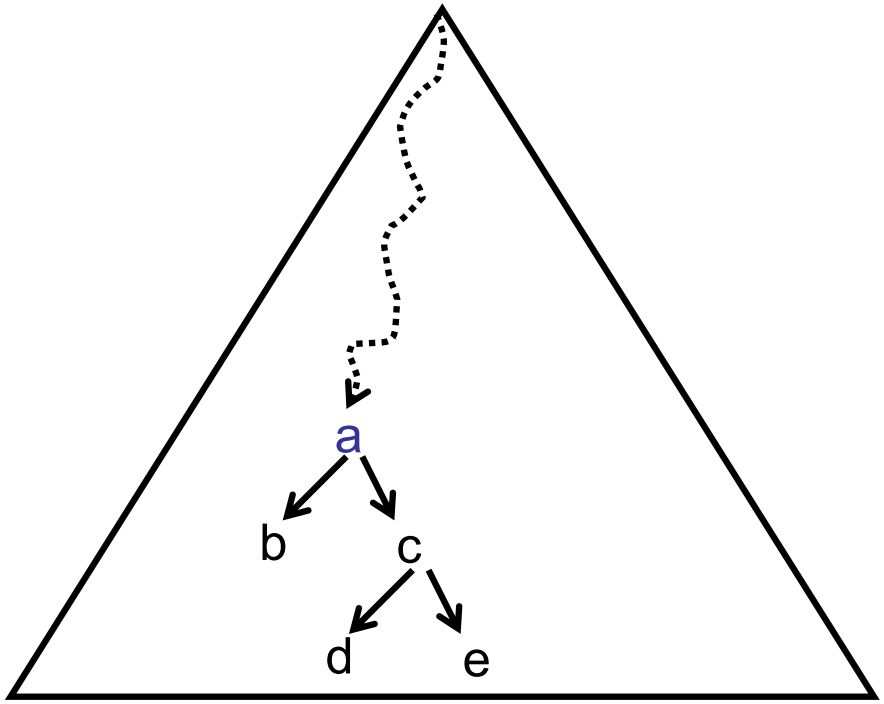
```
anc(x,y) :- m_anc(x), par(x,y)
anc(x,y) :- m_anc(x), anc(x,z), par(z,y)
m_anc("a") :-
```

par:

m_anc___

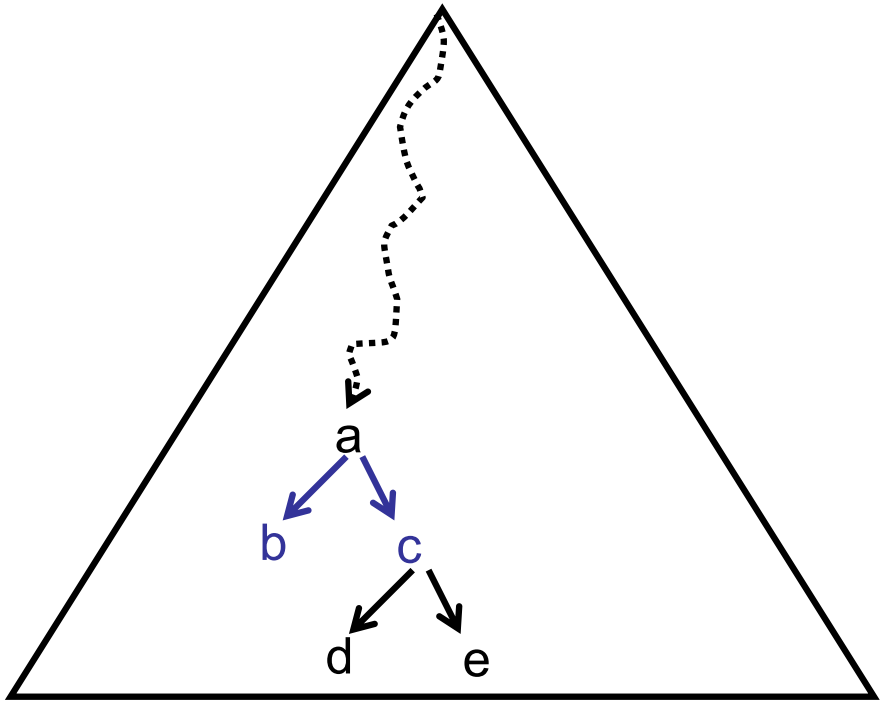
a

anc___



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

par:




