Updated 2/7/2024

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 8

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

2/6/2024

Where We Are

- Relational query languages we have seen so far:
  - SQL
  - Relational Calculus
  - Relational Algebra
- They can express the same class of relational queries (ignoring extensions, such as grouping, aggregates, or sorting)
  - How powerful are they? What kind of useful queries are missing?

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

?

- Find all people who are friends with everyone who is not a friend of Bob

?

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

NO: needs higher math; not possible with RA
(unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

?

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

NO: needs higher math; not possible with RA
(unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y) \}$ DI?

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

NO: needs higher math; not possible with RA
(unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y) \}$

DI?

$\{x \mid \text{Person}(x) \wedge \forall y.[\text{Person}(y) \wedge \neg\text{Friend}(y,\text{'Bob'}) \Rightarrow \text{Friend}(x,y)]\}$

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

?

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

NO: needs higher math; not possible with RA
(unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y) \}$

$\{x \mid \text{Person}(x) \wedge \forall y.[\text{Person}(y) \wedge \neg\text{Friend}(y,\text{'Bob'}) \Rightarrow \text{Friend}(x,y)]\}$

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

NO: equivalent to 3-coloring; NP-complete

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

?

Source: Dan Suciu, CSE 554, 2011.

Which are Relational Queries? Which are not? And Why?

- Given Friend(X,Y): Find all people X whose number of friends is a prime number

NO: needs higher math; not possible with RA
(unless we have access to a relation Prime(x)...)

- Find all people who are friends with everyone who is not a friend of Bob

YES: $\{x \mid \forall y.(\neg \text{Friend}(y, \text{'Bob'}) \Rightarrow \text{Friend}(x,y)\ \}$

$\{x \mid \text{Person}(x)\ \wedge\ \forall y.[\text{Person}(y)\ \wedge\ \neg\text{Friend}(y,\text{'Bob'}) \Rightarrow \text{Friend}(x,y)]\}$

- Partition all people into three sets P1(X),P2(X),P3(X) s.t. any two friends are in different partitions

NO: equivalent to 3-coloring; NP-complete

- Find all people who are direct or indirect friends with Alice (connected in arbitrary length)

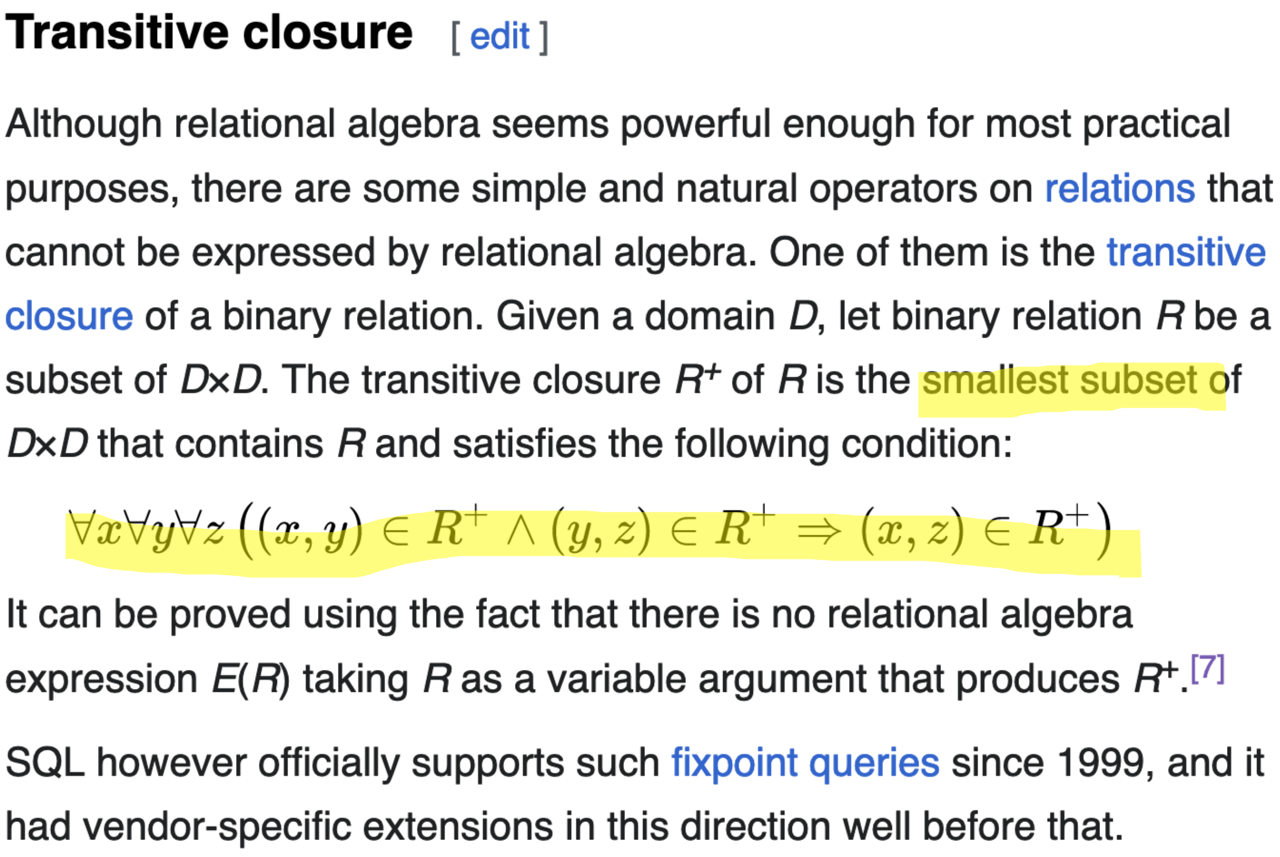
NO: recursive query; PTIME yet not expressible in RA

Next: Datalog: extends monotone RA with recursion

Source: Dan Suciu, CSE 554, 2011.

Transitive closure (not expressible with RA)

THEOREM: Datalog can express queries that RA (RC) cannot (e.g., transitive closure of a graph)



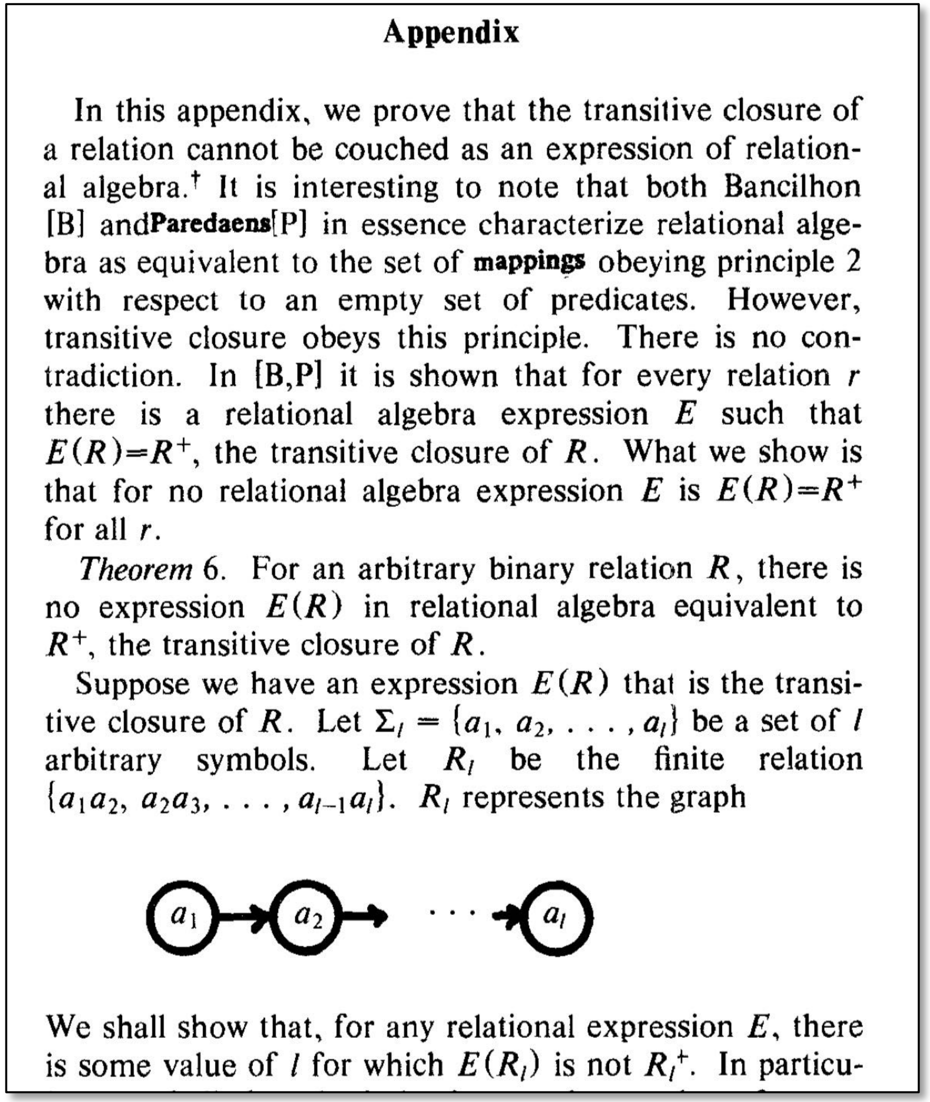
Transitive closure [edit]

Although relational algebra seems powerful enough for most practical purposes, there are some simple and natural operators on relations that cannot be expressed by relational algebra. One of them is the transitive closure of a binary relation. Given a domain *D*, let binary relation *R* be a subset of *D*×*D*. The transitive closure R^+ of *R* is the smallest subset of *D*×*D* that contains *R* and satisfies the following condition:

$$\forall x \forall y \forall z \left((x,y) \in R^+ \wedge (y,z) \in R^+ \Rightarrow (x,z) \in R^+ \right)$$

It can be proved using the fact that there is no relational algebra expression *E*(*R*) taking *R* as a variable argument that produces R^+.[7]

SQL however officially supports such fixpoint queries since 1999, and it had vendor-specific extensions in this direction well before that.



Appendix

In this appendix, we prove that the transitive closure of a relation cannot be couched as an expression of relational algebra.† It is interesting to note that both Bancilhon [B] and Paredaens [P] in essence characterize relational algebra as equivalent to the set of mappings obeying principle 2 with respect to an empty set of predicates. However, transitive closure obeys this principle. There is no contradiction. In [B,P] it is shown that for every relation r there is a relational algebra expression E such that $E(R){=}R^+$, the transitive closure of R. What we show is that for no relational algebra expression E is $E(R){=}R^+$ for all r.

Theorem 6. For an arbitrary binary relation R, there is no expression $E(R)$ in relational algebra equivalent to R^+, the transitive closure of R.

Suppose we have an expression $E(R)$ that is the transitive closure of R. Let $\Sigma_l = \{a_1, a_2, \ldots, a_l\}$ be a set of l arbitrary symbols. Let R_l be the finite relation $\{a_1a_2, a_2a_3, \ldots, a_{l-1}a_l\}$. R_l represents the graph

We shall show that, for any relational expression E, there is some value of l for which $E(R_l)$ is not R_l^+. In particu-

Source: https://en.wikipedia.org/wiki/Relational_algebra#Transitive_closure

Appendix from: Aho, Ullman. "Universality of data retrieval languages". POPL 1979. https://doi.org/10.1145%2F567752.567763

Datalog & ASP

```
Path(x,y) :- Arc(x,y).
Path(x,z) :- Arc(x,y), Path(y,z).
InCycle(x) :- Path(x,x).
```

- Datalog
  - Database query language designed in the 80’s
  - Simple, concise, elegant
    - "Clean" (syntactic) restriction of Prolog with DB access
    - Expressive & declarative: Set-of-rules semantics, Independence of execution order, Invariance under logical equivalence
  - Several open source implementations, mostly academic implementations
  - Recently a hot topic, beyond databases:
    - network protocols, static program analysis, DB+ML

Soufflé

LogicBlox

RelationalAI

- Answer Set Programming (ASP):
  - very powerful extension (with negation) that can model hard computational problems

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

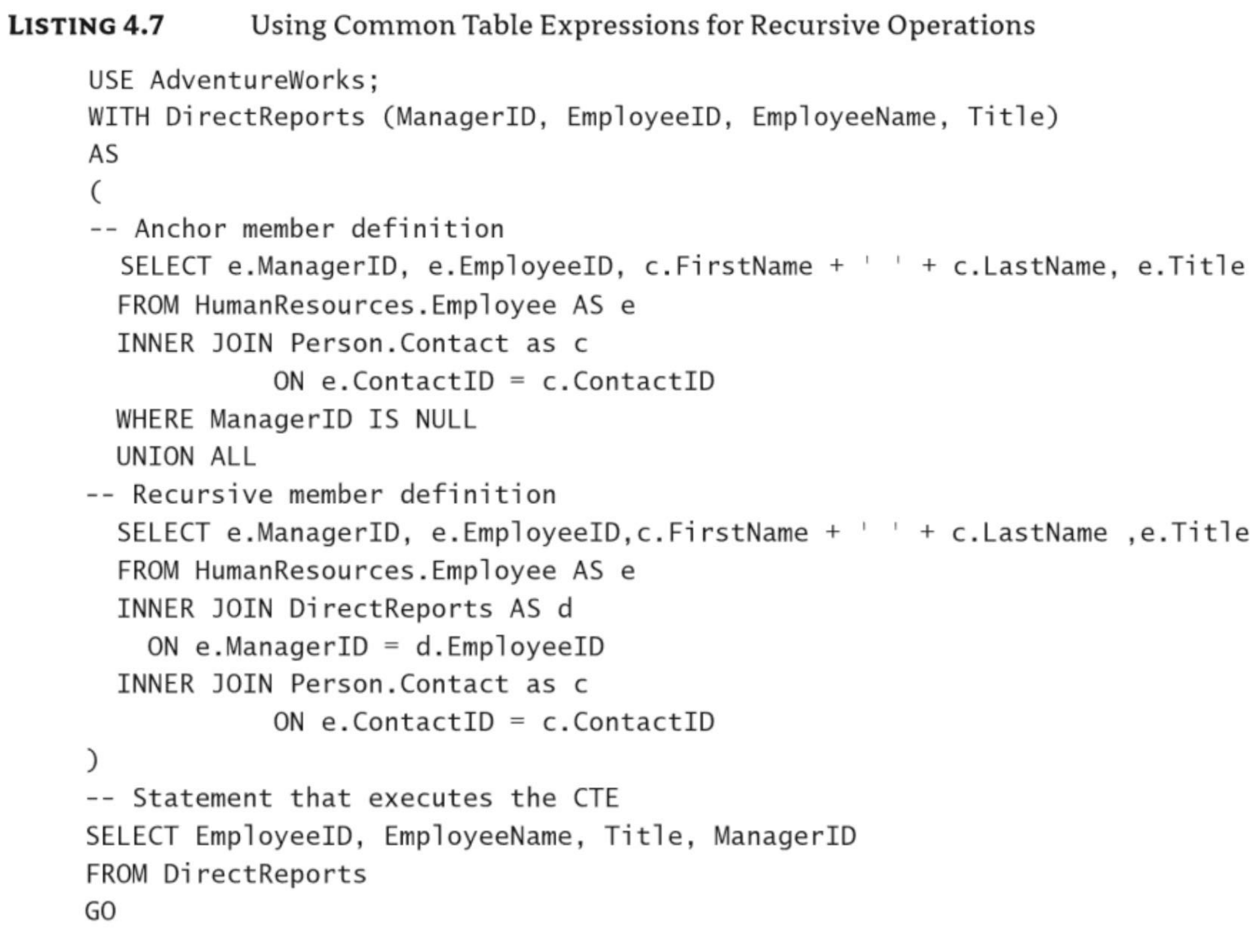
Potassco Solutions

Originally based on slides by Dan Suciu

We will later see and use in class: Souffle (https://souffle-lang.github.io/simple) and Postassco/Clingo: Download: https://potassco.org/clingo/, Running in the browser: https://potassco.org/clingo/run/, More resources on clingo: https://teaching.potassco.org/

Recursion with SQL server vs. Datalog

SQL



LISTING 4.7 Using Common Table Expressions for Recursive Operations

```
USE AdventureWorks;
WITH DirectReports (ManagerID, EmployeeID, EmployeeName, Title)
AS
(
-- Anchor member definition
  SELECT e.ManagerID, e.EmployeeID, c.FirstName + ' ' + c.LastName, e.Title
  FROM HumanResources.Employee AS e
  INNER JOIN Person.Contact as c
            ON e.ContactID = c.ContactID
  WHERE ManagerID IS NULL
  UNION ALL
-- Recursive member definition
  SELECT e.ManagerID, e.EmployeeID,c.FirstName + ' ' + c.LastName ,e.Title
  FROM HumanResources.Employee AS e
  INNER JOIN DirectReports AS d
    ON e.ManagerID = d.EmployeeID
  INNER JOIN Person.Contact as c
            ON e.ContactID = c.ContactID
)
-- Statement that executes the CTE
SELECT EmployeeID, EmployeeName, Title, ManagerID
FROM DirectReports
GO
```

Datalog

Manager(eid) :- Manages(_, eid)

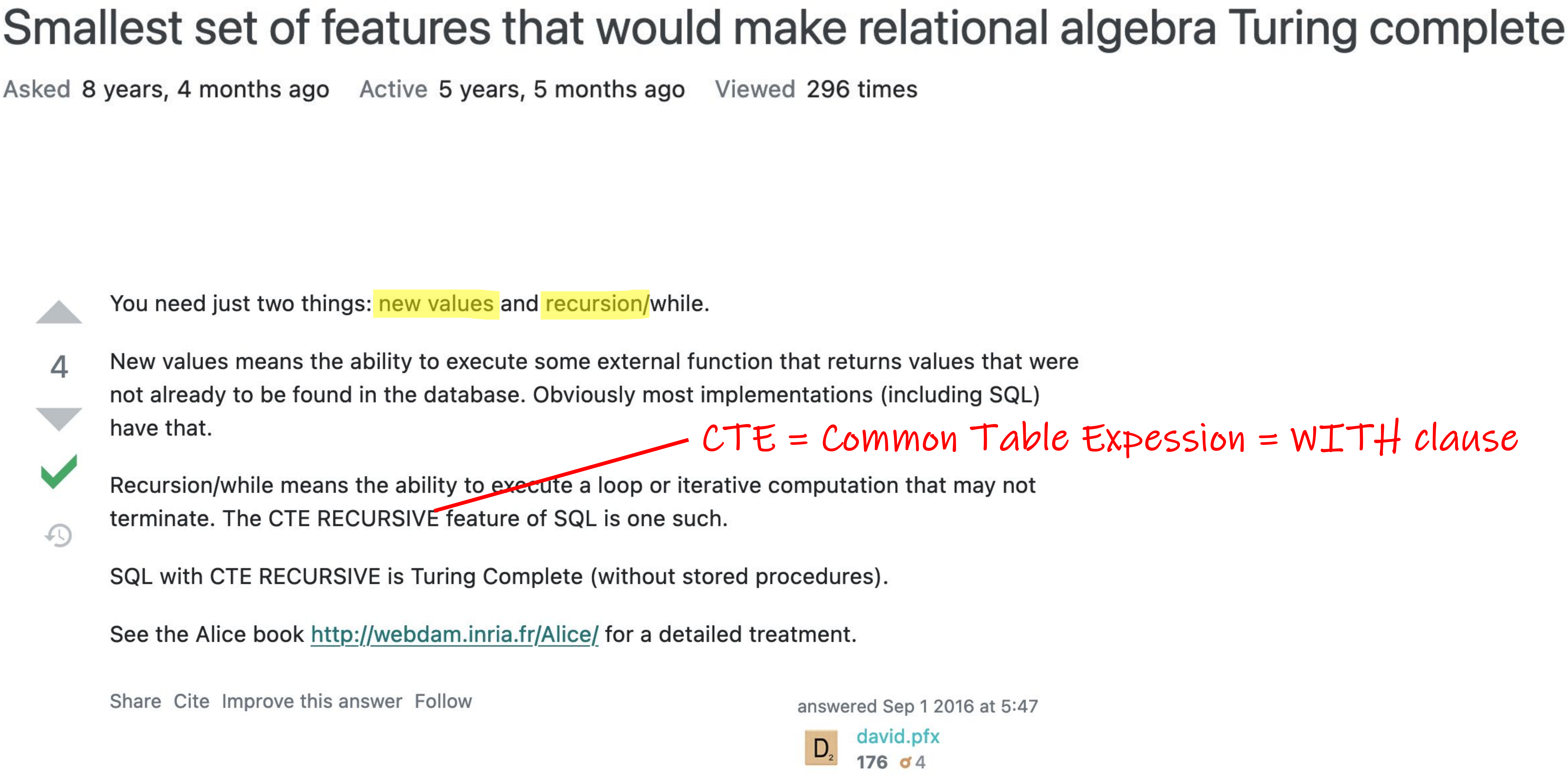
DirectReports(eid, 0) :-
    Employee(eid), not Manager(eid)

DirectReports(eid, level+1) :-
    DirectReports(mid, level), Manages(mid, eid)

SQL Query vs. Datalog: which would you rather write?

Possible scribe: to fix that example ☺

Query on the left from Bieker, Lee. Mastering SQL server 2008. Example on the right by Dan Suciu



Smallest set of features that would make relational algebra Turing complete

Asked 8 years, 4 months ago Active 5 years, 5 months ago Viewed 296 times

4

You need just two things: new values and recursion/while.

New values means the ability to execute some external function that returns values that were not already to be found in the database. Obviously most implementations (including SQL) have that.

Recursion/while means the ability to execute a loop or iterative computation that may not terminate. The CTE RECURSIVE feature of SQL is one such.

SQL with CTE RECURSIVE is Turing Complete (without stored procedures).

See the Alice book http://webdam.inria.fr/Alice/ for a detailed treatment.

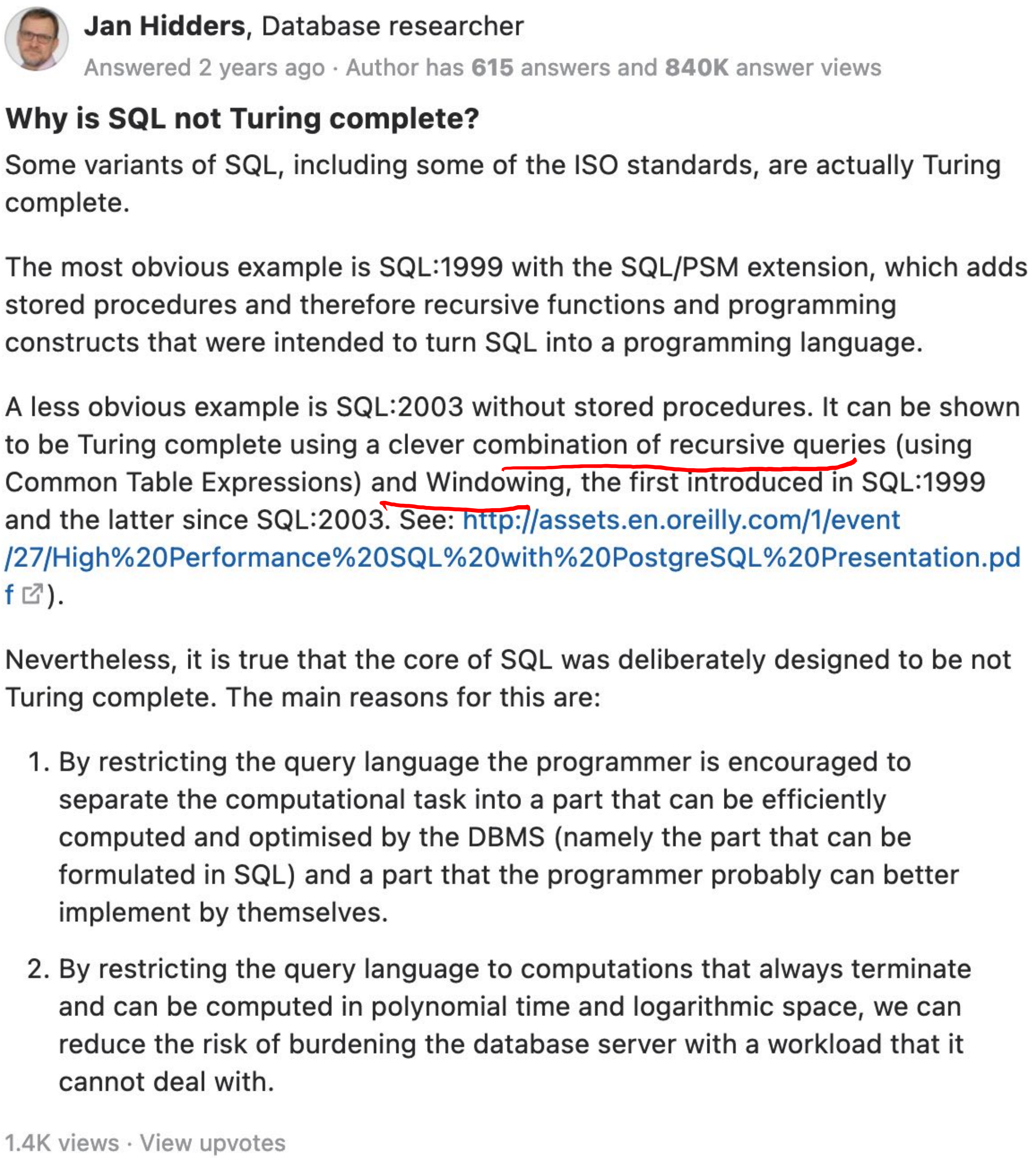
answered Sep 1 2016 at 5:47

david.pfx

176 ●4

CTE = Common Table Expession = WITH clause

https://cs.stackexchange.com/questions/14694/smallest-set-of-features-that-would-make-relational-algebra-turing-complete



Jan Hidders, Database researcher

Answered 2 years ago · Author has **615** answers and **840K** answer views

Why is SQL not Turing complete?

Some variants of SQL, including some of the ISO standards, are actually Turing complete.

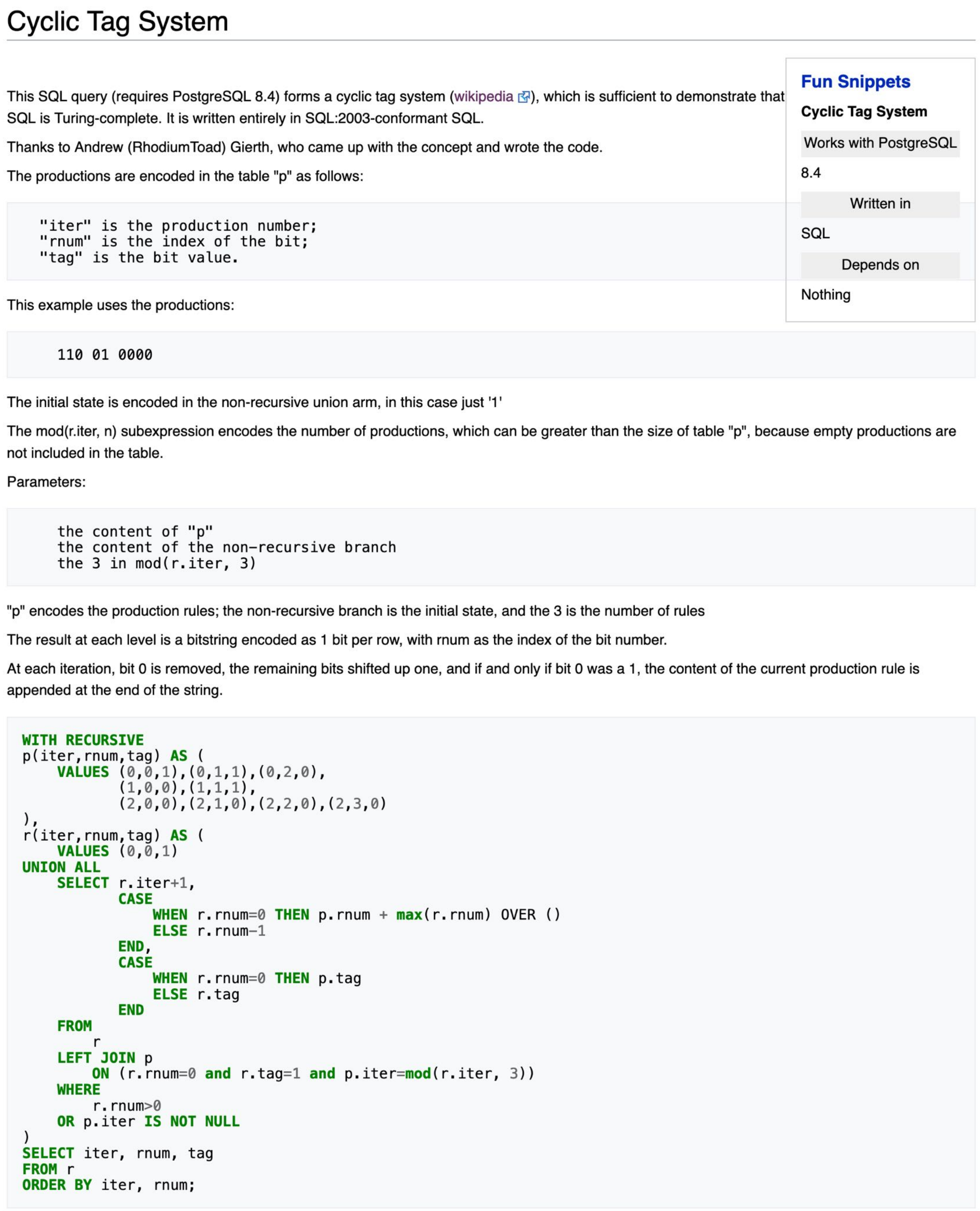
The most obvious example is SQL:1999 with the SQL/PSM extension, which adds stored procedures and therefore recursive functions and programming constructs that were intended to turn SQL into a programming language.

A less obvious example is SQL:2003 without stored procedures. It can be shown to be Turing complete using a clever combination of recursive queries (using Common Table Expressions) and Windowing, the first introduced in SQL:1999 and the latter since SQL:2003. See: http://assets.en.oreilly.com/1/event/27/High%20Performance%20SQL%20with%20PostgreSQL%20Presentation.pdf).

Nevertheless, it is true that the core of SQL was deliberately designed to be not Turing complete. The main reasons for this are:

1. By restricting the query language the programmer is encouraged to separate the computational task into a part that can be efficiently computed and optimised by the DBMS (namely the part that can be formulated in SQL) and a part that the programmer probably can better implement by themselves.
2. By restricting the query language to computations that always terminate and can be computed in polynomial time and logarithmic space, we can reduce the risk of burdening the database server with a workload that it cannot deal with.

1.4K views · View upvotes



Cyclic Tag System

This SQL query (requires PostgreSQL 8.4) forms a cyclic tag system (wikipedia), which is sufficient to demonstrate that SQL is Turing-complete. It is written entirely in SQL:2003-conformant SQL.

Thanks to Andrew (RhodiumToad) Gierth, who came up with the concept and wrote the code.

The productions are encoded in the table "p" as follows:

```
"iter" is the production number;
"rnum" is the index of the bit;
"tag" is the bit value.
```

This example uses the productions:

```
110 01 0000
```

The initial state is encoded in the non-recursive union arm, in this case just '1'

The mod(r.iter, n) subexpression encodes the number of productions, which can be greater than the size of table "p", because empty productions are not included in the table.

Parameters:

```
the content of "p"
the content of the non-recursive branch
the 3 in mod(r.iter, 3)
```

"p" encodes the production rules; the non-recursive branch is the initial state, and the 3 is the number of rules

The result at each level is a bitstring encoded as 1 bit per row, with rnum as the index of the bit number.

At each iteration, bit 0 is removed, the remaining bits shifted up one, and if and only if bit 0 was a 1, the content of the current production rule is appended at the end of the string.

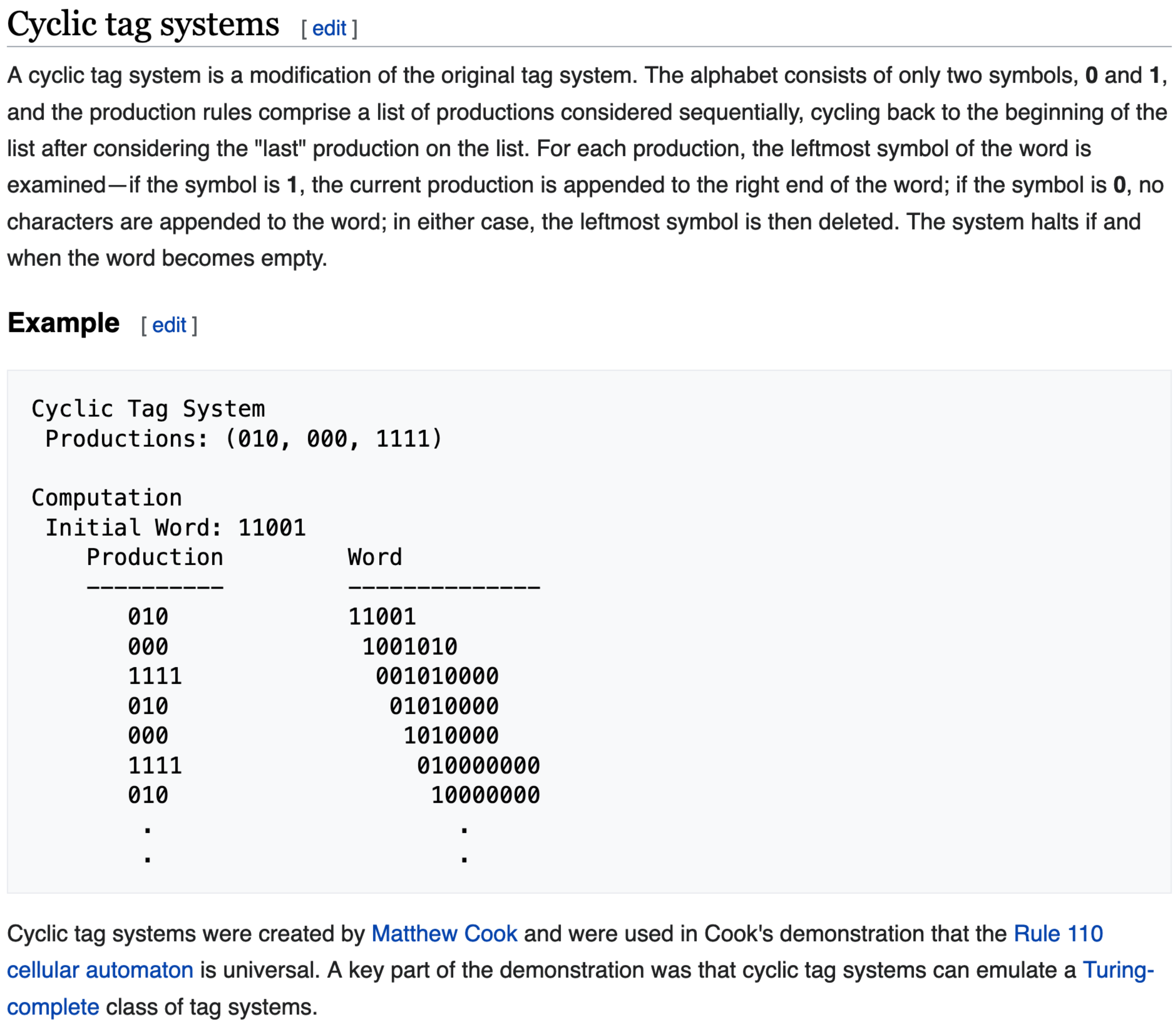
```
WITH RECURSIVE
p(iter,rnum,tag) AS (
    VALUES (0,0,1),(0,1,1),(0,2,0),
           (1,0,0),(1,1,1),
           (2,0,0),(2,1,0),(2,2,0),(2,3,0)
),
r(iter,rnum,tag) AS (
    VALUES (0,0,1)
UNION ALL
    SELECT r.iter+1,
           CASE
               WHEN r.rnum=0 THEN p.rnum + max(r.rnum) OVER ()
               ELSE r.rnum-1
           END,
           CASE
               WHEN r.rnum=0 THEN p.tag
               ELSE r.tag
           END
    FROM
        r
    LEFT JOIN p
        ON (r.rnum=0 and r.tag=1 and p.iter=mod(r.iter, 3))
    WHERE
        r.rnum>0
    OR p.iter IS NOT NULL
)
SELECT iter, rnum, tag
FROM r
ORDER BY iter, rnum;
```

Fun Snippets

Cyclic Tag System

Works with PostgreSQL	8.4
Written in	SQL
Depends on	Nothing

https://www.quora.com/Why-is-relational-algebra-not-Turing-complete , https://wiki.postgresql.org/wiki/Cyclic_Tag_System , https://en.wikipedia.org/wiki/Tag_system#Cyclic_tag_systems



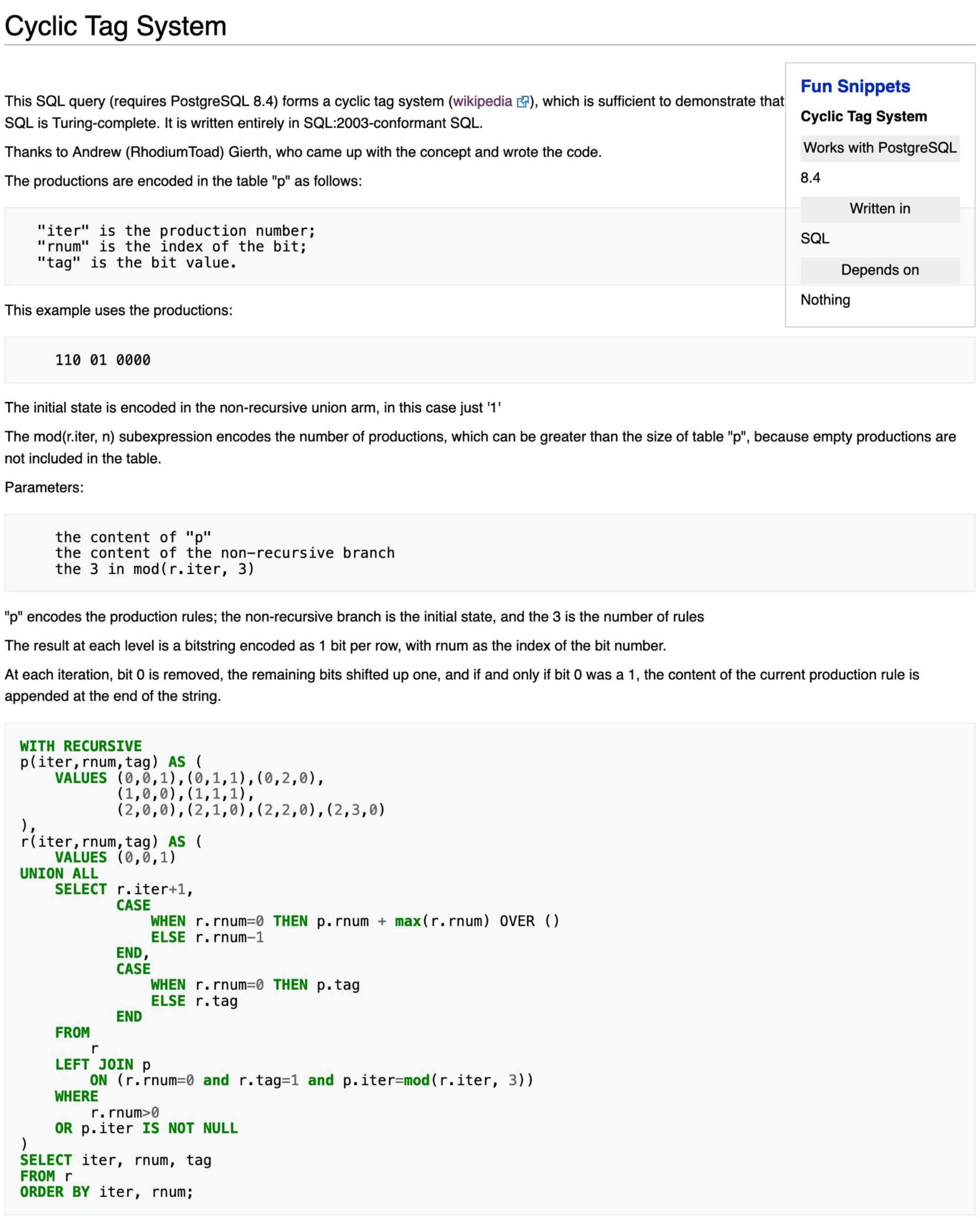
Cyclic tag systems [edit]

A cyclic tag system is a modification of the original tag system. The alphabet consists of only two symbols, **0** and **1**, and the production rules comprise a list of productions considered sequentially, cycling back to the beginning of the list after considering the "last" production on the list. For each production, the leftmost symbol of the word is examined—if the symbol is **1**, the current production is appended to the right end of the word; if the symbol is **0**, no characters are appended to the word; in either case, the leftmost symbol is then deleted. The system halts if and when the word becomes empty.

Example [edit]

```
Cyclic Tag System
 Productions: (010, 000, 1111)

Computation
 Initial Word: 11001
    Production             Word
    ----------             --------------
       010                 11001
       000                  1001010
       1111                  001010000
       010                    01010000
       000                     1010000
       1111                     010000000
       010                       10000000
        .                          .
        .                          .
```

Cyclic tag systems were created by Matthew Cook and were used in Cook's demonstration that the Rule 110 cellular automaton is universal. A key part of the demonstration was that cyclic tag systems can emulate a Turing-complete class of tag systems.



Cyclic Tag System

Fun Snippets
Cyclic Tag System
Works with PostgreSQL
8.4
Written in
SQL
Depends on
Nothing

This SQL query (requires PostgreSQL 8.4) forms a cyclic tag system (wikipedia), which is sufficient to demonstrate that SQL is Turing-complete. It is written entirely in SQL:2003-conformant SQL.

Thanks to Andrew (RhodiumToad) Gierth, who came up with the concept and wrote the code.

The productions are encoded in the table "p" as follows:

```
  "iter" is the production number;
  "rnum" is the index of the bit;
  "tag" is the bit value.
```

This example uses the productions:

```
    110 01 0000
```

The initial state is encoded in the non-recursive union arm, in this case just '1'

The mod(r.iter, n) subexpression encodes the number of productions, which can be greater than the size of table "p", because empty productions are not included in the table.

Parameters:

```
    the content of "p"
    the content of the non-recursive branch
    the 3 in mod(r.iter, 3)
```

"p" encodes the production rules; the non-recursive branch is the initial state, and the 3 is the number of rules

The result at each level is a bitstring encoded as 1 bit per row, with rnum as the index of the bit number.

At each iteration, bit 0 is removed, the remaining bits shifted up one, and if and only if bit 0 was a 1, the content of the current production rule is appended at the end of the string.

```sql
WITH RECURSIVE
p(iter,rnum,tag) AS (
    VALUES (0,0,1),(0,1,1),(0,2,0),
           (1,0,0),(1,1,1),
           (2,0,0),(2,1,0),(2,2,0),(2,3,0)
),
r(iter,rnum,tag) AS (
    VALUES (0,0,1)
UNION ALL
    SELECT r.iter+1,
           CASE
               WHEN r.rnum=0 THEN p.rnum + max(r.rnum) OVER ()
               ELSE r.rnum-1
           END,
           CASE
               WHEN r.rnum=0 THEN p.tag
               ELSE r.tag
           END
    FROM
        r
    LEFT JOIN p
        ON (r.rnum=0 and r.tag=1 and p.iter=mod(r.iter, 3))
    WHERE
        r.rnum>0
    OR p.iter IS NOT NULL
)
SELECT iter, rnum, tag
FROM r
ORDER BY iter, rnum;
```

https://www.quora.com/Why-is-relational-algebra-not-Turing-complete , https://wiki.postgresql.org/wiki/Cyclic_Tag_System , https://en.wikipedia.org/wiki/Tag_system#Cyclic_tag_systems

Query Language Design

Query language design is still a popular topic, especially for graphs. See e.g. https://www.tigergraph.com/gsql/

And the slides
https://courses.cs.washington.edu/courses/csed516/20au/lectures/lecture05-advanced-query-evaluation.pdf
from "DATA516/CSED516: Scalable Data Systems and Algorithms!" Dan Suciu
https://courses.cs.washington.edu/courses/csed516/20au/

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^{\neg}$: Datalog with stratified negation
  - Datalog$^{\pm}$
- Answer Set Programming (ASP)

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

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

?

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

?

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

?

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

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

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

?

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

?

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

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

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

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

?

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

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

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

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

Find actors who played in a movie from 1910 and from 1940

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

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

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

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

Find actors who played in a movie from 1910 ~~and~~ from 1940

OR

?

Examples by Dan Suciu

Datalog: Facts and Rules

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

Facts: tuples in the database

```
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).
```

Rules: queries

(notice position matters: unnamed perspective)

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

Find movies from 1940

```
Q2(f,l) :- Actor(u,f,l), Plays(u,x),
           Movie(x,y,z), z<1940.
```

Find actors who played in a movie before 1940

```
Q4(f,l) :- Actor(z,f,l), Plays(z,x1), Movie(x1,y1,1910).
Q4(f,l) :- Actor(z,f,l), Plays(z,x2), Movie(x2,y2,1940).
```

Find actors who played in a movie from 1910 ~~and~~ OR from 1940

Extensional Database (EDB) predicates: Actor, Plays, Movie

Intensional Database (IDB) predicates: Q1, Q2, Q3, Q4

Examples by Dan Suciu

Example with Souffle

Soufflé

Schema

```
Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)
```

movie

command line if run from the same directory:

```
souffle movie.dl
```

movie.dl

```
.decl Actor(id:number, fname:symbol, lname:symbol)
.decl Plays(aid:number, mid:number)
.decl Movie(id:number, name:symbol, year:number)
Actor(344759,"Douglas", "Fowley").
Plays(344759, 7909).
Plays(344759, 29000).
Movie(7909, "A Night in Armour", 1910).
Movie(29000, "Arizona", 1940).
Movie(29445, "Ave Maria", 1940).

.decl Q2(fname:symbol, lname:symbol)
Q2(f,l) :- Actor(u,f,l), Plays(u,x), Movie(x,_,z), z<1940.
.output Q2
```

also allows to specify specific input and output directories

```
souffle -F. -D. movie.dl
```

tab-separated output, filename: ".csv"

output

Q2.csv

```
Douglas	Fowley
```

For more help on Souffle, see: https://souffle-lang.github.io/simple

Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

Updated 2/10/2024

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 9

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

2/9/2024

Pre-class conversations

- Last class summary
- Project discussions (in 1 weeks: Fri 2/16: first project ideas)

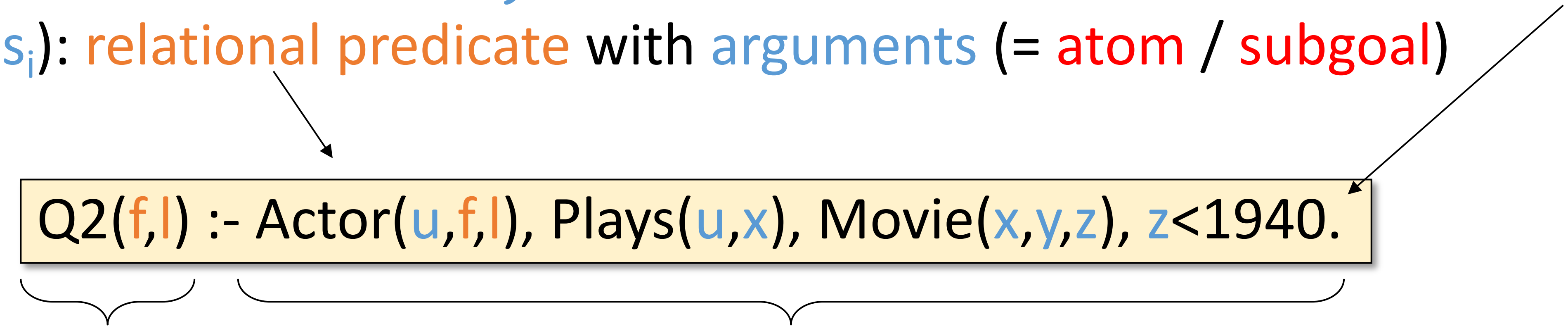
- today:
  - Recursion (Datalog)
- next week:
  - what happens if we add negation? Answer: it depends on how we do it.
    - Datalog with stratified negation
    - Datalog with more genal negation (stable models), leads to ASP

Syntax of rules

- evaluates to true when relation R_i contains the tuple described by $args_i$
- e.g. Actor(344759,"Douglas","Fowley") is true

$R_i(args_i)$: relational predicate with arguments (= atom / subgoal)

arithmetic predicate



Q2(f,l) :- Actor(u,f,l), Plays(u,x), Movie(x,y,z), z<1940.

head

(or consequent)
single IDB atom

body

(or antecedent)
conjunction of atoms

{f,l}: head variables
{u,x,y,z}: existential variables

Alternative notation: Q(args) <- R1(args) AND R2(args) .... / or variables begin with a capital letter, predicates with lower-case letters (problem: can't have "Boston")
Based upon an example by Dan Suciu from CSE 554, 2018.

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

?

Based upon class material from Dan Suciu for CSE 554, 2018.

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \land z<1940) \Rightarrow Q(y)]$$

Ignoring the case of an empty movie table, logically equivalent to

?

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\mathrm{Movie}(x,y,z) \land z<1940) \Rightarrow Q(y)]$$

Ignoring the case of an empty movie table, logically equivalent to

$$\forall y\ [\exists x,z\ [\mathrm{Movie}(x,y,z) \land z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

compare with DRC

?

Based upon class material from Dan Suciu for CSE 554, 2018.

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

Ignoring the case of an empty movie table, logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

We want the smallest set Q with this property (why?)

?

compare with DRC

$$\{(y) \mid \exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940]\ \}$$

Based upon class material from Dan Suciu for CSE 554, 2018.

Logical interpretation of a single rule

Actor(id, fname, lname)
Plays(aid, mid)
Movie(id, name, year)

Q(y) :- Movie(x,y,z), z<1940.

Meaning of a Datalog rule is a logical statement:

For all x,y,z: if (x,y,z) ∈ Movies and z<1940 then y is in Q (i.e. is part of the answer)

$$\forall x,y,z\ [(\text{Movie}(x,y,z) \wedge z<1940) \Rightarrow Q(y)]$$

Ignoring the case of an empty movie table, logically equivalent to

$$\forall y\ [\exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940] \Rightarrow Q(y)\ ]$$

Thus, non-head variables are called "existential variables"

We want the smallest set Q with this property (why?)

That takes care of the empty movie table ☺: a rules only fires if the antecedent is fulfilled …

compare with DRC

$$\{(y) \mid \exists x,z\ [\text{Movie}(x,y,z) \wedge z<1940]\ \}$$

Based upon class material from Dan Suciu for CSE 554, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Syntactic Constraints

head variables

existential variables

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

$$Q(\mathbf{x}) \text{ :- } R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m).$$

The rule stands for the following logical formula:

$$\forall \mathbf{x}[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}[R_1(\mathbf{x}_1,\mathbf{y}_1)\wedge\cdots\wedge R_m(\mathbf{x}_m,\mathbf{y}_m)]]$$

Recall we want the smallest set Q with this property

Two restrictions:

1. Safety: every head variable should occur in the body at least once

R(x,z) :- S(x,y), R(y,x).

?

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Syntactic Constraints

$$Q(\mathbf{x}) \text{ :- } R_1(\mathbf{x}_1,\mathbf{y}_1),\ldots,R_m(\mathbf{x}_m,\mathbf{y}_m).$$

$\mathbf{x}_i \subseteq \mathbf{x}, \mathbf{y}_i \subseteq \mathbf{y}$

(bold = vector notation)

The rule stands for the following logical formula:

$$\forall \mathbf{x}\left[Q(\mathbf{x}) \Leftarrow \exists \mathbf{y}\left[R_1(\mathbf{x}_1,\mathbf{y}_1)\wedge\cdots\wedge R_m(\mathbf{x}_m,\mathbf{y}_m)\right]\right]$$

Two restrictions:

1. Safety: every head variable should occur in the body at least once

~~R(x,z) :- S(x,y), R(y,x).~~

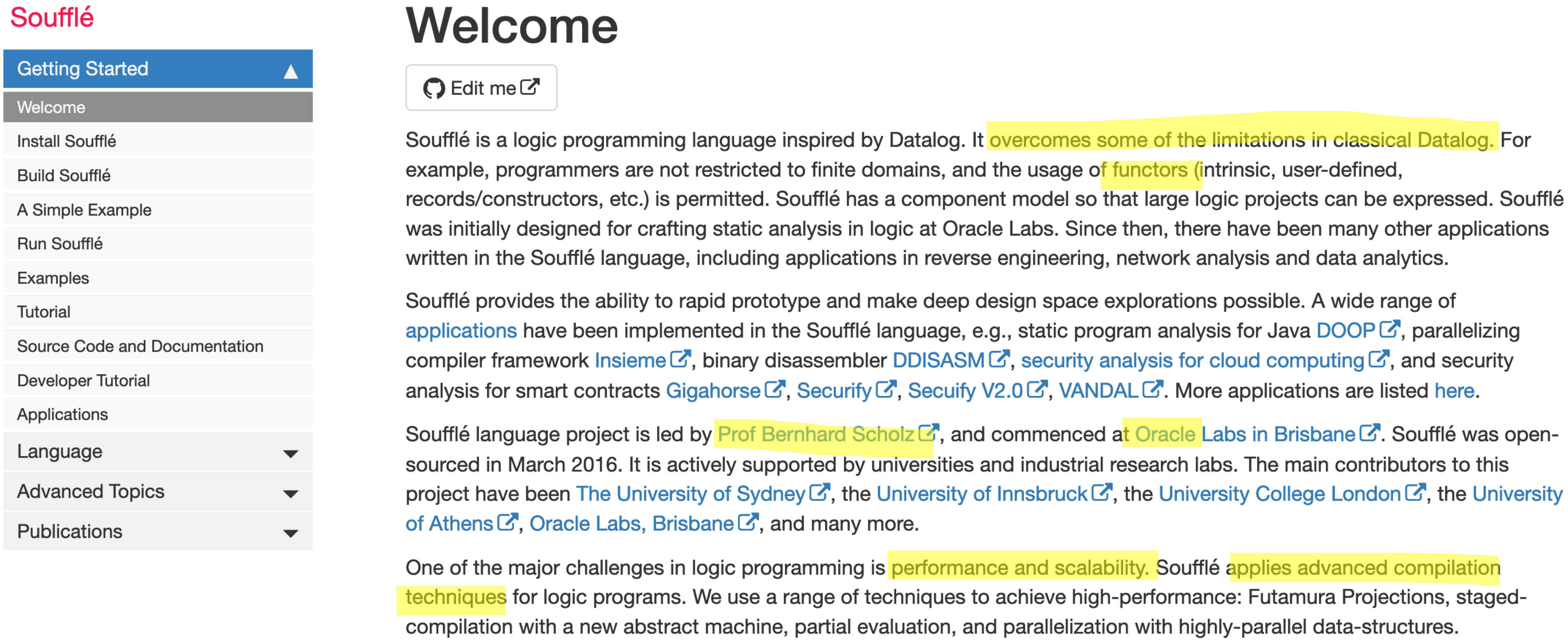
forbidden rule: z not in body

2. The head predicate must be an IDB (Intensional) predicate
   (Body can include both EDBs and IDBs)

~~Arc(x,y) :- Arc(x,z),Arc(z,y).~~

This is mostly of theoretic interest. Souffle calls EDBs the "facts ... sourced from tab-separated input files" but allows them also to appear in the head of a rule (https://souffle-lang.github.io/execute)

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.



Welcome

Soufflé is a logic programming language inspired by Datalog. It overcomes some of the limitations in classical Datalog. For example, programmers are not restricted to finite domains, and the usage of functors (intrinsic, user-defined, records/constructors, etc.) is permitted. Soufflé has a component model so that large logic projects can be expressed. Soufflé was initially designed for crafting static analysis in logic at Oracle Labs. Since then, there have been many other applications written in the Soufflé language, including applications in reverse engineering, network analysis and data analytics.

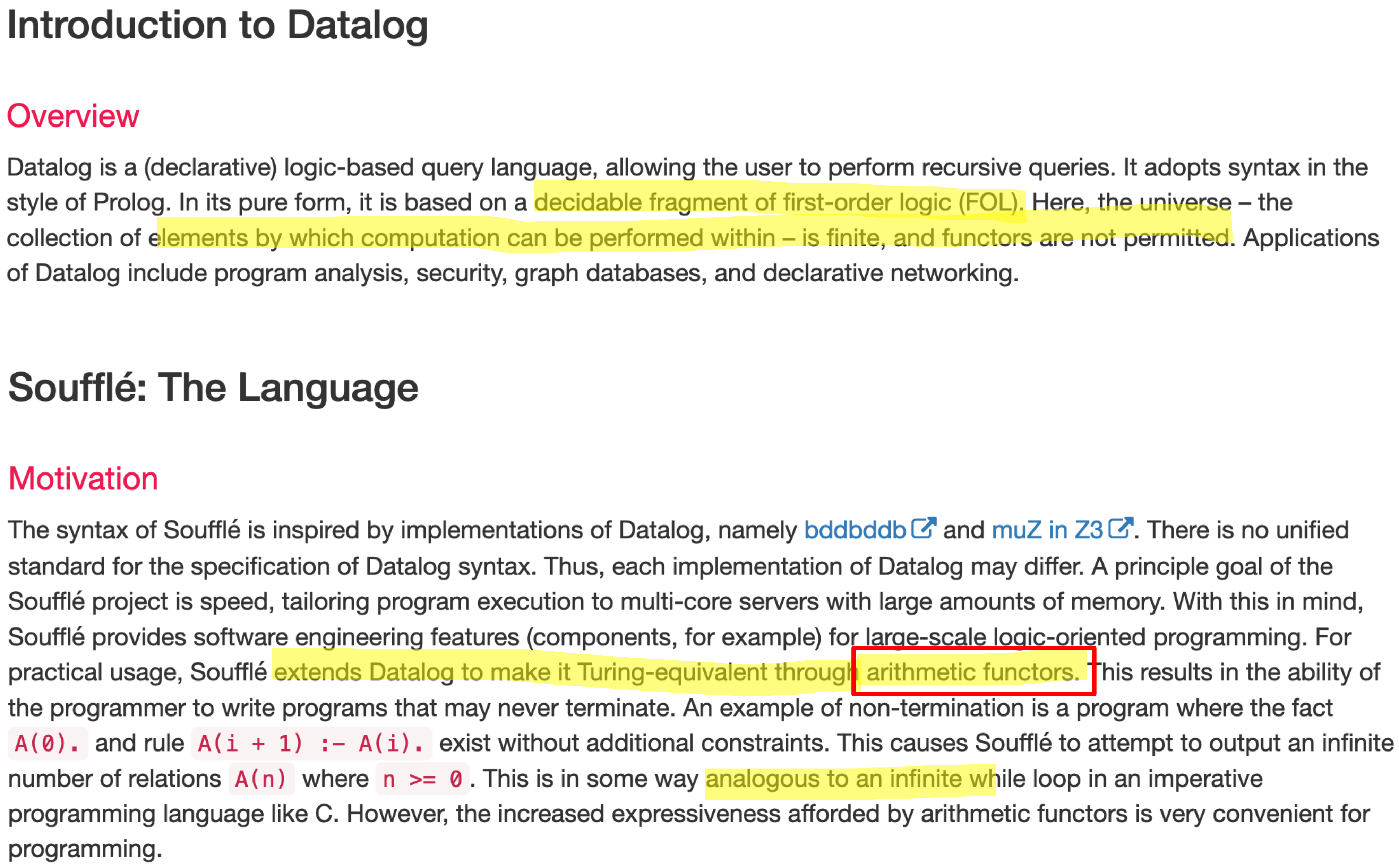
Soufflé provides the ability to rapid prototype and make deep design space explorations possible. A wide range of applications have been implemented in the Soufflé language, e.g., static program analysis for Java DOOP, parallelizing compiler framework Insieme, binary disassembler DDISASM, security analysis for cloud computing, and security analysis for smart contracts Gigahorse, Securify, Secuify V2.0, VANDAL. More applications are listed here.

Soufflé language project is led by Prof Bernhard Scholz, and commenced at Oracle Labs in Brisbane. Soufflé was open-sourced in March 2016. It is actively supported by universities and industrial research labs. The main contributors to this project have been The University of Sydney, the University of Innsbruck, the University College London, the University of Athens, Oracle Labs, Brisbane, and many more.

One of the major challenges in logic programming is performance and scalability. Soufflé applies advanced compilation techniques for logic programs. We use a range of techniques to achieve high-performance: Futamura Projections, staged-compilation with a new abstract machine, partial evaluation, and parallelization with highly-parallel data-structures.

Source: https://souffle-lang.github.io/docs.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/



Introduction to Datalog

Overview

Datalog is a (declarative) logic-based query language, allowing the user to perform recursive queries. It adopts syntax in the style of Prolog. In its pure form, it is based on a decidable fragment of first-order logic (FOL). Here, the universe – the collection of elements by which computation can be performed within – is finite, and functors are not permitted. Applications of Datalog include program analysis, security, graph databases, and declarative networking.

Soufflé: The Language

Motivation

The syntax of Soufflé is inspired by implementations of Datalog, namely bddbddb and muZ in Z3. There is no unified standard for the specification of Datalog syntax. Thus, each implementation of Datalog may differ. A principle goal of the Soufflé project is speed, tailoring program execution to multi-core servers with large amounts of memory. With this in mind, Soufflé provides software engineering features (components, for example) for large-scale logic-oriented programming. For practical usage, Soufflé extends Datalog to make it Turing-equivalent through arithmetic functors. This results in the ability of the programmer to write programs that may never terminate. An example of non-termination is a program where the fact `A(0).` and rule `A(i + 1) :- A(i).` exist without additional constraints. This causes Soufflé to attempt to output an infinite number of relations `A(n)` where `n >= 0`. This is in some way analogous to an infinite while loop in an imperative programming language like C. However, the increased expressiveness afforded by arithmetic functors is very convenient for programming.

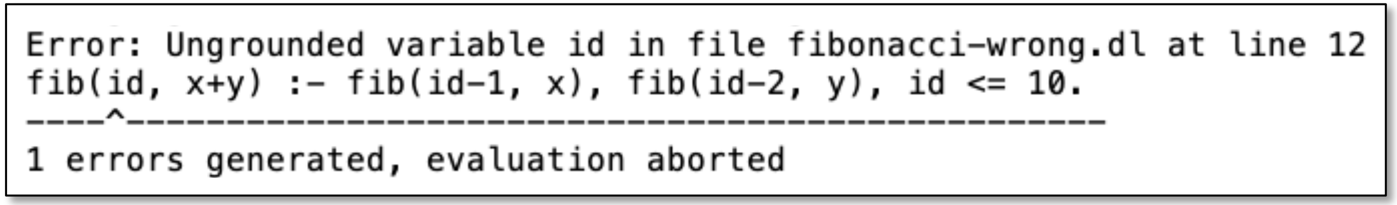
Source: https://souffle-lang.github.io/tutorial

Grounded variables

However, note that the following example has an *ungrounded* variable:

```
.decl fib(idx:number, value:number)
fib(1,1).
fib(2,1).
fib(idx, x + y) :- fib(idx-1, x), fib(idx-2, y), idx <= 10.
.output fib
```

The reason for this is that variable `idx` is not bound as an argument of a positive predicate in the body. In the example, variable `idx` occurs in the predicates `fib(idx-1, x)` and `fib(idx-2, y)` but as arguments of a functor rather than as a direct argument.

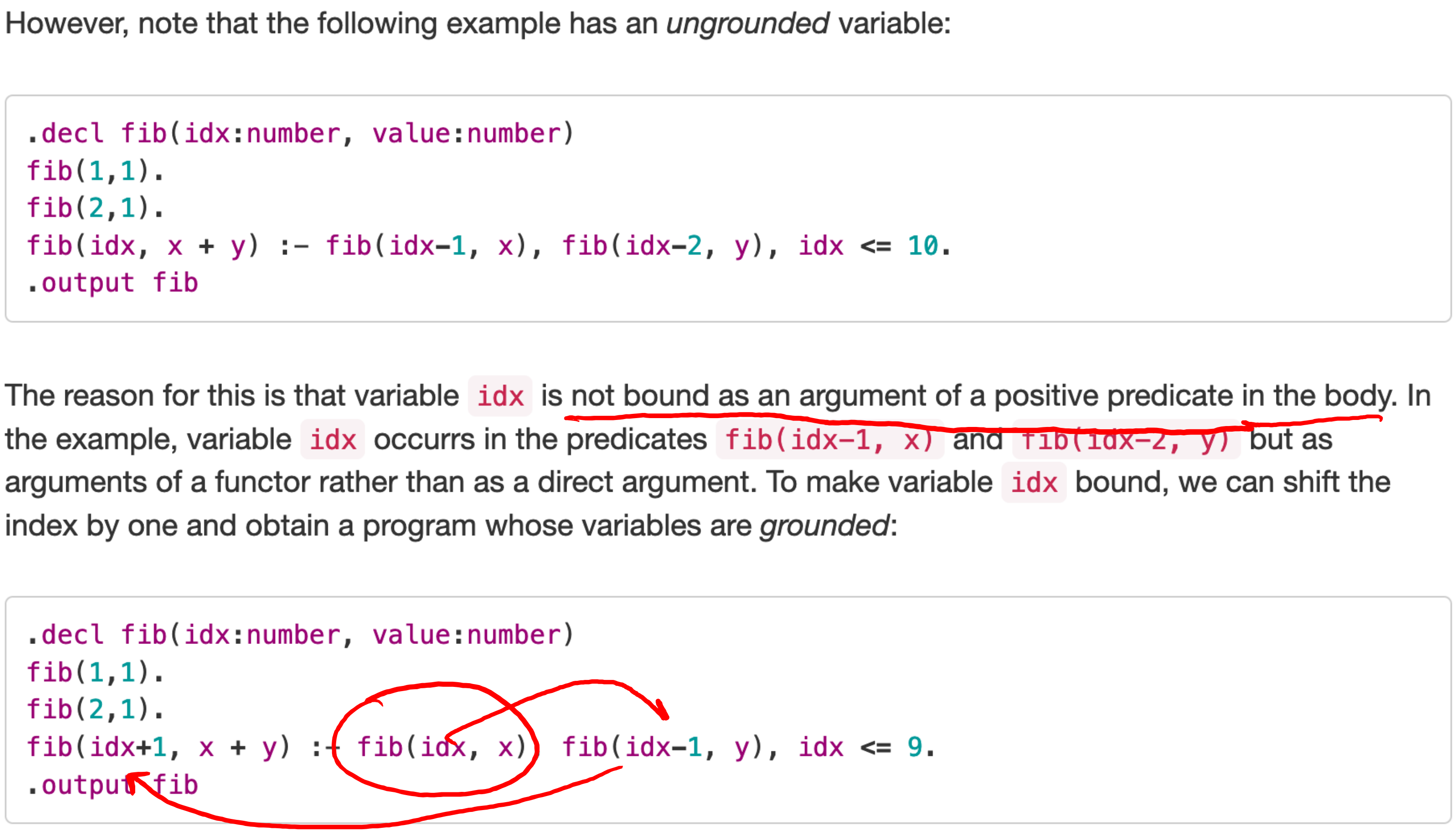


```
Error: Ungrounded variable id in file fibonacci-wrong.dl at line 12
fib(id, x+y) :- fib(id-1, x), fib(id-2, y), id <= 10.
----^-------------------------------------------------
1 errors generated, evaluation aborted
```

What can be done?

Source: https://souffle-lang.github.io/rules

Grounded variables

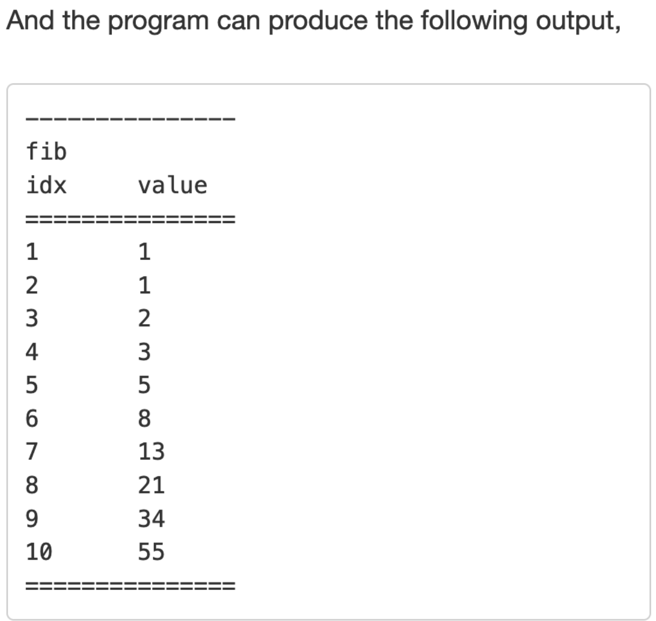


However, note that the following example has an *ungrounded* variable:

```
.decl fib(idx:number, value:number)
fib(1,1).
fib(2,1).
fib(idx, x + y) :- fib(idx-1, x), fib(idx-2, y), idx <= 10.
.output fib
```

The reason for this is that variable `idx` is not bound as an argument of a positive predicate in the body. In the example, variable `idx` occurs in the predicates `fib(idx-1, x)` and `fib(idx-2, y)` but as arguments of a functor rather than as a direct argument. To make variable `idx` bound, we can shift the index by one and obtain a program whose variables are *grounded*:

```
.decl fib(idx:number, value:number)
fib(1,1).
fib(2,1).
fib(idx+1, x + y) :- fib(idx, x), fib(idx-1, y), idx <= 9.
.output fib
```



And the program can produce the following output,

```
---------------
fib
idx       value
===============
1         1
2         1
3         2
4         3
5         5
6         8
7         13
8         21
9         34
10        55
===============
```

Source: https://souffle-lang.github.io/rules

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Grounded variables

fibonacci

```
souffle fibonacci.dl
```

fibonacci.dl

```
.decl fib(key:number, value:number)
.output fib

fib(1, 1).
fib(2, 1).
fib(id+2, x+y) :- fib(id, x), fib(id+1, y), id <= 13.
```

?

Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

Source: https://souffle-lang.github.io/rules

Grounded variables

fibonacci

```
souffle fibonacci.dl
```

fibonacci.dl

```
.decl fib(key:number, value:number)
.output fib

fib(1, 1).
fib(2, 1).
fib(id+2, x+y) :- fib(id, x), fib(id+1, y), id <= 13.
```

fib.csv

```
1  1
2  1
3  2
4  3
5  5
6  8
7  13
8  21
9  34
10 55
11 89
12 144
13 233
14 377
15 610
```

Source: https://souffle-lang.github.io/rules

Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^\neg$: Datalog with stratified negation
  - Datalog$^\pm$
- Answer Set Programming (ASP)

RA to Datalog by examples: Union

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R(A,B,C) ∪ S(D,E,F)

Datalog:

?

RA to Datalog by examples: Union

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R(A,B,C) ∪ S(D,E,F)

Datalog:

Q(x,y,z) :- R(x,y,z)
Q(x,y,z) :- S(x,y,z)

IDB EDB

?

RA to Datalog by examples: Union

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R(A,B,C) ∪ S(D,E,F)

Datalog:

Q(x,y,z) :- R(x,y,z)
Q(x,y,z) :- S(x,y,z)

IDB EDB

RA to Datalog by examples: Intersection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R(A,B,C) ∩ S(D,E,F)

Datalog:

?

RA to Datalog by examples: Intersection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R(A,B,C) ∩ S(D,E,F)

Datalog:

Q(x,y,z) :- R(x,y,z), S(x,y,z)

RA to Datalog by examples: Selection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\sigma_{B=\text{'Alice'} \wedge C>10}(R)$

Datalog:

?

RA to Datalog by examples: Selection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\sigma_{B='Alice' \wedge C>10} (R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice', z > 10

(also: Q(x,y,z) :- R(x,'Alice',z), z > 10)

RA to Datalog by examples: Selection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\sigma_{B='Alice' \wedge C>10}(R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice', z > 10

RA:

$\sigma_{B='Alice' \vee C>10}(R)$

?

RA to Datalog by examples: Selection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\sigma_{B='Alice' \wedge C>10}(R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice', z > 10

RA:

$\sigma_{B='Alice' \vee C>10}(R)$

Datalog:

Q(x,y,z) :- R(x,y,z), y='Alice'

Q(x,y,z) :- R(x,y,z), z > 10

RA to Datalog by examples: Projection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\pi_A(R)$ $\pi_{-B,C}(R)$

Datalog:

?

RA to Datalog by examples: Projection

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$\pi_A(R)$ $\pi_{-B,C}(R)$

Datalog:

```
Q(x) :- R(x,y,z)
Q(x) :- R(x,_,_)
```

Underscore denotes an "anonymous variable".
Each occurrence of an underscore represents a different variable

RA to Datalog by examples: Equi-join

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$$\pi_{-D,E}(\ R \bowtie_{A=D \wedge B=E} S)$$

Datalog:

?

RA to Datalog by examples: Equi-join

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

$$\overset{\Pi_{A,B,C,F}}{\pi_{-D,E}}\left(\underbrace{R \bowtie_{A=D \wedge B=E} S}_{A,B,C,\cancel{D},\cancel{E},F}\right)$$

Datalog:

Q(x,y,z,w) :- R(x,y,z), S(x,y,w)

(also: Q(x,y,z,w) :- R(x,y,z), S(u,v,w), x=u, y=v)

RA to Datalog by examples: Difference

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R–S

Datalog:

?

RA to Datalog by examples: Difference

R(A,B,C)
S(D,E,F)
T(G,H)

RA:

R–S

Datalog$^{\neg}$: (we need to add negation)

Q(x,y,z) :- R(x,y,z), not S(x,y,z)

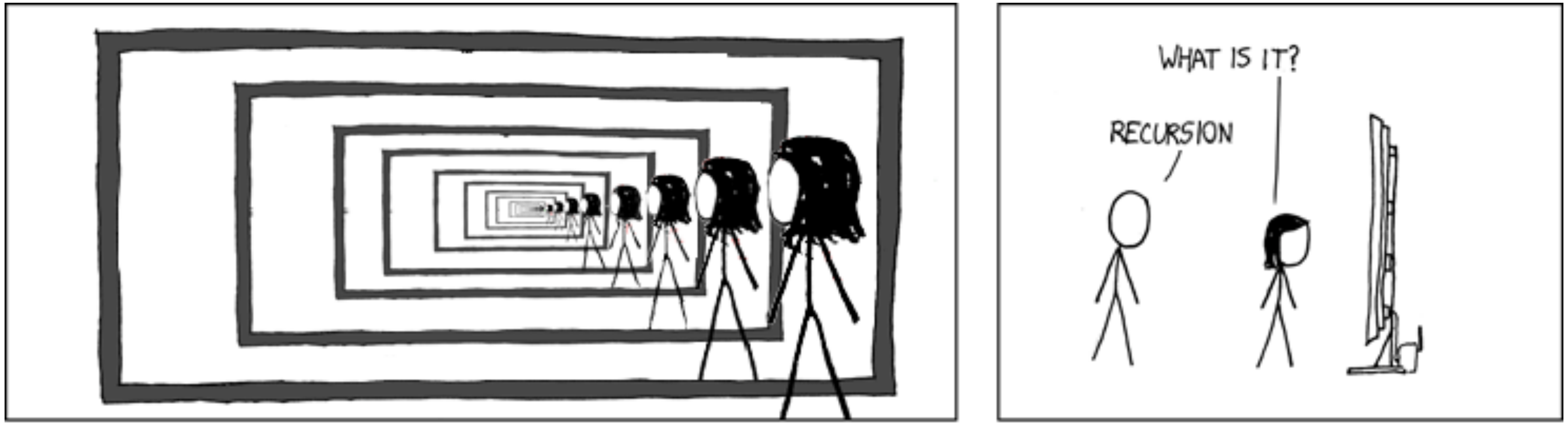
SAFETY

we have a long discussion later on what can go wrong if you are not careful about how you define negation

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^\neg$: Datalog with stratified negation
  - Datalog$^\pm$
- Answer Set Programming (ASP)

Recursion



Recursion occurs when a thing is defined in terms of itself (self-repetition).

Recursion and Iteration both repeatedly execute a set of instructions.

- Recursion (self-similarity) is when a statement in a function calls itself repeatedly.
- Iteration (repetition) is when a loop repeatedly executes until the controlling condition becomes false.

A Datalog program consists of several rules:

- Usually there is one distinguished predicate that's the output
- Rules can be recursive!

Figure Source: Fake XKCD: http://xkcdsw.com/1105

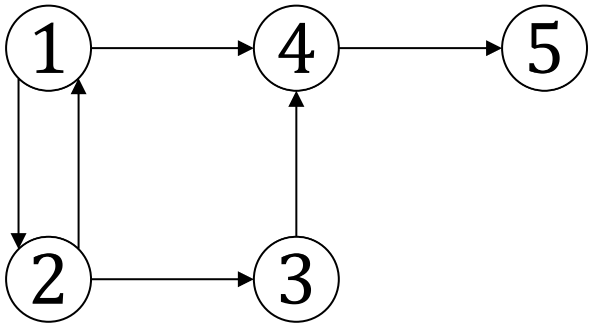
Example

A(S,T)

EDB P(x,y) :- A(x,y).

IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

What does this query compute?

?

Based upon an example by Dan Suciu from CSE 554, 2018.

Example

A(S,T)

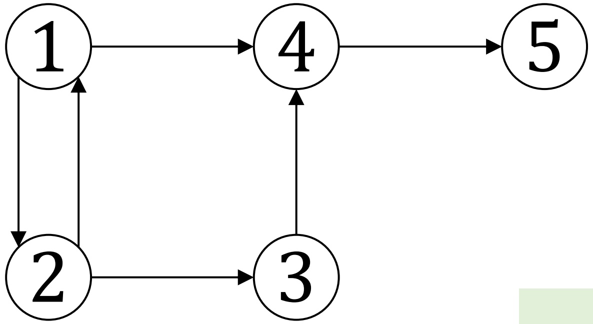
EDB P(x,y) :- A(x,y).

IDB P(x,y) :- A(x,z), P(z,y).

recursion due to head in rule body

$x \rightarrow z \rightsquigarrow y$

Calculates all paths (transitive closure)



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

For all nodes x and y:
If there is an Arc from x to y,
then there is a Path from x to y.

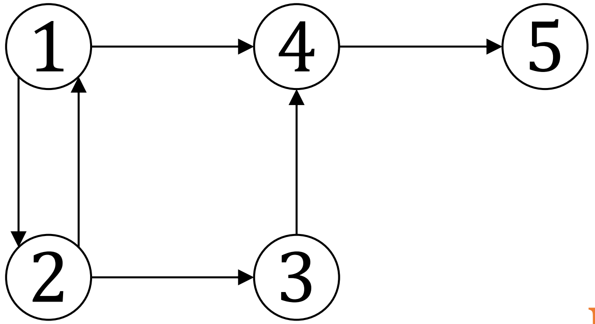
For all nodes x, z, and y:
If there is an Arc from x to z, and there is a Path from z to y
then there is a Path from x to y.

Based upon an example by Dan Suciu from CSE 554, 2018.

Example

A(S,T)

EDB P(x,y) :- A(x,y).
IDB P(x,y) :- A(x,z), P(z,y).



Initially: P is empty

1^{st} iteration

P

?

A

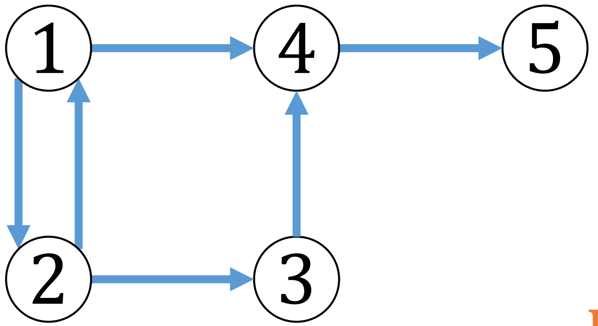
S	T
1	2
1	4
2	1
2	3
3	4
4	5

Example

A(S,T)

EDB P(x,y) :- A(x,y).

IDB P(x,y) :- A(x,z), P(z,y).



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

1st iteration

P

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

P=A from 1st rule

2nd iteration

P

?

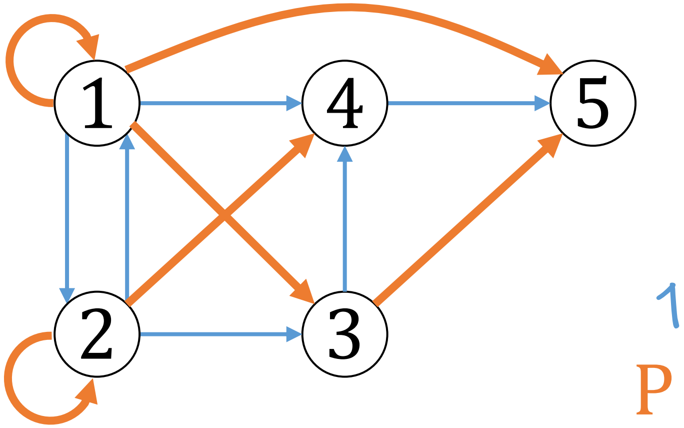
2nd rule generates nothing (because P is empty)

Example

A(S,T)

EDB $P(x,y)$:- $A(x,y)$.

IDB $P(x,y)$:- $A(x,z)$, $P(z,y)$.



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

1st iteration

P

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

P=A from 1st rule

2nd rule generates nothing (because P is empty)

2nd iteration

P

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

1st rule

2nd rule

New facts from 2nd rule

3rd iteration

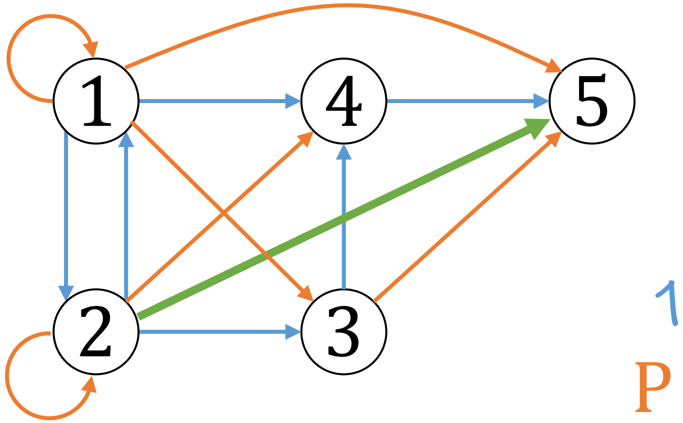
?

Example

A(S,T)

EDB P(x,y) :- A(x,y).

IDB P(x,y) :- A(x,z), P(z,y).



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

recall set semantics!

(No new facts)

1st iteration

P

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

P=A from 1st rule

2nd rule generates nothing (because P is empty)

2nd iteration

P

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

1st rule

2nd rule

3rd iteration = 4th iteration

P

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

1st + 2nd rule

1st rule

2nd rule

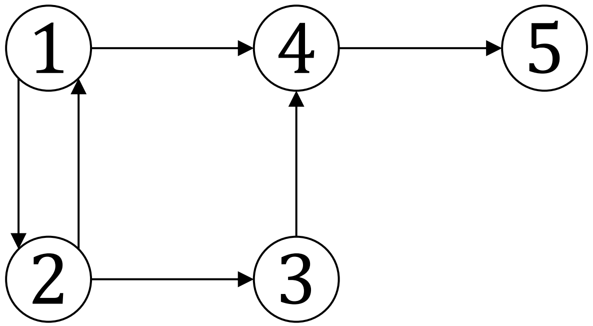
New facts from 2nd rule

Example with Souffle

Soufflé

A(S,T)

graph1



```
souffle graph1.dl
```

graph1.dl

```
.decl A(S:number, T:number)
A(1,2).
A(2,1).
A(2,3).
A(1,4).
A(3,4).
A(4,5).

.decl P(S:number, T:number)
P(x, y) :- A(x, y).
P(x, y) :- A(x, z), P(z, y).

.output P
```

output

P.csv

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

tab-separated,
output filename: ".csv"

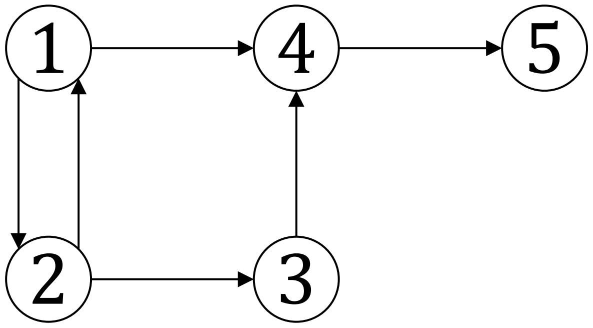
For more help on Souffle, see: https://souffle-lang.github.io/simple
Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example with Souffle

Soufflé

A(S,T)

graph2



```
souffle graph2.dl
```

graph2.dl

```
.decl A(S:number, T:number)
.decl P(S:number, T:number)
.input A
.output P

P(x, y) :- A(x, y).
P(x, y) :- A(x, z), P(z, y).
```

input

A.facts

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

tab-separated,
input filename:
".facts"

output

P.csv

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

tab-separated,
output filename: ".csv"

For more help on Souffle, see: https://souffle-lang.github.io/simple
Datalog example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

What is a principled process to determine if a program is recursive?

1

```
   Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
 Invited(y) :- Relative('myself',y),Local(y).
```

?

2

```
   Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
 Invited(y) :- Relative('myself',y),Local(y).
```

?

3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
  Visit(x,y) :- MayLike(x,y).
  Close(x,z) :- Visit(x,y),Visit(z,y).
```

?

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.

Dependency Graph

- The dependency graph of a Datalog program is the directed graph (V,E) where
  - V is the set of IDB predicates (relation names)
  - E contains an arc S ⟶ T whenever there is a rule with T in the head and S in the body

- A Datalog program is recursive if its dependency graph contains a cycle

Which of these programs is recursive?

1

```
   Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
 Invited(y) :- Relative('myself',y),Local(y).
```

?

2

```
   Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
 Invited(y) :- Relative('myself',y),Local(y).
```

?

3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
  Visit(x,y) :- MayLike(x,y).
  Close(x,z) :- Visit(x,y),Visit(z,y).
```

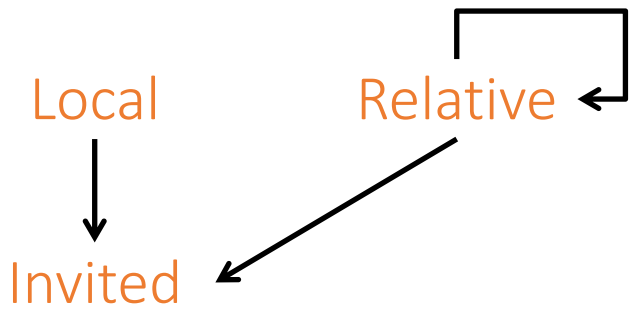
?

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.

Which of these programs is recursive?

1

```
       Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
   Invited(y) :- Relative('myself',y),Local(y).
```



2

```
       Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
   Invited(y) :- Relative('myself',y),Local(y).
```

?

3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
    Visit(x,y) :- MayLike(x,y).
   Close(x,z) :- Visit(x,y),Visit(z,y).
```

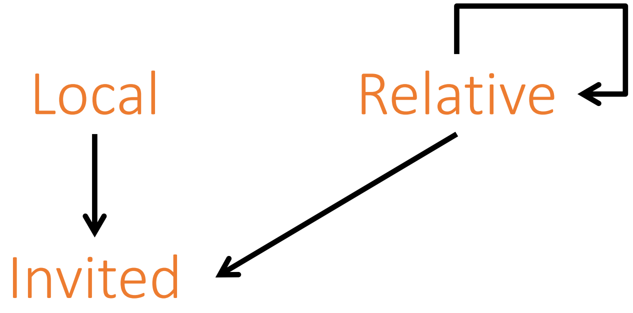
?

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Which of these programs is recursive?

1

```
      Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
   Invited(y) :- Relative('myself',y),Local(y).
```



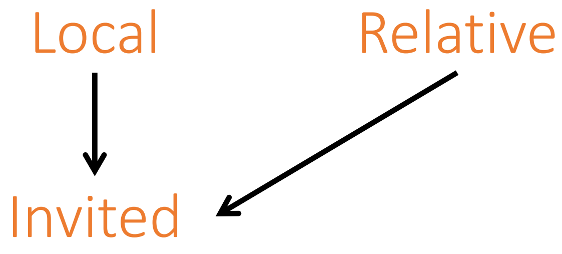
Local

Relative

Invited

2

```
      Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
   Invited(y) :- Relative('myself',y),Local(y).
```



Local

Relative

Invited

3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
    Visit(x,y) :- MayLike(x,y).
   Close(x,z) :- Visit(x,y),Visit(z,y).
```

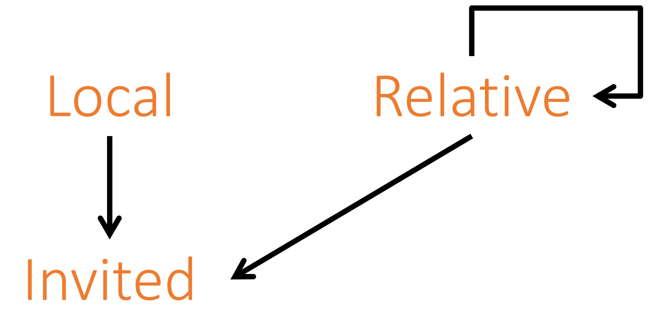
?

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.

Which of these programs is recursive?

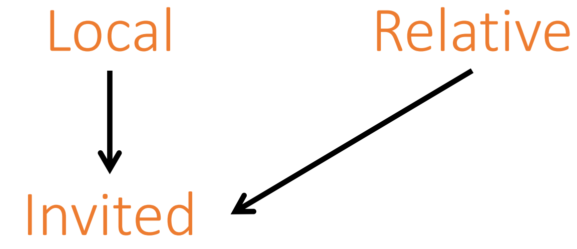
1

```
    Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
Relative(x,y) :- Relative(x,z),Parent(z,y).
Relative(x,y) :- Relative(x,z),Parent(y,z).
Relative(x,y) :- Relative(x,z),Spouse(z,y).
  Invited(y) :- Relative('myself',y),Local(y).
```



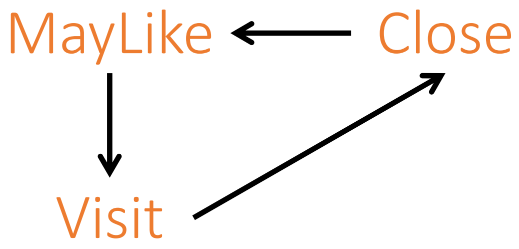
2

```
    Local(x) :- Person(x,y,'MA').
Relative(x,x) :- Person(x,y,z).
  Invited(y) :- Relative('myself',y),Local(y).
```



3

```
MayLike(x,y) :- Close(x,z),Likes(z,y).
   Visit(x,y) :- MayLike(x,y).
  Close(x,z) :- Visit(x,y),Visit(z,y).
```



Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Expressiveness of Non-recursive Datalog

THEOREM: Non-recursive Datalog with built-in predicates (<,>,≤,≥,!=) has the same expressive power as the positive algebra {σ,π,×,∪}

If we restrict selection to $\sigma_=$ (i.e. selection with a single equality), this fragment is also called at times UCQs (Union of Conjunctive Queries) or USPJ (Union-Select-Project-Join) queries.

Outline: T1-4: Datalog & ASP

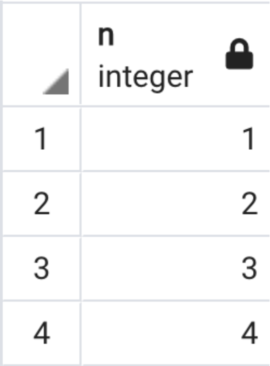
- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^{\neg}$: Datalog with stratified negation
  - Datalog$^{\pm}$
- Answer Set Programming (ASP)

1. A simple recursive query

non-recursive part (here same as "select 1")

recursive part, contains reference to the query's output

```
WITH RECURSIVE T(n) as (
    values (1)
  UNION ALL
    select n+1
    from T
    where n<=3)
SELECT n FROM T
```



	n integer
1	1
2	2
3	3
4	4

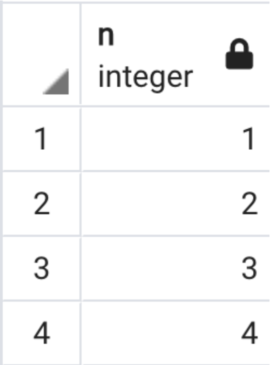
Example slightly adapted from: https://www.postgresql.org/docs/current/queries-with.html#QUERIES-WITH-RECURSIVE

1. A simple recursive query

non-recursive part (here same as "select 1")

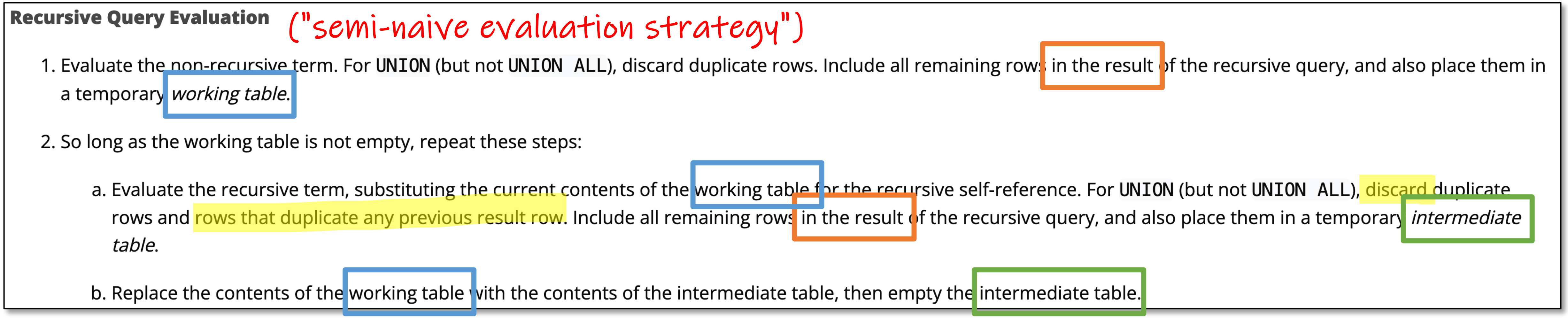
recursive part, contains reference to the query's output

```
WITH RECURSIVE T(n) as (
    values (1)
  UNION ALL
    select n+1
    from T
    where n<=3)
SELECT n FROM T
```



	n integer
1	1
2	2
3	3
4	4

Step	WT_{start}	$\Delta R=IT=Wt_{end}$	Results
1.	?	?	?
2.			
3.			
4.			
5.			



Recursive Query Evaluation ("semi-naive evaluation strategy")

1. Evaluate the non-recursive term. For `UNION` (but not `UNION ALL`), discard duplicate rows. Include all remaining rows in the result of the recursive query, and also place them in a temporary *working table*.

2. So long as the working table is not empty, repeat these steps:

   a. Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference. For `UNION` (but not `UNION ALL`), discard duplicate rows and rows that duplicate any previous result row. Include all remaining rows in the result of the recursive query, and also place them in a temporary *intermediate table*.

   b. Replace the contents of the working table with the contents of the intermediate table, then empty the intermediate table.

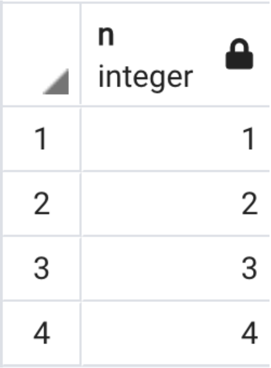
Example slightly adapted from: https://www.postgresql.org/docs/current/queries-with.html#QUERIES-WITH-RECURSIVE

1. A simple recursive query

non-recursive part (here same as "select 1")

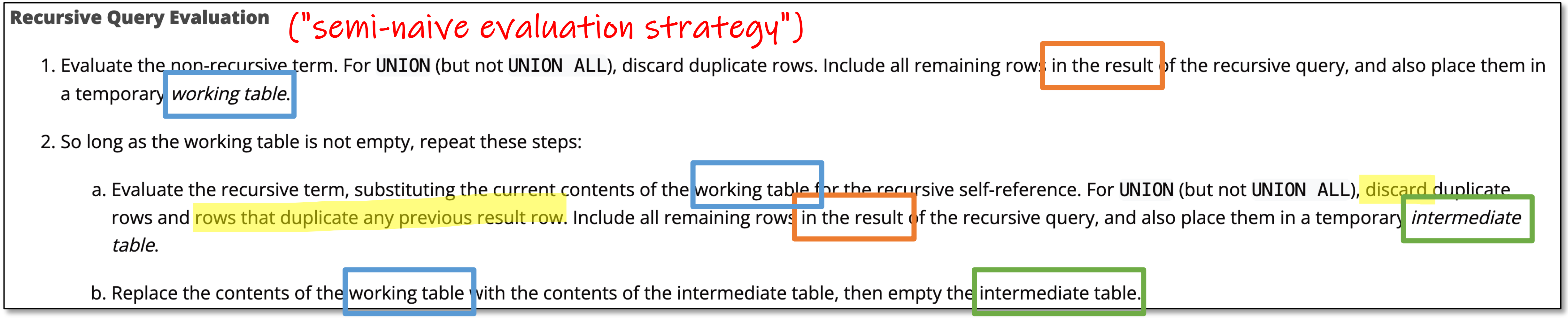
recursive part, contains reference to the query's output

```
WITH RECURSIVE T(n) as (
    values (1)
  UNION ALL
    select n+1
    from T
    where n<=3)
SELECT n FROM T
```



	n integer
1	1
2	2
3	3
4	4

Step	WT_{start}	$\Delta R=IT=Wt_{end}$	Results
1.		{1}	{1}
2.	{1}	{2}	{1,2}
3.	{2}	{3}	{1,2,3}
4.	{3}	{4}	{1,2,3,4}
5.	{4}	∅	{1,2,3,4}



Recursive Query Evaluation ("semi-naive evaluation strategy")

1. Evaluate the non-recursive term. For `UNION` (but not `UNION ALL`), discard duplicate rows. Include all remaining rows in the result of the recursive query, and also place them in a temporary *working table*.

2. So long as the working table is not empty, repeat these steps:

   a. Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference. For `UNION` (but not `UNION ALL`), discard duplicate rows and rows that duplicate any previous result row. Include all remaining rows in the result of the recursive query, and also place them in a temporary *intermediate table*.

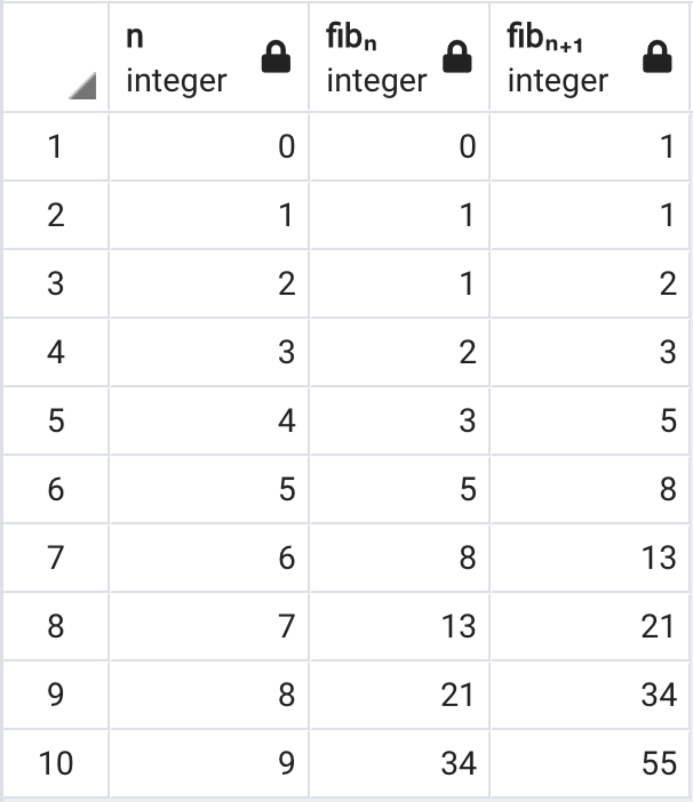
   b. Replace the contents of the working table with the contents of the intermediate table, then empty the intermediate table.