m_anc___

a

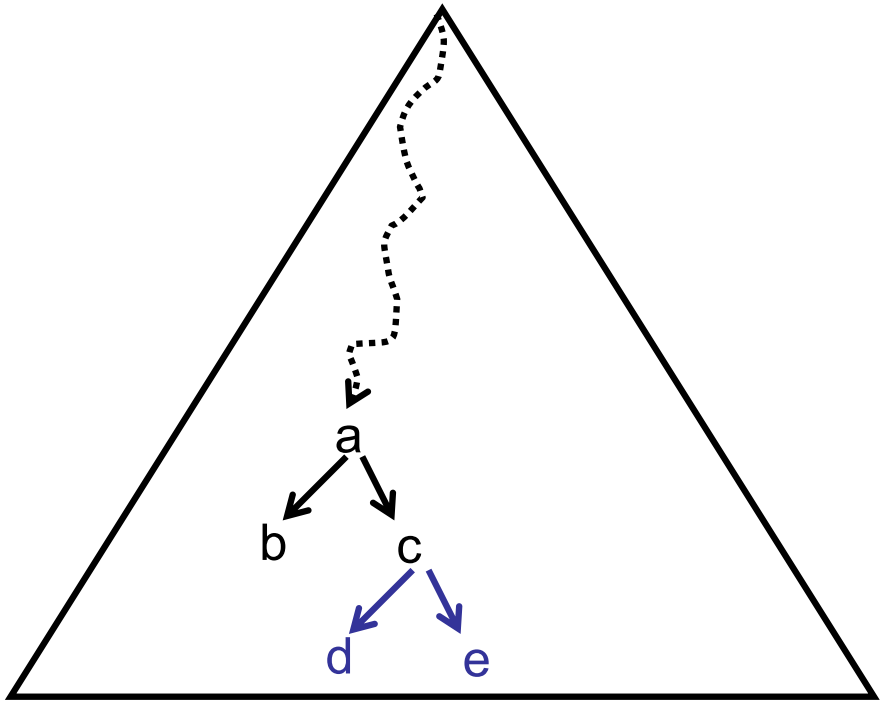
anc___

a | b

a | c

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

par




m_anc___

| |
|---|
| a |

anc___

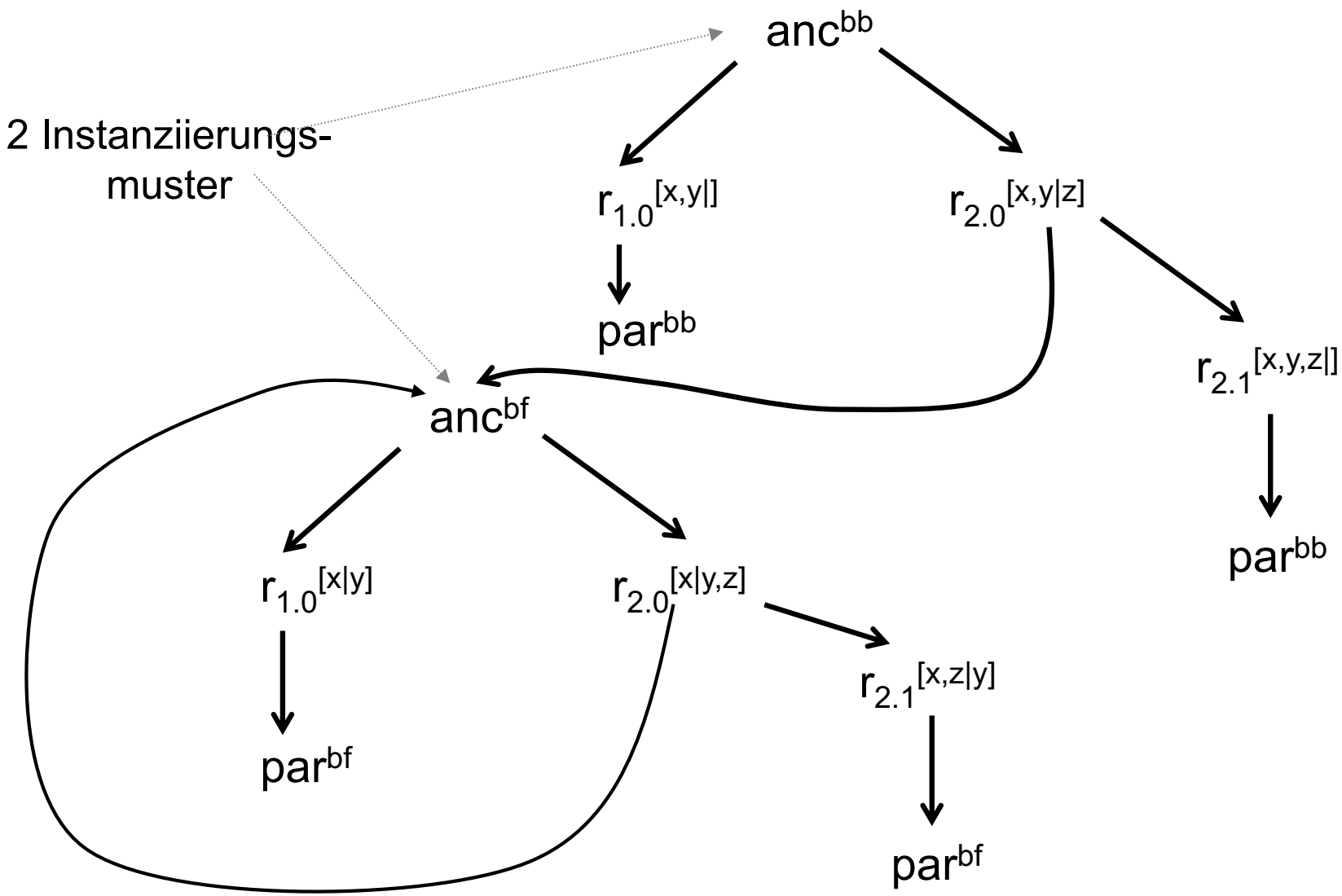
| | |
|---|---|
| a | b |
| a | c |
| a | d |
| a | e |

Die spezifische Instanziierung mit „a“ in der Anfrage wurde als Fakt **`m_anc("a") :-`** modelliert; und die Regeln wurden abhängig von dem magic-Prädikat **`m_anc`** gemacht.