Example slightly adapted from: https://www.postgresql.org/docs/current/queries-with.html#QUERIES-WITH-RECURSIVE

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, ...

```
WITH RECURSIVE Fib as (

      ?

   UNION ALL

      ?

SELECT * FROM Fib
LIMIT 10;
```

Fib



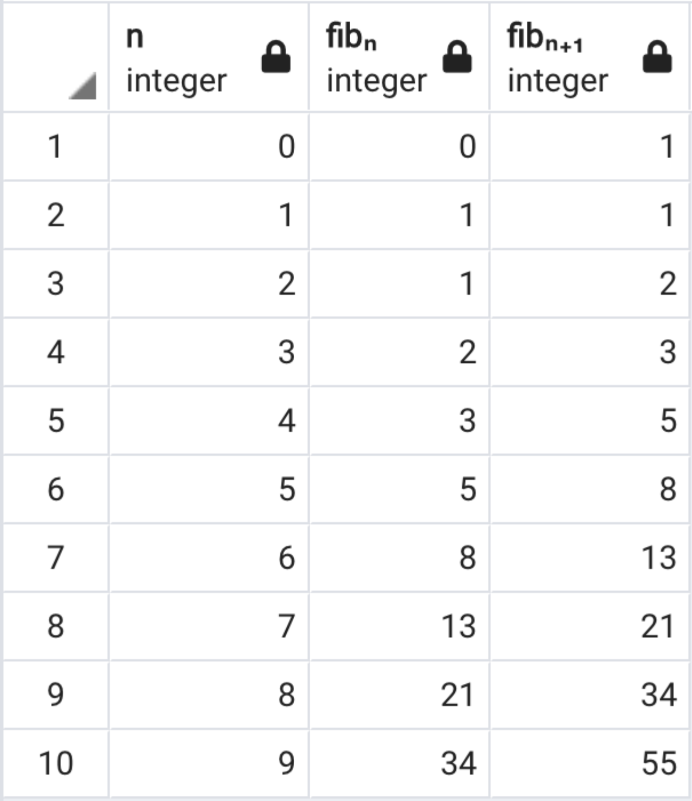
	n integer	fib_n integer	fib_{n+1} integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, …

```
WITH RECURSIVE Fib as (
      select 0 as n,
                  0 as "fib_n",
                  1 as "fib_n+1"
   UNION ALL

         ?

SELECT * FROM Fib
LIMIT 10;
```

Fib



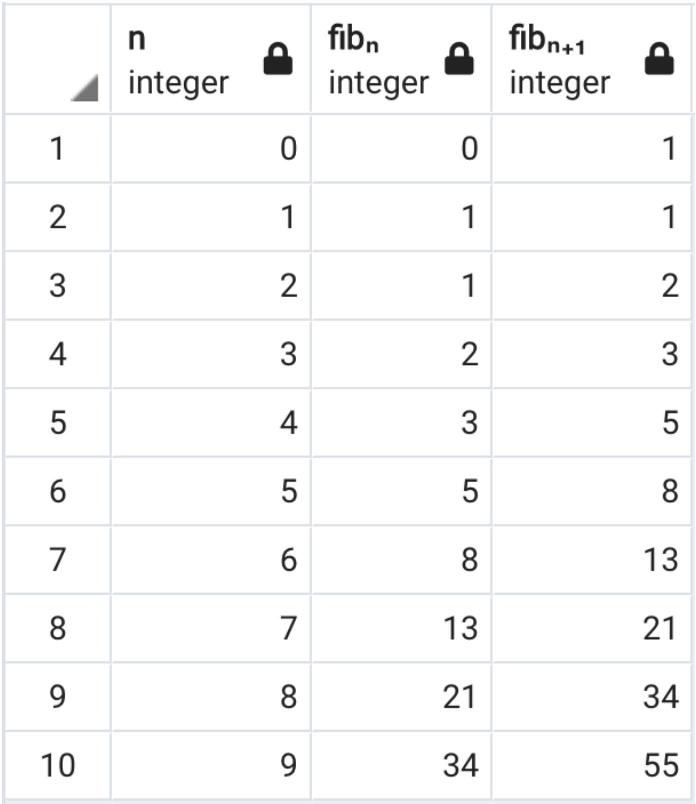
	n integer	fib_n integer	fib_{n+1} integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, …

```
WITH RECURSIVE Fib as (
    select 0 as n,
           0 as "fib_n",
           1 as "fib_n+1"
  UNION ALL
    select n+1,
                ?

    from Fib)
SELECT * FROM Fib
LIMIT 10;
```

Fib



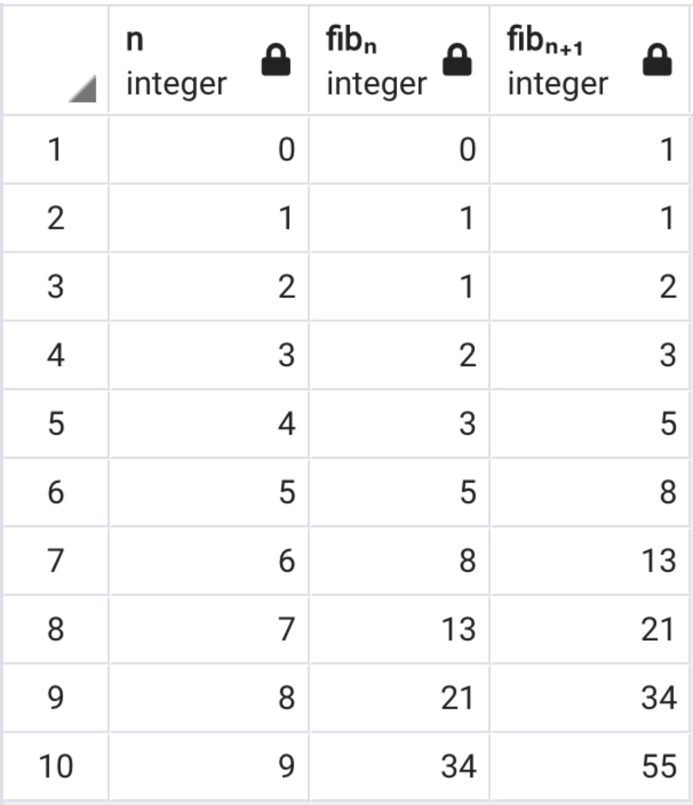
	n integer	fib$_n$ integer	fib$_{n+1}$ integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, …

```
WITH RECURSIVE Fib as (
      select 0 as n,
               0 as "fib_n",
               1 as "fib_n+1"
   UNION ALL
      select n+1,
               "fib_n+1", ?

      from Fib)
SELECT * FROM Fib
LIMIT 10;
```

Fib

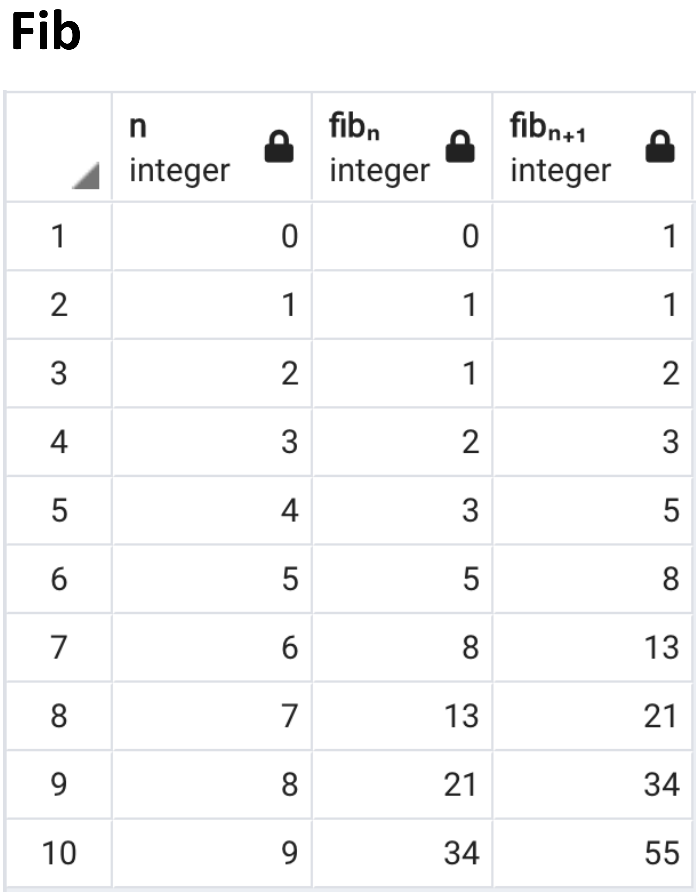


	n integer	fib_n integer	fib_{n+1} integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, …

```
WITH RECURSIVE Fib as (
      select 0 as n,
               0 as "fib_n",
               1 as "fib_n+1"
   UNION ALL
      select n+1,
               "fib_n+1",
               "fib_n" + "fib_n+1"
      from Fib)
SELECT * FROM Fib
LIMIT 10;
```



Fib

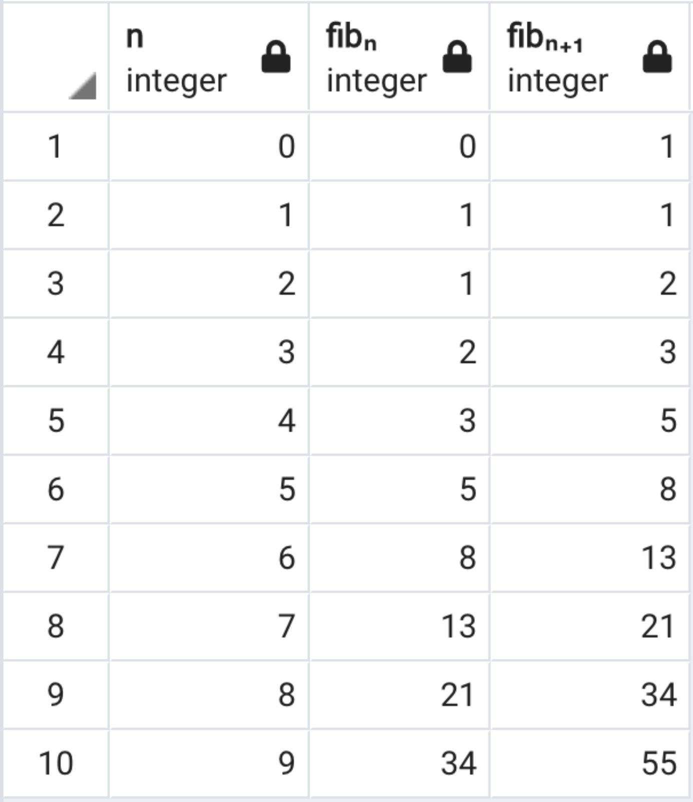
	n integer	fib_n integer	fib_{n+1} integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, ...

```
WITH RECURSIVE Fib(n,"fib_n","fib_n+1") as(
      select 0, 0, 1


   UNION ALL
      select n+1,
               "fib_n+1",
               "fib_n" + "fib_n+1"
      from Fib)
SELECT * FROM Fib
LIMIT 10;
```

Fib



	n integer	fib_n integer	fib_{n+1} integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

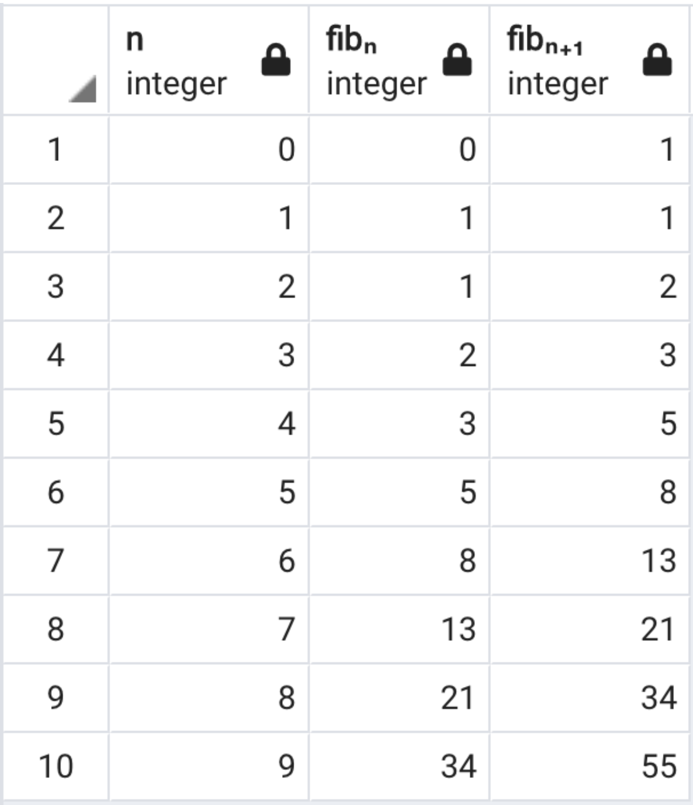
"This works because PostgreSQL's implementation evaluates only as many rows of a WITH query as are actually fetched by the parent query. Using this trick in production is not recommended, because other systems might work differently." Source: https://www.postgresql.org/docs/current/queries-with.html#QUERIES-WITH-RECURSIVE

Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

2. Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, ...

```
WITH RECURSIVE Fib(n,"fib_n","fib_n+1") as(
      select 0, 0, 1


   UNION ALL
      select n+1,
               "fib_n+1",
               "fib_n" + "fib_n+1"
      from Fib
      where n<9)
SELECT * FROM Fib;
```

Fib



	n integer	fib_n integer	fib_{n+1} integer
1	0	0	1
2	1	1	1
3	2	1	2
4	3	2	3
5	4	3	5
6	5	5	8
7	6	8	13
8	7	13	21
9	8	21	34
10	9	34	55

condition in WHERE clause is a more general way to write this query

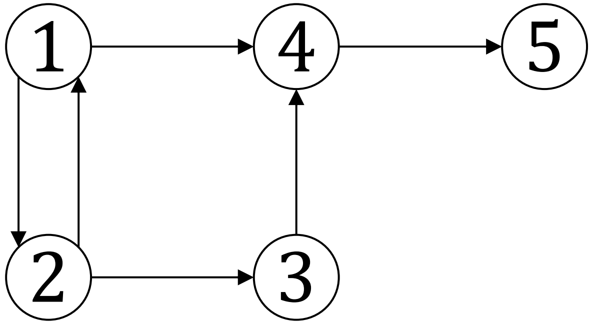
Example slightly adapted from: https://www.cybertec-postgresql.com/en/recursive-queries-postgresql/

3. Recursion on graphs

A for arcs or adjacencies (directed edges), S for source, T for target; another relation E (edges) have both directions

A(S,T)

501



"Find all paths (transitive closure)"

A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

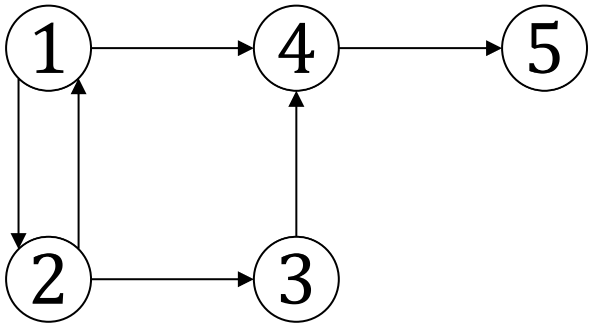
?

SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

3. Recursion on graphs

A for arcs or adjacencies (directed edges), S for source, T for target; another relation E (edges) have both directions → A(S,T)

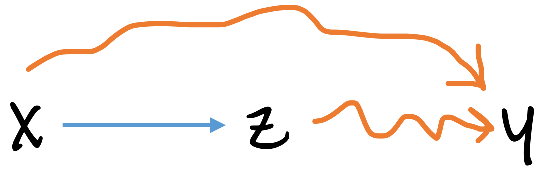
501



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

“Find all paths (transitive closure)”



1. Create a path for every arc

2. An arc + a path can make another path

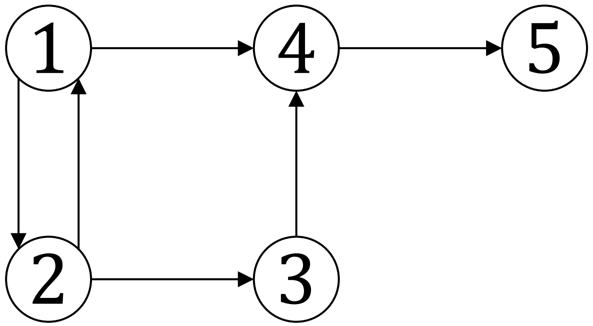
SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

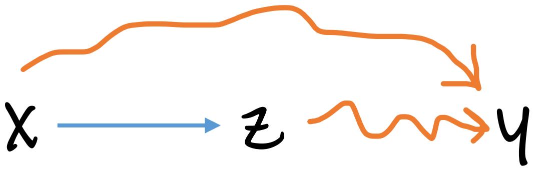
3. Recursion on graphs

A for arcs or adjacencies (directed edges), S for source, T for target; another relation E (edges) have both directions → A(S,T)

501



"Find all paths (transitive closure)"



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

For all nodes x and y:
If there is an arc from x to y,
then there is a path from x to y.

For all nodes x, z, and y:
If there is an arc from x to z, and there is a path from z to y
then there is a path from x to y.

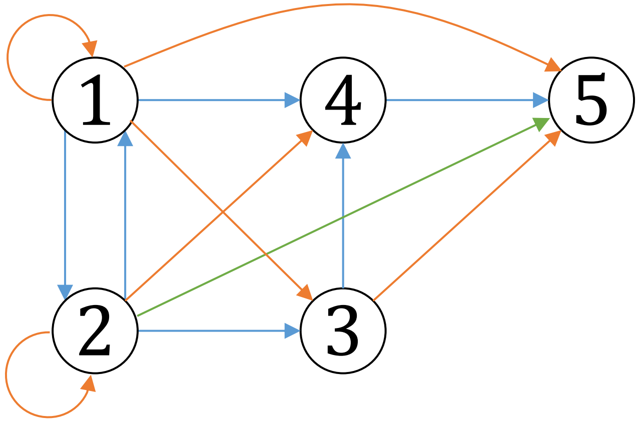
```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```

3. Recursion on graphs

P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).

A(S,T)

501



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

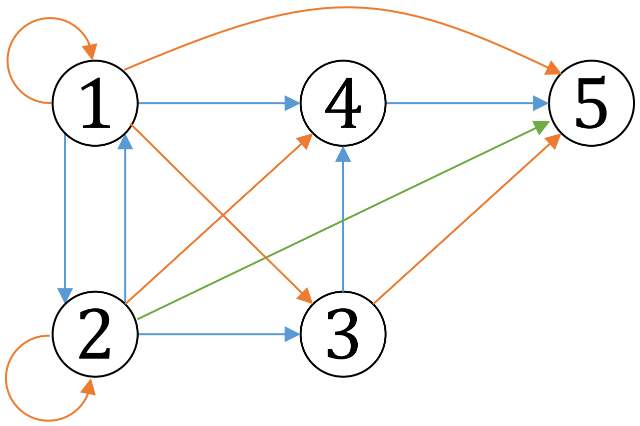
In SQL ?

3. Recursion on graphs

A(S,T)

501

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

```
WITH RECURSIVE P AS (
        ?
   UNION

        ?

SELECT *
FROM P
```

SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

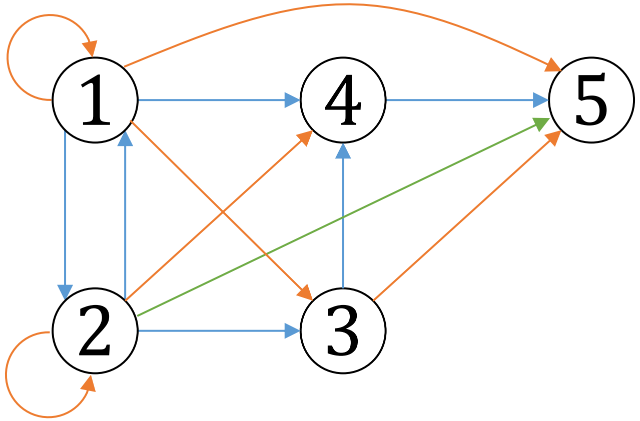
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

3. Recursion on graphs

P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).

A(S,T)

501



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION

      ?

SELECT *
FROM P
```

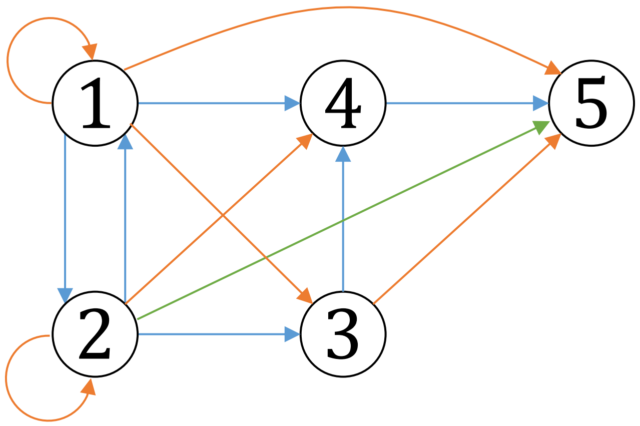
SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

3. Recursion on graphs

P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).

A(S,T)

501



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A.S, P.T
    FROM A, P
    WHERE A.T = P.S)
SELECT *
FROM P
```

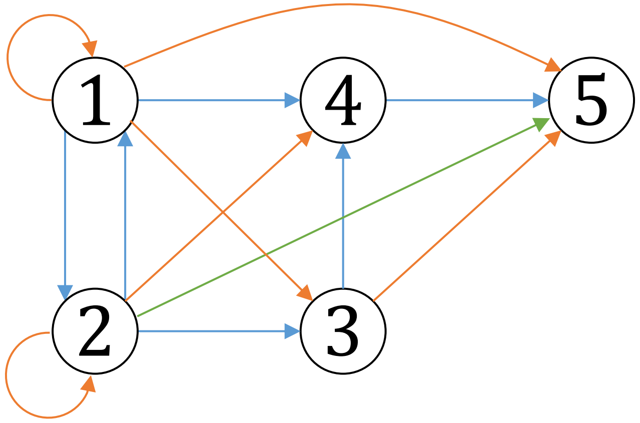
SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

3. Recursion on graphs

A(S,T)

501

```
P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).
```



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

Strictly speaking, this process is iteration, not recursion:

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A.S, P.T
    FROM A, P
    WHERE A.T = P.S)
SELECT *
FROM P
```

Recursion and Iteration both repeatedly execute a set of instructions.

- Recursion (self-similarity) is when a statement in a function calls itself repeatedly.
- Iteration (repetition) is when a loop repeatedly executes until the controlling condition becomes false.

See also: https://www.postgresql.org/docs/14/queries-with.html#QUERIES-WITH-RECURSIVE

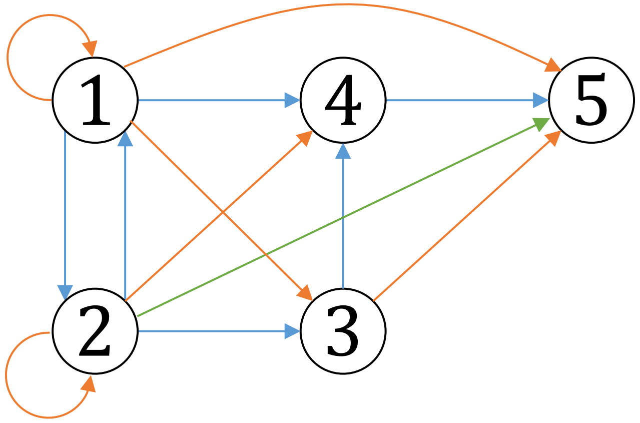
SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

3. Recursion on graphs

A(S,T)

501

P(x,y) :- A(x,y).
P(x,y) :- A(x,z), A(z,y).



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

Probe for understanding: how does the output change with this little change in the query

?

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A1.S, A2.T
    FROM A A1, A A2
    WHERE A1.T = A2.S)
SELECT *
FROM P
```

P

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

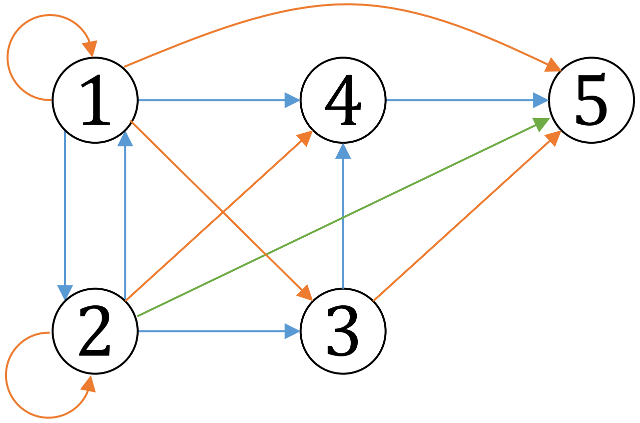
?

SQL database available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

3. Recursion on graphs

A(S,T)

501



A

S	T
1	2
1	4
2	1
2	3
3	4
4	5

P(x,y) :- A(x,y).
P(x,y) :- A(x,z), A(z,y).

Probe for understanding: how the output changes with this little change in the query:

```
WITH RECURSIVE P AS (
    SELECT S, T
    FROM A
  UNION
    SELECT A1.S, A2.T
    FROM A A1, A A2
    WHERE A1.T = A2.S)
SELECT *
FROM P
```

P

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

Challenge

???

- Write a query that finds the shortest path to each node from a starting node
- Create an interesting minimum database instance
- Show interesting variations
- https://www.postgresql.org/docs/14/queries-with.html

?

Updated 2/16/2024

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 10

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

2/16/2024

Pre-class conversations

- Last class summary
- Project discussions (today: first project ideas)

- today:
  - More on Datalog
  - What happens if we add negation? Answer: it depends on how we do it.
    - Datalog with stratified negation
    - Datalog with more genal negation (stable models), leads to ASP

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^{\neg}$: Datalog with stratified negation
  - Datalog$^{\pm}$
- Answer Set Programming (ASP)

Semantics of Datalog Programs

- Let **S** be a schema, **D** a database over **S**, and **P** be a Datalog program over **S** (i.e., all EDBs predicates belong to **S**)
- The result of evaluating **P** over **D** is a database **I** over the IDB schema of **P**
- We give 2 definitions:
  1. Fixpoint semantics operative (think procedural)
  2. model-theoretic declarative

1. Fixpoint semantics via the chase (operative definition)

Pseudo-code of a chase procedure:

Chase(P,D)

```
I := empty
repeat {
    if(D∪I satisfies all the rules of P), then return I
    Find a rule head(x) :- body(x,y) and constants a,b
        s.t. that D∪I contains body(a,b) but not head(a)
    I := I ∪ {head(a)}
}
```

("D∪I" is here just a set of tuples)

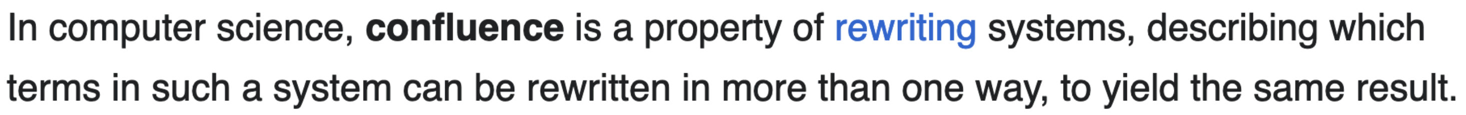
Notice since rules are monotone, I is also monotonically increasing

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Nondeterminism

- Note: the chase is underspecified (i.e., not fully defined)
  - There can be many ways of choosing the next violation to handle
  - And each choice can lead to new violations, and so on
- We can view the choice of a new violation as nondeterministic

Church-Rosser property (defined for term reduction):
*If term **a** can be reduced to both **b** and **c**, then there must be a further term **d** (possibly equal to either b or c) to which both **b** and **c** can be reduced.*

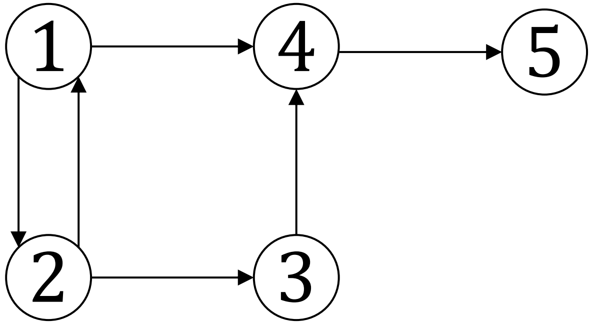


In computer science, **confluence** is a property of rewriting systems, describing which terms in such a system can be rewritten in more than one way, to yield the same result.

$$\begin{tikzcd}
\mathbf{a} \arrow[r, "*"] \arrow[d, "*"'] & \mathbf{b} \arrow[d, dashed, "*"] \\
\mathbf{c} \arrow[r, dashed, "*"] & \mathbf{d}
\end{tikzcd}$$

Also see: https://en.wikipedia.org/wiki/Church%E2%80%93Rosser_theorem , https://en.wikipedia.org/wiki/Confluence_(abstract_rewriting)

Example



```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

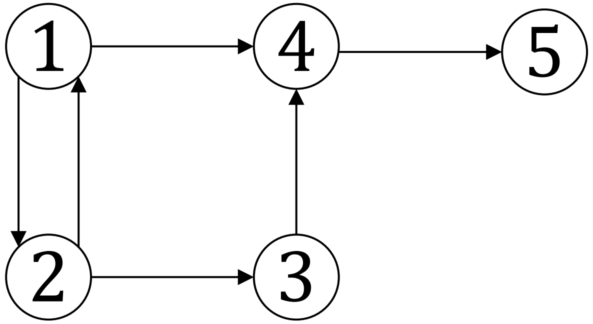
Arc

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

Path

Reachable

Example



```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

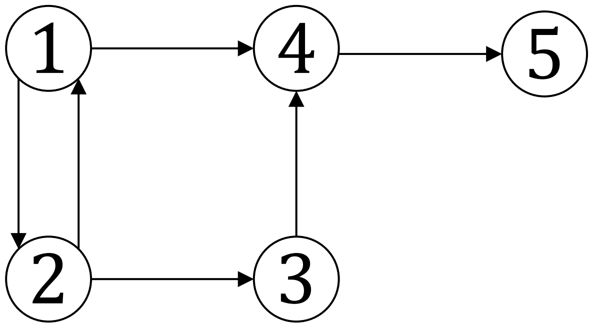
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2

Reachable

Example



```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

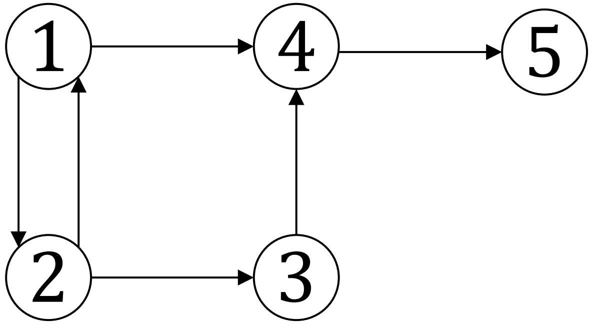
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2
2	1

Reachable

Example



```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

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

Path

1	2
2	1
2	3

Reachable

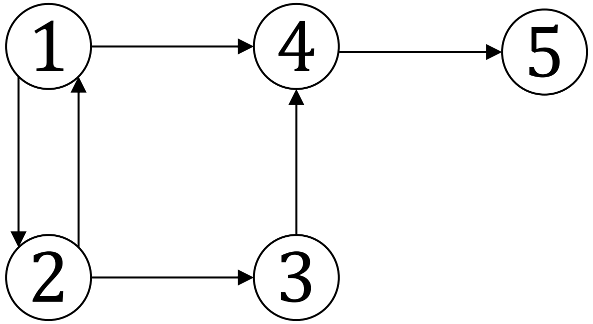
Example

$\{x,y,z\} \rightarrow \{1,2\}$

x → 1

z → 2

y → 1



Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).

Arc

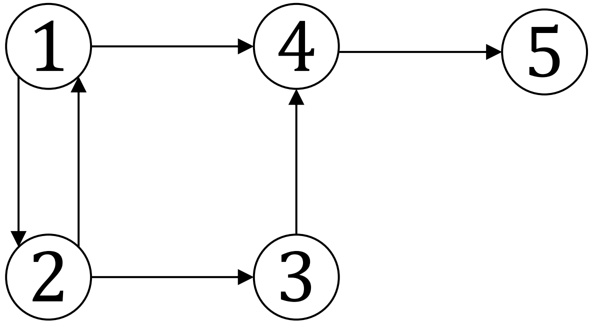
1	2
2	1
2	3
1	4
3	4
4	5

Path

1	2
2	1
2	3
1	1

Reachable

Example



```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
Reachable(y) :- Path('1',y).
```

Arc

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

Path

1	2
2	1
2	3

Reachable

2

2. Minimal model semantics (model-theoretic definition)

- We say that IDB **I** is a model of Datalog program **P** (w.r.t. **EDB D**) if **D**U**I** satisfies all the rules of **P**

$$\forall \text{var}[\text{Head}(\text{IDB}) \Leftarrow \text{Body}(\text{EDB}, \text{IDB})]$$

- We say that **I** is a minimal model if **I** does not properly contain any other model

- Theorem: there exists one minimal model

Illustration with our example

```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
```

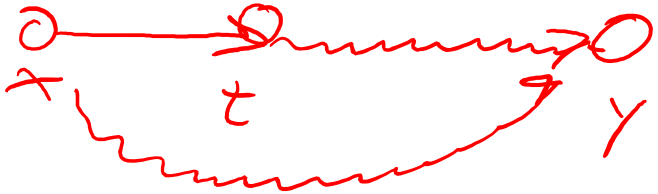
1. Fixpoint semantics

2. Minimal model semantics: smallest Path s.t.

Illustration with our example

Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).

1. Fixpoint semantics



```
$Path^{(0)} := \emptyset$, $t := 0$
Repeat {
    inc(t)
    $Path^{(t)}(x, y) := Arc(x,y) \cup \Pi_{xy}(Arc(x,z) \bowtie Path^{(t-1)}(z,y))$
until $Path^{(t)} = Path^{(t-1)}$}
```

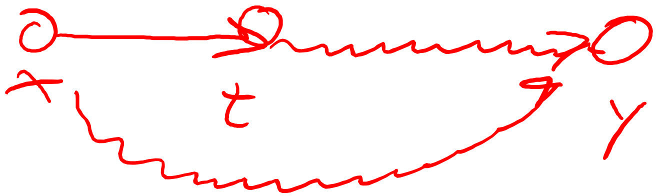
immediate consequence operator "T_P":
$P^{(t)} = T_P(P^{(t-1)})$

2. Minimal model semantics: smallest relation Path s.t.

Illustration with our example

```
Path(x,y) :- Arc(x,y).
Path(x,y) :- Arc(x,z), Path(z,y).
```

1. Fixpoint semantics



```
$\text{Path}^{(0)} := \emptyset$, t:=0
Repeat {
    inc(t)
    $\text{Path}^{(t)}(x,y) := \text{Arc}(x,y) \cup \Pi_{xy}(\text{Arc}(x,z) \bowtie \text{Path}^{(t-1)}(z,y))$
until $\text{Path}^{(t)} = \text{Path}^{(t-1)}$}
```

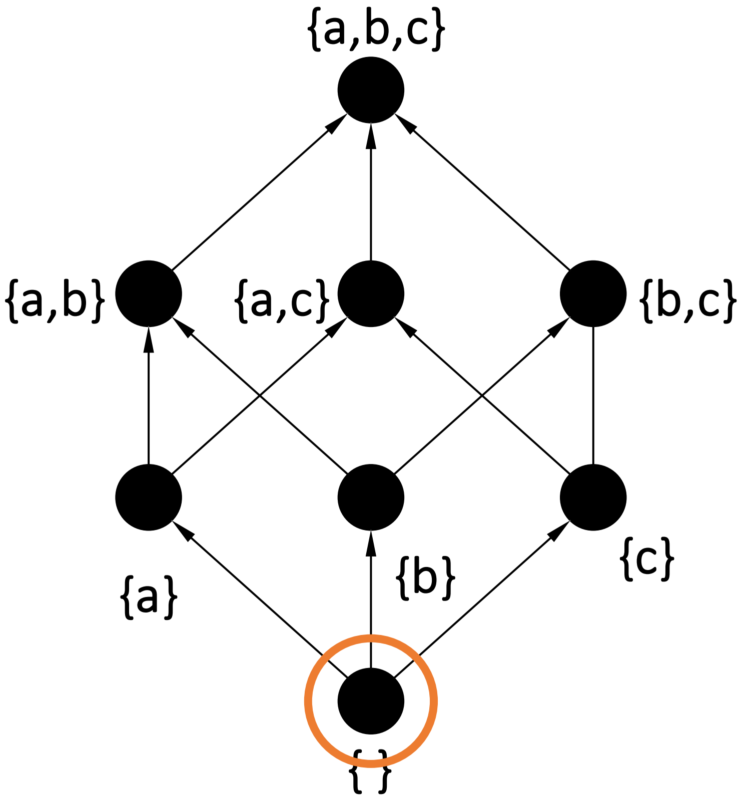
immediate consequence operator "T_P":

$P^{(t)} = T_P(P^{(t-1)})$

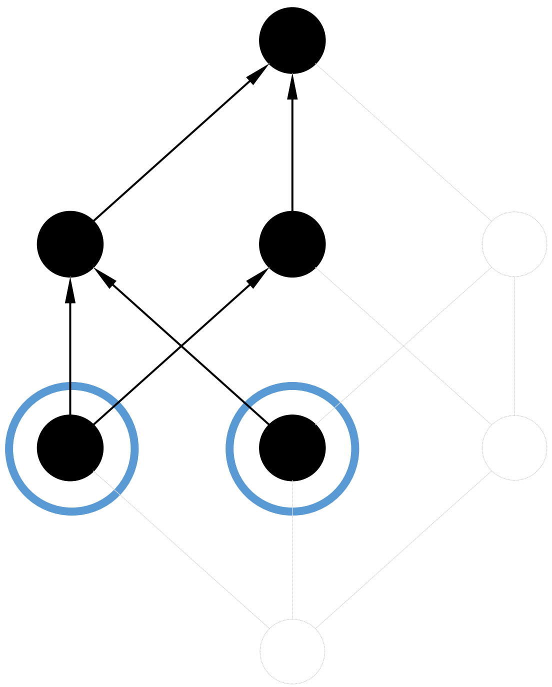
2. Minimal model semantics: smallest relation Path s.t.

$$\forall x,y\,[\text{Arc}(x,y) \Rightarrow \text{Path}(x,y)] \land$$
$$\forall x,y,z\,[\text{Arc}(x,z) \land \text{Path}(z,y) \Rightarrow \text{Path}(x,y)]$$

Minimum (least) element vs minimal elements in partial orders



Consider a partial order (S,≤). The set of elements from S are represented by black circles, arrows show partial order between elements.



1 least element

An element a in S is called a least (or minimum) element of S if a ≤ x for all x in S.

2 minimal elements

An element a in S is called a minimal element of S if there is no element b in A such that b ≤ a.

For more details see e.g. "Davey, Priestley. Introduction To Lattices And Order (book, 2nd ed). 2002", https://doi.org/10.1017/CBO9780511809088

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Datalog Semantics & equivalence b/w the definitions

(nondeterministic)

1. The fixpoint semantics tells us how to compute a Datalog query
2. The minimal model semantics is more declarative: only says what we get

THEOREM: For all Datalog programs P and DBs D
there is a unique minimal model,
and every chase returns this model

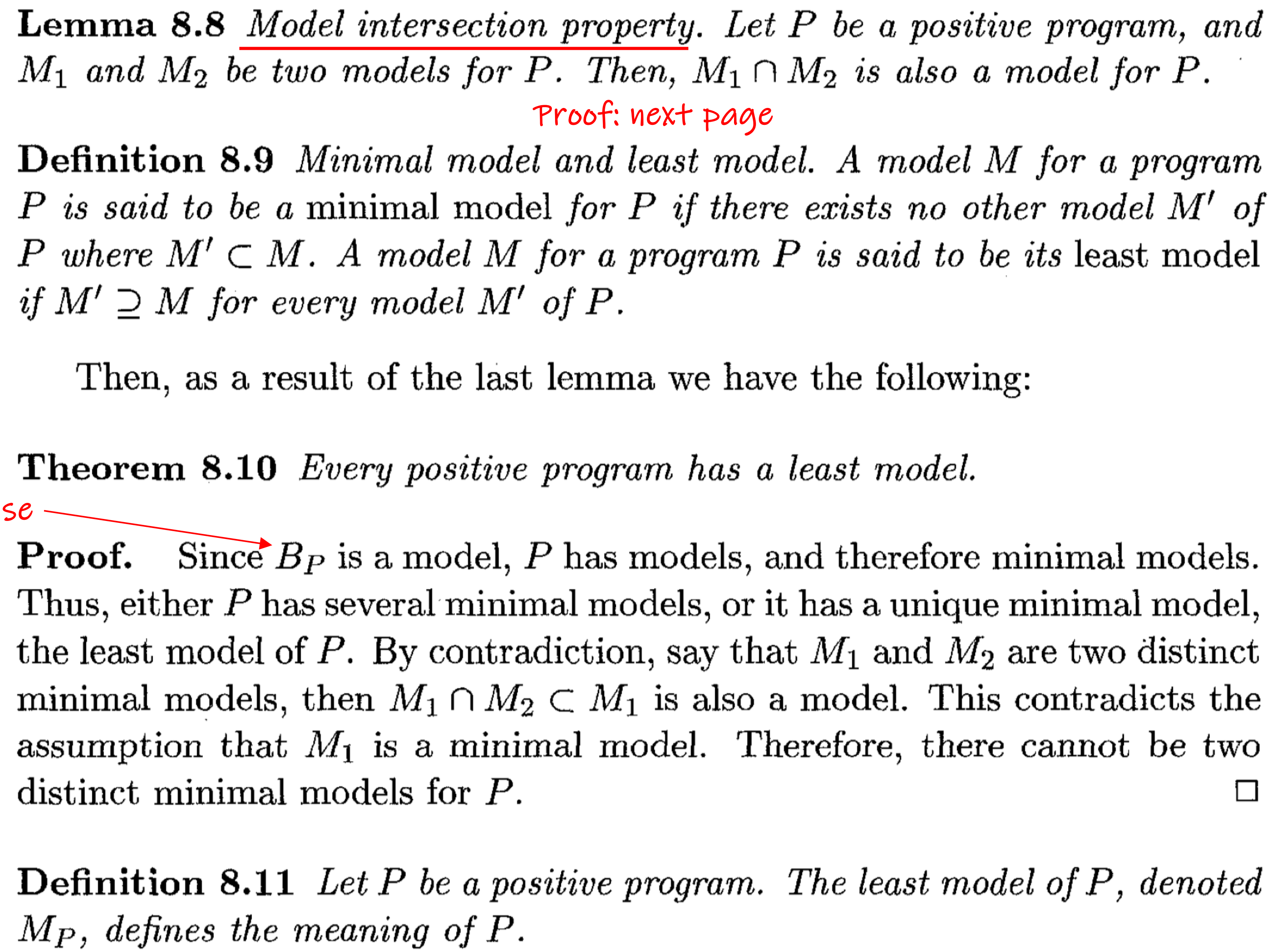
Proof sketch:

1. If I_1 and I_2 are models, so are $\mathrm{I}_1 \cap \mathrm{I}_2$
2. Every chase returns a model (finite)
3. Pick a chase and prove by induction: If I' is a model, then every intermediate I is contained in I' (monotonicity)

The minimal model is the *result*, denoted P(D)

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Details



Lemma 8.8 *Model intersection property. Let P be a positive program, and M_1 and M_2 be two models for P. Then, $M_1 \cap M_2$ is also a model for P.*

Proof: next page

Definition 8.9 *Minimal model and least model. A model M for a program P is said to be a* minimal model *for P if there exists no other model M' of P where $M' \subset M$. A model M for a program P is said to be its* least model *if $M' \supseteq M$ for every model M' of P.*

Then, as a result of the last lemma we have the following:

Theorem 8.10 *Every positive program has a least model.*

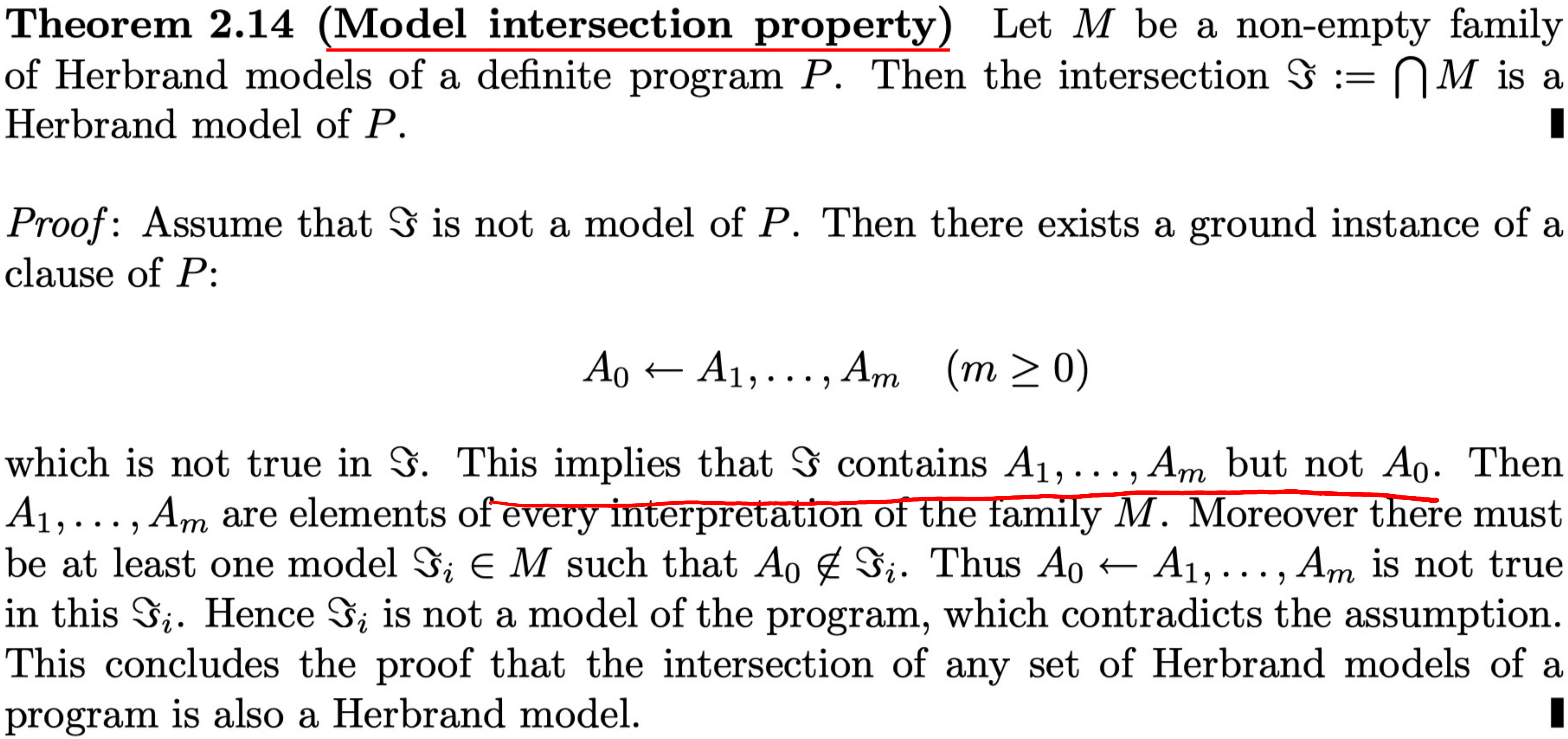
Herbrand base

Proof. Since B_P is a model, P has models, and therefore minimal models. Thus, either P has several minimal models, or it has a unique minimal model, the least model of P. By contradiction, say that M_1 and M_2 are two distinct minimal models, then $M_1 \cap M_2 \subset M_1$ is also a model. This contradicts the assumption that M_1 is a minimal model. Therefore, there cannot be two distinct minimal models for P. □

Definition 8.11 *Let P be a positive program. The least model of P, denoted M_P, defines the meaning of P.*

Source: Zaniolo et al. Advanced Database systems. 1997. Section 8.9. https://dl.acm.org/doi/book/10.5555/260822
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Details



Theorem 2.14 (Model intersection property) Let M be a non-empty family of Herbrand models of a definite program P. Then the intersection $\Im := \bigcap M$ is a Herbrand model of P. ▮

Proof: Assume that $\Im$ is not a model of P. Then there exists a ground instance of a clause of P:

$$A_0 \leftarrow A_1, \ldots, A_m \quad (m \geq 0)$$

which is not true in $\Im$. This implies that $\Im$ contains $A_1, \ldots, A_m$ but not A_0. Then $A_1, \ldots, A_m$ are elements of every interpretation of the family M. Moreover there must be at least one model $\Im_i \in M$ such that $A_0 \notin \Im_i$. Thus $A_0 \leftarrow A_1, \ldots, A_m$ is not true in this $\Im_i$. Hence $\Im_i$ is not a model of the program, which contradicts the assumption. This concludes the proof that the intersection of any set of Herbrand models of a program is also a Herbrand model. ▮

Source: Nilsson, Maluszynski. Logic, Programming and Prolog, 2nd ed, 2000. Chapter 2: http://tinman.cs.gsu.edu/~raj/8710/f02/bok.pdf

Semantics Summary

1. Fixpoint-theoretic
   - Most "operational": Based on the immediate consequence operator for a Datalog program.

2. Model-theoretic
   - Most "declarative": Based on model-theoretic semantics of first order logic. View rules as logical constraints.

Semantics Summary

1. Fixpoint-theoretic
   - Most "operational": Based on the immediate consequence operator for a Datalog program.
   - Least fixpoint is reached after finitely many iterations of the immediate consequence operator.
   - Basis for practical, bottom-up evaluation strategy.
2. Model-theoretic
   - Most "declarative": Based on model-theoretic semantics of first order logic. View rules as logical constraints.
   - Given input DB D and Datalog program P, find the smallest possible DB instance D' that extends D and satisfies all constraints in P.

Monotonicity

- Can Datalog express difference?
  - Answer: No!
- Proof: Datalog is monotone, difference is not
  - That is, if D and D' are such that every relation of D is contained in the corresponding relation of D' ($\mathrm{D} \subseteq \mathrm{D}'$), then $\mathrm{P}(\mathrm{D}) \subseteq \mathrm{P}(\mathrm{D}')$

$$\mathrm{D} \subseteq \mathrm{D}' \quad \Rightarrow \quad \mathrm{P}(\mathrm{D}) \subseteq \mathrm{P}(\mathrm{D}')$$

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^\neg$: Datalog with stratified negation
  - Datalog$^\pm$
- Answer Set Programming (ASP)

Datalog Evaluation Algorithms

- Goal: preserve the efficiency of query optimizers, yet extend them to recursion
- Two general strategies we will discuss:
  - 1. Naive Datalog evaluation
  - 2. Semi-naive Datalog evaluation
- More powerful optimizations:
  - 3. Magic sets (which we will not cover, or may revisit later under "Topic 3: efficient query evaluation & factorized representations")

1. Naive Datalog evaluation

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.

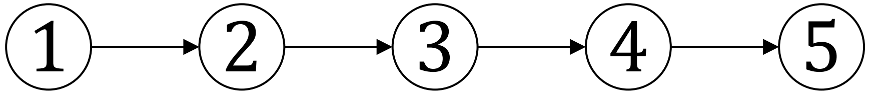
```
$P^{(0)} := \emptyset$, $t := 0$
Repeat {
    inc(t)
    $P^{(t)}(x,y) := A(x,y) \cup \Pi_{-z}(A(x,z) \bowtie P^{(t-1)}(z,y))$
until $P^{(t)} = P^{(t-1)}$}
```

immediate consequence operator "T_P":
$P^{(t)} = T_P(P^{(t-1)})$

- Problem: The same facts are discovered over and over again
- Goal: The semi-naive algorithm tries to reduce the number of facts discovered multiple times

Example

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.



A

1	2
2	3
3	4
4	5

$P^{(1)}$?

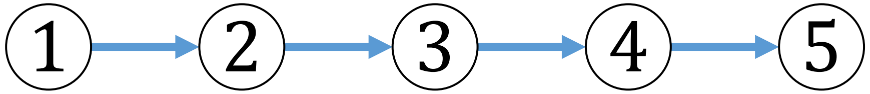
$P^{(2)}$?

$P^{(3)}$?

$P^{(4)}$?

Example

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.



paths of LEN ≤ 1

A

1	2
2	3
3	4
4	5

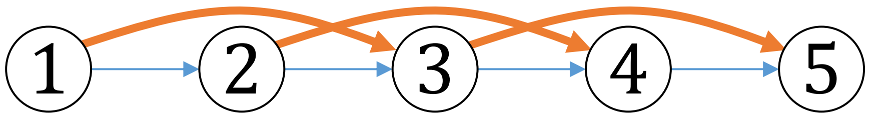
$P^{(1)}$

1	2
2	3
3	4
4	5

L=1

Example

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.



A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

$P^{(1)}$

1	2
2	3
3	4
4	5

L=1

paths of LEN ≤ 2

$P^{(2)}$

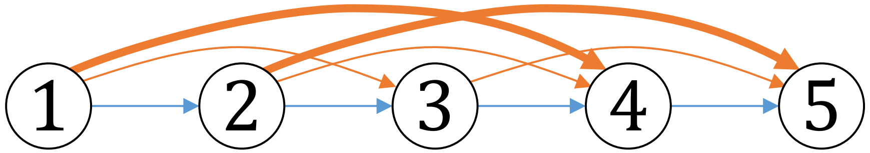
1	2
2	3
3	4
4	5
1	3
2	4
3	5

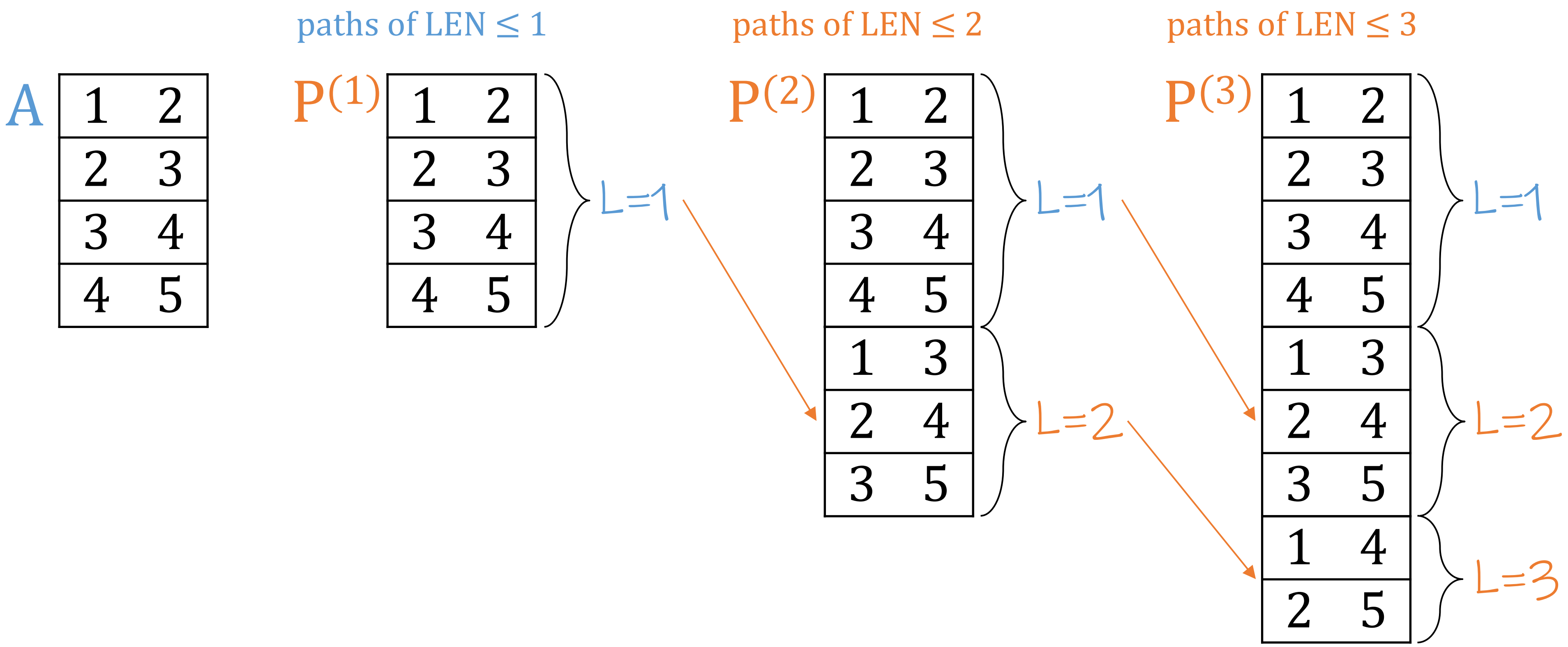
L=1

L=2

Example

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.





A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

$P^{(1)}$

1	2
2	3
3	4
4	5

paths of LEN ≤ 2

$P^{(2)}$

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

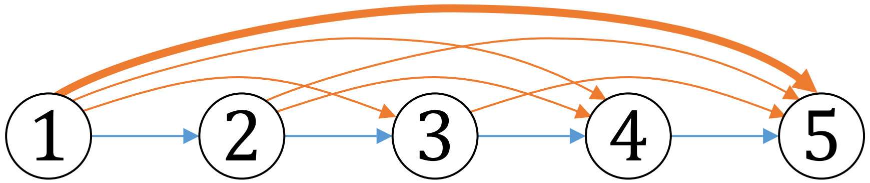
paths of LEN ≤ 3

$P^{(3)}$

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

Example

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.



A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

$P^{(1)}$

1	2
2	3
3	4
4	5

L=1

paths of LEN ≤ 2

$P^{(2)}$

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

L=1
L=2

paths of LEN ≤ 3

$P^{(3)}$

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

L=1
L=2
L=3

paths of LEN ≤ 4

$P^{(4)}$

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

L=1
L=2
L=3
L=4

Side-topic: Incremental View Maintentance

Background: Incremental View Maintenace

Let Q be a "view" computed by a single Datalog rule without recursion, thus a simple conjunctive query

$Q \text{ :- } R_1, R_2, \ldots$

```
SELECT ...
FROM R1
NATURAL JOIN R2
NATURAL JOIN R3 ...
```

Add tuples to some of the relations:

$$R_1 \longleftarrow R_1 \cup \Delta R_1,\ R_2 \longleftarrow R_2 \cup \Delta R_2,\ \ldots$$

Then the view Q will also increase in size:

$$Q \longleftarrow Q \cup \Delta Q$$

Incremental view maintenance problem:
Compute ΔQ without having to recompute **Q** from scratch

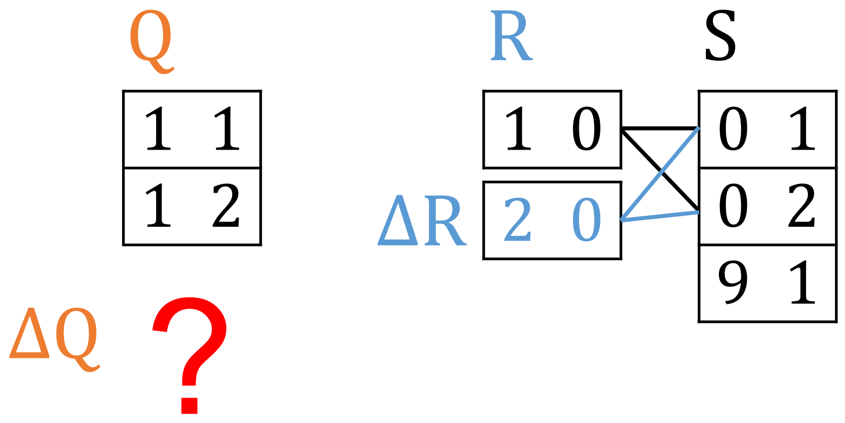
Background: Incremental View Maintenace

Example 1:

Q(x,y) :- R(x,z), S(z,y)

ΔQ(x,y):- ?

If R ⟵ R ∪ ΔR,
then what is ΔQ ?



Q

1	1
1	2

R

1	0

ΔR

2	0

S

0	1
0	2
9	1

ΔQ ?

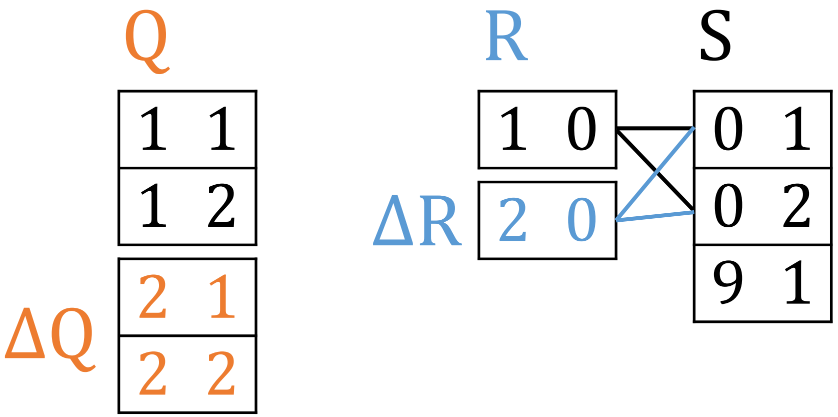
Background: Incremental View Maintenace

Example 1:

Q(x,y) :- R(x,z), S(z,y)

ΔQ(x,y):- ?

If R ⟵ R ∪ ΔR,
then what is ΔQ ?



Q

1	1
1	2

ΔQ

2	1
2	2

R

1	0

ΔR

2	0

S

0	1
0	2
9	1

Background: Incremental View Maintenace

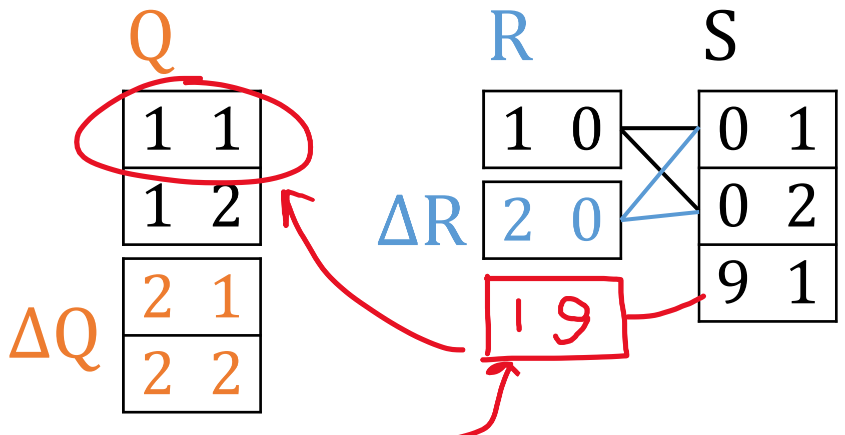
Example 1:

Q(x,y) :- R(x,z), S(z,y)

ΔQ(x,y):- ΔR(x,z), S(z,y)

If R ⟵ R ∪ ΔR,
then what is ΔQ ?

(to be more precise: we still need to subtract Q:
$\Delta Q = \Delta R \bowtie S - Q$, e.g. for ΔR = (1,1). More on that later)



Q

1	1
1	2

ΔQ

2	1
2	2

R

1	0

ΔR

2	0
1	9

S

0	1
0	2
9	1

Relational Algebra:

$$Q = R \bowtie S$$

$$Q \cup \Delta Q = (R \cup \Delta R) \bowtie S$$

?

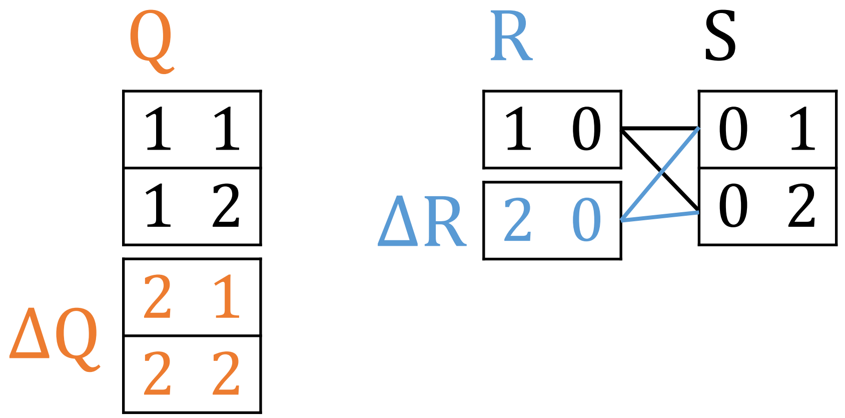
Background: Incremental View Maintenace

Example 1:

Q(x,y) :- R(x,z), S(z,y)

ΔQ(x,y):- ΔR(x,z), S(z,y)

If R ⟵ R ∪ ΔR,
then what is ΔQ ?



Q

1	1
1	2

ΔQ

2	1
2	2

R

1	0

ΔR

2	0

S

0	1
0	2

(to be more precise: we still need to subtract Q:
$\Delta Q = \Delta R \bowtie S - Q$, e.g. for $\Delta R = (1,1)$. More on that later)

$$z = x \cdot y$$

$$z + \Delta z = (x + \Delta x) \cdot y$$

?

Relational Algebra:

$$Q = R \bowtie S$$

$$Q \cup \Delta Q = (R \cup \Delta R) \bowtie S$$

?

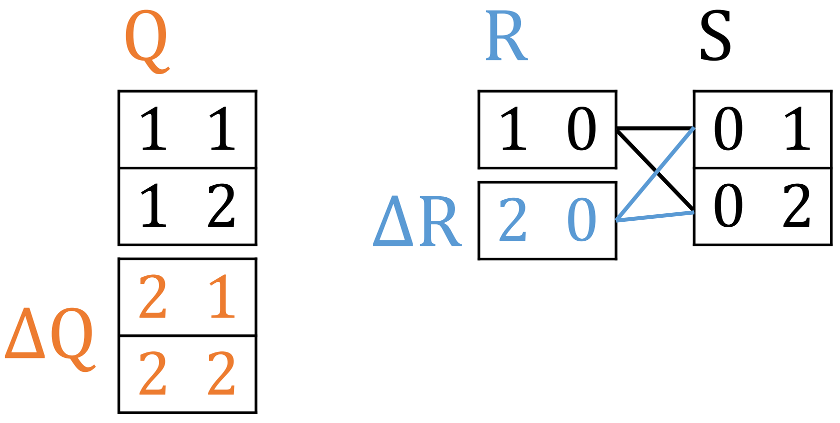
Background: Incremental View Maintenace

Example 1:

Q(x,y) :- R(x,z), S(z,y)

ΔQ(x,y):- ΔR(x,z), S(z,y)

If R ⟵ R ∪ ΔR,
then what is ΔQ ?



Q

1	1
1	2

ΔQ

2	1
2	2

R

1	0

ΔR

2	0

S

0	1
0	2

(to be more precise: we still need to subtract Q:
ΔQ=ΔR⋈S – Q, e.g. for ΔR = (1,1). More on that later)

Multiplication ⊗ distributes
over Addition ⊕

(a+b)c
= ac+bc

$z = x \cdot y$

$z+\Delta z = (x+\Delta x)\cdot y$

$z+\Delta z = (x\cdot y)+(\Delta x\cdot y)$

$z+\Delta z = \quad z \quad +(\Delta x\cdot y)$

$\Delta z = \Delta x\cdot y$

Relational Algebra:

$Q = R \bowtie S$

$Q \cup \Delta Q = (R \cup \Delta R) \bowtie S$

$Q \cup \Delta Q = (R \bowtie S) \cup (\Delta R \bowtie S)$

$Q \cup \Delta Q = \quad Q \quad \cup (\Delta R \bowtie S)$

$\Delta Q = \Delta R \bowtie S$

Join ⋈ distributes
over union ∪

(a∪b)⋈c
= a⋈c ∪ b⋈c

Background: Incremental View Maintenace

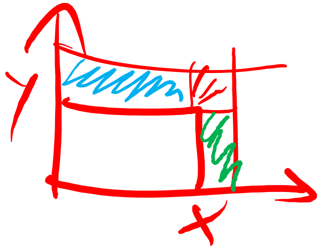
Example 2:

Q(x,y) :- R(x,z), S(z,y)

If R ⟵ R ∪ ΔR, and S ⟵ S ∪ ΔS,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)

?



$z = x \cdot y$

$z + \Delta z = (x + \Delta x) \cdot (y + \Delta y)$

?

Background: Incremental View Maintenace

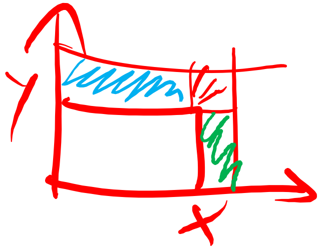
Example 2:

Q(x,y) :- R(x,z), S(z,y)

If R ⟵ R ∪ ΔR, and S ⟵ S ∪ ΔS,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)

?



$z = x \cdot y$

$z + \Delta z = (x + \Delta x) \cdot (y + \Delta y)$

$z + \Delta z = (x \cdot y) + (\Delta x \cdot y) + (x \cdot \Delta y) + (\Delta x \cdot \Delta y)$

$z + \Delta z = \quad z \quad + (\Delta x \cdot y) + (x \cdot \Delta y) + (\Delta x \cdot \Delta y)$

$\Delta z = (\Delta x \cdot y) + (x \cdot \Delta y) + (\Delta x \cdot \Delta y)$

Relational Algebra:

$Q = R \bowtie S$

$Q \cup \Delta Q = (R \cup \Delta R) \bowtie (S \cup \Delta S)$

$Q \cup \Delta Q = (R \bowtie S) \cup (\Delta R \bowtie S) \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie \Delta S)$

$Q \cup \Delta Q = \quad Q \quad \cup (\Delta R \bowtie S) \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie \Delta S)$

$\Delta Q = (\Delta R \bowtie S) \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie \Delta S)$

Background: Incremental View Maintenace

Example 2:

Q(x,y) :- R(x,z), S(z,y)

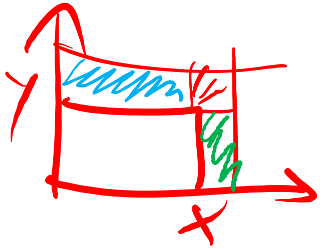
ΔQ(x,y):- ΔR(x,z), S(z,y)

ΔQ(x,y):- R(x,z), ΔS(z,y)

ΔQ(x,y):- ΔR(x,z), ΔS(z,y)

If R ⟵ R ∪ ΔR, and S ⟵ S ∪ ΔS,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)



$$z = x\cdot y$$
$$z+\Delta z = (x+\Delta x)\cdot(y+\Delta y)$$
$$z+\Delta z = (x\cdot y)+(\Delta x\cdot y)+(x\cdot \Delta y)+(\Delta x\cdot \Delta y)$$
$$z+\Delta z = \quad z \quad +(\Delta x\cdot y)+(x\cdot \Delta y)+(\Delta x\cdot \Delta y)$$
$$\Delta z = (\Delta x\cdot y)+(x\cdot \Delta y)+(\Delta x\cdot \Delta y)$$

Relational Algebra:

$$Q = R \bowtie S$$
$$Q\cup\Delta Q = (R\cup\Delta R) \bowtie (S\cup\Delta S)$$
$$Q\cup\Delta Q = (R\bowtie S)\cup(\Delta R\bowtie S)\cup(R\bowtie \Delta S)\cup(\Delta R\bowtie \Delta S)$$
$$Q\cup\Delta Q = \quad Q \quad \cup(\Delta R\bowtie S)\cup(R\bowtie \Delta S)\cup(\Delta R\bowtie \Delta S)$$
$$\Delta Q = (\Delta R\bowtie S)\cup(R\bowtie \Delta S)\cup(\Delta R\bowtie \Delta S)$$

Background: Incremental View Maintenace

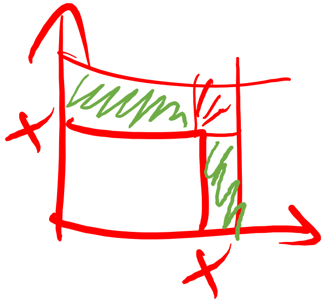
Example 3:

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

If $R \longleftarrow R \cup \Delta R$,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)

?



$z = x^2$

$z+\Delta z = (x+\Delta x)^2$

?

Background: Incremental View Maintenace

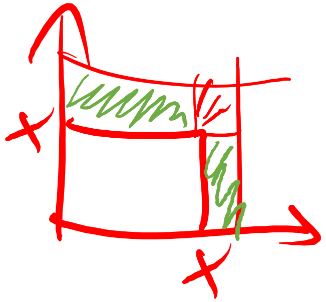
Example 3:

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

If $R \longleftarrow R \cup \Delta R$,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)

?



$$z = x^2$$

$$z+\Delta z = (x+\Delta x)^2$$

$$z+\Delta z = x^2+(\Delta x\cdot x)+(x\cdot\Delta x)+\Delta x^2$$

$$z+\Delta z = z + 2x\Delta x+\Delta x^2$$

$$\Delta z = 2x\Delta x+\Delta x^2$$

Relational Algebra:

$$Q = R \bowtie_c R$$

$$Q\cup\Delta Q = (R\cup\Delta R) \bowtie_c (R\cup\Delta R)$$

?

Background: Incremental View Maintenace

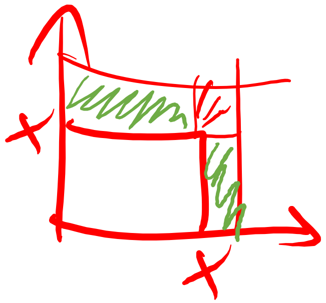
Example 3:

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

If $R \longleftarrow R \cup \Delta R$,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)

?



$$z = x^2$$

$$z+\Delta z = (x+\Delta x)^2$$

$$z+\Delta z = x^2+(\Delta x\cdot x)+(x\cdot\Delta x)+\Delta x^2$$

$$z+\Delta z = z + 2x\Delta x+\Delta x^2$$

$$\Delta z = 2x\Delta x+\Delta x^2$$

Relational Algebra:

$$Q = R \bowtie_c R$$

$$Q\cup\Delta Q = (R\cup\Delta R) \bowtie_c (R\cup\Delta R)$$

$$Q\cup\Delta Q = (R\bowtie_c R)\cup(\Delta R\bowtie_c R)\cup(R\bowtie_c \Delta R)\cup(\Delta R\bowtie_c \Delta R)$$

$$Q\cup\Delta Q = Q \cup(\Delta R\bowtie_c R)\cup(R\bowtie_c \Delta R)\cup(\Delta R\bowtie_c \Delta R)$$

$$\Delta Q = (\Delta R\bowtie_c R)\cup(R\bowtie_c \Delta R)\cup(\Delta R\bowtie_c \Delta R)$$

Background: Incremental View Maintenace

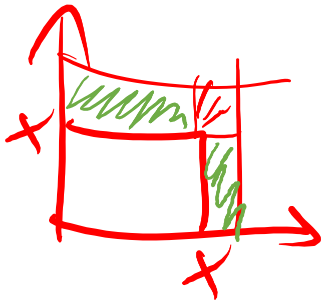
Example 3:

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

ΔQ(x,y):- ΔR(x,z), R(z,y)
ΔQ(x,y):- R(x,z), ΔR(z,y)
ΔQ(x,y):- ΔR(x,z), ΔR(z,y)

If $R \longleftarrow R \cup \Delta R$,
then what is ΔQ ?

(as before, we ignore the subtraction of Q here)



$$z = x^2$$
$$z+\Delta z = (x+\Delta x)^2$$
$$z+\Delta z = x^2+(\Delta x\cdot x)+(x\cdot\Delta x)+\Delta x^2$$
$$z+\Delta z = z + 2x\Delta x+\Delta x^2$$
$$\Delta z = 2x\Delta x+\Delta x^2$$

Relational Algebra:

$$Q = R \bowtie_c R$$
$$Q \cup \Delta Q = (R \cup \Delta R) \bowtie_c (R \cup \Delta R)$$
$$Q \cup \Delta Q = (R \bowtie_c R) \cup (\Delta R \bowtie_c R) \cup (R \bowtie_c \Delta R) \cup (\Delta R \bowtie_c \Delta R)$$
$$Q \cup \Delta Q = Q \cup (\Delta R \bowtie_c R) \cup (R \bowtie_c \Delta R) \cup (\Delta R \bowtie_c \Delta R)$$
$$\Delta Q = (\Delta R \bowtie_c R) \cup (R \bowtie_c \Delta R) \cup (\Delta R \bowtie_c \Delta R)$$

Back to Datalog evaluation

2. Semi-Naive Datalog evaluation

P(x,y) :- A(x,y).
P(x,y) :- A(x,z), P(z,y).

Recall the naive evaluation:

```
$P^{(0)} := \emptyset$, t:=0
Repeat {
    inc(t)
    $P^{(t)}(x,y) := A(x,y) \cup \pi_{xy}(A(x,z) \bowtie P^{(t-1)}(z,y))$
until $P^{(t)} = P^{(t-1)}$}
```

immediate consequence operator "T_P":
$P^{(t)} = T_P(P^{(t-1)})$

Semi-naive evaluation:

```
$P := A(x,z)$; $\Delta P^{(0)} := A(x,z)$
Repeat {
    inc(t)
    $\Delta P^{(t)}(x,y) := \pi_{xy}(A(x,z) \bowtie \Delta P^{(t-1)}(z,y)) - P(x,y)$
    $P := P \cup \Delta P^{(t)}$
until $\Delta P^{(t)} = \emptyset$}
```

"incrementalized" immediate consequence operator:
$\Delta P^{(t)} = T_P(\Delta P^{(t-1)}) - P^{(t-1)}$

The idea of semi-naive evaluation predates following paper which is often cited as main reference:
Bancilhon, Ramakrishnan. An Amateur's Introduction to Recursive Query Processing Strategies. SIGMOD 1986. https://doi.org/10.1145/16894.16859 (the 1988 revision is better)
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

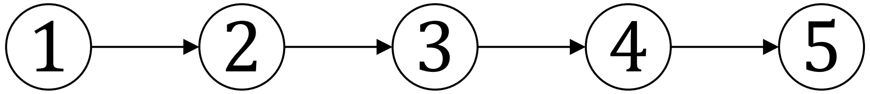
Example

$P^{(t)}(x,y) \text{ :- } A(x,y).$

$P^{(t)}(x,y) \text{ :- } A(x,z),\ P^{(t-1)}(z,y).$

$\Delta P^{(t)}(x,y) \text{ :- } A(x,z),\ \Delta P^{(t-1)}(z,y),\ \text{not } P(x,y).$

$P(x,y) \text{ :- } \Delta P^{(t)}(x,y).$



paths of LEN ≤ 1

A

1	2
2	3
3	4
4	5

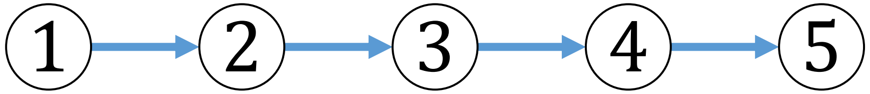
P

?

Example

$P^{(t)}(x,y) \text{ :- } A(x,y).$
$P^{(t)}(x,y) \text{ :- } A(x,z),\ P^{(t-1)}(z,y).$

$\Delta P^{(t)}(x,y) \text{ :- } A(x,z),\ \Delta P^{(t-1)}(z,y), \text{ not } P(x,y).$
$P(x,y) \text{ :- } \Delta P^{(t)}(x,y).$



A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

P

1	2
2	3
3	4
4	5

$\Delta P^{(1)}$

paths of LEN ≤ 2

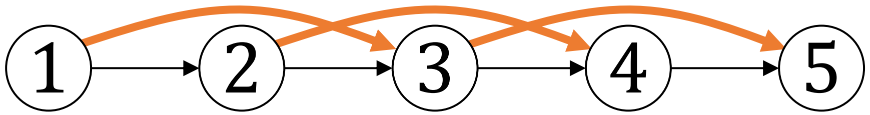
P

?

Example

$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.

$\Delta P^{(t)}(x,y)$:- $A(x,z)$, $\Delta P^{(t-1)}(z,y)$, not $P(x,y)$.
$P(x,y)$:- $\Delta P^{(t)}(x,y)$.



A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

P

1	2
2	3
3	4
4	5

$\Delta P^{(1)}$

paths of LEN ≤ 2

P

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

$\Delta P^{(2)}$

paths of LEN ≤ 3

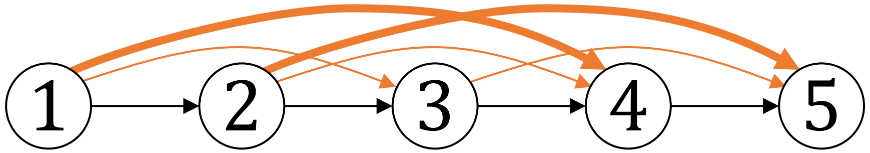
P

?

Example

$P^{(t)}$(x,y) :- A(x,y).
$P^{(t)}$(x,y) :- A(x,z), $P^{(t-1)}$(z,y).

$\Delta P^{(t)}$(x,y) :- A(x,z), $\Delta P^{(t-1)}$(z,y), not P(x,y).
P(x,y) :- $\Delta P^{(t)}$(x,y).



A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

P

1	2
2	3
3	4
4	5

$\Delta P^{(1)}$

paths of LEN ≤ 2

P

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

$\Delta P^{(2)}$

paths of LEN ≤ 3

P

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

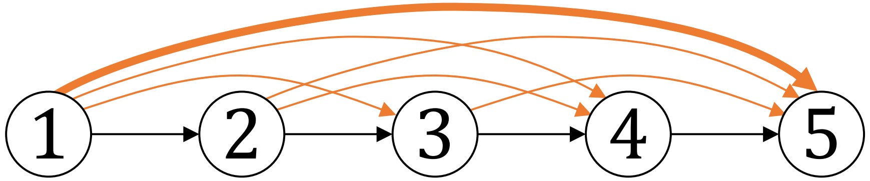
$\Delta P^{(3)}$

paths of LEN ≤ 4

P

?

Example



$P^{(t)}(x,y)$:- $A(x,y)$.
$P^{(t)}(x,y)$:- $A(x,z)$, $P^{(t-1)}(z,y)$.

$\Delta P^{(t)}(x,y)$:- $A(x,z)$, $\Delta P^{(t-1)}(z,y)$, not $P(x,y)$.
$P(x,y)$:- $\Delta P^{(t)}(x,y)$.

A

1	2
2	3
3	4
4	5

paths of LEN ≤ 1

P

$\Delta P^{(1)}$	1	2
	2	3
	3	4
	4	5

paths of LEN ≤ 2

P

	1	2
	2	3
	3	4
	4	5
$\Delta P^{(2)}$	1	3
	2	4
	3	5

paths of LEN ≤ 3

P

	1	2
	2	3
	3	4
	4	5
	1	3
	2	4
	3	5
$\Delta P^{(3)}$	1	4
	2	5

paths of LEN ≤ 4

P

	1	2
	2	3
	3	4
	4	5
	1	3
	2	4
	3	5
	1	4
	2	5
$\Delta P^{(4)}$	1	5

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^{\neg}$: Datalog with stratified negation
  - Datalog$^{\pm}$
- Answer Set Programming (ASP)

The Chase

- A simple fixed-point algorithm to test implication of data dependencies.
- In its simplest incarnation it tests whether the projection of a relation schema constrained by some functional dependencies onto a given decomposition can be recovered by rejoining the projections
  - i.e. whether a particular decomposition is "lossless"
  - Problem is motivated by from schema normalization (decomposition of relations)
- The interesting aspect is that this algorithms is confluent: we can apply rules in any order and will still arrive at a unique fixed-point

The term ‘‘chase’’ was coined in "Maier, Mendelzon, Sagiv: Testing implications of data dependencies, TODS 1979. https://doi.org/10.1145/320107.320115", where it was used to test the logical implication of dependencies. "Aho, Sagiv, Ullman: Equivalences among relational expressions, SICOMP 1979. https://doi.org/10.1137/0208017" introduced tableaux queries with an algorithm that coincides with the chase with functional dependencies. "Aho, Beeri, Ullman: The theory of joins in relational databases, TODS 1979. https://doi.org/10.1145/320083.320091" extends this algorithm to include also multivalued dependencies, for the purpose of checking whether the join of several relations is lossless. See also "Deutsch, Nash: Chase. Encyclopedia of Database Systems. 2009. https://doi.org/10.1007/978-0-387-39940-9_1250" for more details

Notation

A B C

- We usually denote relations by a name and an ordered set of attributes
  - $R_1(A, B, C, D), R_2(D, E, F)$
- We can can also ignore relation names and the order among attributes. A relation is then just a set of attributes (unordered or named perspective)
  - $S_1 = \{A, B, C, D\}, S_2 = \{D, E, F\}$
- We can then view a relational schema R as a pair (S, Σ) where:
  - S is a finite set of attributes
    - $S = \{A, B, C, D, E, F\},$
  - Σ is a set of functional dependencies (FDs) over S
    - $\Sigma = \{D \rightarrow E, D \rightarrow F\}$
- We want to know if we can always decompose S into S_1 and S_2, s.t.:
  - $R_1 = \pi_{S_1}(R), R_2 = \pi_{S_2}(R), R = R_1 \bowtie R_2$

A possibly familiar example

Assume we decompose $R(A, B, C, D, E, F)$ with $\Sigma = \{D \rightarrow E, D \rightarrow F\}$ into $R_1(A, B, C, D)$ and $R_2(D, E, F)$. Is $R = R_1 \bowtie R_2$ for every database over this schema?

R

A	B	C	D	E	F
a	e	i	l	o	s
b	f	i	l	o	s
c	g	j	m	p	t
d	h	k	n	q	t

Σ:

$D \rightarrow E$

$D \rightarrow F$

$\pi_{A,B,C,D}$

$\pi_{D,E,F}$

R_1

A	B	C	D
a	e	i	l
b	f	i	l
c	g	j	m
d	h	k	n

R_2

D	E	F
l	o	s
m	p	t
n	q5	t

A possibly familiar example: now even more familiar ☺

Assume we decompose **Item**(N,P,C,M,S,C) with $\Sigma = \{M \rightarrow S, M \rightarrow C\}$ into **Product**(N,P,C,M) and **Company**(M,S,C). Is $\mathbf{Item} = \mathbf{Product} \bowtie \mathbf{Company}$ for every database?

Item

Name	Price	Category	Manufacturer	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	15	Japan

Σ:

$M \rightarrow S$

$M \rightarrow C$

$\pi_{\text{N,P,C,M}}$

$\pi_{\text{M,S,C}}$

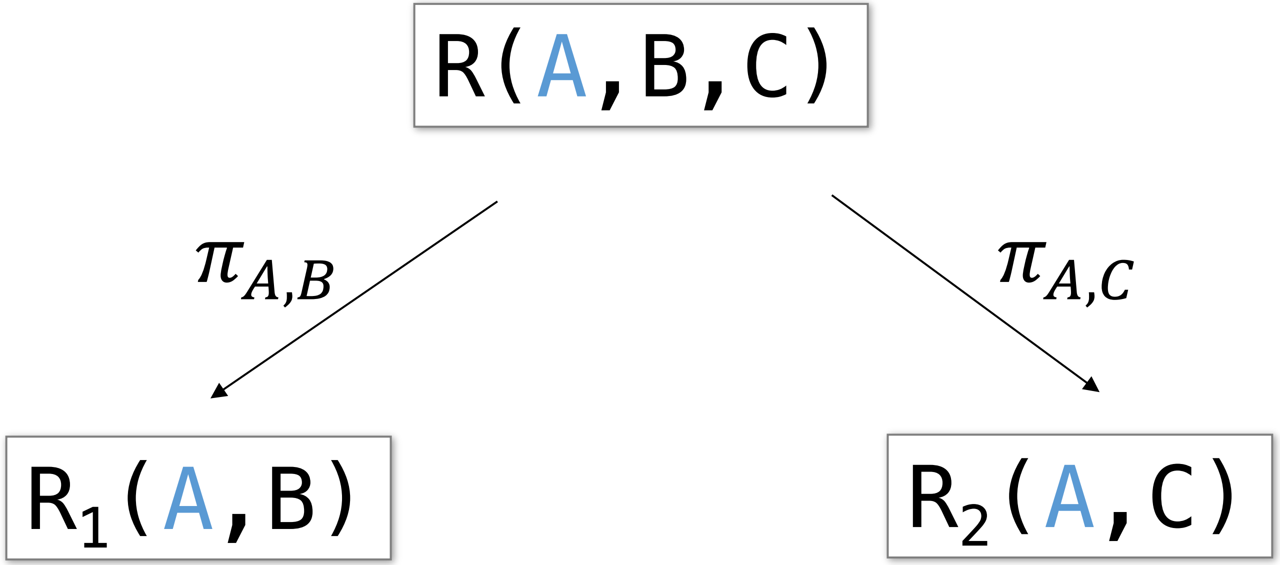
Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Manufacturer	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Decompositions in General

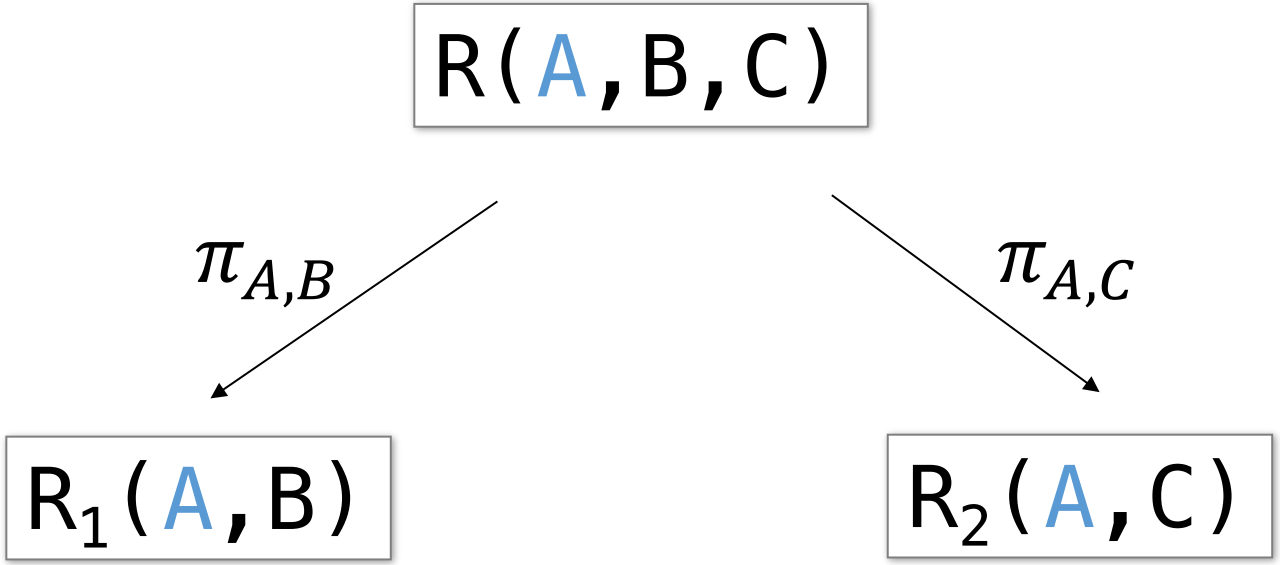


Notice that $R \subseteq R_1 \bowtie R_2$ for every database over any schema (we never loose tuples).

But we want that $R = R_1 \bowtie R_2$ for every database over this schema.
We then say that the decomposition of R into (R_1, R_2) is lossless if $R = R_1 \bowtie R_2$.

When is this the case? ?

Decompositions in General



R(A,B,C)

$\pi_{A,B}$

$\pi_{A,C}$

R$_1$(A,B)

R$_2$(A,C)

The decomposition is <u>lossless</u> iff:

- $A \rightarrow B$, even if we don't have $A \rightarrow C$ at the same time, or
- $A \rightarrow B$, even if we don't have $A \rightarrow C$ at the same time, or

Notice that $R \subseteq R_1 \bowtie R_2$ <u>for every database</u> over any schema (we never loose tuples).

But we want that $R = R_1 \bowtie R_2$ <u>for every database</u> over this schema.
We then say that the decomposition of R into (R_1, R_2) is <u>lossless</u> if $R = R_1 \bowtie R_2$.

When is this the case?

Lossless Decomposition

A	B	C
Name	**Price**	**Category**
Gizmo	19	Gadget
OneClick	24	Camera
Gizmo	19	Camera

Is this decomposition lossless = correct? ?

A	B
Name	**Price**
Gizmo	19
OneClick	24
~~Gizmo~~	~~19~~

A	C
Name	**Category**
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Lossless Decomposition

A → B

A B C

Name	Price	Category
Gizmo	19	Gadget
OneClick	24	Camera
Gizmo	19	Camera

A B

Name	Price
Gizmo	19
OneClick	24
~~Gizmo~~	~~19~~

A C

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Is this decomposition lossless = correct?

Yes, we don't loose information

Lossless Decomposition

C B A

Name	Price	Category
Gizmo	19	Gadget
OneClick	24	Camera
Gizmo	19	Camera

Is this decomposition lossless = correct? ?

C A

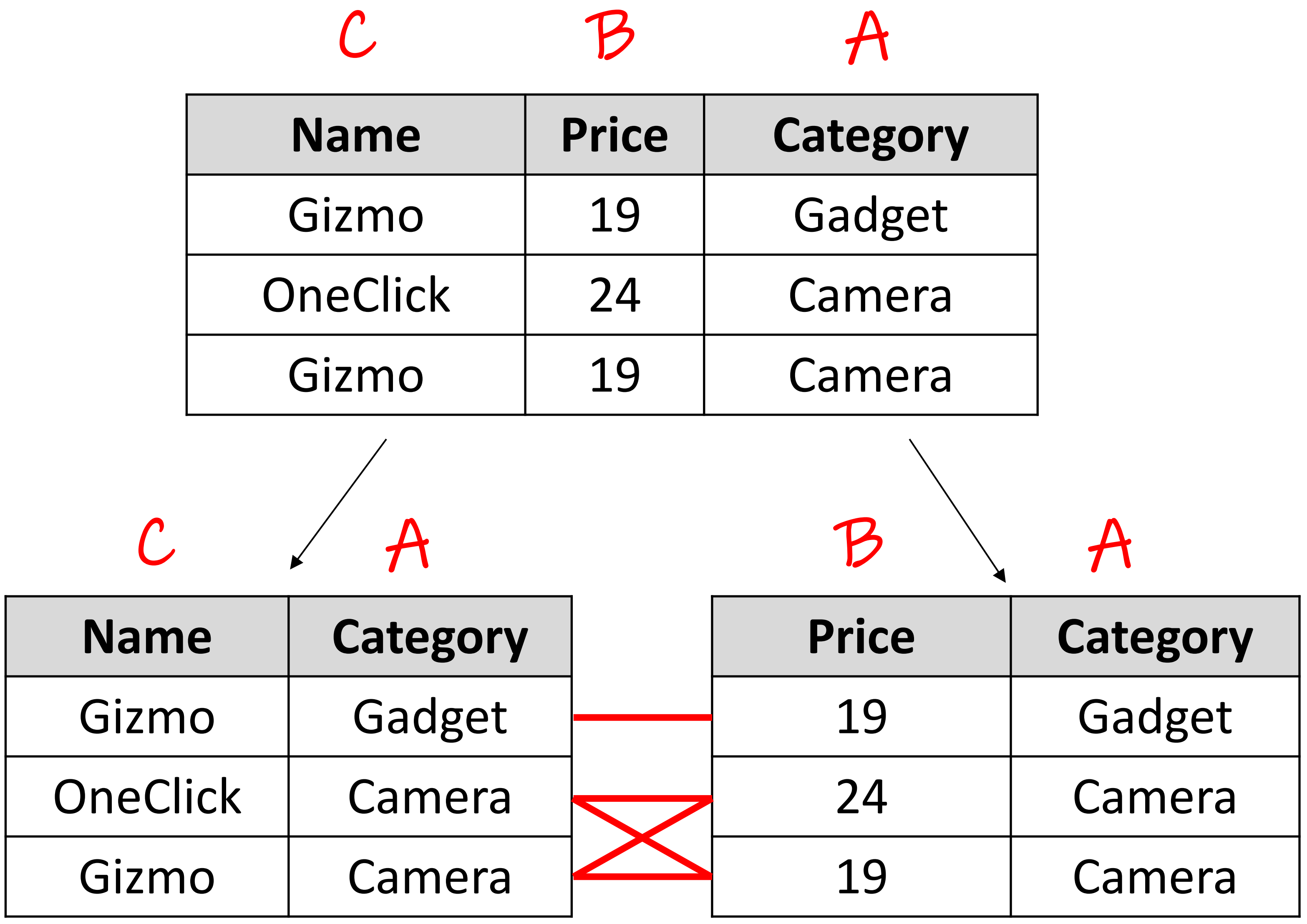
Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

B A

Price	Category
19	Gadget
24	Camera
19	Camera

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Lossless Decomposition



Name	Price	Category
Gizmo	19	Gadget
OneClick	24	Camera
Gizmo	19	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19	Gadget
24	Camera
19	Camera

Is this decomposition lossless = correct?

No, here we lost information (Does Gizmo cost 19 or 24?).

Why does this happen?

(Neither $A \to B$, nor $A \to C$)

More general question: is a given decomposition lossless?

- Given a relation R with attributes S, a set of FDs Σ over S, and a set of subsets of S: $S_1, S_2, \ldots, S_k$.
- Is the decomposition of R into $R_1 = \pi_{S_1}(R), \ldots, R_k = \pi_{S_k}(R)$ lossless? I.e. Is it true that $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k = R$?
- All we need to prove is that
  - …

?

More general question: is a given decomposition lossless?

- Given a relation R with attributes S, a set of FDs Σ over S, and a set of subsets of S: $S_1, S_2, \ldots, S_k$.
- Is the decomposition of R into $R_1 = \pi_{S_1}(R)$, ..., $R_k = \pi_{S_k}(R)$ lossless? I.e. Is it true that $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k = R$?
- All we need to prove is that
  - $R \supseteq R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k$
- because we already know that we never loose tuples:
  - $R \subseteq R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k$

The chase in a page (a test for lossless join decomposition)

- Given $R(A, B, C, D)$, is the decomposition into $R_1 = \pi_{A,D}(R), R_2 = \pi_{A,C}(R)$, $R_3 = \pi_{B,C,D}(R)$ lossless, if R satisifies $\Sigma = \{A \rightarrow B, B \rightarrow C, CD \rightarrow A\}$?
- We need to check that $R \supseteq R_1 \bowtie R_2 \bowtie R_3$:
  - Suppose $(a, b, c, d) \in R_1 \bowtie R_2 \bowtie R_3$. Question: Is it also in R?
  - Since $(a, b, c, d) \in R_1 \bowtie R_2 \bowtie R_3$, therefore also $(a, d) \in R_1, (a, c) \in R_2, (b, c, d) \in R_3$
  - We therefor know that R must contain the following tuples (Irrespective of the FDs Σ):

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

why?

?

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

The chase in a page (a test for lossless join decomposition)

- Given $R(A,B,C,D)$, is the decomposition into $R_1 = \pi_{A,D}(R), R_2 = \pi_{A,C}(R)$, $R_3 = \pi_{B,C,D}(R)$ lossless, if R satisifies $\Sigma = \{A \rightarrow B, B \rightarrow C, CD \rightarrow A\}$?
- We need to check that $R \supseteq R_1 \bowtie R_2 \bowtie R_3$:
  - Suppose $(a,b,c,d) \in R_1 \bowtie R_2 \bowtie R_3$. Question: Is it also in R?
  - Since $(a,b,c,d) \in R_1 \bowtie R_2 \bowtie R_3$, therefore also $(a,d) \in R_1, (a,c) \in R_2, (b,c,d) \in R_3$
  - We therefor know that R must contain the following tuples (Irrespective of the FDs Σ):

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

why?

because $(a,d) \in R_1$ which was derived from R as $\pi_{A,D}(R)$ ← $R_1(A,D)$

because $(a,c) \in R_2$ which was derived from from R as $\pi_{A,C}(R)$

because $(b,c,d) \in R_3$ which was derived from from R as $\pi_{B,C,D}(R)$

- Idea: "Chase" them (apply given FDs Σ by equating constants) until we can either prove that $(a,b,c,d) \in R$ or until we cannot apply any more FDs.

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

The chase in a page (a test for lossless join decomposition)

- Idea: “Chase” them (apply given FDs Σ by equating constants) until we can either prove that $(a, b, c, d) \in R$ or until we cannot apply any more FDs.
- Our FDs Σ:
  - $A \rightarrow B$
  - $B \rightarrow C$
  - $CD \rightarrow A$

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

apply:
$A \rightarrow B$

?

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

The chase in a page (a test for lossless join decomposition)

- Idea: “Chase” them (apply given FDs Σ by equating constants) until we can either prove that $(a, b, c, d) \in R$ or until we cannot apply any more FDs.
- Our FDs Σ:
  - $A \rightarrow B$
  - $B \rightarrow C$
  - $CD \rightarrow A$

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

apply:
$A \rightarrow B$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

apply:
$B \rightarrow C$

?

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

The chase in a page (a test for lossless join decomposition)

- Idea: "Chase" them (apply given FDs Σ by equating constants) until we can either prove that $(a, b, c, d) \in R$ or until we cannot apply any more FDs.
- Our FDs Σ:
  - $A \rightarrow B$
  - $B \rightarrow C$
  - $CD \rightarrow A$

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

apply: $A \rightarrow B$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

apply: $B \rightarrow C$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

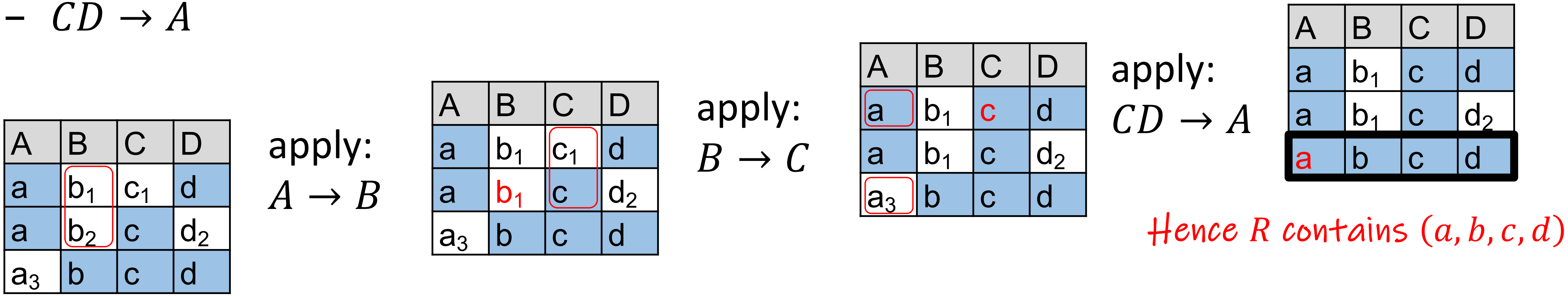
apply: $CD \rightarrow A$

?

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

The chase in a page (a test for lossless join decomposition)

- Idea: “Chase” them (apply given FDs Σ by equating constants) until we can either prove that $(a, b, c, d) \in R$ or until we cannot apply any more FDs.
- Our FDs Σ:
  - $A \rightarrow B$
  - $B \rightarrow C$
  - $CD \rightarrow A$



A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

apply:
$A \rightarrow B$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

apply:
$B \rightarrow C$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

apply:
$CD \rightarrow A$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Hence R contains (a, b, c, d)

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

The chase in a page (a test for lossless join decomposition)

- Idea: "Chase" them (apply given FDs Σ by equating constants) until we can either prove that $(a, b, c, d) \in R$ or until we cannot apply any more FDs.
- Our FDs Σ:
  - $A \rightarrow B$
  - $B \rightarrow C$
  - $CD \rightarrow A$

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

apply: $A \rightarrow B$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

apply: $B \rightarrow C$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

apply: $CD \rightarrow A$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

Hence R contains (a, b, c, d)

(

apply: $A \rightarrow B$

A	B	C	D
~~a~~	~~b~~	~~c~~	~~d~~
a	b	c	d_2
a	b	c	d

unique fix point

)

Example taken from Example 3.22, Section 3.4.2, "Garcia-Molina, Ullman, Widom. Database Systems: The Complete Book. 2nd ed. 2009

Chase example 2

C→E D→E E→B
E→D

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_2
a_3	b_3	c_3	d	e

?

Chase example 2

C→E D→E E→B E→D

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_2
a_3	b_3	c_3	d	e

C→E

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_1
a_3	b_3	c_3	d	e

D→E

$e = e_1$

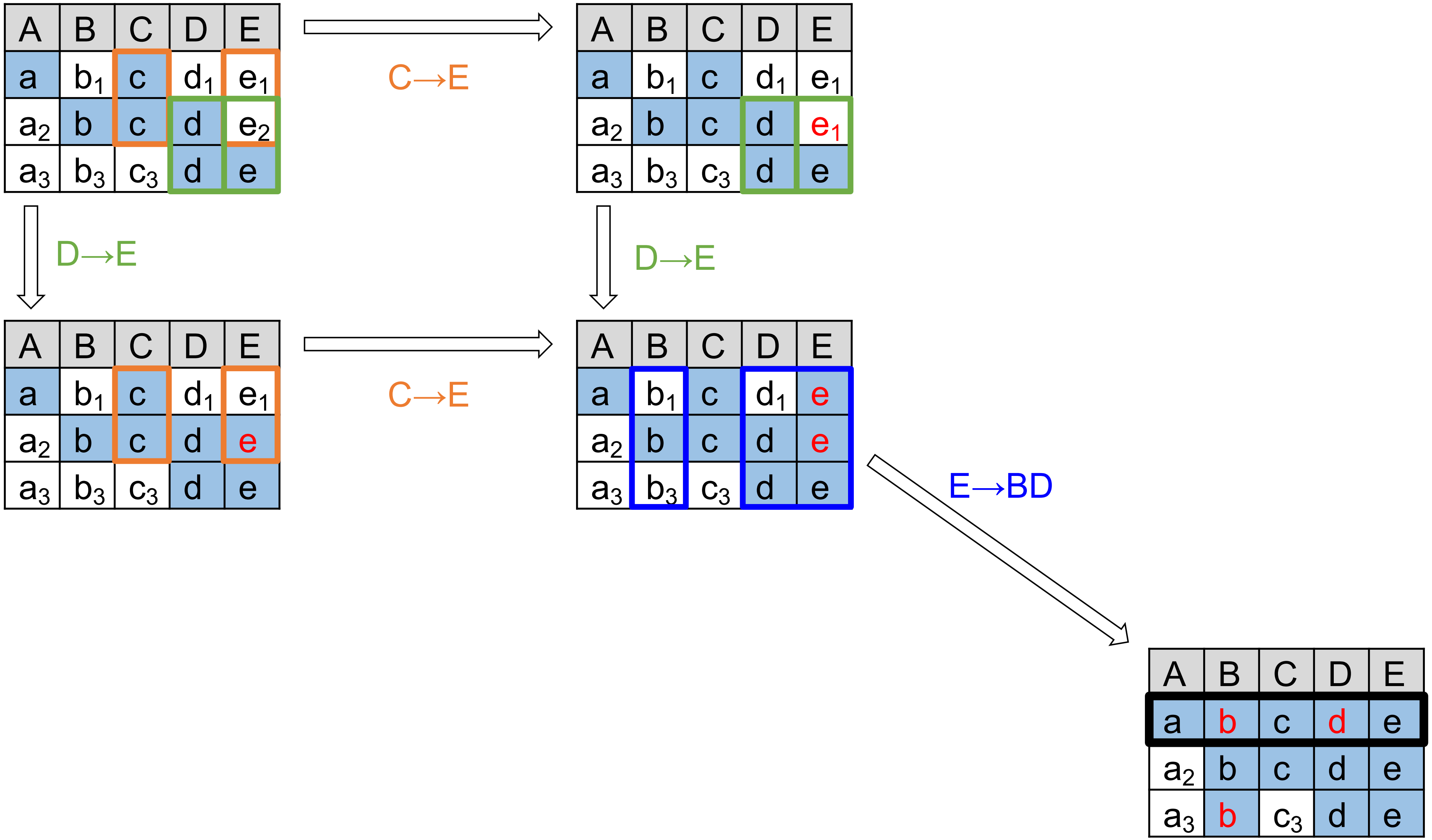
A	B	C	D	E
a	b_1	c	d_1	e
a_2	b	c	d	e
a_3	b_3	c_3	d	e

E→BD

A	B	C	D	E
a	b	c	d	e
a_2	b	c	d	e
a_3	b	c_3	d	e

Chase example 2

C→E D→E E→B
E→D



A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_2
a_3	b_3	c_3	d	e

C→E

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_1
a_3	b_3	c_3	d	e

D→E

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e
a_3	b_3	c_3	d	e

C→E

D→E

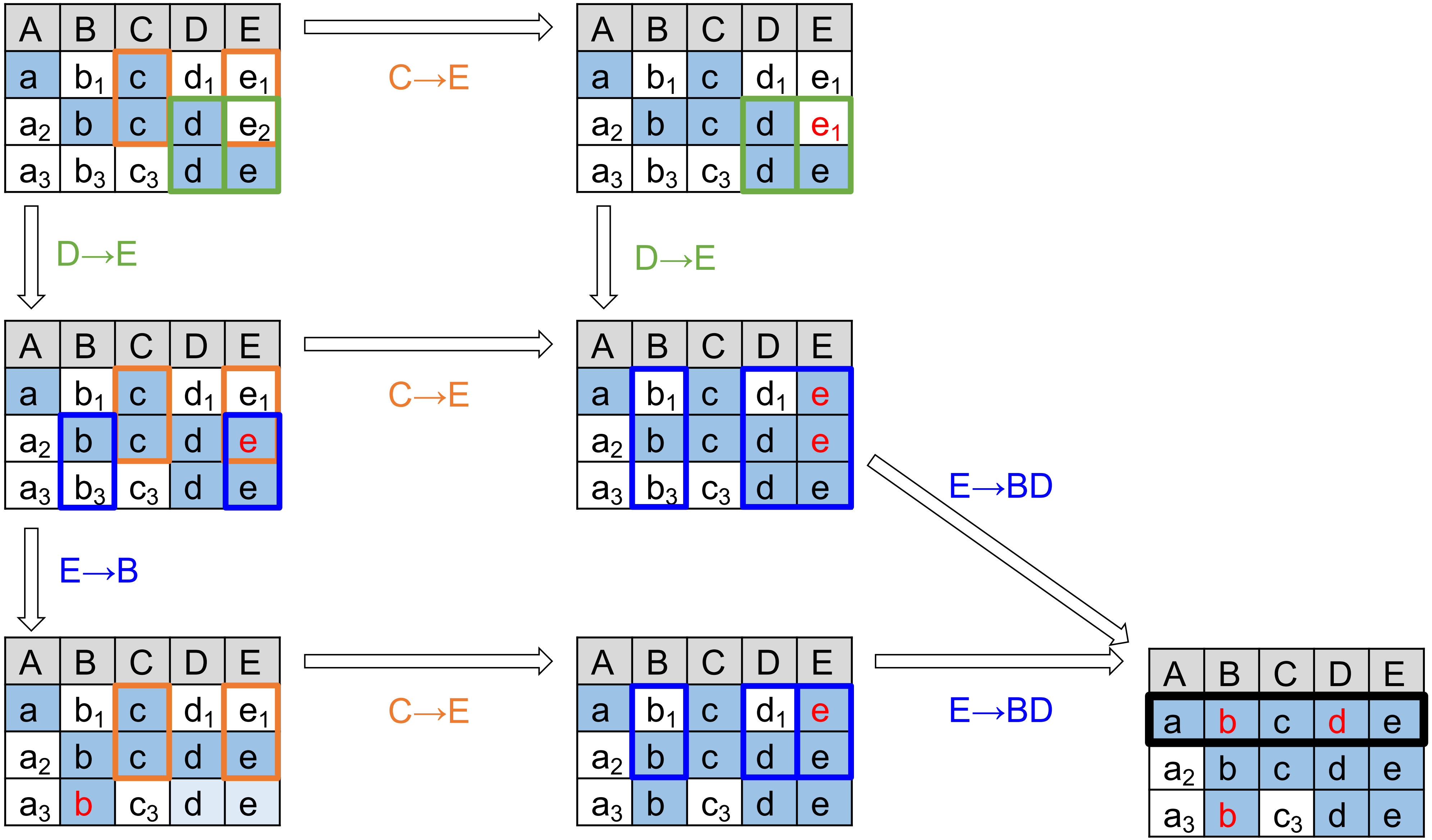
A	B	C	D	E
a	b_1	c	d_1	e
a_2	b	c	d	e
a_3	b_3	c_3	d	e

E→BD

A	B	C	D	E
a	b	c	d	e
a_2	b	c	d	e
a_3	b	c_3	d	e

Chase example 2

C→E D→E E→B
E→D



A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_2
a_3	b_3	c_3	d	e

C→E

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_1
a_3	b_3	c_3	d	e

D→E

D→E

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e
a_3	b_3	c_3	d	e

C→E

A	B	C	D	E
a	b_1	c	d_1	e
a_2	b	c	d	e
a_3	b_3	c_3	d	e

E→BD

E→B

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e
a_3	b	c_3	d	e

C→E

A	B	C	D	E
a	b_1	c	d_1	e
a_2	b	c	d	e
a_3	b	c_3	d	e

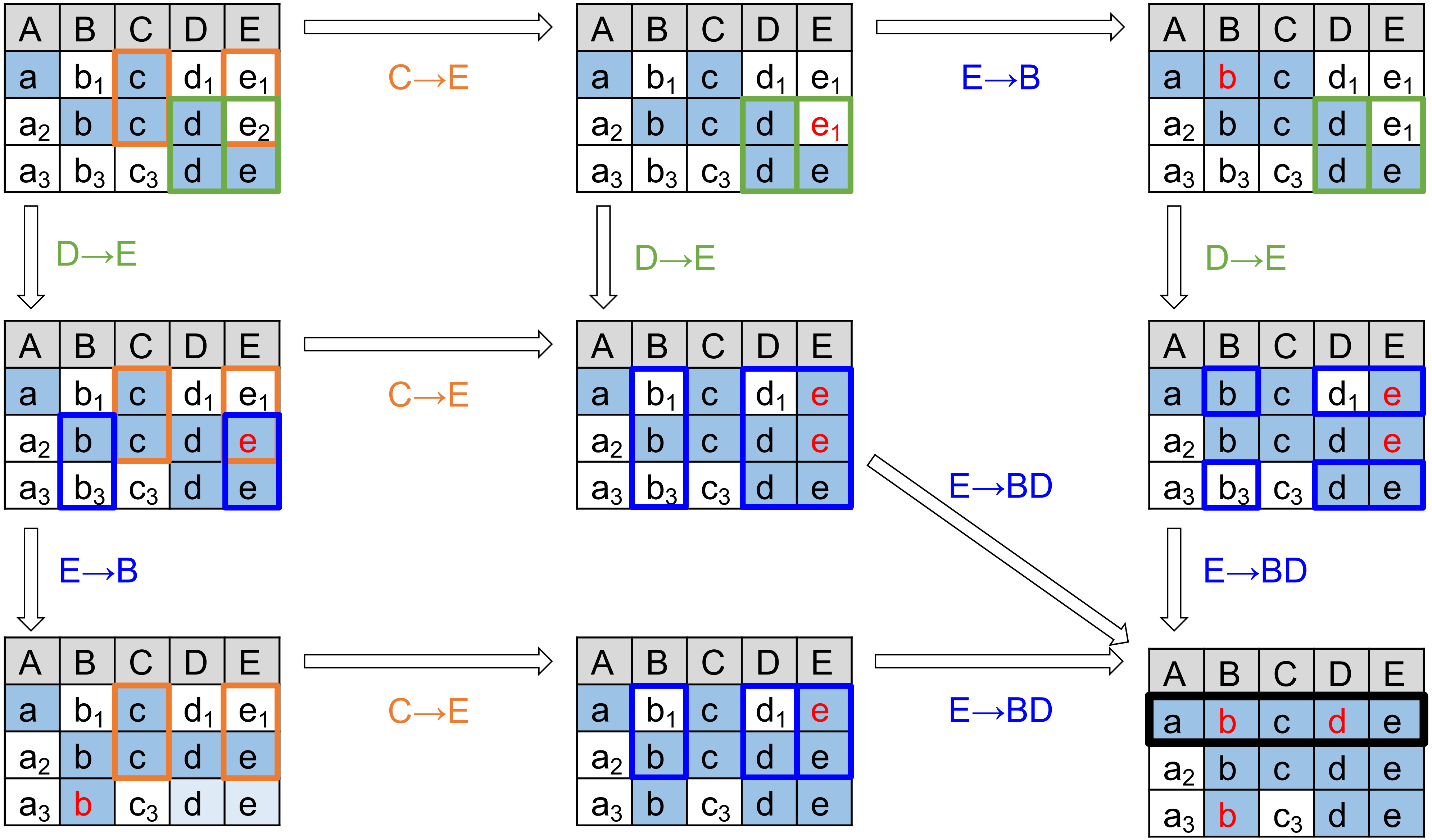
E→BD

A	B	C	D	E
a	b	c	d	e
a_2	b	c	d	e
a_3	b	c_3	d	e

The chase is
confluent
(has a unique
fix point)

Chase example 2

C→E D→E E→B E→D



A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_2
a_3	b_3	c_3	d	e

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e_1
a_3	b_3	c_3	d	e

A	B	C	D	E
a	b	c	d_1	e_1
a_2	b	c	d	e_1
a_3	b_3	c_3	d	e

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e
a_3	b_3	c_3	d	e

A	B	C	D	E
a	b_1	c	d_1	e
a_2	b	c	d	e
a_3	b_3	c_3	d	e

A	B	C	D	E
a	b	c	d_1	e
a_2	b	c	d	e
a_3	b_3	c_3	d	e

A	B	C	D	E
a	b_1	c	d_1	e_1
a_2	b	c	d	e
a_3	b	c_3	d	e

A	B	C	D	E
a	b_1	c	d_1	e
a_2	b	c	d	e
a_3	b	c_3	d	e

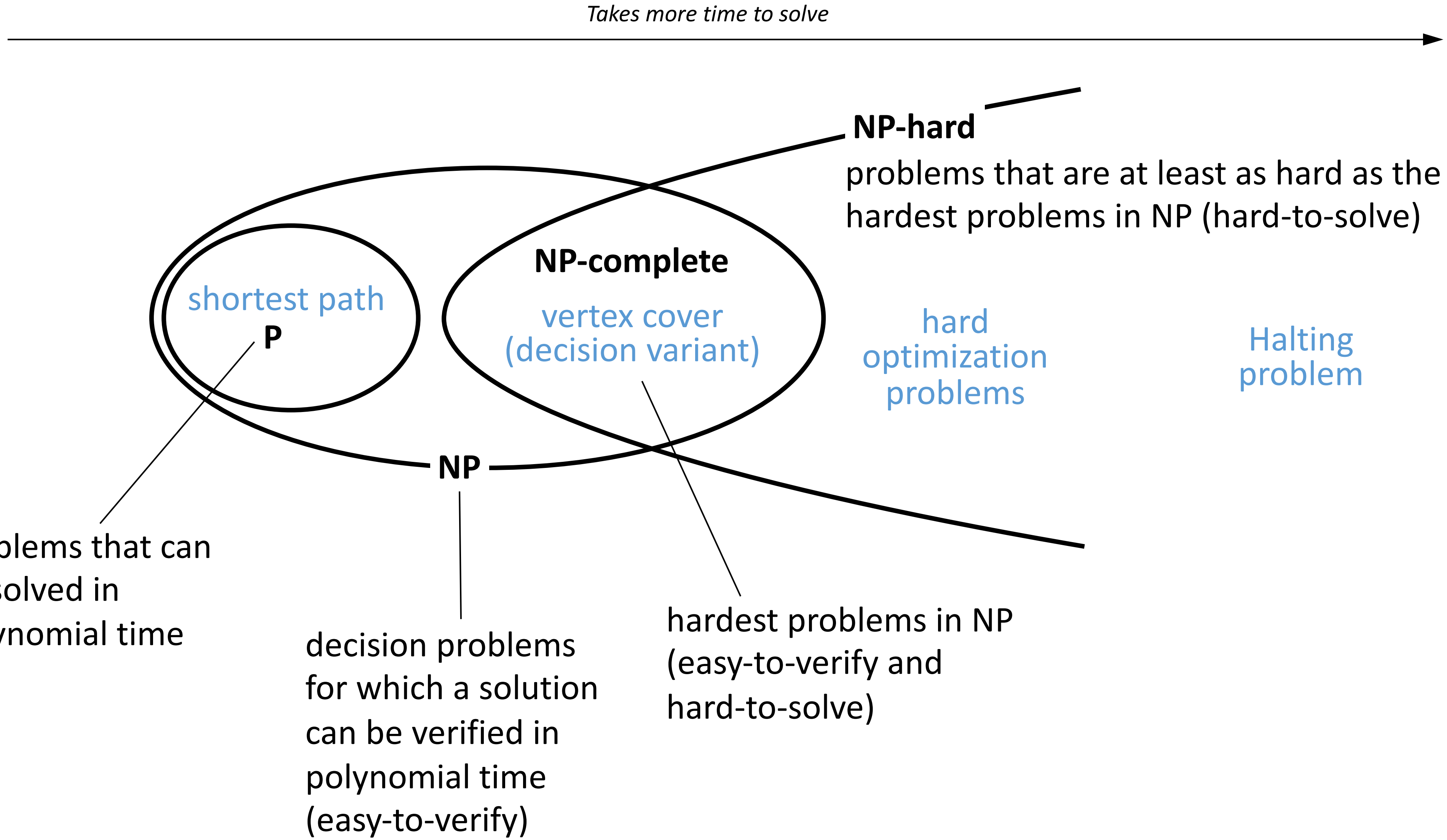
A	B	C	D	E
a	b	c	d	e
a_2	b	c	d	e
a_3	b	c_3	d	e

The chase is confluent (has a unique fix point)

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^{\neg}$: Datalog with stratified negation
  - Datalog$^{\pm}$
- Answer Set Programming (ASP)

NP-hardness (assuming P≠NP)



Takes more time to solve
NP-hard
problems that are at least as hard as the hardest problems in NP (hard-to-solve)
NP-complete
shortest path
P
vertex cover (decision variant)
hard optimization problems
Halting problem
NP
problems that can be solved in polynomial time
decision problems for which a solution can be verified in polynomial time (easy-to-verify)
hardest problems in NP (easy-to-verify and hard-to-solve)

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3).

?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

Box(x) :- Item(x), ¬Box(x).

Item('ball') → ?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

Box(x) :- Item(x), ¬Box(x).

Item('ball') → Box('ball') ???

LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).

Item('ball') → ?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

Box(x) :- Item(x), ¬Box(x).

Item('ball') → Box('ball') ???

LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).

Item('ball') → LeftBox('ball') ???
unsafe!

LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- Item(x), ¬LeftBox(x).

Item('ball') ?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

What should be the Semantics?