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

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

Anfrage: `query() :- anc("a","b")`



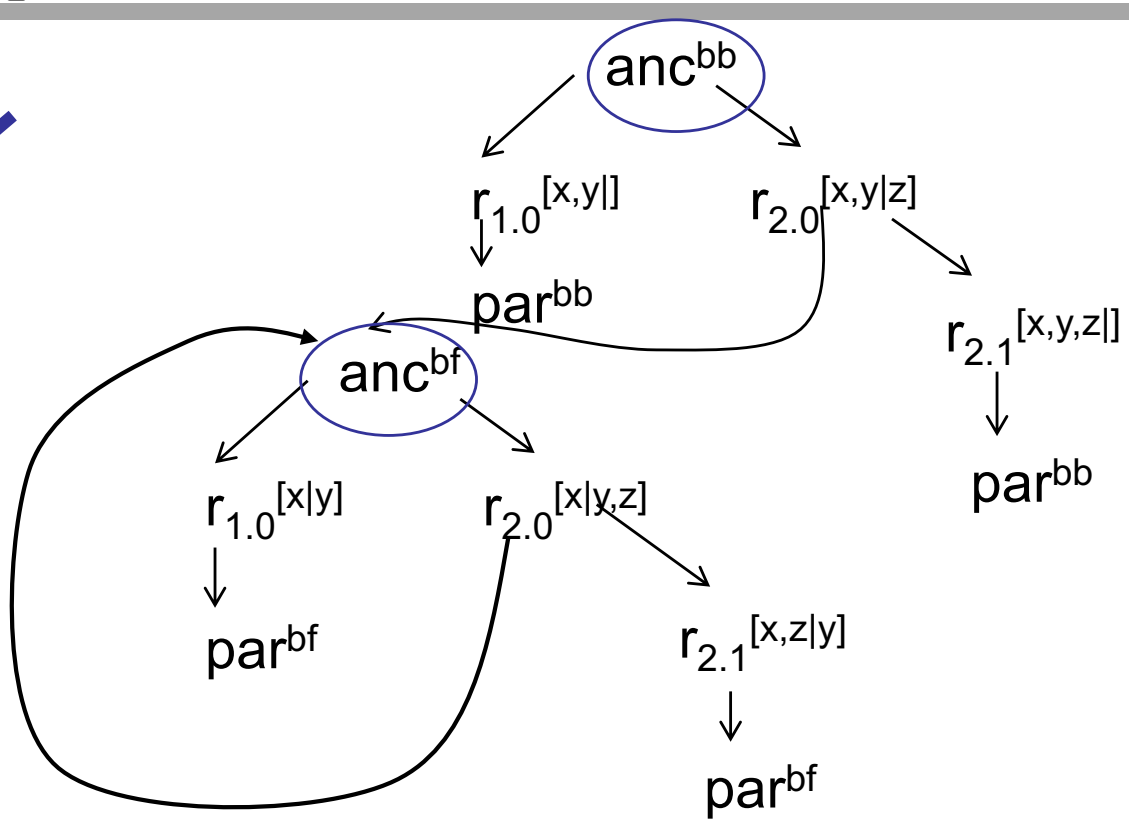
# Magic-Set-Transformation

Beginnend mit einem Datalog-Programm und einem Instanziierungsmuster aus der Anfrage führe folgende Schritte aus:

1. Splitte Prädikate, um eindeutige Instanziierungsmuster zu bekommen
2. Rektifiziere Teilziele (behebe mehrfache Vorkommnisse einer Variablen in einem IDB-Prädikat)
3. Führe Magie-Prädikate (und Hilfsprädikate) ein

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

Splitten für eindeutiges Instanziierungsmuster für Prädikate




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

# Magic-Set-Transformation

Beginnend mit einem Datalog-Programm und einem Instanziierungsmuster aus der Anfrage führe folgende Schritte aus:

1. Splitte Prädikate, um eindeutige Instanziierungsmuster zu bekommen
2. Rektifiziere Teilziele (behebe mehrfache Vorkommnisse einer Variablen in einem IDB-Prädikat)
3. Führe Magie-Prädikate (und Hilfsprädikate) ein

# Rektifizierung

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

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

- `p(z,z)` ist nicht rektifiziert
- Erstelle `q(z):- p(z,z).`
- Unifiziere Köpfe von Regeln mit mit `p(z,z)`. Es wird r1 zu

  `q(z) :- a(z,z)`

  und r2 zu

  `q(z) :- b(z,w),q(w),b(w,z)`
- Im ursprünglichen r2 ersetze Teilziel `p(z,z)` mit `q(z)`.

- Resultierende Regeln:

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

r2´: `p(x,y):- b(x,z),q(z),b(z,y)`

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

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

# Nächste Woche

- Zurück in den Kerker:
  - Kostenabschätzung zur Auswahl bester Pläne

# Nächste Woche

- Zurück in den Kerker:
  - Kostenabschätzung zur Auswahl bester Pläne

Seven Q + three Q for your attention!