```
Friend(x,y) :- Likes(x,y),¬Parent(y,x).
```

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

```
Box(x) :- Item(x), ¬Box(x).
```

Item('ball') → Box('ball') ???

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).
```

Item('ball') → LeftBox('ball') ???

unsafe!

```
LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- Item(x), ¬LeftBox(x).
```

Item('ball') → LeftBox('ball')
Item('ball') → RightBox('ball')

⇒ *Adding negation to Datalog is not straightforward!*

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

What should be the Semantics?

Friend(x,y) :- Likes(x,y),¬Parent(y,x).

Likes (1,2). Parent (2,1).
Likes (1,3). → Friend(1,3)

Box(x) :- Item(x), ¬Box(x).

Item('ball') → ~~Box('ball')~~

no "stable" model

LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- ¬LeftBox(x).

Item('ball') → ~~LeftBox('ball')~~

unsafe!

LeftBox(x) :- Item(x), ¬RightBox(x).
RightBox(x) :- Item(x), ¬LeftBox(x).

Item('ball') → LeftBox('ball')
Item('ball') → RightBox('ball')

Later discussed "stable model" semantics
(intended models = answer sets)

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

Negation in Datalog

- Various semantics have been proposed for supporting negation in Datalog that still allow tractability

- We will first look at two:
  - 1. Semipositive Datalog$^{\neg}$ (restricted): PTIME
  - 2. Stratified Datalog$^{\neg}$ (standard): PTIME

- We will later look at a more powerful (but intractable) semantics
  - Stable Models semantics (or answer set programming ASP): NP-complete and beyond!

1. Semipositive Programs and Safety

```
Friend(x,y) :- Likes(x,y), ¬Parent(y,x).
```

$\text{Likes} - \pi_{y,x}\text{Parent}$

A semipositive program is a program where only EDBs may be negated

- Semantics: same as ordinary Datalog programs
- Safety: rule is safe if <u>every variable occurs in a positive</u> (= unnegated) relational atom (ensures domain independence: the results of programs are finite and depend only on the actual contents of the database)

Exercise: Are following rules safe?

```
S(x) :- T(y), Arc(z,y), ¬Arc(x,y).
```

?

```
S(x) :- T(y), ¬T(x).
```

?

Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

1. Semipositive Programs and Safety

Friend(x,y) :- Likes(x,y), ¬Parent(y,x).

$\text{Likes} - \pi_{y,x}\text{Parent}$

A semipositive program is a program where only EDBs may be negated

- Semantics: same as ordinary Datalog programs
- Safety: rule is safe if every variable occurs in a positive (= unnegated) relational atom (ensures domain independence: the results of programs are finite and depend only on the actual contents of the database)

Exercise: Are following rules safe?

S(x) :- T(y), Arc(z,y), ¬Arc(x,y).

unsafe (what is the domain for "x"?)

S(x) :- T(y), ¬T(x).

unsafe

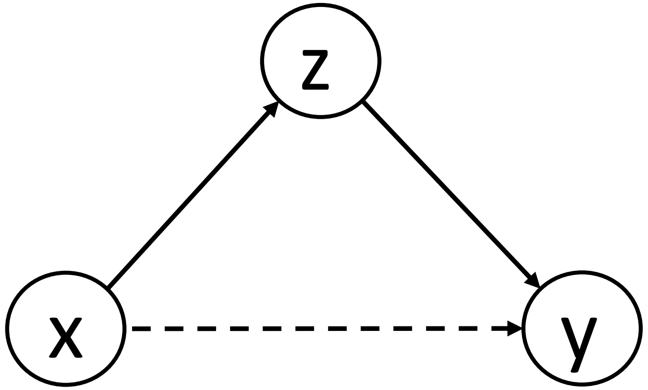
Alternative notations to "¬ Parent(y,x)" are "! Parent(y,x)" or "~Parent(y,x)" or "NOT Parent(y,x)"

1. Semipositive: Negated Atoms

Arc(Source,Target)

- We may put ¬, !, ~, or not in front of an EDB atom to negate its meaning.
- EXAMPLE: Return all pairs of nodes (x,y) where y is two hops away from x, but not an immediate neighbor of x.

?

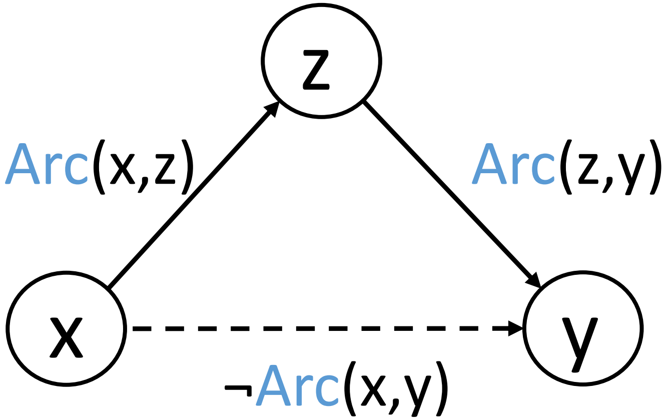


1. Semipositive: Negated Atoms

Arc(Source,Target)

- We may put ¬, !, ~, or not in front of an EDB atom to negate its meaning.
- EXAMPLE: Return all pairs of nodes (x,y) where y is two hops away from x, but not an immediate neighbor of x.

TwoHopsAway(x,y) :- Arc(x,z), Arc(z,y), ¬Arc(x,y).



A(S,T)

501

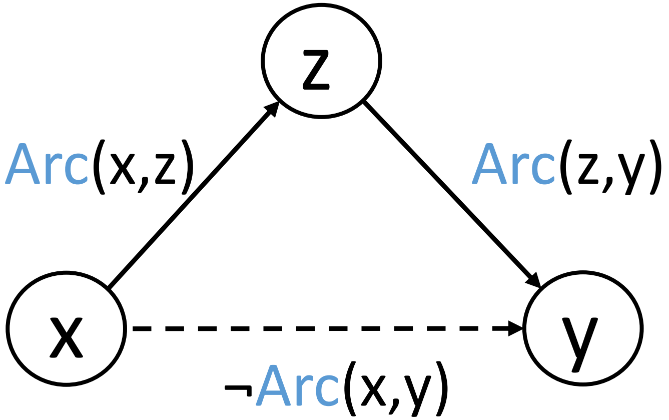
SQL ?

1. Semipositive: Negated Atoms

Arc(Source,Target)

- We may put ¬, !, ~, or not in front of an EDB atom to negate its meaning.
- EXAMPLE: Return all pairs of nodes (x,y) where y is two hops away from x, but not an immediate neighbor of x.

TwoHopsAway(x,y) :- Arc(x,z), Arc(z,y), ¬Arc(x,y).



A(S,T)

501

```
SELECT A1.S, A2.T
FROM A A1, A A2
WHERE A1.T = A2.S
AND NOT EXISTS
    (SELECT *
    FROM A A3
    WHERE A3.S = A1.S
    AND A3.T = A2.T)
```

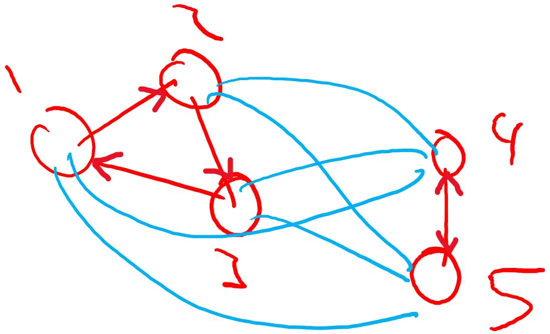
Exampe: beyond Semipositive

Node is basically ADom:
Node(x) :- Arc(x,y)
Node(y) :- Arc(x,y)

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

?



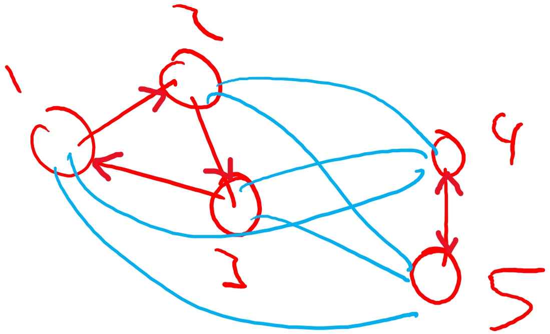
Exampe: beyond Semipositive

Node is basically ADom:
Node(x) :- Arc(x,y)
Node(y) :- Arc(x,y)

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Arc(x,y).
Reachable(x,y) :- Arc(x,z), Reachable(z,y).
```



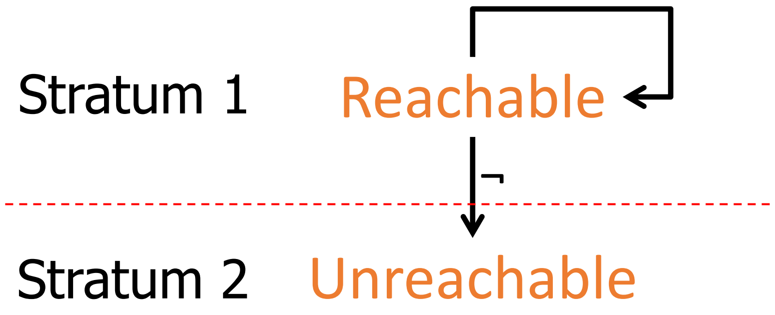
Exampe: beyond Semipositive

Node is basically ADom:
Node(x) :- Arc(x,y)
Node(y) :- Arc(x,y)

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Arc(x,y).
Reachable(x,y) :- Arc(x,z), Reachable(z,y).
Unreachable(x,y) :- Node(x), Node(y), ¬Reachable(x,y).
```



- Straightforward syntactic restriction.
- When the Datalog program is stratified, we can evaluate IDB predicates stratum-by-stratum
- Once evaluated, treat it as EDB for higher strata.

Precedence graph

- Nodes = IDB predicates
- Arc p→q if predicate q depends on p
- Label this arc "¬" if predicate p is negated

think: "topological sort"

Non-stratified example: LeftBox(x) :- ¬LeftBox(x), Item(x).

?

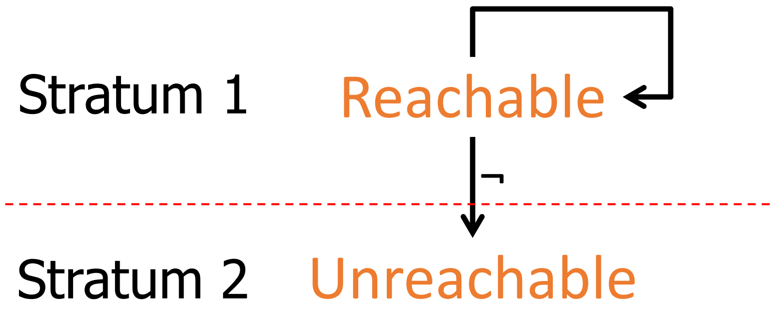
Exampe: beyond Semipositive

Node is basically ADom:
Node(x) :- Arc(x,y)
Node(y) :- Arc(x,y)

Arc(Source,Target)
Node(id)

Compute all pairs of disconnected nodes in a graph.

```
Reachable(x,y) :- Arc(x,y).
Reachable(x,y) :- Arc(x,z), Reachable(z,y).
Unreachable(x,y) :- Node(x), Node(y), ¬Reachable(x,y).
```



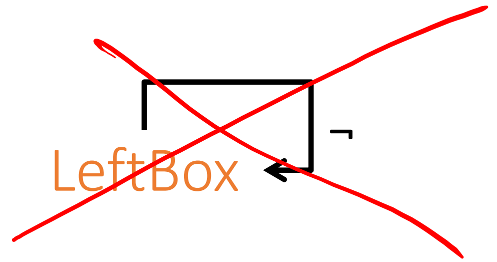
- Straightforward syntactic restriction.
- When the Datalog program is stratified, we can evaluate IDB predicates stratum-by-stratum
- Once evaluated, treat it as EDB for higher strata.

Precedence graph

- Nodes = IDB predicates
- Arc p→q if predicate q depends on p
- Label this arc "¬" if predicate p is negated

think: "topological sort"

Non-stratified example: LeftBox(x) :- ¬LeftBox(x), Item(x).



2. Stratified Programs: Definition and Semantics

- DEFINITION: Let **P** be a Datalog program, **E** be the set of EDB predicates, and **I** be the set of IDB predicates. A stratification of **P** is a partitioning of the IDB predicates into disjoint sets $\mathbf{I}_1,\ldots,\mathbf{I}_k$ such that:
  - For i=1,...,k, every rule with head in I_i has possible body predicates only from **E**, $\mathbf{I}_1,\ldots,\mathbf{I}_i$
  - For i=1,...,k, every rule with head in I_i has negated body predicates only from **E**, $\mathbf{I}_1,\ldots,\mathbf{I}_{i-1}$

- SEMANTICS:
  - For i=1,...,k:
    - Compute the IDBs of the stratum $\mathbf{I}_i$, possibly via recursion
    - Add computed IDBs to the EDBs
  - Due to the definition of stratification, each E_i can be viewed as semipositive

Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.

2. Theorems on Stratification

Contrast with our earlier definition of recursive programs!

- THEOREM 1: A program has a stratification if and only if its dependency graph does not contain a cycle with a "negated edge"
  - Dependency graph is defined as previously, except that edges can be labeled with negation
  - Hence, we can test for stratifiability efficiently, via graph reachability

```
A(x) :- B(x).
B(x) :- C(x).
C(x) :- ¬A(x).
```

Can it be stratified ?

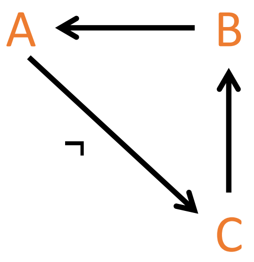
Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.

2. Theorems on Stratification

Contrast with our earlier
definition of recursive programs!

- THEOREM 1: A program has a stratification if and only if its dependency graph does not contain a cycle with a "negated edge"
  - Dependency graph is defined as previously, except that edges can be labeled with negation
  - Hence, we can test for stratifiability efficiently, via graph reachability

```
A(x) :- B(x).
B(x) :- C(x).
C(x) :- ¬A(x).
```

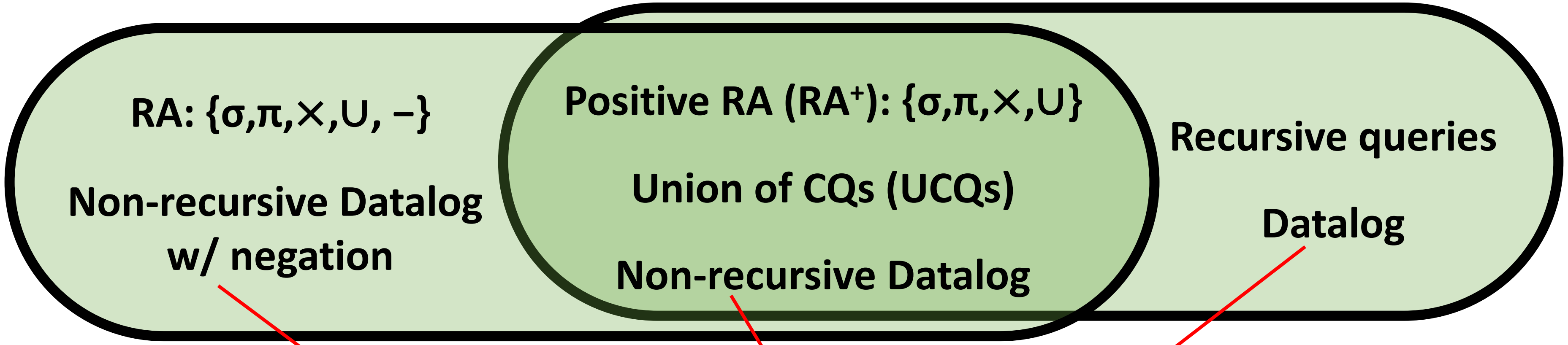


- THEOREM 2: Non-recursive Datalog with negation can always be stratified via the topological order

- THEOREM 3: Non-recursive Datalog with negation has the same expressive power as the algebra $\{\sigma_=, \pi, \times, \cup, -\}$
  - Extendable to RA if we add the comparison predicates <, >, !=, <=, >=

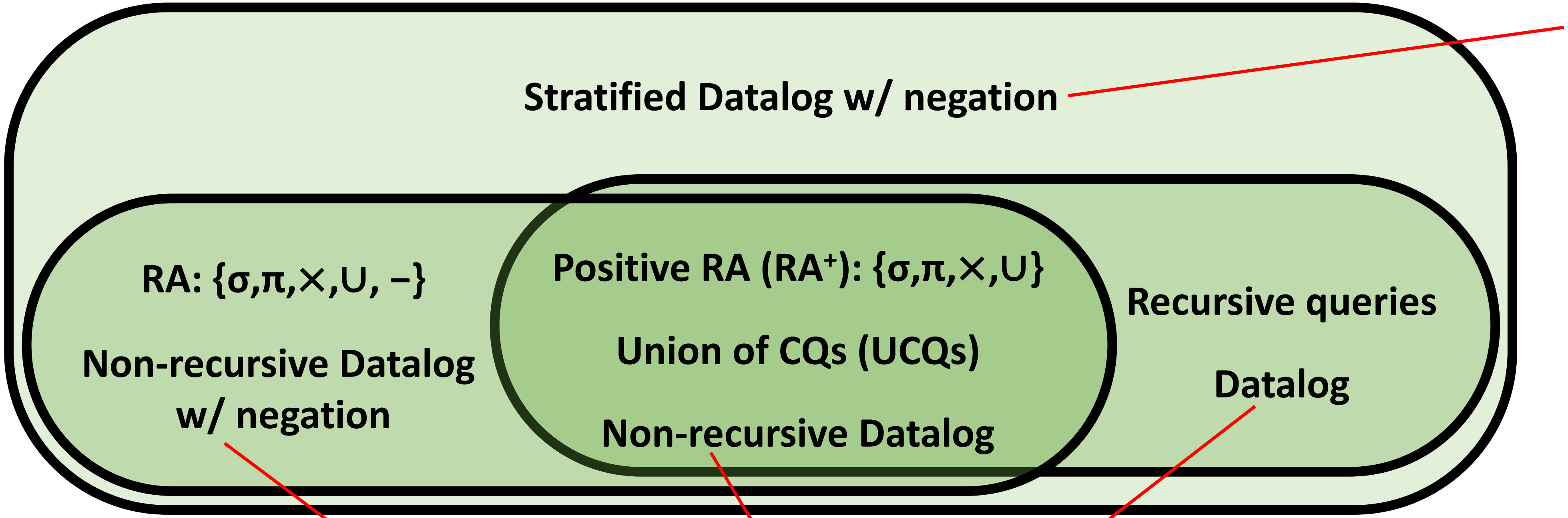
Based on material by Benny Kimelfeld and Oded Shmueli for 236363 Database Management Systems, Technion, 2018.
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Hierarchy of expressiveness



Notice that Datalog and UCQs often assume an unordered domain and no built-in predicates. For equality, we assume here an ordered domain and allow built-in predicates (>,<,≤,≥,!=).

Hierarchy of expressiveness

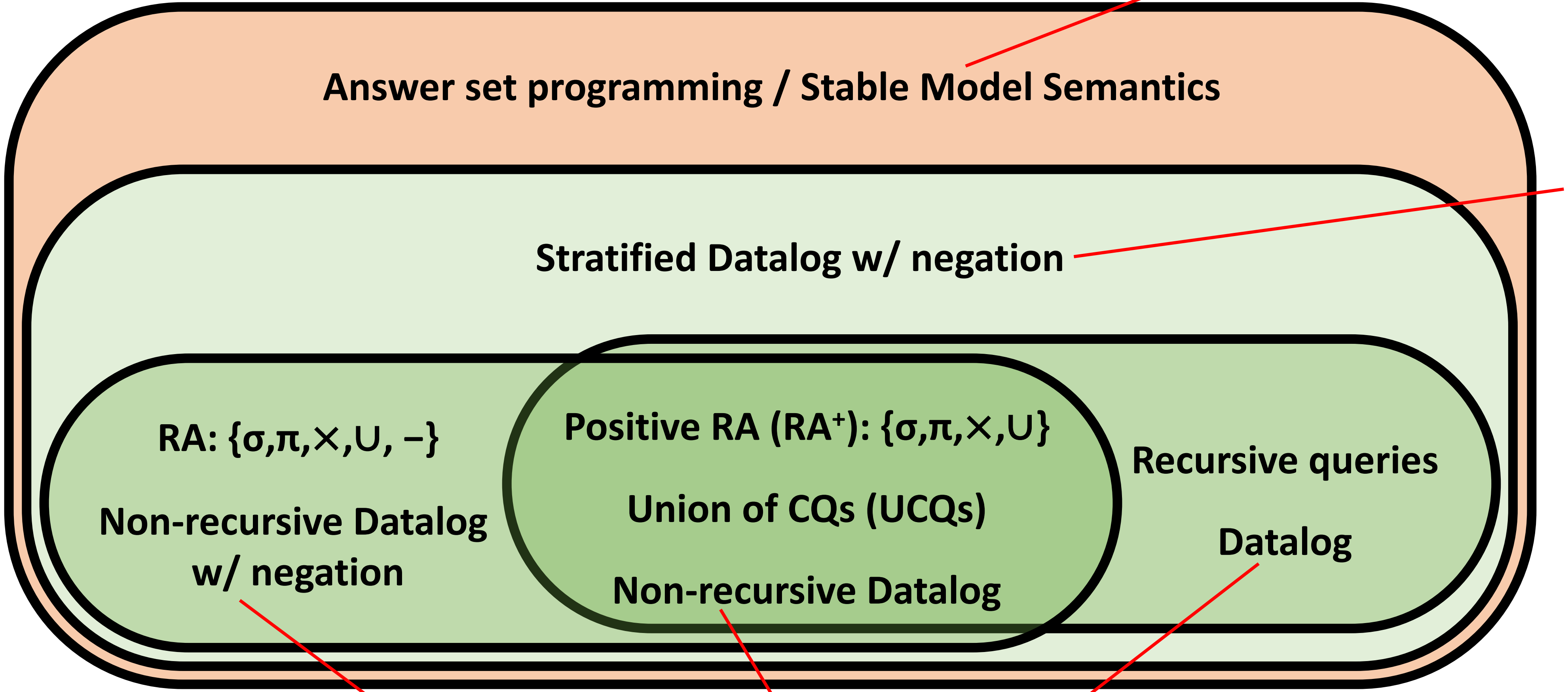

Stratified Datalog w/ negation
RA: {σ,π,×,∪, −}
Non-recursive Datalog w/ negation
Positive RA (RA⁺): {σ,π,×,∪}
Union of CQs (UCQs)
Non-recursive Datalog
Recursive queries
Datalog

can express all polynomial time queries on ordered databases relying on only information encoded in tables (e.g. excludes arithmetical functions)

Notice that Datalog and UCQs often assume an unordered domain and no built-in predicates.
For equality, we assume here an ordered domain and allow built-in predicates (>,<,≤,≥,!=).

Hierarchy of expressiveness

ASP can express NP-complete problems (and even problems higher in the Polynomial hierarchy)
(For Turing-completeness, we would only have to add functions, i.e. the ability to create new values not previously found in the EDB)


Answer set programming / Stable Model Semantics
Stratified Datalog w/ negation
RA: {σ,π,×,∪, −}
Non-recursive Datalog w/ negation
Positive RA (RA⁺): {σ,π,×,∪}
Union of CQs (UCQs)
Non-recursive Datalog
Recursive queries
Datalog

can express all polynomial time queries on ordered databases relying on only information encoded in tables (e.g. excludes arithmetical functions)

Notice that Datalog and UCQs often assume an unordered domain and no built-in predicates.
For equality, we assume here an ordered domain and allow built-in predicates (>,<,≤,≥,!=).

2. Stratification practice

Parent(P,C)

Q: Find all descendants of Alice, who are not descendants of Bob

?

2. Stratification practice

Parent(P,C)

Q: Find all descendants of Alice,
who are not descendants of Bob

first compute for each
person their descendants

then use negation

?

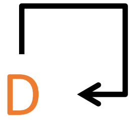
2. Stratification practice

Parent(P,C)

Q: Find all descendants of Alice,
who are not descendants of Bob

```
D(x,y) :- Parent(x,y).
D(x,z) :- Parent(y,z), D(x,y).
```

first compute for each person their descendants



then use negation

?

2. Stratification practice

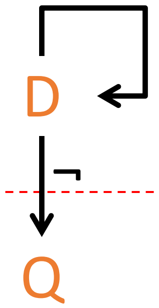
Parent(P,C)

Q: Find all descendants of Alice,
who are not descendants of Bob

```
D(x,y) :- Parent(x,y).
D(x,z) :- Parent(y,z), D(x,y).
Q(x) :- D('Alice',x), ¬D('Bob',x).
```

first compute for each
person their descendants

then use negation



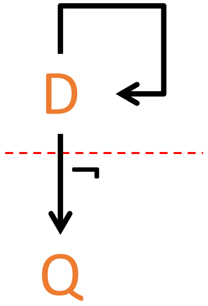
2. Stratification practice

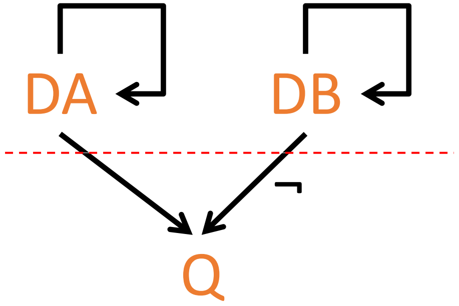
Parent(P,C)

Q: Find all descendants of Alice, who are not descendants of Bob

```
D(x,y) :- Parent(x,y).
D(x,z) :- Parent(y,z), D(x,y).
Q(x) :- D('Alice',x), ¬D('Bob',x).
```

```
DA(y) :- Parent('Alice',y).
DA(y) :- Parent(x,y), DA(x).
DB(y) :- Parent('Bob',y).
DB(y) :- Parent(x,y), DB(x).
Q(x) :- DA(x), ¬DB(x).
```





Datalog Summary

- EDB (extensional/base relations), IDB (intentional/derived relations)
- Datalog program = set of rules; base relations are also rules
- Datalog can be recursive
  - Stratified Datalog with negation still PTIME
  - Non-stratified Datalog: stable model semantics, ASP, can model NPC problems
- SQL has also been extended to express limited form of recursion
  - Using a recursive "with" clause, also called CTE (Common Table Expression)
  - Can only have a single IDB

For more details on recursion in SQL see https://www.postgresql.org/docs/14/queries-with.html#QUERIES-WITH-RECURSIVE

Updated 2/20/2024

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 11

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

2/20/2024

Pre-class conversations

- Last class summary
- Project discussions (in class and after)
- Faculty candidates (today, Feb 29, March 20)

- today:
  - More on Datalog
  - What happens if we add negation? Answer: it depends on how we do it.
    - Datalog with stratified negation
    - Datalog with more genal negation (stable models), leads to ASP
  - Later: Beyond NP with ASP (including 3-colorability in 2 lines)

Outline: T1-4: Datalog & ASP

- Datalog
  - Datalog rules
  - Datalog vs. RA
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Naive and Semi-naive evaluation (Incremental View Maintenance)
  - Chase Procedure (and Decompositions=Factorizations)
  - Datalog$^{\neg}$: Datalog with stratified negation
  - Datalog$^{\pm}$
- Answer Set Programming (ASP)

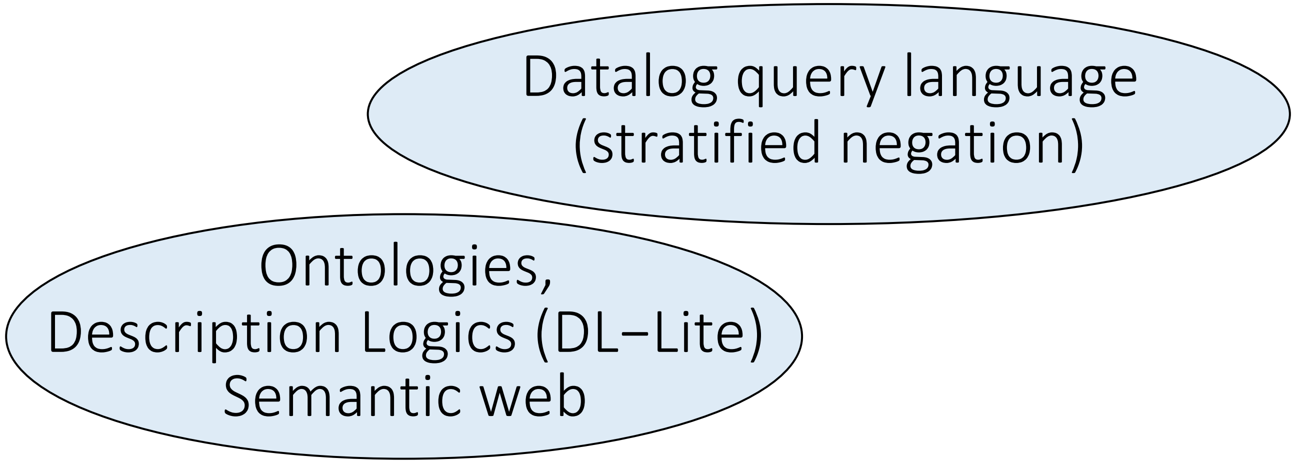
Datalog$^{\pm}$: background

Datalog query language
(stratified negation)

- Much is possible with Datalog

Based on a presentation by Andrea Cali

Datalog$^{\pm}$: background



Datalog query language
(stratified negation)

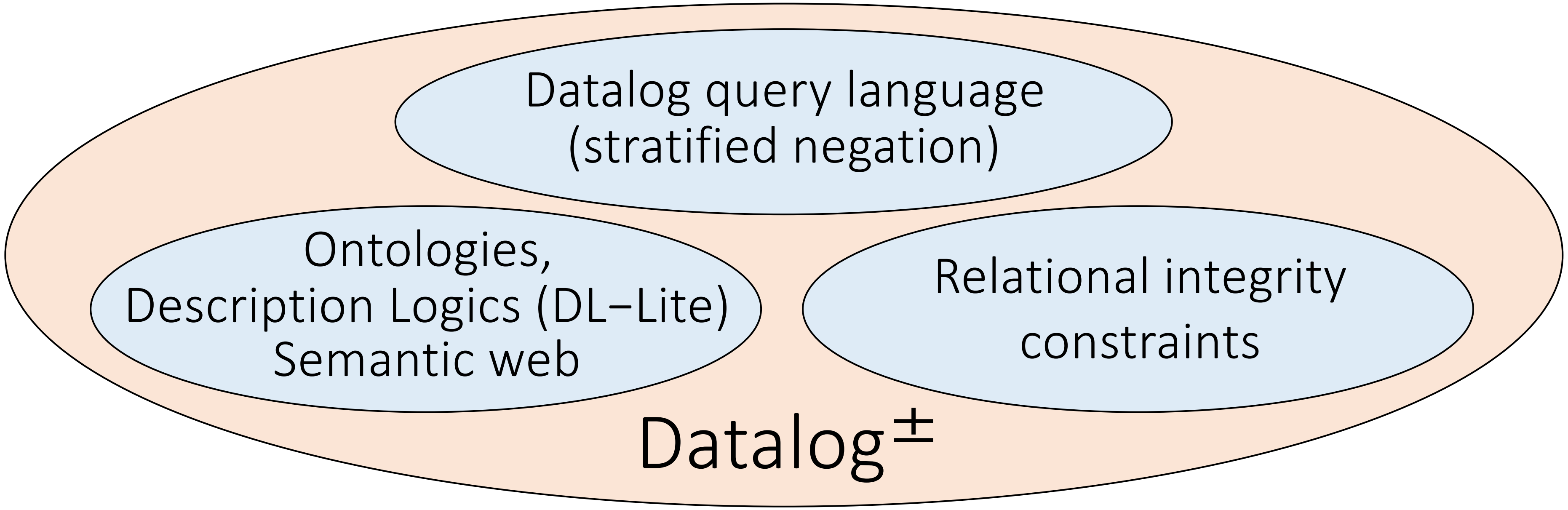
Ontologies,
Description Logics (DL−Lite)
Semantic web

- Much is possible with Datalog
- Much is not (observed e.g. by [Patel-Schneider, Horrocks 2006])

Patel-Schneider, Horrocks. Position paper: A comparison of two modelling paradigms in the Semantic Web. WWW (Semantic Web track). 2006. https://dl.acm.org/doi/10.1145/1135777.1135784

Based on a presentation by Andrea Cali

Datalog$^{\pm}$: goal



- Much is possible with Datalog
- Much is not (observed e.g. by [Patel-Schneider, Horrocks 2006])
- Datalog$^{\pm}$ is a framework that extends Datalog with:
  - value invention (∃-variables in the head): TGDs (Tuple-Generating Dependencies)
  - equality predicate in the head: EGDs (Equality Generating Dependencies)
  - constant ⊥ in the head: negative constraints (disjointness)

Patel-Schneider, Horrocks. Position paper: A comparison of two modelling paradigms in the Semantic Web. WWW (Semantic Web track). 2006. https://dl.acm.org/doi/10.1145/1135777.1135784
Cali, Gottlob, Lukasiewicz, Marnette, Pieris. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. LICS 2010. https://doi.org/10.1109/LICS.2010.27
Based on a presentation by Andrea Cali

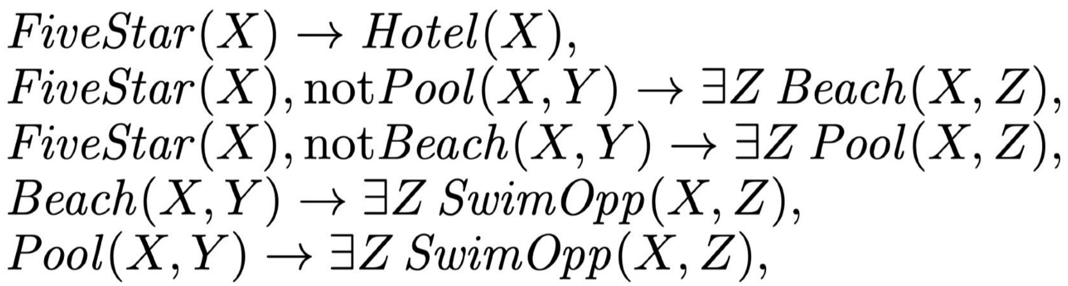
Datalog and expressiveness for ontological reasoning

Assertion type	Datalog rule
Inclusion	emp(X) → person(X)
(Inverse) role inclusion	reportsTo(X , Y) → manages(Y , X)
Reflexive expansion	boss(X) → manages(X , X)
Transitivity	manages(X , Y), manages(Y , Z) → manages(X, Z)
Concept product	seniorEmp(X), emp(Y) → higher(X, Y)
Participation	?
Disjointness	?
Functionality	?

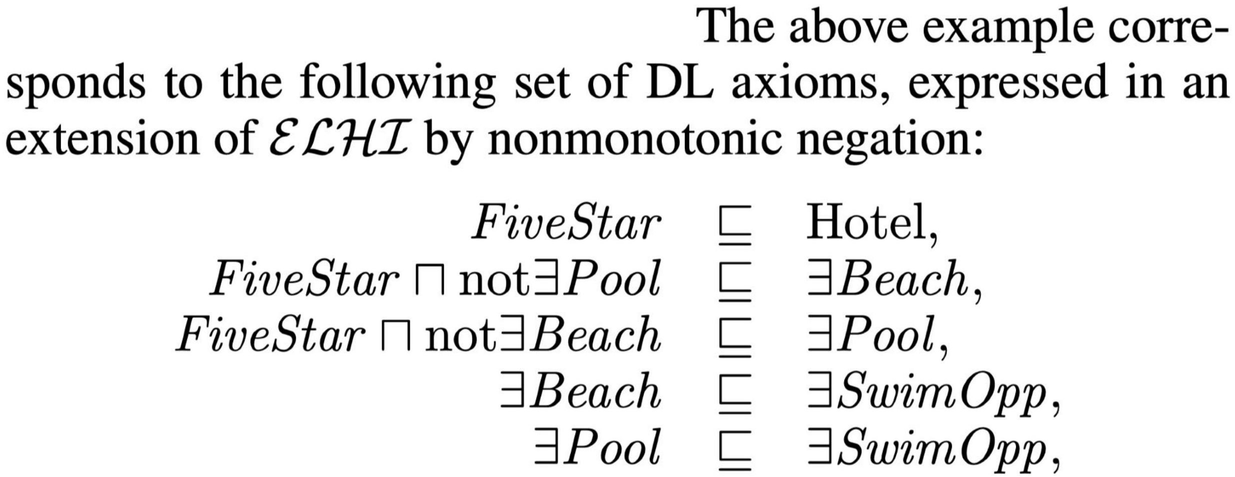
Ontology assertion	Datalog$^{\pm}$ rule
Participation	boss(X) → ∃Y reports(Y , X)
Disjointness	customer(X), boss(X) → ⊥
Functionality	reports(X , Y1), reports (X , Y2) → Y1 = Y2

Based on a presentation by Andrea Cali

Datalog$^{\pm}$ vs. DL



$$\begin{array}{l}
\mathit{FiveStar}(X) \rightarrow \mathit{Hotel}(X),\\
\mathit{FiveStar}(X), \mathrm{not}\,\mathit{Pool}(X,Y) \rightarrow \exists Z\; \mathit{Beach}(X,Z),\\
\mathit{FiveStar}(X), \mathrm{not}\,\mathit{Beach}(X,Y) \rightarrow \exists Z\; \mathit{Pool}(X,Z),\\
\mathit{Beach}(X,Y) \rightarrow \exists Z\; \mathit{SwimOpp}(X,Z),\\
\mathit{Pool}(X,Y) \rightarrow \exists Z\; \mathit{SwimOpp}(X,Z),
\end{array}$$



The above example corresponds to the following set of DL axioms, expressed in an extension of $\mathcal{ELHI}$ by nonmonotonic negation:

$$\begin{array}{rcl}
\mathit{FiveStar} & \sqsubseteq & \mathrm{Hotel},\\
\mathit{FiveStar} \sqcap \mathrm{not}\exists \mathit{Pool} & \sqsubseteq & \exists \mathit{Beach},\\
\mathit{FiveStar} \sqcap \mathrm{not}\exists \mathit{Beach} & \sqsubseteq & \exists \mathit{Pool},\\
\exists \mathit{Beach} & \sqsubseteq & \exists \mathit{SwimOpp},\\
\exists \mathit{Pool} & \sqsubseteq & \exists \mathit{SwimOpp},
\end{array}$$

Source: Gottlob, Lukasiewicz, Pieris. Datalog+/-: Questions and Answers. AAAI 2014. https://www.aaai.org/ocs/index.php/KR/KR14/paper/viewPaper/7965

Interesting Observations

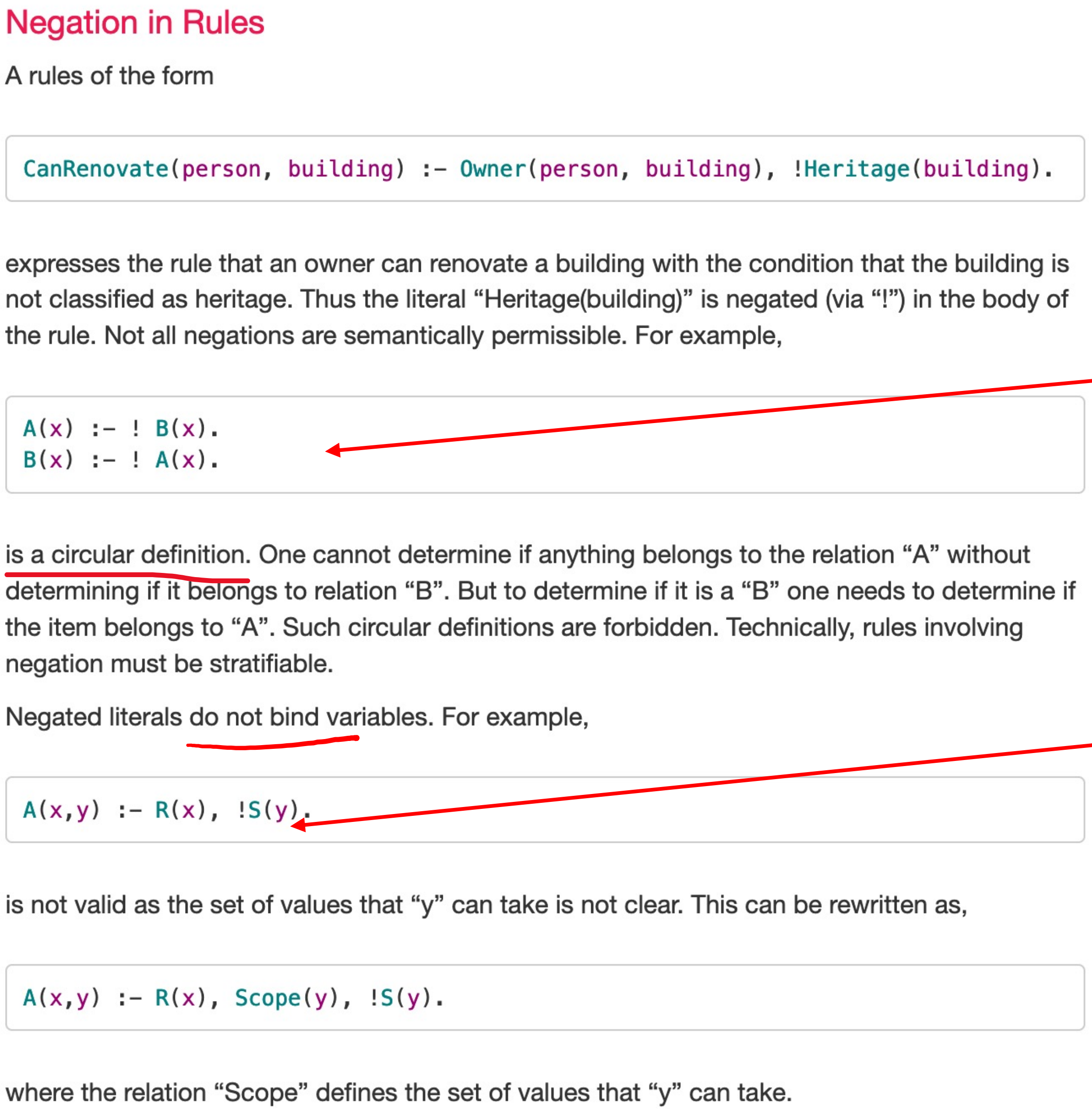
- Exploiting schema knowledge in query answering is not trivial
- Languages and algorithms exist that allow for tractable query answering
- Applications in real-world scenarios are possible
  - Industrial applications in data integration, Semantic Web, ontological reasoning
  - Companies and Products: RelationalAI, Deepreason.ai, Oracle Semantic Technologies, IBM IODT, OntoDLV (Vienna)

Based on a presentation by Andrea Cali

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]

Negation in Souffle vs. Negation in ASP



Negation in Rules

A rules of the form

```
CanRenovate(person, building) :- Owner(person, building), !Heritage(building).
```

expresses the rule that an owner can renovate a building with the condition that the building is not classified as heritage. Thus the literal “Heritage(building)” is negated (via “!”) in the body of the rule. Not all negations are semantically permissible. For example,

```
A(x) :- ! B(x).
B(x) :- ! A(x).
```

is a circular definition. One cannot determine if anything belongs to the relation “A” without determining if it belongs to relation “B”. But to determine if it is a “B” one needs to determine if the item belongs to “A”. Such circular definitions are forbidden. Technically, rules involving negation must be stratifiable.

Negated literals do not bind variables. For example,

```
A(x,y) :- R(x), !S(y).
```

is not valid as the set of values that “y” can take is not clear. This can be rewritten as,

```
A(x,y) :- R(x), Scope(y), !S(y).
```

where the relation “Scope” defines the set of values that “y” can take.

YES: stable model semantics as used by ASP can deal with this circular definition

NO: safety conditions are still the same as for souffle

Source: https://souffle-lang.github.io/rules

Answer Set Programming (ASP)

- Programming paradigm that can model AI problems (e.g, planning, combinatorics)
- Basic idea
  - Allow non-stratified negation and encode problem (specification & "instance") as logic program rules
  - Solutions are so-caled "stable models" of the program
- Semantics based on Possible Worlds and Stable Models
  - Given an answer set program P, there can be multiple solutions (stable models, answer sets)
  - Each model **M**: assignment of true/false value to propositions to make all formulas true (combinatorial)
  - Captures default reasoning, non-monotonic reasoning, constrained optimization, exceptions, weak exceptions, preferences, etc., in a natural way
- Finding stable models of answer set programs is not easy
  - Current systems CLASP, DLV, clingo, Smodels, etc., extremely sophisticated
  - Work by first grounding the program (= replacing variables with constants), suitably transforming it to a propositional theory whose models are stable models of the original program (contrast with "lifted inference" later)
  - These models are found using a SAT solver or solvers using similar ideas to SAT solvers

Rules with Negation

- Closed world assumption (CWA) as used in standard Datalog:
  - If a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.
- Problem: CWA can lead to inconsistencies when negation is allowed in rule bodies. Intuition: we can have multiple minimal models ("Herbrand models")

Example 1:

```
boring(chess) :- boring(chess).
```

⇨ ?

What are all the possible *minimal* models:

- Herbrand universe U_P (set of all constants) ={chess}
- Herbrand base B_P (set of grounded atoms) = {boring(chess)}
- Interpretations (all subsets of B_P) = { {}, {boring(chess)} }
- Model: interpretation that makes each ground instance of each rule true

The "boring chess" example is taken from "Ceri, Gottlob, Tanca. What you always wanted to know about Datalog (and never dared to ask). TKDE 1989. https://doi.org/10.1109/69.43410

Rules with Negation

- Closed world assumption (CWA) as used in standard Datalog:
  - If a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.
- Problem: CWA can lead to inconsistencies when negation is allowed in rule bodies. Intuition: we can have multiple minimal models ("Herbrand models")

Example 1:

boring(chess) :- boring(chess). ⇒ $M_1 = \{\}$

What are all the possible *minimal* models:

M_2 = {boring(chess)} is a model, but not minimal

The "boring chess" example is taken from "Ceri, Gottlob, Tanca. What you always wanted to know about Datalog (and never dared to ask). TKDE 1989. https://doi.org/10.1109/69.43410

Rules with Negation

- Closed world assumption (CWA) as used in standard Datalog:
  - If a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.
- Problem: CWA can lead to inconsistencies when negation is allowed in rule bodies. Intuition: we can have multiple minimal models ("Herbrand models")

Example 1:

boring(chess) :- boring(chess).

⇨

$M_1 = \{\}$

What are all the possible *minimal* models:

$M_2 = \{boring(chess)\}$ is a model, but not minimal

Example 2:

boring(chess) :- ¬interesting(chess).

⇨

?

Possible interpretations:
{ {}, {b(c)}, {i(c)},
{b(c),i(c)} }

What are all the possible *minimal* models:

The "boring chess" example is taken from "Ceri, Gottlob, Tanca. What you always wanted to know about Datalog (and never dared to ask). TKDE 1989. https://doi.org/10.1109/69.43410

Rules with Negation

- Closed world assumption (CWA) as used in standard Datalog:
  - If a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.
- Problem: CWA can lead to inconsistencies when negation is allowed in rule bodies. Intuition: we can have multiple minimal models ("Herbrand models")

Example 1:

boring(chess) :- boring(chess). ⇒ M_1 = {}

What are all the possible *minimal* models:

M_2 = {boring(chess)} is a model, but not minimal

Example 2:

boring(chess) :- ¬interesting(chess). ⇒ M_1 = {boring(chess)}

M_2 = {interesting(chess)}

What are all the possible *minimal* models:

The "boring chess" example is taken from "Ceri, Gottlob, Tanca. What you always wanted to know about Datalog (and never dared to ask). TKDE 1989. https://doi.org/10.1109/69.43410

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]

Horn clauses and logic programming

A clause is a disjunction of literals.

$\bar{a} \vee \bar{b} \vee c \vee d$

$a \wedge b \Rightarrow c \vee d$

$1 \wedge a \wedge b \Rightarrow c \vee d \vee 0$

A Horn clause has at most one positive (i.e. unnegated) literal.

?

Alfred Horn, ~1973
https://en.wikipedia.org/wiki/Alfred_Horn

Recall: $\bar{a} = \neg a = !a = \sim a =$ NOT a

Horn clauses and logic programming

A clause is a disjunction of literals.

$\bar{a} \lor \bar{b} \lor c \lor d$ $\quad$ $a \land b \Rightarrow c \lor d$

$1 \land a \land b \Rightarrow c \lor d \lor 0$

Alfred Horn, ~1973

https://en.wikipedia.org/wiki/Alfred_Horn

A Horn clause has at most one positive (i.e. unnegated) literal.

$\bar{a} \lor \bar{b} \lor c$? definite clause (exactly one positive)

c ? unit clause (facts, unconditional knowledge, empty body)

$\bar{a} \lor \bar{b}$? goal clause

Recall: $\bar{a} = \neg a = !a = {\sim}a = \text{NOT } a$

Horn clauses and logic programming

A clause is a disjunction of literals.

$\bar{a} \vee \bar{b} \vee c \vee d$ $\quad a \wedge b \Rightarrow c \vee d$

$1 \wedge a \wedge b \Rightarrow c \vee d \vee 0$

Alfred Horn, ~1973
https://en.wikipedia.org/wiki/Alfred_Horn

A Horn clause has at most one positive (i.e. unnegated) literal.

$\bar{a} \vee \bar{b} \vee c$ $\quad a \wedge b \Rightarrow c$ definite clause (exactly one positive)

c $\quad 1 \Rightarrow c$ unit clause (facts, unconditional knowledge, empty body)

$\bar{a} \vee \bar{b}$ $\quad a \wedge b \Rightarrow 0$ goal clause

Universal quantification (everything above was propositional)

$\neg\text{human}(X) \vee \text{mortal}(X)$

? ?

Recall: $\bar{a} = \neg a = !a = {\sim}a =$ NOT a

Horn clauses and logic programming

A clause is a disjunction of literals.

$\bar{a} \vee \bar{b} \vee c \vee d$ $\qquad$ $a \wedge b \Rightarrow c \vee d$

$1 \wedge a \wedge b \Rightarrow c \vee d \vee 0$

Alfred Horn, ~1973
https://en.wikipedia.org/wiki/Alfred_Horn

A Horn clause has at most one positive (i.e. unnegated) literal.

$\bar{a} \vee \bar{b} \vee c$ $\qquad$ $a \wedge b \Rightarrow c$ $\qquad$ definite clause (exactly one positive)

c $\qquad$ $1 \Rightarrow c$ $\qquad$ unit clause (facts, unconditional knowledge, empty body)

$\bar{a} \vee \bar{b}$ $\qquad$ $a \wedge b \Rightarrow 0$ $\qquad$ goal clause

Universal quantification (everything above was propositional)

$\neg \text{human}(X) \vee \text{mortal}(X)$

$\forall X[\neg \text{human}(X) \vee \text{mortal}(X)]$ $\qquad$ $\forall X[\text{ human}(X) \Rightarrow \text{mortal}(X)]$

Recall: $\bar{a} = \neg a = !a = \sim a =$ NOT a

Datalog grammar

P ∈ program	=	$r_1.\ r_2.\ \ldots\ r_n.$
r ∈ rule	=	a_0 :- $a_1, \ldots, a_m.$
a ∈ atom	=	$p(t_1, \ldots, t_k)$
t ∈ term	=	x | "c"

p = set of predicate symbols

x = set of variable symbols
c = set of constants

a ground atom has only constants as terms (no variables)

Concepts from logic programming

- P: Program

?

- U_P: Herbrand universe (or Herbrand domain or vocabulary)

?

- B_P: Herbrand base (or alphabet)

?

- I: Interpretation (or database instance or dataset)

?

- M: Model of P

?

- A model is minimal if ?

Jacques Herbrand, 1931
https://en.wikipedia.org/wiki/Jacques_Herbrand

Concepts from logic programming

- P: Program
  - set of facts (assertions) and rules (sentences that allow to infer new facts from existing ones)
- U_P: Herbrand universe (or Herbrand domain or vocabulary)

  ?
- B_P: Herbrand base (or alphabet)

  ?
- I: Interpretation (or database instance or dataset)

  ?
- M: Model of P

  ?
- A model is minimal if ?

Jacques Herbrand, 1931
https://en.wikipedia.org/wiki/Jacques_Herbrand

Concepts from logic programming

- P: Program
  - set of facts (assertions) and rules (sentences that allow to infer new facts from existing ones)
- U_P: Herbrand universe (or Herbrand domain or vocabulary)
  - set of all constants (variable-free terms) appearing in P (cp. with active domain interpretation)
- B_P: Herbrand base (or alphabet)
  - set of all ground atoms (variable-free) constructible with predicates from P and terms from U_P
- I: Interpretation (or database instance or dataset)

?

- M: Model of P

?

- A model is minimal if ?

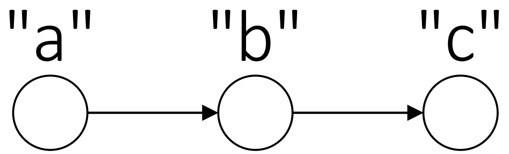
Concepts from logic programming

- P: Program
  - set of facts (assertions) and rules (sentences that allow to infer new facts from existing ones)
- U_P: Herbrand universe (or Herbrand domain or vocabulary)
  - set of all constants (variable-free terms) appearing in P (cp. with active domain interpretation)
- B_P: Herbrand base (or alphabet)
  - set of all ground atoms (variable-free) constructible with predicates from P and terms from U_P
- I: Interpretation (or database instance or dataset)
  - any subset of B_P
- M: Model of P
  - an interpretation that makes each ground instance of each rule in P true (a ground instance of a rule is obtained by replacing all variables in the rule by elements from U_P)
- A model is minimal if ?

Concepts from logic programming

- P: Program
  - set of facts (assertions) and rules (sentences that allow to infer new facts from existing ones)
- U_P: Herbrand universe (or Herbrand domain or vocabulary)
  - set of all constants (variable-free terms) appearing in P (cp. with active domain interpretation)
- B_P: Herbrand base (or alphabet)
  - set of all ground atoms (variable-free) constructible with predicates from P and terms from U_P
- I: Interpretation (or database instance or dataset)
  - any subset of B_P
- M: Model of P
  - an interpretation that makes each ground instance of each rule in P true (a ground instance of a rule is obtained by replacing all variables in the rule by elements from U_P)
- A model is minimal if it does not properly contain any other model

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Interpretation

?

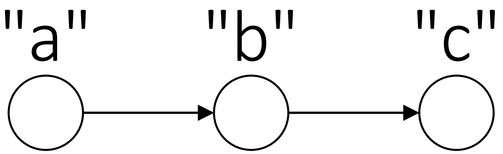
Herbrand universe U_P

?

Herbrand base B_P

?

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Interpretation

?

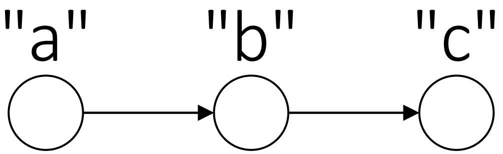
Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P

?

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Interpretation

?

Herbrand universe U_P

{"a", "b", "c"}

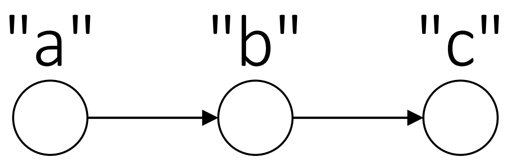
Herbrand base B_P $|B_P| = 18$

```
{ arc("a","a").    path("a","a").
  arc("a","b").    path("a","b").
  arc("a","c").    path("a","c").
      ⋮                ⋮
  arc("c","b").    path("c","b").
  arc("c","c").    path("c","c"). }
```

Contains a wild mix of

- explicit facts that we know (IDB) like arc("a","b"),
- facts that can be inferred (EDB) like path("a","b"), and
- facts that cannot be inferred like path("c","a") or arc("a","a")

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Interpretation one of many interpretations

```
arc("a","b"). arc("b","c"). arc("b","a").
path("a","b"). path("b","c"). path("b","a").
path("a","c"). path("a","a").
```

Is this interpretation a model? ?

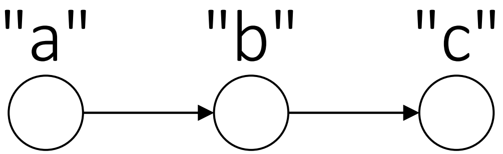
Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P $|B_P| = 18$

```
{ arc("a","a").     path("a","a").
  arc("a","b").     path("a","b").
  arc("a","c").     path("a","c").
        ⋮                 ⋮
  arc("c","b").     path("c","b").
  arc("c","c").     path("c","c"). }
```

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P

$|B_P| = 18$

{ arc("a","a").
arc("a","b").
arc("a","c").
⋮
arc("c","b").
arc("c","c").

path("a","a").
path("a","b").
path("a","c").
⋮
path("c","b").
path("c","c"). }

Interpretation

one of many interpretations

arc("a","b"). arc("b","c"). arc("b","a").
path("a","b"). path("b","c"). path("b","a").
path("a","c"). path("a","a").

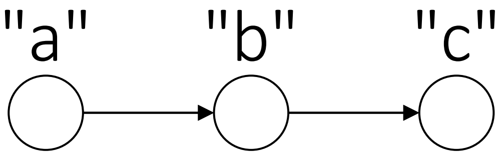
Is this interpretation a model?

No! There is a rule for which there is a ground instance that is not true in this interpretation

x→"b", y→"b", z→"a":
path("b","b") :- arc("b","a"), path("a","b").

This is an example grounding of a rule.

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P

$|B_P| = 18$

{ arc("a","a").
arc("a","b").
arc("a","c").
⋮
arc("c","b").
arc("c","c").

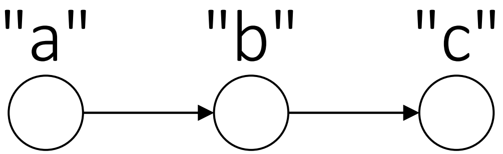
path("a","a").
path("a","b").
path("a","c").
⋮
path("c","b").
path("c","c"). }

Interpretation

arc("a","b"). arc("b","c"). arc("b","a").
path("a","b"). path("b","c"). path("b","a").
path("a","c"). path("a","a"). **path("b","b").**

Is this new interpretation a model? ?

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Interpretation

arc("a","b"). arc("b","c"). arc("b","a").
path("a","b"). path("b","c"). path("b","a").
path("a","c"). path("a","a"). **path("b","b").**

Herbrand universe U_P

{"a", "b", "c"}

Is this new interpretation a model?

Yes!

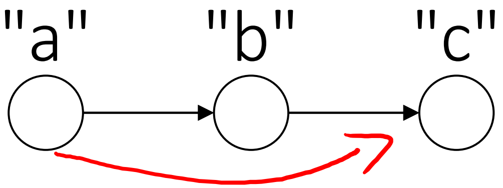
Herbrand base B_P $|B_P| = 18$

Is this model minimal?

?

{ arc("a","a"). path("a","a").
arc("a","b"). path("a","b").
arc("a","c"). path("a","c").
⋮ ⋮
arc("c","b"). path("c","b").
arc("c","c"). path("c","c"). }

Herbrand, interpretations, models



Program P

```
arc("a","b"). arc("b","c").
path(x,y) :- arc(x,y).
path(x,y) :- arc(x,z), path(z,y).
```

Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P $|B_P| = 18$

{ arc("a","a").
arc("a","b").
arc("a","c").
⋮
arc("c","b").
arc("c","c").

path("a","a").
path("a","b").
path("a","c").
⋮
path("c","b").
path("c","c"). }

Interpretation

arc("a","b"). arc("b","c"). ~~arc("b","a").~~
path("a","b"). path("b","c"). ~~path("b","a").~~
path("a","c"). ~~path("a","a"). path("b","b").~~

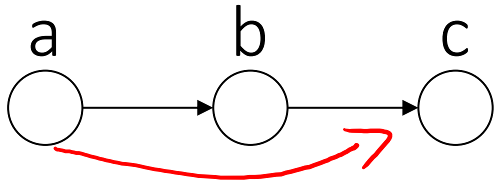
Is this new interpretation a model?

Yes!

Is this model minimal?

No! There is a properly contained model

Herbrand, interpretations, models



Program P

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```

Interpretation

arc(a,b). arc(b,c). ~~arc(b,a).~~
path(a,b). path(b,c). ~~path(b,a).~~
path(a,c). ~~path(a,a). path(b,b).~~

Convention in ASP:

- Variables begin with upper-case
- constants begin with lower-case

Herbrand universe U_P

{a, b, c}

Herbrand base B_P $|B_P| = 18$

{ arc(a,a).
arc(a,b).
arc(a,c).
⋮
arc(c,b).
arc(c,c).
path(a,a).
path(a,b).
path(a,c).
⋮
path(c,b).
path(c,c). }

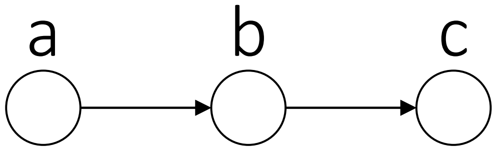
Is this new interpretation a model?

Yes!

Is this model minimal?

No! There is a properly contained model

Evaluating ASP's with Clingo



paths1

paths1.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```

```
clingo paths1.txt
```

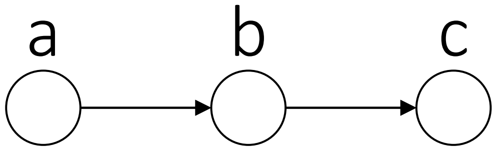
?

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Postassco/Clingo available for download at: https://teaching.potassco.org/

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Evaluating ASP's with Clingo



paths1

paths1.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```

```
clingo paths1.txt
```

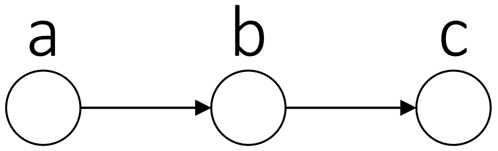
```
Solving...
Answer: 1
arc(a,b) arc(b,c) path(a,b)
path(b,c) path(a,c)
SATISFIABLE
```

Shows all predicates, including EDBs

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Postassco/Clingo available for download at: https://teaching.potassco.org/

Evaluating ASP's with Clingo



paths2

paths2.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
#show path/2.
```

Show only the facts in the predicate named "path" with arity "2"

```
clingo paths2.txt
```

```
Solving...
Answer: 1
path(a,b) path(b,c) path(a,c)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Postassco/Clingo available for download at: https://teaching.potassco.org/
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]

Semantics: Informally

- Informally, a stable model M of a ground program P is a set of ground atoms such that
  1. Every rule is satisfied:

     i.e., for any rule in P

     h :- $a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n$.

     if each atom a_i is satisfied (a_i's are in M) and no atom b_i is satisfied (i.e. no b_i is in M), then h is in M.
  2. Every $h \in M$ can be derived from a rule by a "non-circular reasoning" (informal for: we are looking for minimal models, or there is some "derivation provenance")

Recall that alternatives to "¬" are "not" and "!"and "~". Writing out "not" explicitly is more common in ASP.

Semantics: "non-circular" more formally

Idea: Guess a model M (= a set of atoms). Then verify M is the exact set of atoms that "can be derived" under standard minimal model semantics on P^M on a modified positive program P^M (called "the reduct") derived from P as follows:

1. Create all possible groundings of the rules of program P
2. Delete all grounded rules that contradict M

$\cancel{h \text{ :- } a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n.}$ if some $b_i \in M$

3. In remaining grounded rules, delete all negative literals

$h \text{ :- } a_1, \ldots, a_m, \cancel{\neg b_1, \ldots, \neg b_n}.$ if no $b_i \in M$

M is a stable model of P iff M is the least model of P^M

Recall that alternatives to "¬" are "not" and "!"and "~". Writing out "not" explicitly is more common in ASP.

Semantics: "non-circular" more concisely

Idea: Guess a model M (= a set of atoms). Then verify M is the exact set of atoms that "can be derived" under standard minimal model semantics on P^M on a modified positive program P^M (called "the reduct") derived from P as follows:

The reduct of P w.r.t M is:

$$P^M = \left\{ \boxed{h \text{ :- } a_1, \ldots, a_m.} \;\middle|\; \boxed{h \text{ :- } a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n.} \in \text{grounding of } P \;\wedge\; \text{no } b_i \in M \right\}$$

M is a stable model of P iff M is the least model of P^M

Recall that alternatives to "¬" are "not" and "!"and "~". Writing out "not" explicitly is more common in ASP.

Examples

"a" is a proposition that is either true or false

P1:

```
a :- a.
```

M={a}   Is M a stable model of P1?   ?

Examples

"a" is a proposition that is either true or false

P1:

```
a :- a.
```

~~M={a}~~ not a stable model (<u>not minimal</u>, derivation of "a" is based on circular reasoning: {a} is not least model of a :- a)

? What is a stable model?

Examples

"a" is a proposition that is either true or false. Intuitively a predicate with zero arguments (arity 0)

P1:

```
a :- a.
```

~~M={a}~~ not a stable model (not minimal, derivation of "a" is based on circular reasoning: {a} is not least model of a :- a)

M={} stable model

P2:

```
a :- not b.
```

Interpretations:

{ {a}, {b},
{}, {a,b} }

?

Examples

"a" is a proposition that is either true or false

P1: a :- a.

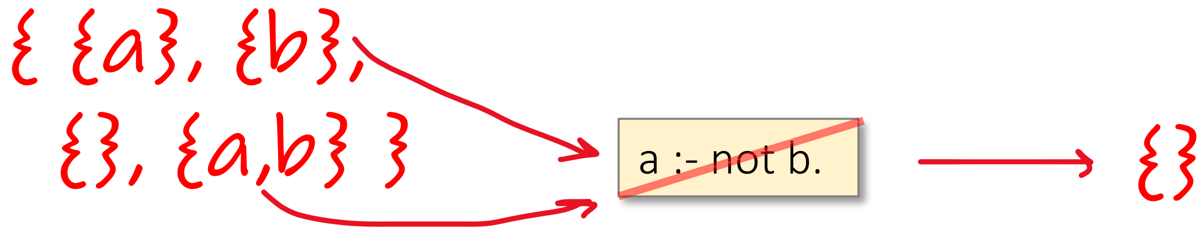
~~M={a}~~ not a stable model (not minimal, derivation of "a" is based on circular reasoning: {a} is not least model of a :- a)

M={} stable model

P2: a :- not b.

?

Interpretations:



{ {a}, {b}, {}, {a,b} }

~~a :- not b.~~ → {}

Examples

"a" is a proposition that is either true or false

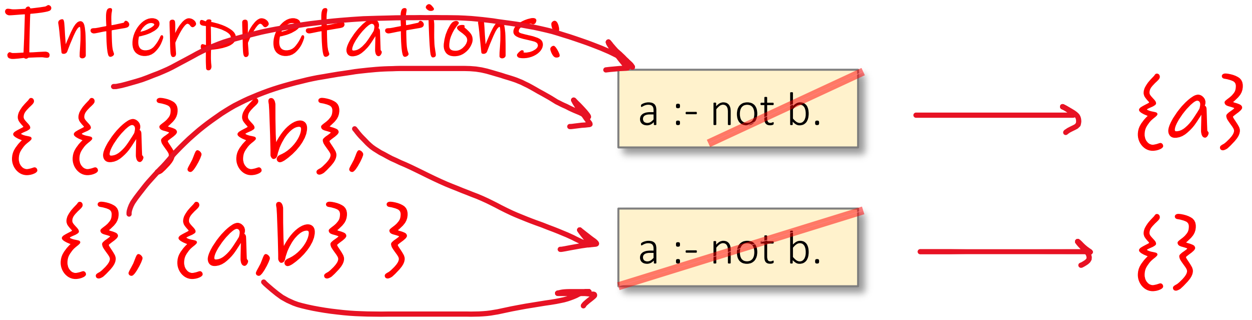
P1: `a :- a.`

~~M={a}~~ not a stable model (<u>not minimal</u>, derivation of "a" is based on circular reasoning: {a} is not least model of a :- a)

M={} stable model

P2: `a :- not b.`

?



Interpretations:

{ {a}, {b}, {}, {a,b} }

a :- ~~not b.~~ ⟶ {a}

~~a :- not b.~~ ⟶ {}

Examples

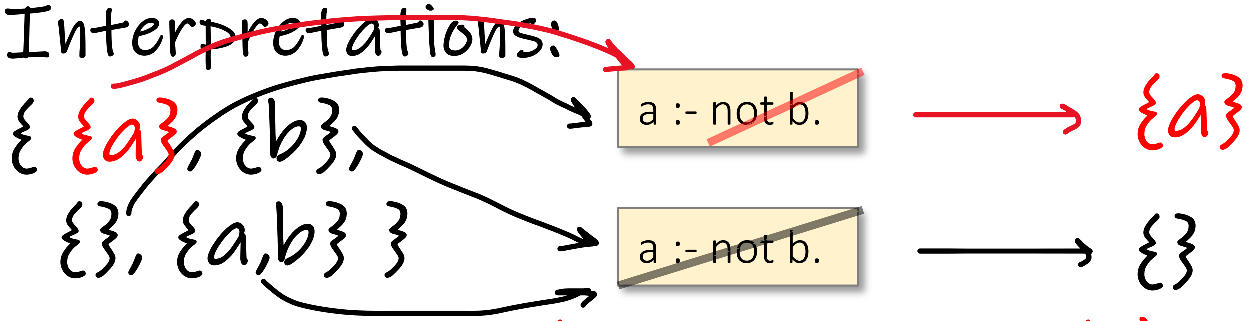
"a" is a proposition that is either true or false

P1: a :- a.

~~M={a}~~ not a stable model (<u>not minimal</u>, derivation of "a" is based on circular reasoning: {a} is not least model of a :- a)

M={} stable model

P2: a :- not b.



Interpretations:

{ {a}, {b}, {}, {a,b} }

a :- ~~not b.~~ → {a}

~~a :- not b.~~ → {}

M={a}

only stable model (compare to the the earlier chess example)

P3: a :- not a.

{ {}, {a} }

?

Examples

"a" is a proposition that is either true or false

P1: a :- a.

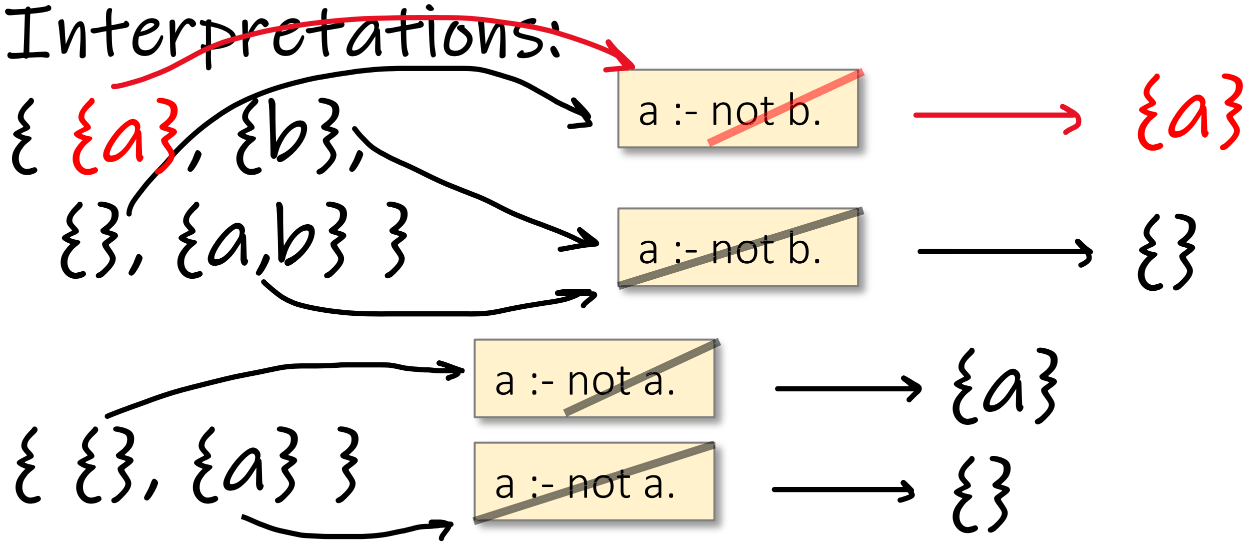
~~M={a}~~ not a stable model (not minimal, derivation of "a" is based on circular reasoning: {a} is not least model of a :- a)

M={} stable model

P2: a :- not b.

M={a}
only stable model

P3: a :- not a.



Interpretations:
{ {a}, {b}, {}, {a,b} }

~~a :- not b.~~ → {a}

~~a :- not b.~~ → {}

{ {}, {a} }

~~a :- not a.~~ → {a}

~~a :- not a.~~ → {}

has no stable model (cp. to earlier "Box(x) :- Item(x), ¬Box(x).")

Examples

P4:

```
a :- not b.
b :- not a.
```

?

Examples

P4:

```
a :- not b.
b :- not a.
```

$M_1=\{a\}$

$M_2=\{b\}$

two stable models

How can you "prove" that M_1 is a stable model?

?

Examples

P4:

```
a :- not b.
b :- not a.
```

a ~~:- not b~~.

~~b :- not a.~~

$M_1=\{a\}$

$M_2=\{b\}$

two stable models

Examples

P4:

```
a :- not b.
b :- not a.
```

a ~~:- not b~~.

~~b :- not a.~~

M_1={a}

M_2={b}

two stable models

P5:

```
a :- not b.
b :- not a.
a :- not a.
```

? { {}, {a}, {b}, {a,b} }

Examples

P4:

```
a :- not b.
b :- not a.
```

$M_1=\{a\}$

$M_2=\{b\}$

two stable models

a ~~:- not b~~.

~~b :- not a.~~

P5:

```
a :- not b.
b :- not a.
a :- not a.
```

$M=\{a\}$

only stable model

How can you "prove" that M is a stable model?

?

Examples

P4:

a :- not b.

b :- not a.

a ~~:- not b~~.

~~b :- not a.~~

M_1={a}

M_2={b}

two stable models

P5:

a :- not b.

b :- not a.

a :- not a.

a ~~:- not b~~.

~~b :- not a.~~

~~a :- not a.~~

M={a}

only stable model

Evaluating ASP's with Clingo

p4, p5

p4.txt

```
a :- not b.
b :- not a.
```

$M_1=\{a\}$

$M_2=\{b\}$

print all stable models (not just one)

```
clingo p4.txt -n 0
```

→

```
Answer: 1
b
Answer: 2
a
SATISFIABLE
```

p5.txt

```
a :- not b.
b :- not a.
a :- not a.
```

$M=\{a\}$

```
clingo p5.txt -n 0
```

→

```
Answer: 1
a
SATISFIABLE
```

Potassco Solutions

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Postassco/Clingo: Download: https://potassco.org/clingo/, Running in the browser: https://potassco.org/clingo/run/, More resources on clingo: https://teaching.potassco.org/

Updated 2/24/2024

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 12

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

2/23/2024

Pre-class conversations

- Last class summary
- Scribe correction: I make a pass on Monday (before next class)
- Project discussions (in class and after)
- Faculty candidates (THU Feb 29, WED March 20)

- Today:
  - Stable models, ASP
  - Later: Beyond NP with ASP (including 3-colorability in 2 lines)

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]

Discussion from last time

P2: `a :- not b.`

M={a} is the only stable model

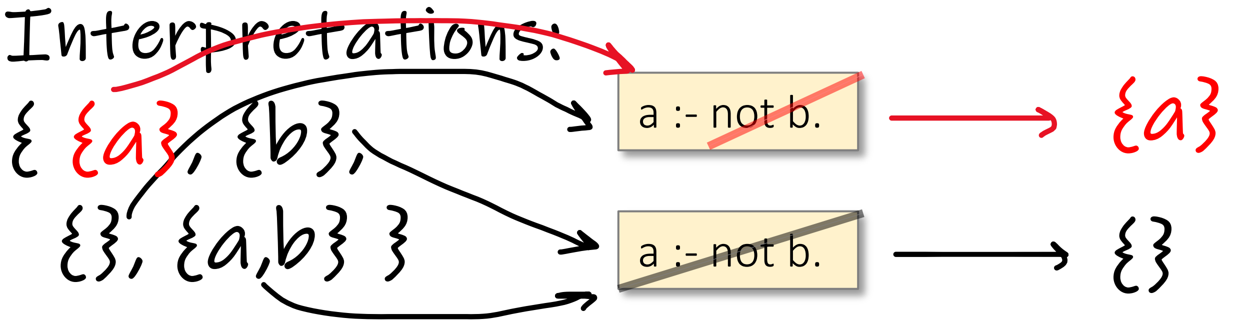
$$\text{not } b \Rightarrow a$$

$$b \vee a$$

$$a \vee b$$

$$\text{not } a \Rightarrow b$$

Logically equivalent



?

"Why should syntax determine the semantics?"

P6: `b :- not a.`

M={b} is the only stable model

Discussion from last time

P2: a :- not b.

M={a} is the only stable model

$\text{not}\ b \Rightarrow a$

$b \vee a$

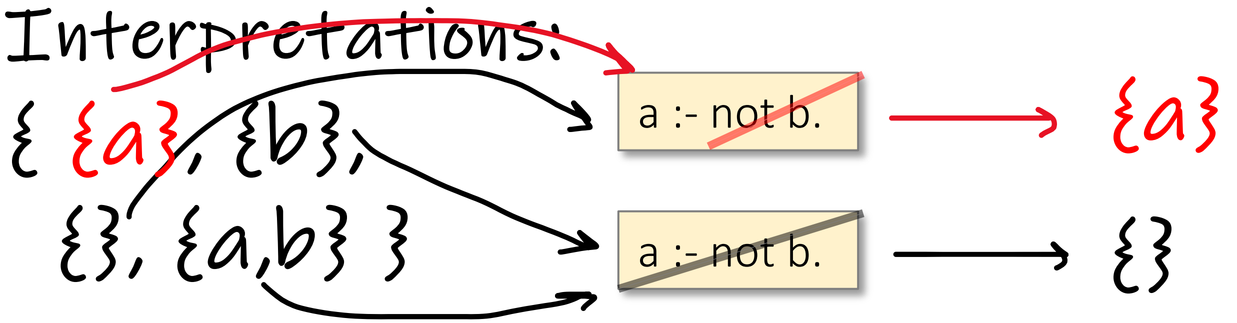
$a \vee b$

$\text{not}\ a \Rightarrow b$

Logically equivalent

P6: b :- not a.

M={b} is the only stable model



Interpretations:

{ {a}, {b}, {}, {a,b} }

a :- ~~not b.~~ → {a}

~~a :- not b.~~ → {}

a :- a.

$a \Rightarrow a$

$\bar{a} \vee a$

recall that we want to have the least model in standard Datalog (non-circular)

What do empty bodies or heads mean in ASP?

```
a :- b, not c.
```

Think of the head as a disjunction, body as conjunction

$0 \vee a \Leftarrow 1 \wedge b \wedge \neg c$

"Disjunctive Logic Programming": disjunctions in the head

Empty body:

```
a.
```

?

Empty head:

```
   :- b, not c.
```

?

What do empty bodies or heads mean in ASP?

```
a :- b, not c.
```

Think of the head as a disjunction, body as conjunction

$0 \lor a \Leftarrow 1 \land b \land \neg c$

"Disjunctive Logic Programming": disjunctions in the head

Empty body:

```
a.
```

$a \Leftarrow 1$

Empty body describes a fact:
"a" needs to be true.
Also in Datalog

Empty head:

```
  :- b, not c.
```

?

What do empty bodies or heads mean in ASP?

```
a :- b, not c.
```

Think of the head as a disjunction, body as conjunction

$$0 \lor a \Leftarrow 1 \land b \land \neg c$$

"Disjunctive Logic Programming": disjunctions in the head

Empty body:

```
a.
```

$$a \Leftarrow 1$$

Empty body describes a fact:
"a" needs to be true.
Also in Datalog

Empty head:

```
 :- b, not c.
```

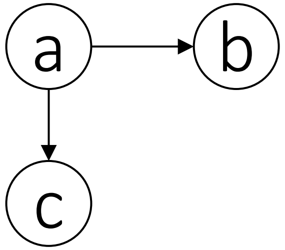
$$0 \qquad \Leftarrow b \land \neg c$$

Empty heads describes a constraint: "b and not c" must not be true in any model. Emtpy head describes a condition in the body which leads to contradiction (false)

3-colorability

vertex	A
	a
	b
	c

E	S	T
	a	b
	a	c



Q: For a graph (V, E) assign each vertex a color in {1, 2, 3} such that no adjacent vertices have the same color.

?

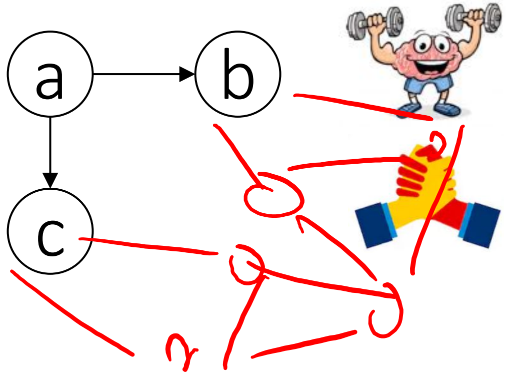
Convention in ASP: Capital letters are variables, lower case letters constants

Cp. edge(X,a)
vs. edge(x,"a")

3-colorability

vertex	X
	a
	b
	c

E	S	T
	a	b
	a	c



Q: For a graph (V, E) assign each vertex a color in {1, 2, 3} such that no adjacent vertices have the same color.

EDB (facts)

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
```

IDB

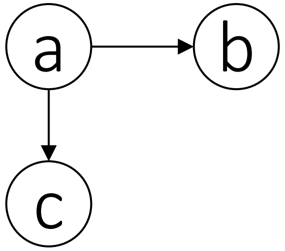
Convention in ASP: Capital letters are variables, lower case letters constants

Cp. edge(X,a)
vs. edge(x,"a")

Every vertex needs to have a color ?

Vertices from an edge can't have same color ?

3-colorability



Q: For a graph (V, E) assign each vertex a color in {1, 2, 3} such that no adjacent vertices have the same color.

EDB (facts)

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
```

IDB

```
color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
```

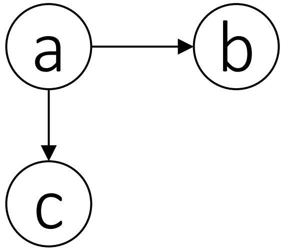
Convention in ASP:
Capital letters are variables, lower case letters constants

Cp. edge(X,a)
vs. edge(x,"a")

Every vertex needs to have a color

Vertices from an edge can't have same color ?

3-colorability



Q: For a graph (V, E) assign each vertex a color in {1, 2, 3} such that no adjacent vertices have the same color.

EDB (facts)

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
```

IDB

```
color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
```

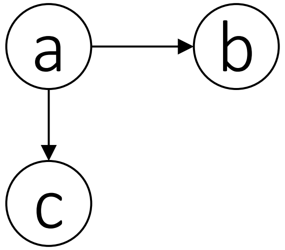
Convention in ASP: Capital letters are variables, lower case letters constants

Cp. edge(X,a)
vs. edge(x,"a")

Vertices from an edge can't have same color ?

":- edge(a,X), edge(b,X)" means that "a" and "b" don't share a neighbor

3-colorability



Q: For a graph (V, E) assign each vertex a color in {1, 2, 3} such that no adjacent vertices have the same color.

EDB (facts)

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
```

IDB

```
color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
```

```
:- edge(V,U), color(V,C), color(U,C).
```

constraint

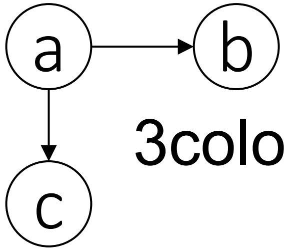
Convention in ASP:
Capital letters are variables, lower case letters constants

Cp. edge(X,a)
vs. edge(x,"a")

Vertices from an edge can't have same color

":- edge(a,X), edge(b,X)" means that "a" and "b" don't share a neighbor

3-colorability with Clingo



3colorability1

```
clingo 3colorability1.txt
```

3colorability1.txt

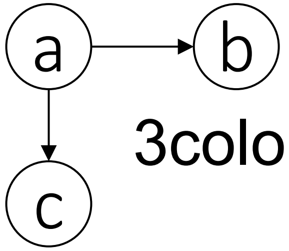
```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
:- edge(V,U), color(V,C), color(U,C).
```

Returns a stable model if it exists. Since there is a stable model, the problem is "satisfiable".

```
Answer: 1
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c) color(a,1)
color(b,3) color(c,3)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability with Clingo



3colorability1

```
clingo 3colorability1.txt –n 0
```

print all stable models (not just one)

3colorability1.txt

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
:- edge(V,U), color(V,C), color(U,C).
```

```
Answer: 1
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c)
color(a,1) color(b,3) color(c,3)
Answer: 2
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c)
color(a,1) color(b,3) color(c,2)
Answer: 3
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c)
color(a,1) color(b,2) color(c,3)

...

Answer: 11
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c)
color(a,3) color(b,2) color(c,2)
Answer: 12
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c)
color(a,3) color(b,1) color(c,2)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]

Data Conflict Resolution Using Trust Mappings

Wolfgang Gatterbauer & Dan Suciu

June 8, Sigmod 2010

Paper: https://doi.org/10.1145/1807167.1807193

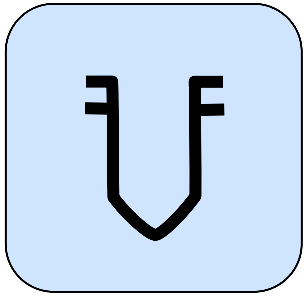
Full version with proofs: http://arxiv.org/pdf/1012.3320

Old Project web page: https://db.cs.washington.edu/projects/beliefdb/

Problem in social data: often no single ground truth

The Indus Script*

What is the origin of this glyph?



Bob : cow

Alice : ship hull

Charlie : jar

* Current state of knowledge on the Indus Script: Rao et al., Science 324(5931):1165, May 2009

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Background: Conflicts & Trust in Community DBs

Conflicting beliefs

glyph	origin	
Ʊ	ship hull	Alice
Ʊ	cow	Bob
Ʊ	jar	Charlie
ᛟ	fish	Bob
ᛟ	knot	Charlie
↟	arrow	Charlie

"Beliefs": annotated (key,value) pairs

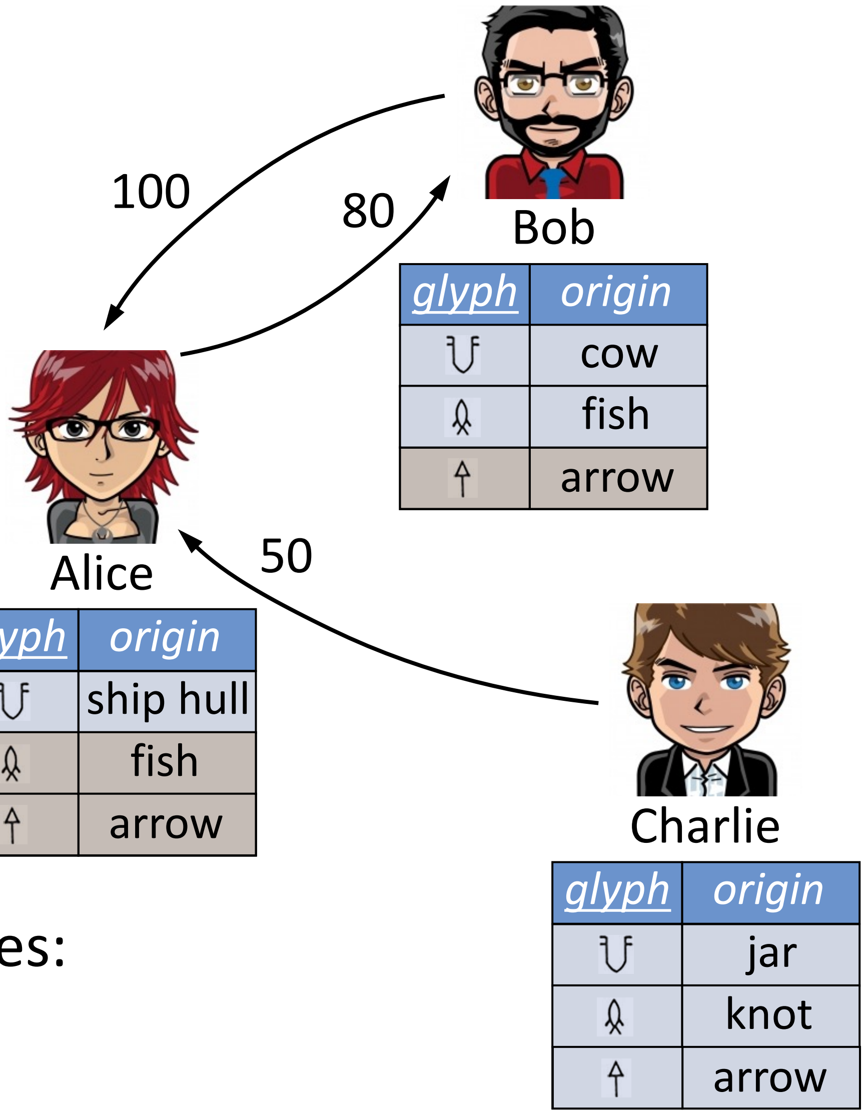
Trust mappings

Alice ← Bob	(100)
Alice ← Charlie	(50)
Bob ← Alice	(80)

Priorities

"Explicit belief"

"Implicit belief"



Bob

glyph	origin
Ʊ	cow
ᛟ	fish
↟	arrow

Alice

glyph	origin
Ʊ	ship hull
ᛟ	fish
↟	arrow

Charlie

glyph	origin
Ʊ	jar
ᛟ	knot
↟	arrow

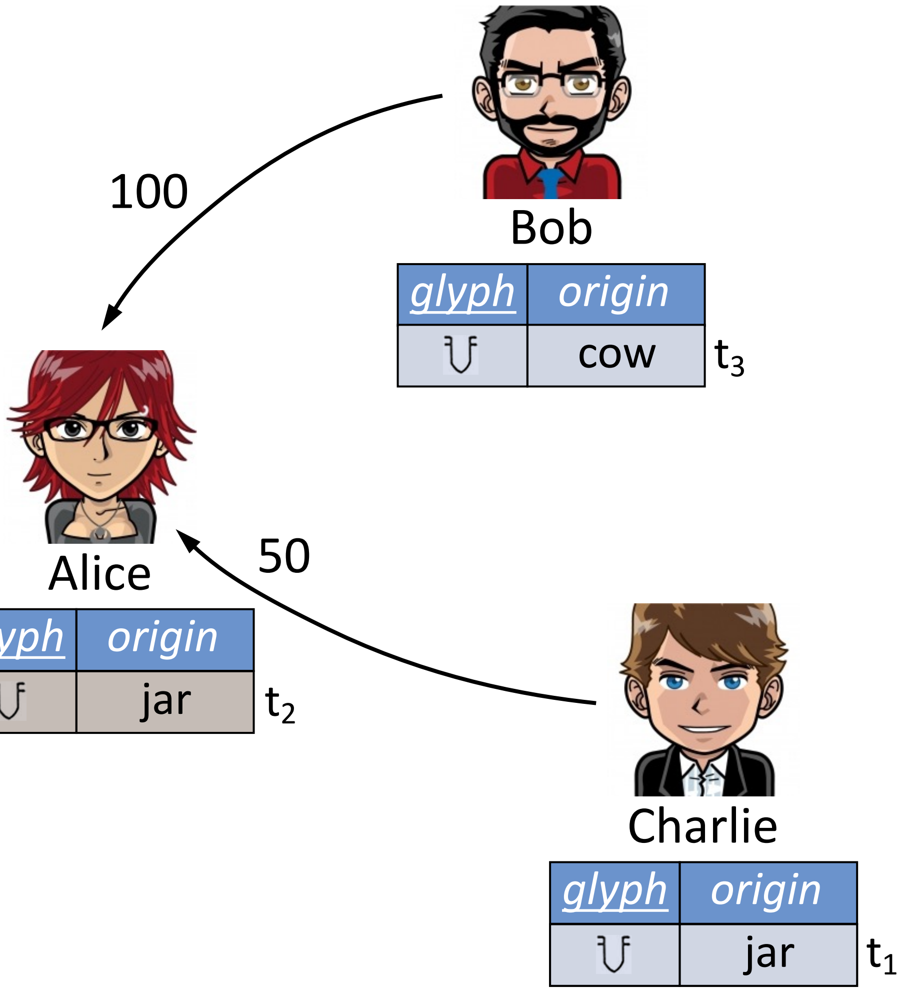
Recent work on community databases:

- Taylor & Ives [SIGMOD'06] ← Orchestra
- Green et al. [VLDB'07] ← Orchestra
- Kot & Koch [VLDB'09] ← Youtopia
- **GBKS** [VLDB'09] ← BeliefDB

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Limitations of previous work: transient effects

1. Incorrect inserts
   - Value depends on order of inserts



glyph	*origin*	
Ʊ	cow	t_3

glyph	*origin*	
Ʊ	jar	t_2

glyph	*origin*	
Ʊ	jar	t_1

ϟ Alice would have preferred Bob's value over Charlie's

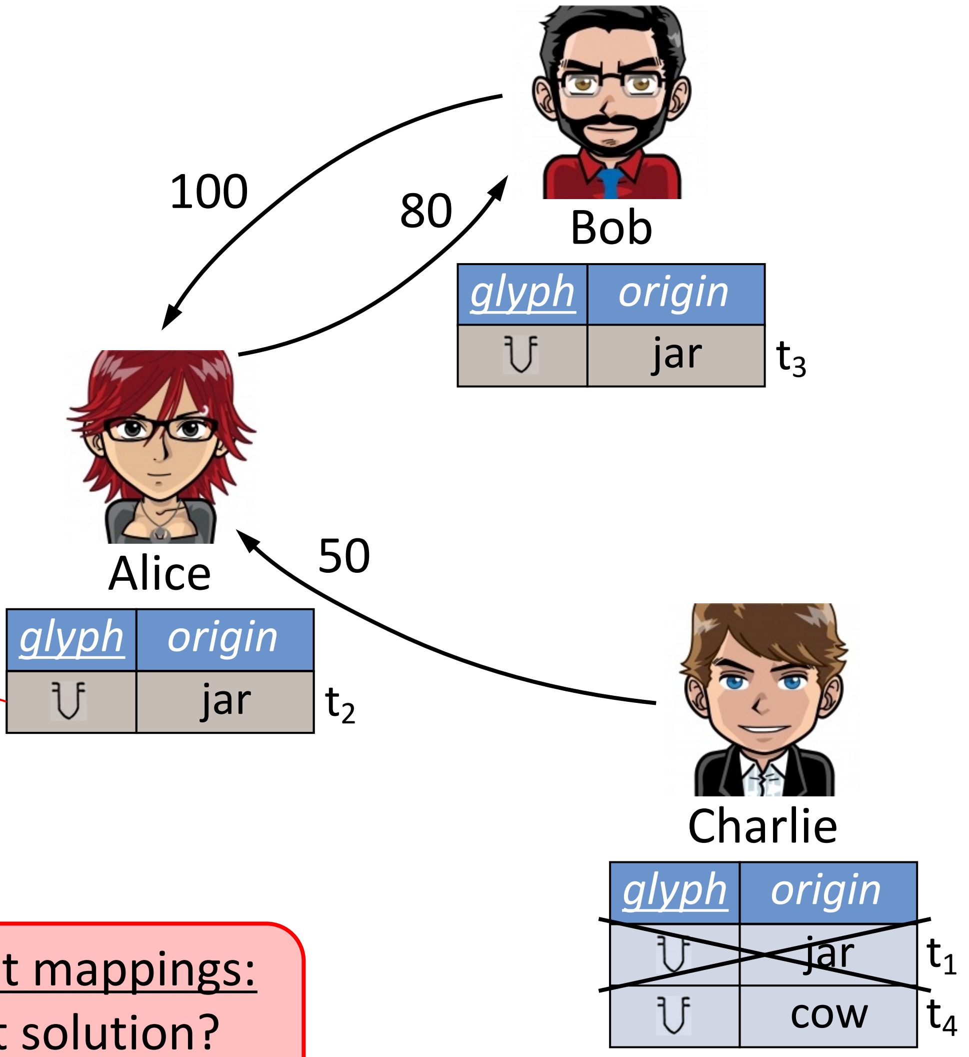
Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Limitations of previous work: transient effects

1. Incorrect inserts
   - Value depends on order of inserts

2. Incorrect updates
   - Mis-handling of revokes

↯ Alice and Bob trust each other most, but have lost "justification" for their beliefs



Bob

glyph	*origin*	
U	jar	t_3

Alice

glyph	*origin*	
U	jar	t_2

Charlie

glyph	*origin*	
~~U~~	~~jar~~	t_1
U	cow	t_4

<u>Automatic conflict resolution with trust mappings:</u>

1. How to define a globally consistent solution?
2. How to calculate it efficiently?

(3. Several extensions)

GS [Sigmod'10]

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Agenda

1. Stable solutions
   – how to define a unique and consistent solution?

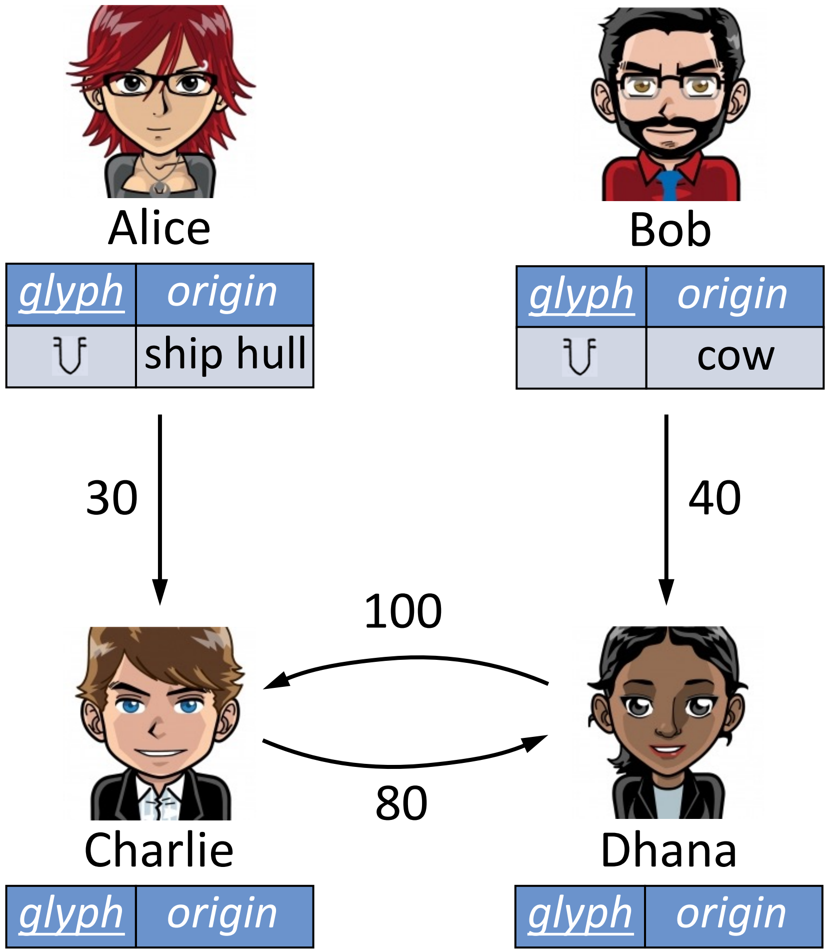
2. Resolution algorithm
   – how to calculate the solution efficiently?

3. Extensions
   – how to deal with "negative beliefs"?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Binary Trust Networks (BTNs)

To simplify presentation: focus on binary TNs



Alice

glyph	origin
Ʉ	ship hull

Bob

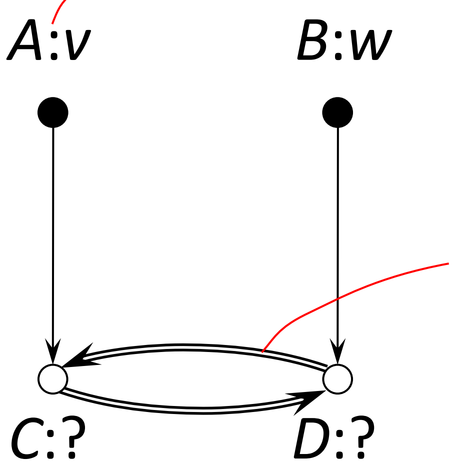
glyph	origin
Ʉ	cow

Charlie

glyph	origin

Dhana

glyph	origin



User A has explicit belief v

User D is user C's "preferred parent"

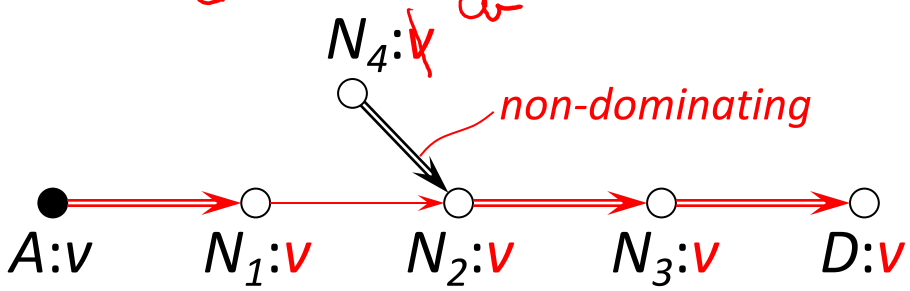
Focus on one single key (we ignore the glyph)

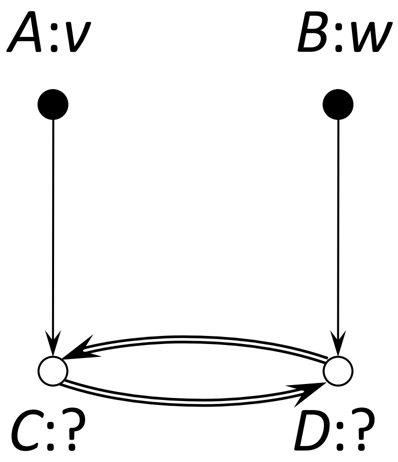
Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

The definition of a globally consistent solution

- Stable solution
  - assignment of values to each node, s.t. each belief has a "*non-dominated ~~lineage~~*" to an explicit belief

Provenance

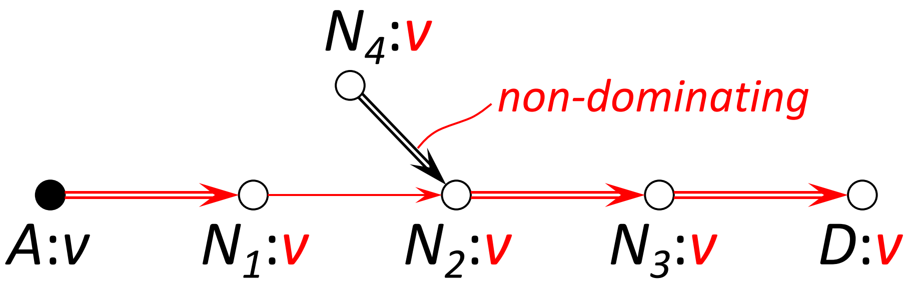


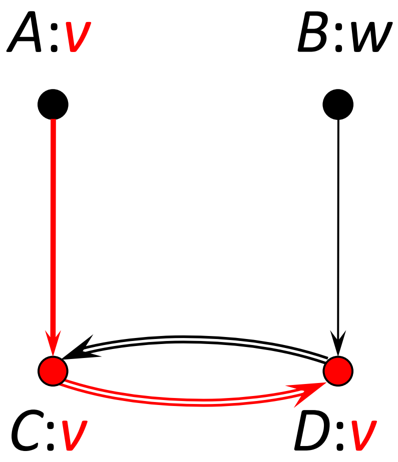


Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

The definition of a globally consistent solution

- Stable solution
  - assignment of values to each node, s.t. each belief has a *"non-dominated lineage"* to an explicit belief



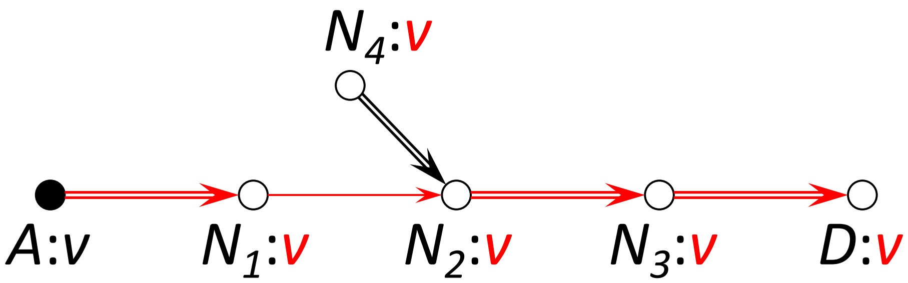


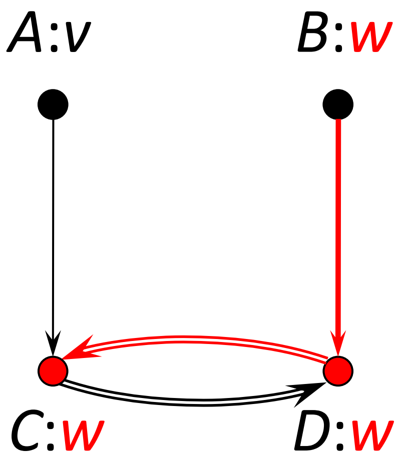
SS_1=(*A*:*v*, *B*:*w*, *C*:*v*, *D*:*v*)

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

The definition of a globally consistent solution

- Stable solution
  - assignment of values to each node, s.t. each belief has a *"non-dominated lineage"* to an explicit belief



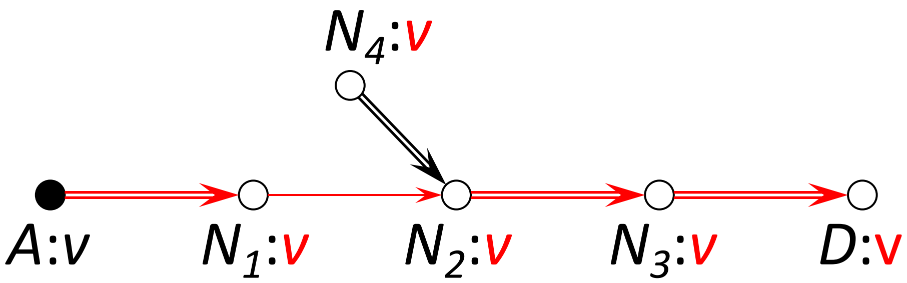


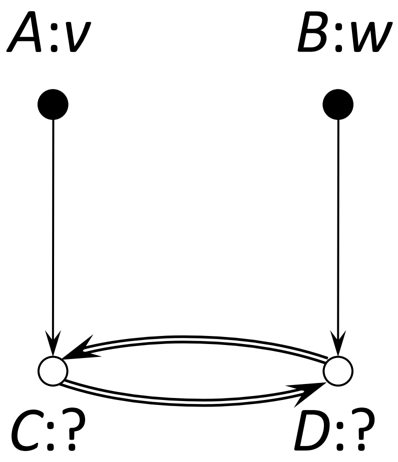
SS_1=(A:v, B:w, C:v, D:v)
SS_2=(A:v, B:w, C:w, D:w)

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Possible and certain values from all stable solutions

- Stable solution
  - assignment of values to each node, s.t. each belief has a *"non-dominated lineage"* to an explicit belief





SS$_1$=(*A*:*v*, *B*:*w*, *C*:*v*, *D*:*v*)
SS$_2$=(*A*:*v*, *B*:*w*, *C*:*w*, *D*:*w*)

- Possible / Certain semantics
  - a stable solution determines, for each node, a possible value ("**poss**")
  - certain value ("**cert**") = intersection of all stable solutions, per user

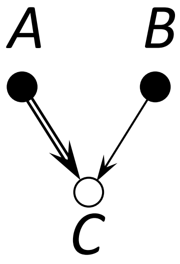
X	**poss**(*X*)	**cert**(*X*)
A	{*v*}	{*v*}
B	{*w*}	{*w*}
C	{*v*,*w*}	∅
D	{*v*,*w*}	∅

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Logic programs (LP) with stable model semantics

Convention from LP solver DLV: constants and predicates start with lowercase letters, variables with uppercase letters.

- LPs can capture this semantics.



```
   poss(c,X) :- poss(a,X).
block(c,b,Y) :-poss(b,Y), poss(c,X), X!=Y.
    poss(c,Y) :- poss(b,Y), not block(c,b,Y).
```

- There exist powerful and free LP solver available.
- Previous work on peer data exchange suggest using LPs.

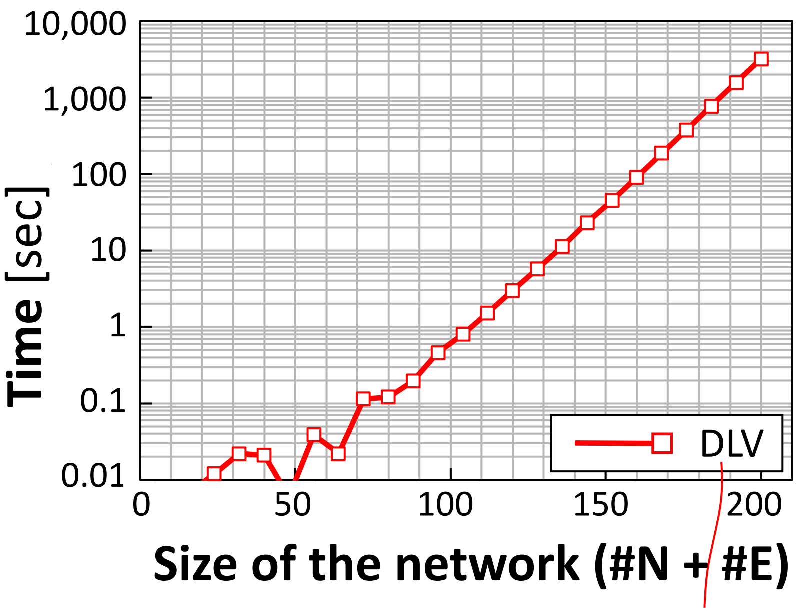
Greco et al. [TKDE'03]

Arenas et al. [TLP'03]

Barcelo, Bertossi [PADL'03]

Bertossi, Bravo [LPAR'07]

But solving LPs is hard ☹

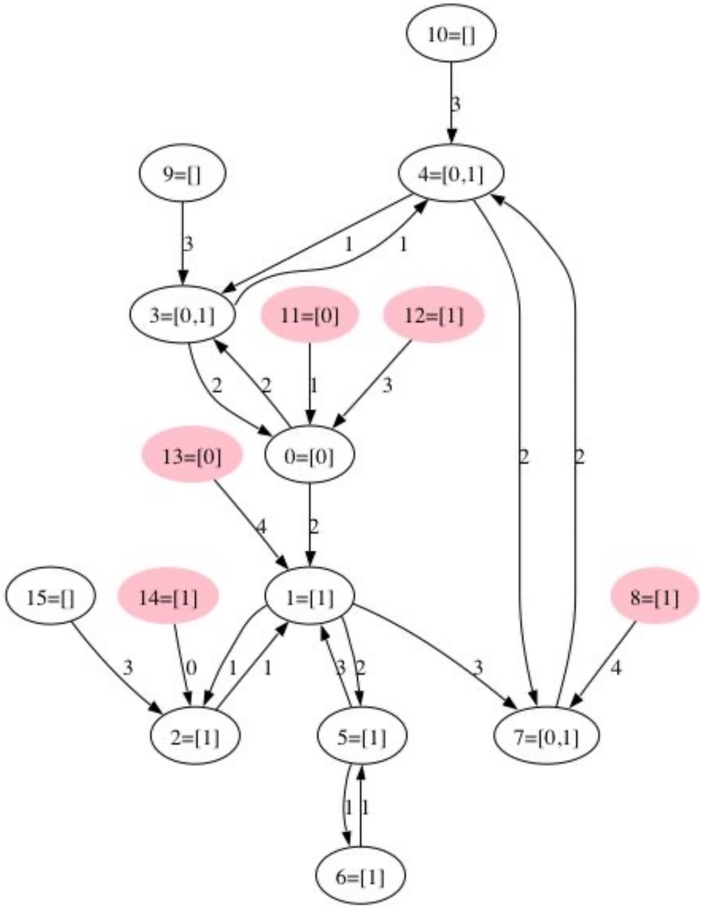


State-of-the-art LP solver

Yet surprisingly, our problem allows a PTIME solution ☺

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

DLV example



Size: 38

input.txt

```
% --- Insert explicit beliefs ---
possH(h8_0,1).
possH(h11_0,0).
possH(h12_0,1).
possH(h13_0,0).
possH(h14_0,1).
% --- Node: 0 ---
possH(h0_1,X)       :- possH(h0_0,X).
block(h0_1,11,X)    :- poss(11,X), possH(h0_1,Y), Y!=X.
possH(h0_1,X)       :- poss(11,X), not block(h0_1,11,X).
possH(h0_2,X)       :- possH(h0_1,X).
block(h0_2,3,X)     :- poss(3,X), possH(h0_2,Y), Y!=X.
possH(h0_2,X)       :- poss(3,X), not block(h0_2,3,X).
possH(h0_3,X)       :- possH(h0_2,X).
block(h0_3,12,X)    :- poss(12,X), possH(h0_3,Y), Y!=X.
possH(h0_3,X)       :- poss(12,X), not block(h0_3,12,X).
poss(0,X)           :- possH(h0_3,X).
% --- Node: 1 ---
possH(h1_1,X)       :- possH(h1_0,X).
block(h1_1,2,X)     :- poss(2,X), possH(h1_1,Y), Y!=X.
possH(h1_1,X)       :- poss(2,X), not block(h1_1,2,X).
possH(h1_2,X)       :- possH(h1_1,X).
block(h1_2,0,X)     :- poss(0,X), possH(h1_2,Y), Y!=X.
possH(h1_2,X)       :- poss(0,X), not block(h1_2,0,X).
possH(h1_3,X)       :- possH(h1_2,X).
block(h1_3,5,X)     :- poss(5,X), possH(h1_3,Y), Y!=X.
possH(h1_3,X)       :- poss(5,X), not block(h1_3,5,X).
possH(h1_4,X)       :- possH(h1_3,X).
block(h1_4,13,X)    :- poss(13,X), possH(h1_4,Y), Y!=X.
possH(h1_4,X)       :- poss(13,X), not block(h1_4,13,X).
poss(1,X)           :- possH(h1_4,X).
% --- Node: 2 ---
. . . . . .

% --- Node: 13 ---
poss(13,X)          :- possH(h13_0,X).
% --- Node: 14 ---
poss(14,X)          :- possH(h14_0,X).
% --- Node: 15 ---
poss(15,X)          :- possH(h15_0,X).
```

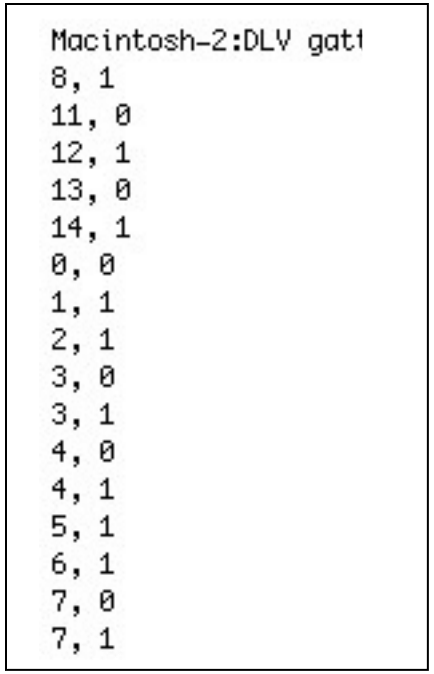
query.txt

```
poss(X,U) ?
```

Executing program

```
./dlv.bin – brave
    input.txt. query-.txt
```

Result



```
Macintosh-2:DLV gat
8, 1
11, 0
12, 1
13, 0
14, 1
0, 0
1, 1
2, 1
3, 0
3, 1
4, 0
4, 1
5, 1
6, 1
7, 0
7, 1
```

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Agenda

1. Stable solutions
   – how to define a unique and consistent solution?

2. Resolution algorithm
   – how to calculate the solution efficiently?

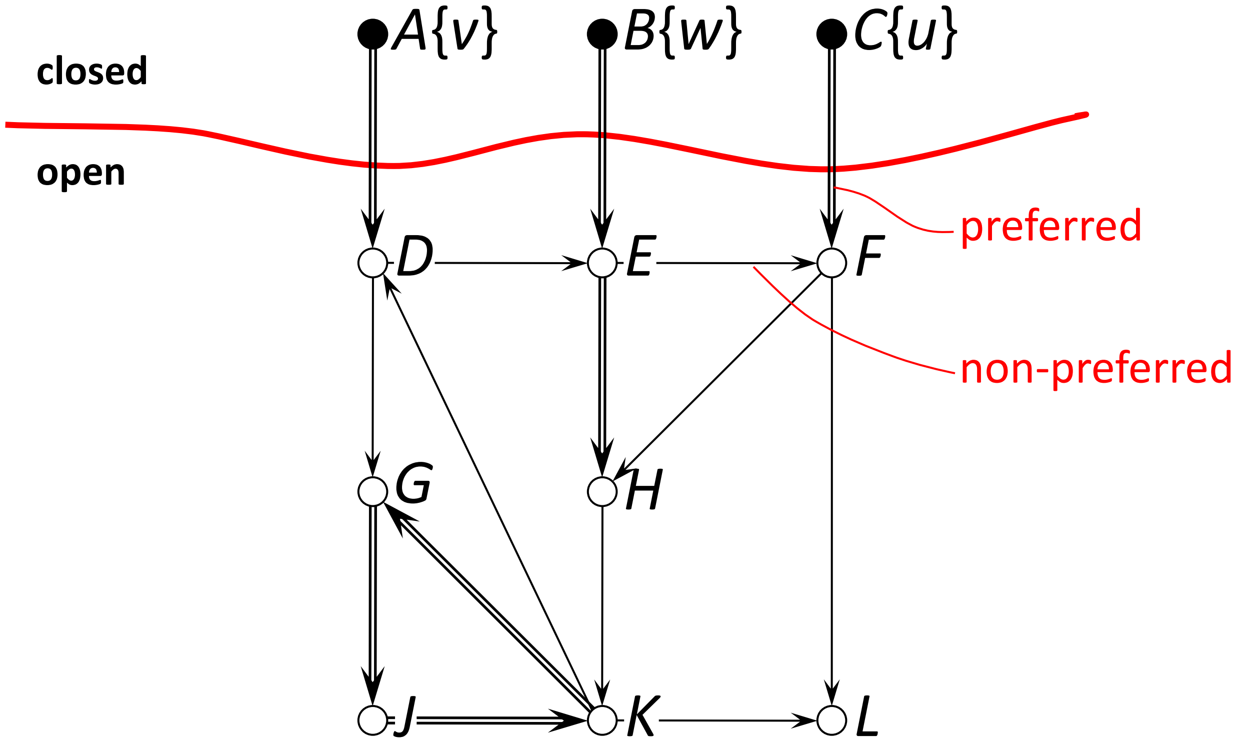
3. Extensions
   – how to deal with “negative beliefs”?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm

Focus on binary trust network

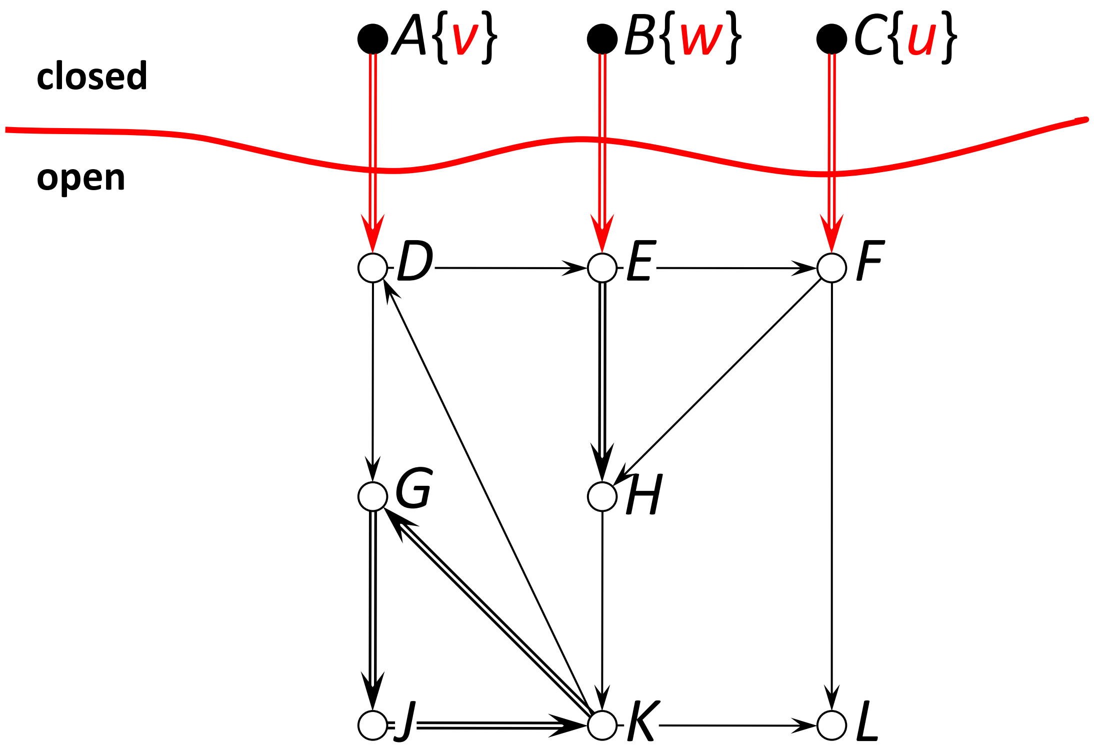
- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs



X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$?$	$?$
E	$?$	$?$
F	$?$	$?$
G	$?$	$?$
H	$?$	$?$
J	$?$	$?$
K	$?$	$?$
L	$?$	$?$

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm

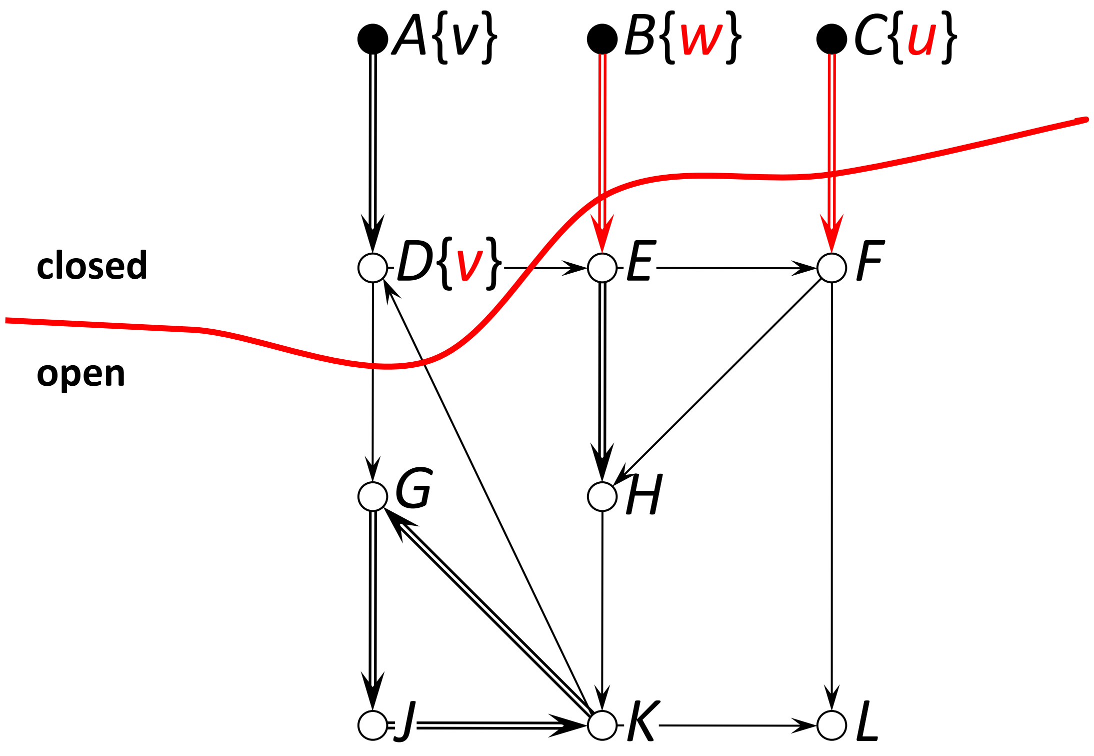


- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	?	?
E	?	?
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm

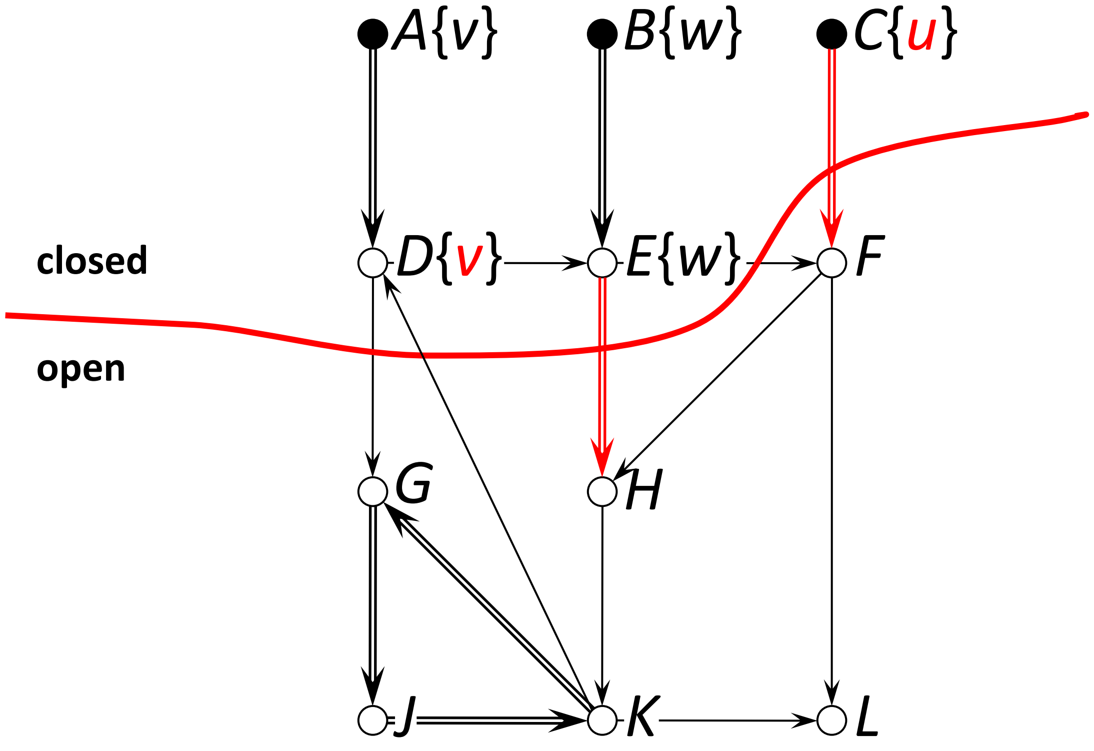


- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from **open** to **closed** → follow

X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	?	?
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm



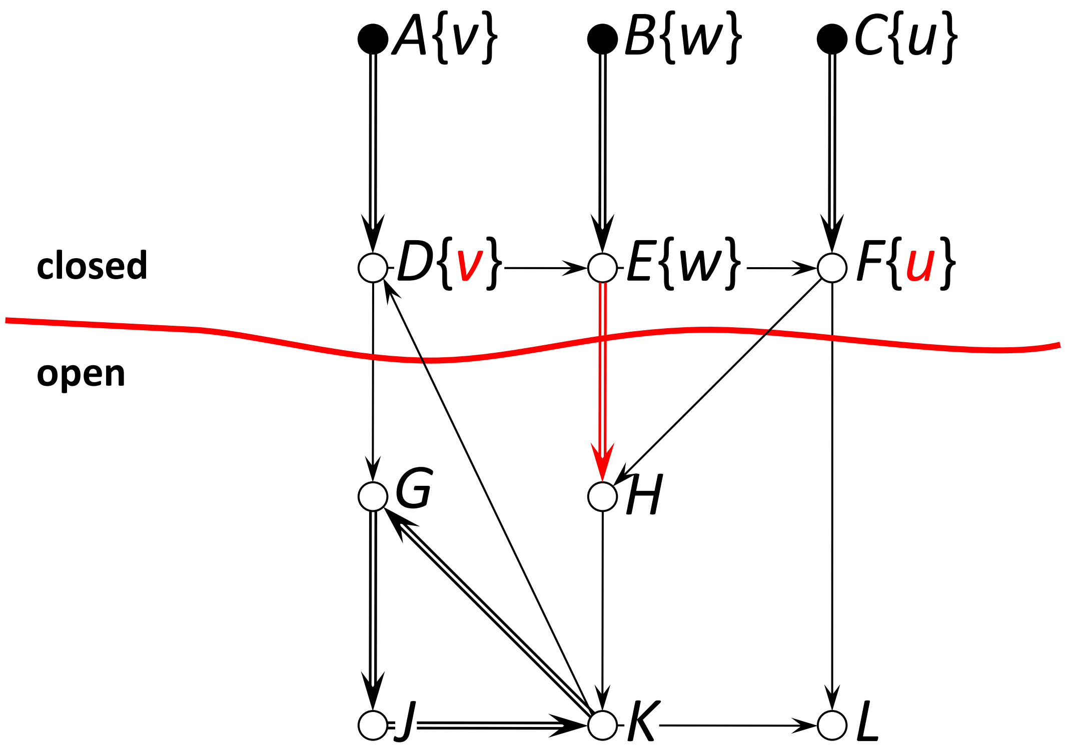
- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from **open** to **closed**
  → follow

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm

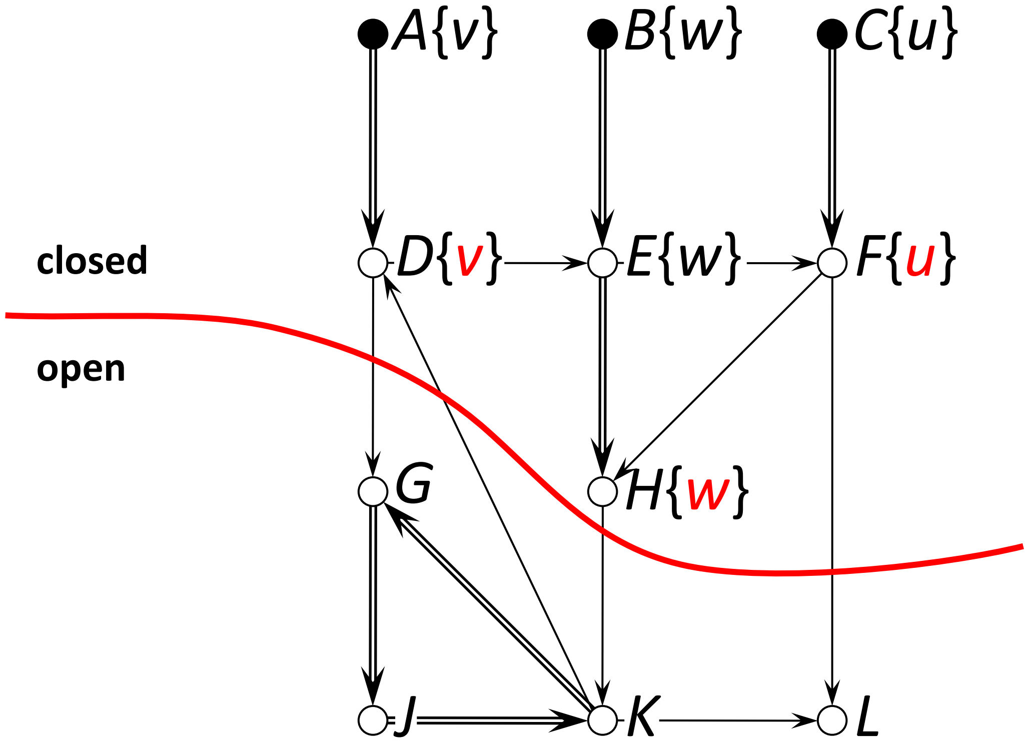
- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow



X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	{w}	{w}
F	{u}	{u}
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm



Now we are stuck!

- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow

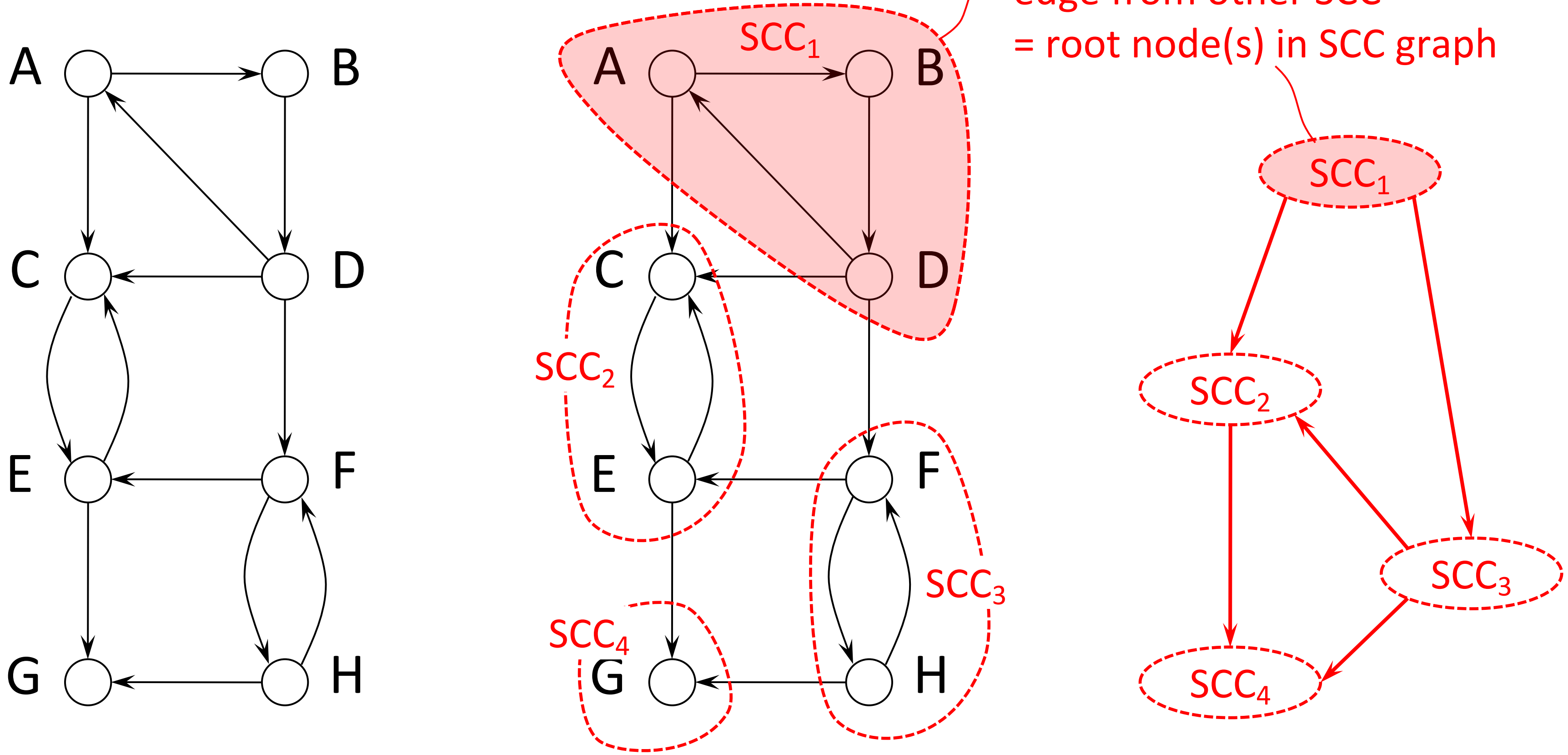
X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	$\{u\}$	$\{u\}$
G	?	?
H	$\{w\}$	$\{w\}$
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Detail: Strongly Connected Components (SCCs)

For every cyclic or acyclic directed graph:

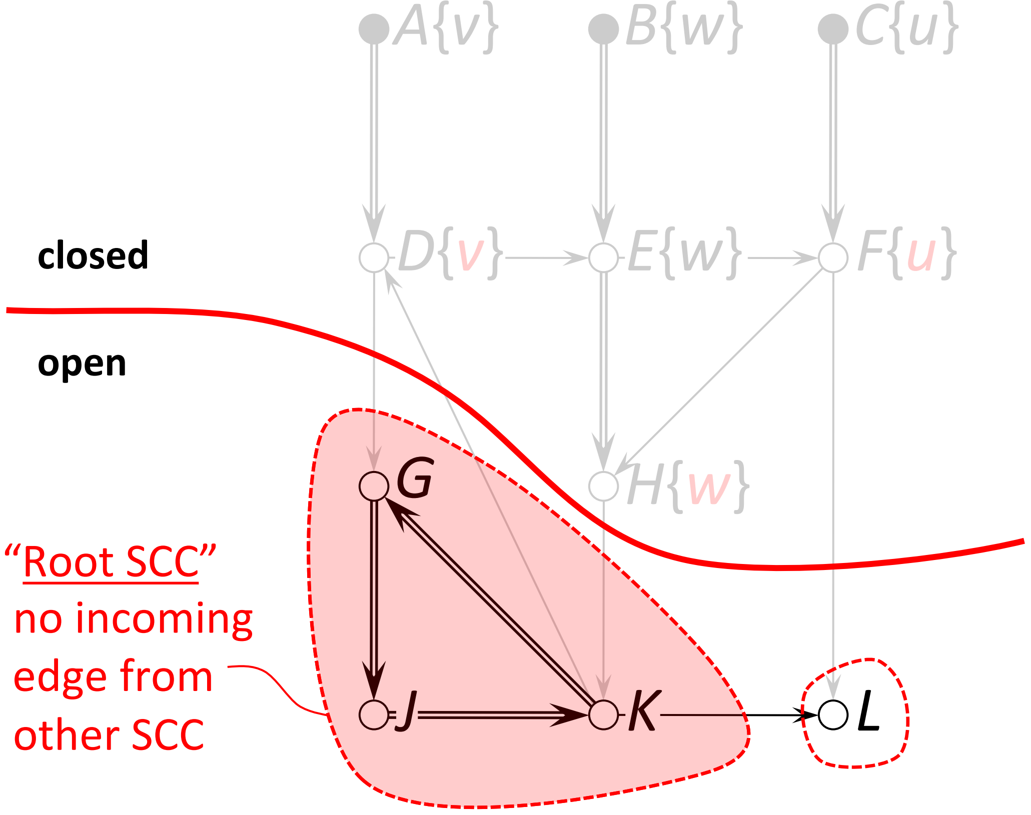
- The Strongly Connected Components graph is a DAG
- can be calculated in **O(*n*)** Tarjan [1972]



"Minimal SCCs": no incoming edge from other SCC
= root node(s) in SCC graph

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm



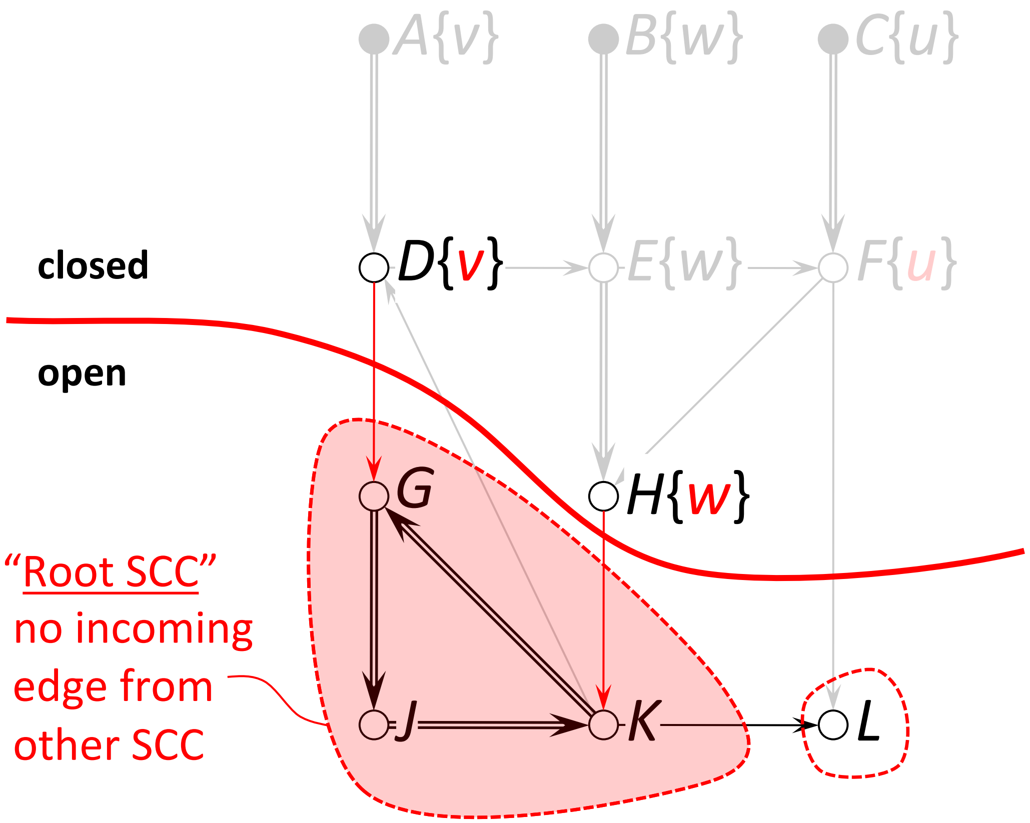
“Root SCC”
no incoming
edge from
other SCC

- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**

X	**poss**(*X*)	**cert**(*X*)
A	{*v*}	{*v*}
B	{*w*}	{*w*}
C	{*u*}	{*u*}
D	{*v*}	{*v*}
E	{*w*}	{*w*}
F	{*u*}	{*u*}
G	?	?
H	{*w*}	{*w*}
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm



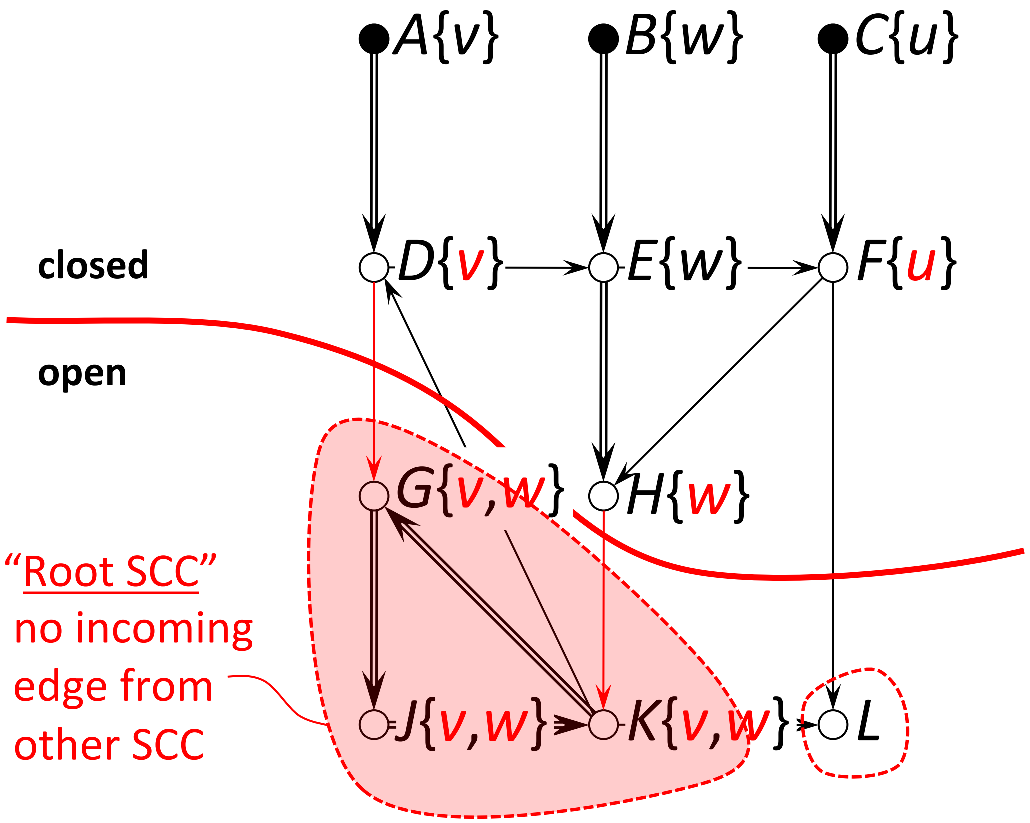
"Root SCC" no incoming edge from other SCC

- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from **open** to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	$\{u\}$	$\{u\}$
G	?	?
H	$\{w\}$	$\{w\}$
J	?	?
K	?	?
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm

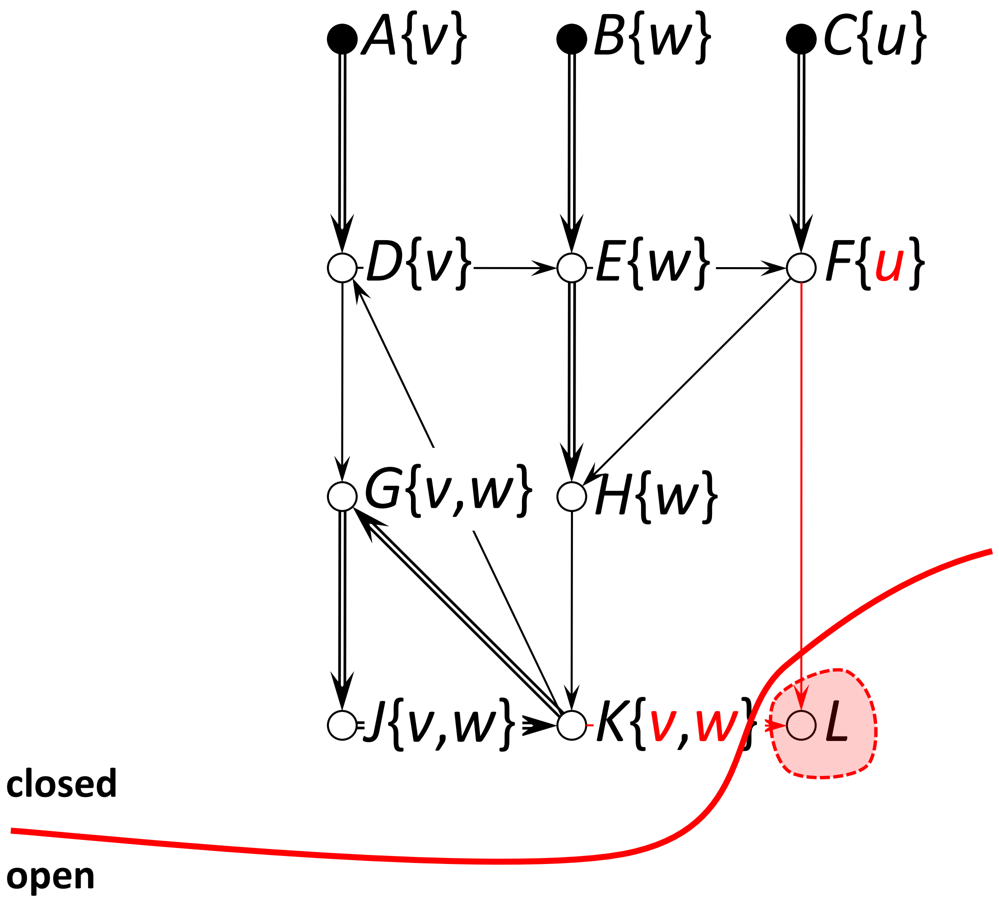


- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  - Step 1: if ∃ preferred edges from **open** to **closed**
    → follow
  - Step 2: else
    → construct SCC graph of **open**
    → resolve minimum SCCs

X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	{w}	{w}
F	{u}	{u}
G	{v,w}	∅
H	{w}	{w}
J	{v,w}	∅
K	{v,w}	∅
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm

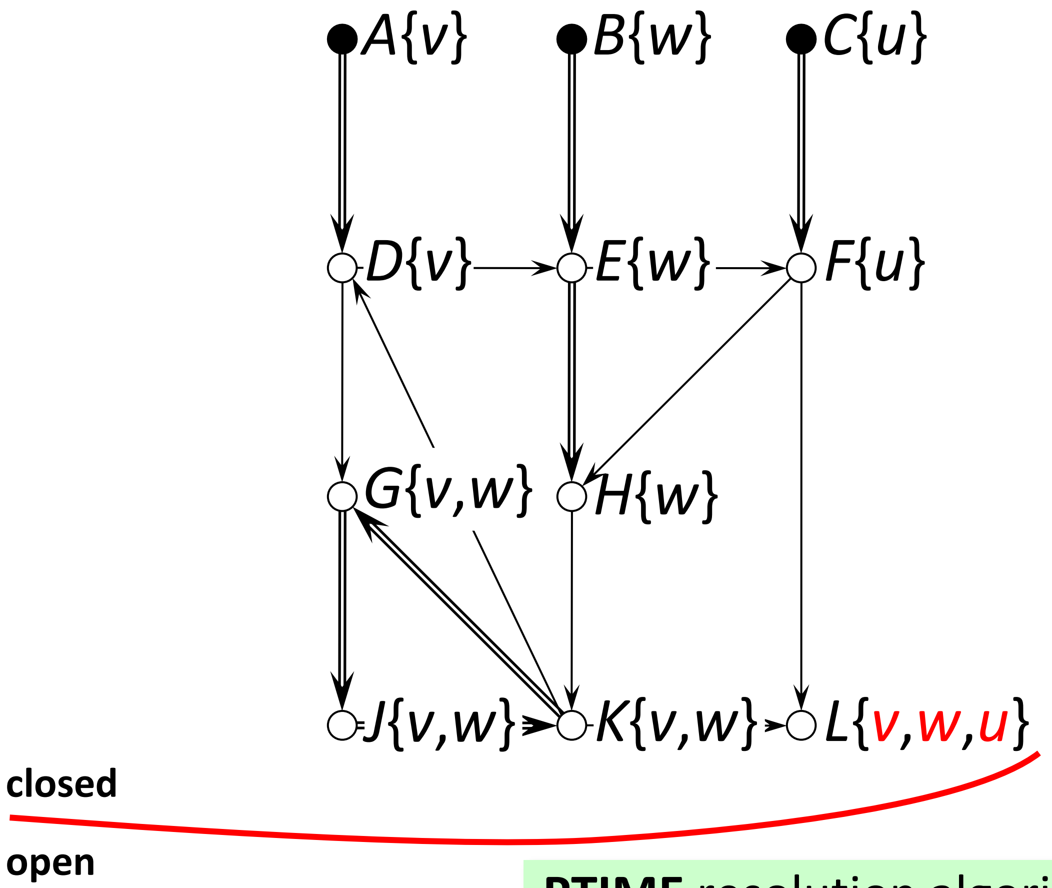


- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**
  → resolve minimum SCCs

X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	{w}	{w}
F	{u}	{u}
G	{v,w}	$\varnothing$
H	{w}	{w}
J	{v,w}	$\varnothing$
K	{v,w}	$\varnothing$
L	?	?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  - Step 1: if ∃ preferred edges from **open** to **closed**
    → follow
  - Step 2: else
    → construct SCC graph of **open**
    → resolve minimum SCCs

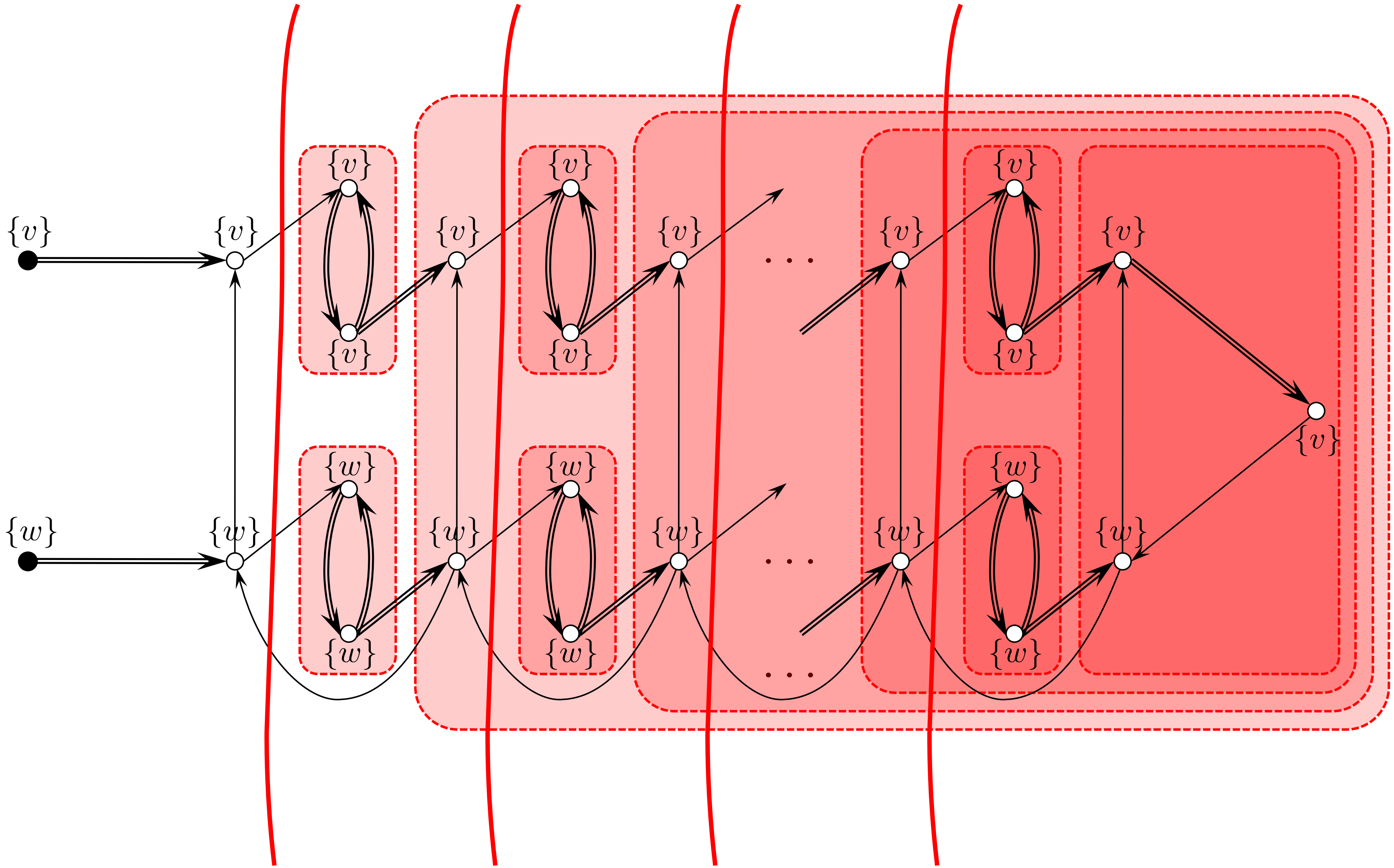
X	**poss**(*X*)	**cert**(*X*)
A	{*v*}	{*v*}
B	{*w*}	{*w*}
C	{*u*}	{*u*}
D	{*v*}	{*v*}
E	{*w*}	{*w*}
F	{*u*}	{*u*}
G	{*v*,*w*}	∅
H	{*w*}	{*w*}
J	{*v*,*w*}	∅
K	{*v*,*w*}	∅
L	{*v*,*w*,*u*}	∅

Can be implemented in current DBMS with transitive closure ☺

PTIME resolution algorithm
O(n^2) worst case
O(n) on reasonable graphs

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

O(n2)-worst-case for Resolution Algorithm



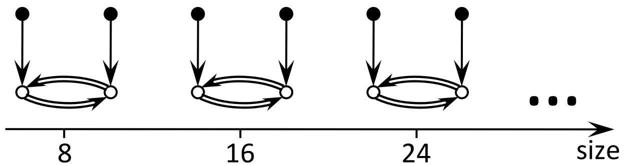
Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Experiments on large network data

Calculating **poss** / **cert** for fixed key

- **DLV**: State-of-the art logic programming solver
- **RA**: Resolution algorithm

Network 1: “Oscillators”

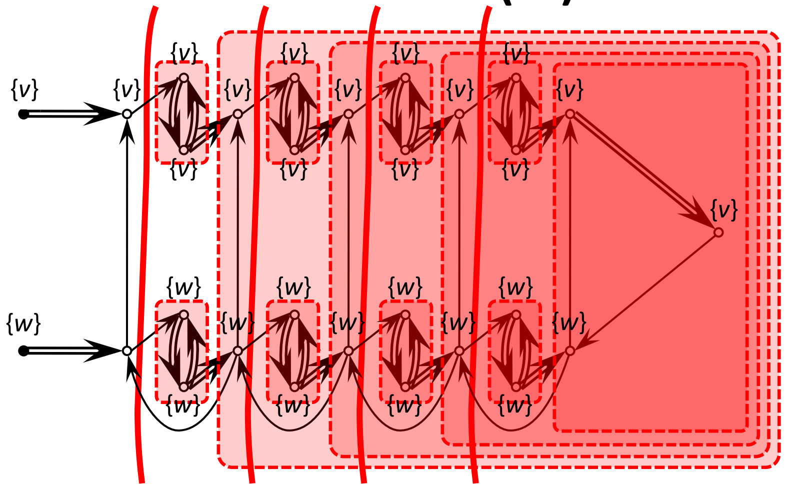


Network 2: “Web link data”

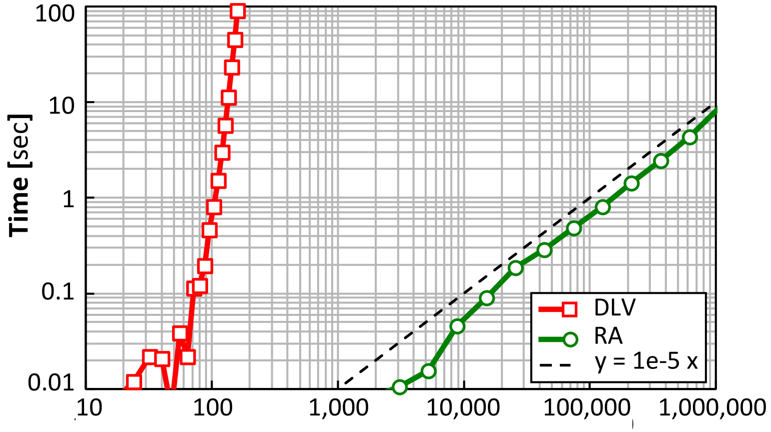
Web data set with 5.4m links between 270k domain names. Approach:

- Sample links with increasing ratio
- Include both nodes in sample
- Assign explicit beliefs randomly

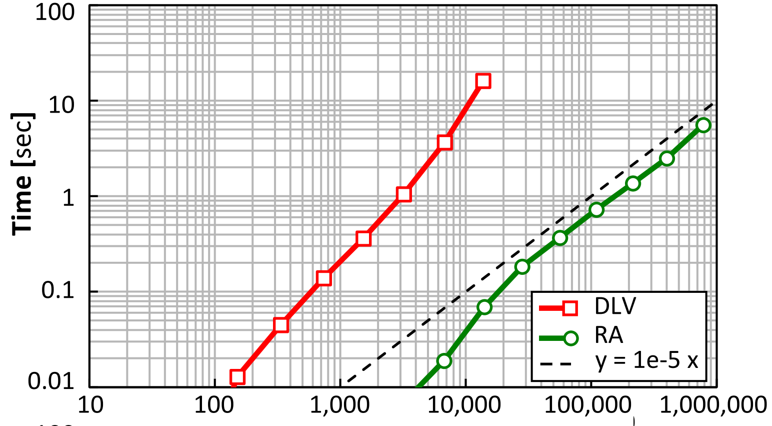
Network 3: “Worst case” **O(n^2)**



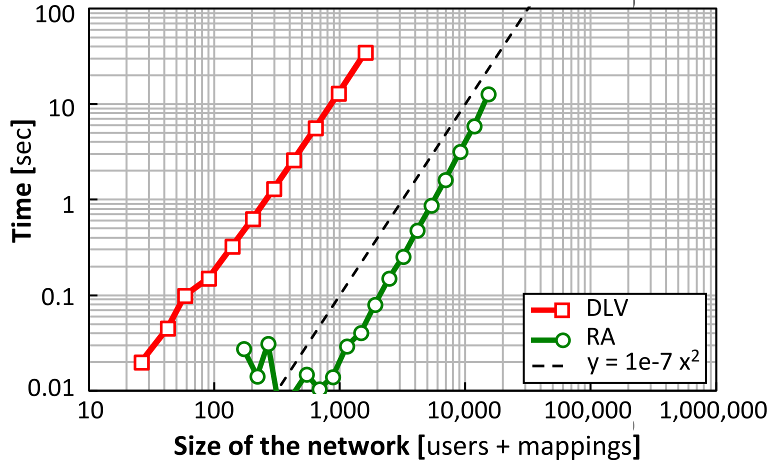
1



2



3



Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010

Agenda

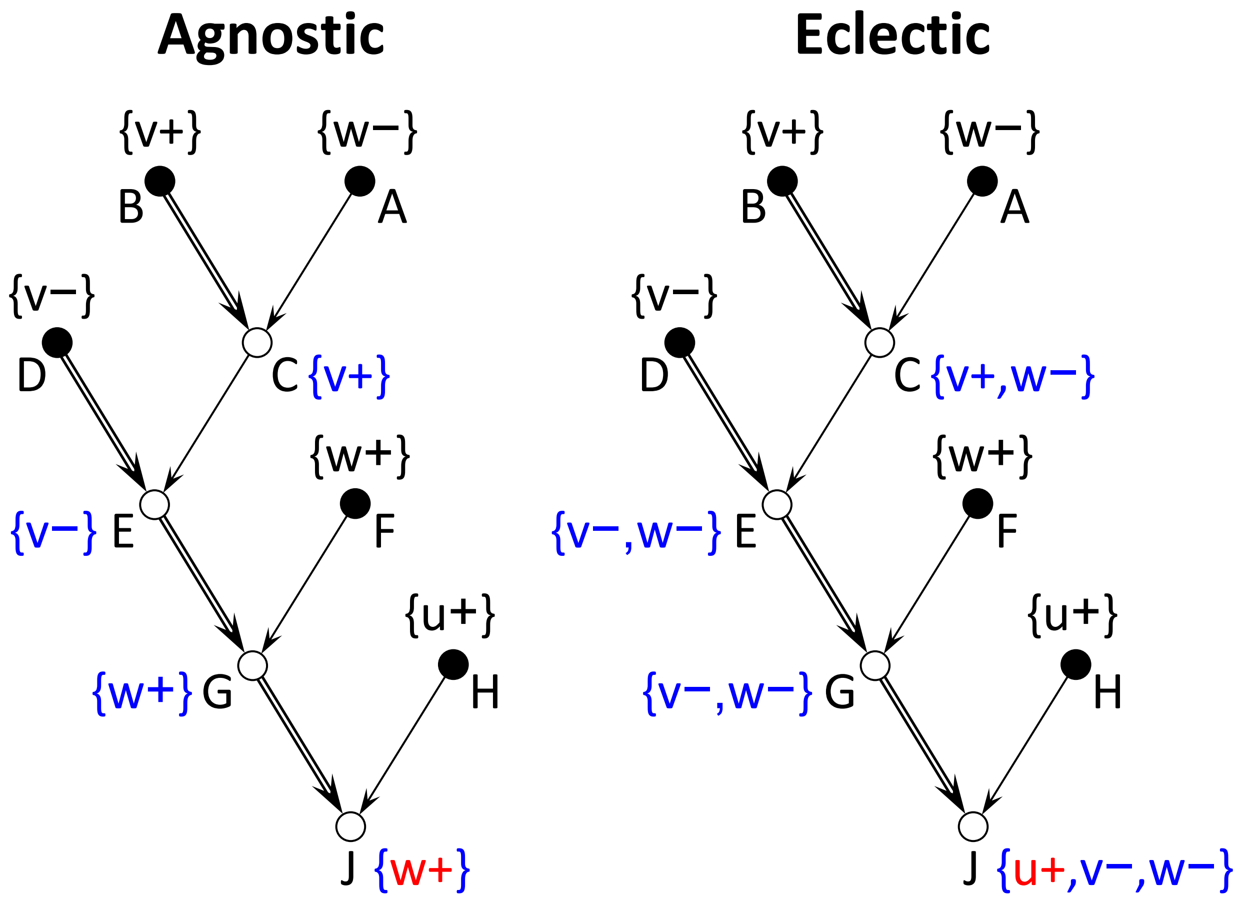
1. Stable solutions
   – how to define a unique and consistent solution?

2. Resolution algorithm
   – how to calculate the solution efficiently?

3. Extensions
   – how to deal with “negative beliefs”?

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

3 semantics for negative beliefs



w/o cycles*	**O(*n*)**	**O(*n*)**
w cycles	**NP-hard**	**NP-hard**

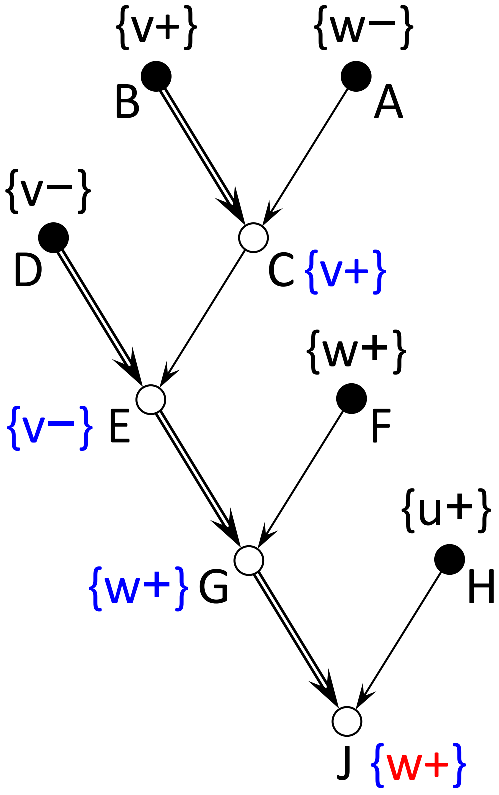
* assuming total order on parents for each node

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

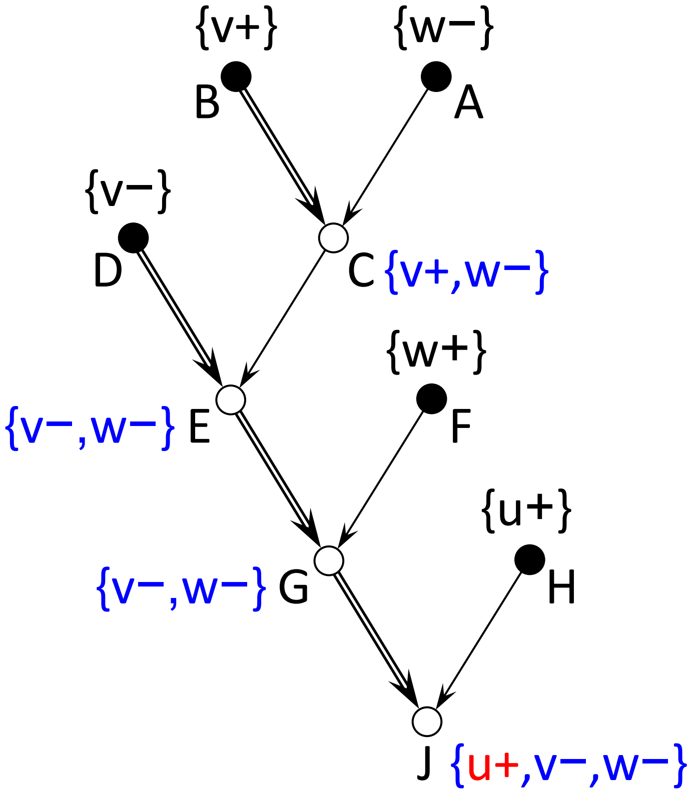
3 semantics for negative beliefs

Our recommendation

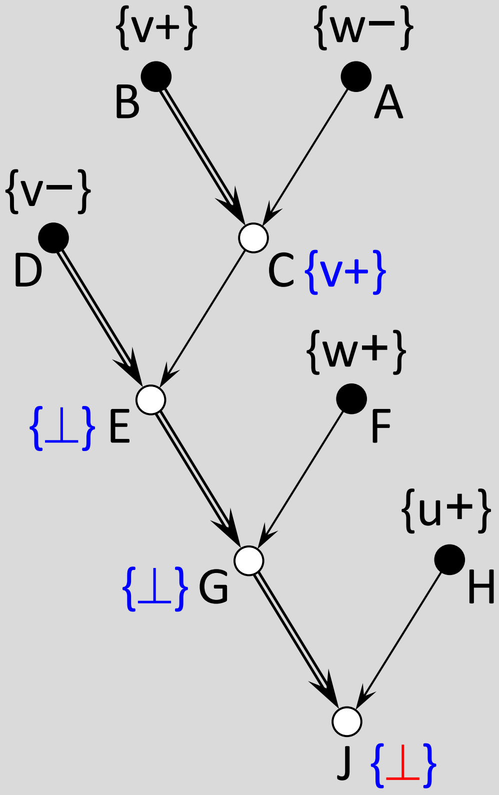
Agnostic



Eclectic



Skeptic



w/o cycles*	**O(n)**	**O(n)**	**O(n)**
w cycles	**NP-hard**	**NP-hard**	**O(n^2)**

with a variation of resolution algorithm

* assuming total order on parents for each node

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Take-aways automatic conflict resolution

Problem

- Given explicit beliefs & trust mappings, how to assign consistent value assignment to users?

Our solution

- Stable solutions with possible/certain value semantics
- PTIME algorithm [**O(n^2)** worst case, **O(n)** experiments]
- Several extensions
  - negative beliefs: 3 semantics, two hard, one **O(n^2)**
  - bulk inserts
  - agreement checking
  - consensus value
  - lineage computation

in the paper & TR

Please visit us at the poster session Th, 3:30pm

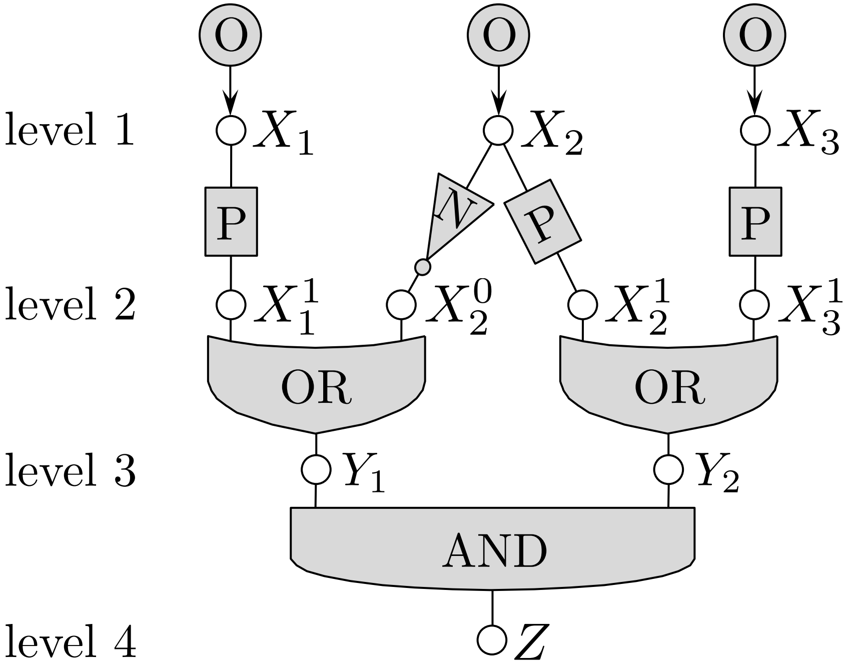
or at: https://db.cs.washington.edu/projects/beliefdb/

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

some details

Fig_ComplexityExampleLong

8-17-2010



Encoding

(0/1) = (a+/b+)

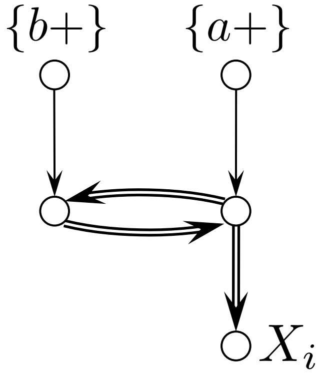
(0/1) = (c+/d+)

(0/1) = (e+/d+)

(0/1) = (e+/f+)

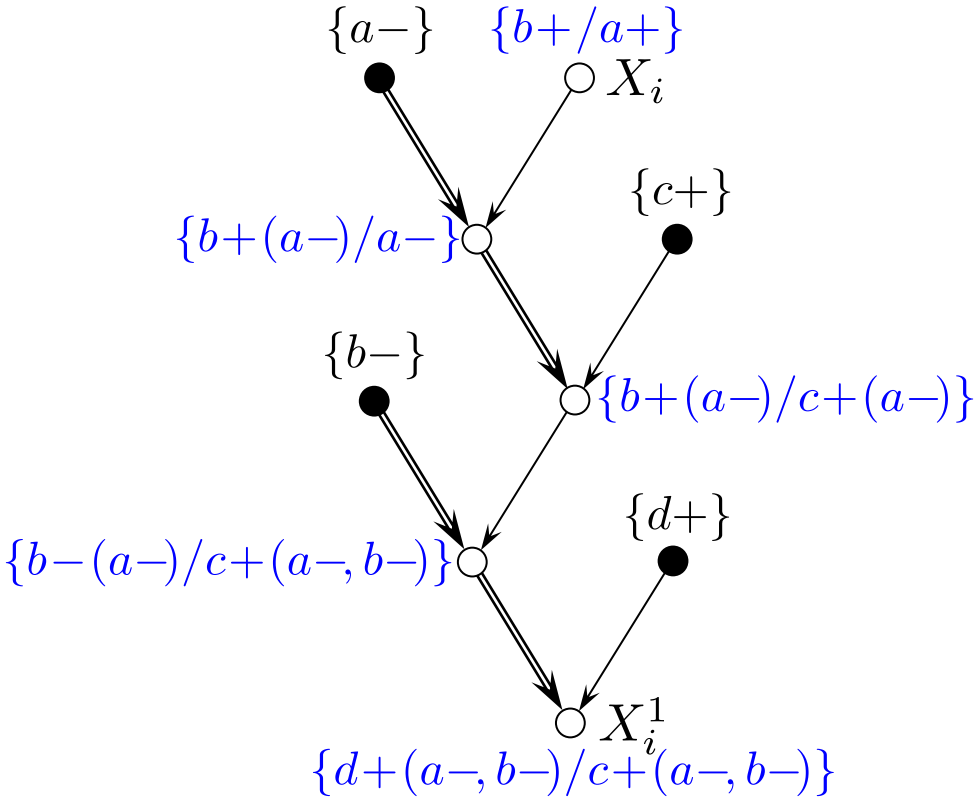
Fig_ComplexityOscillator

8-16-2010



Fig_ComplexityPassLong

8-17-2010



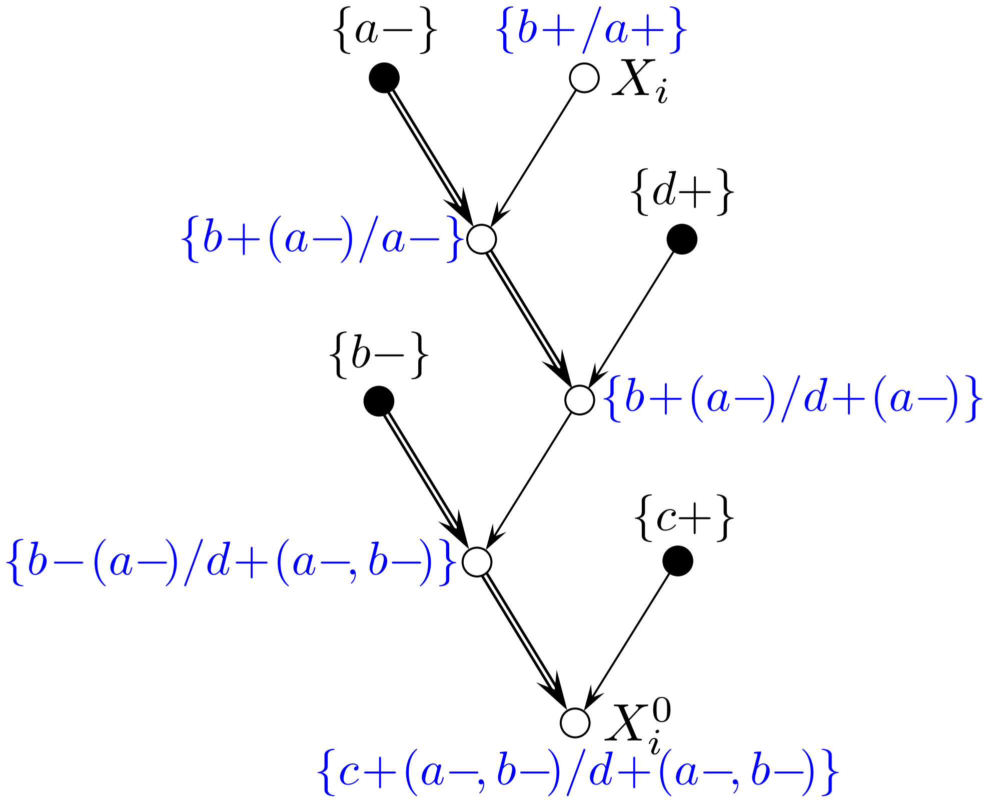
Encoding

(0/1) = (a+/b+)

(0/1) = (c+/d+)

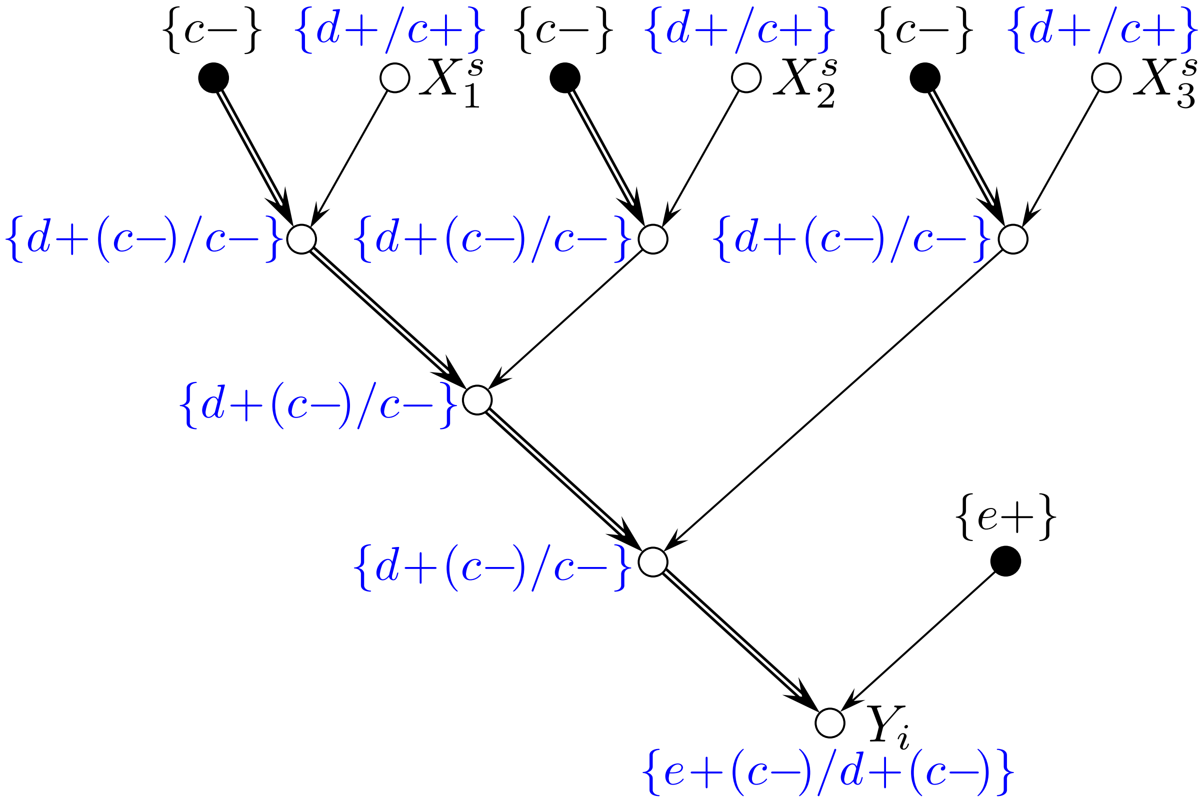
Fig_ComplexityNotLong

8-17-2010



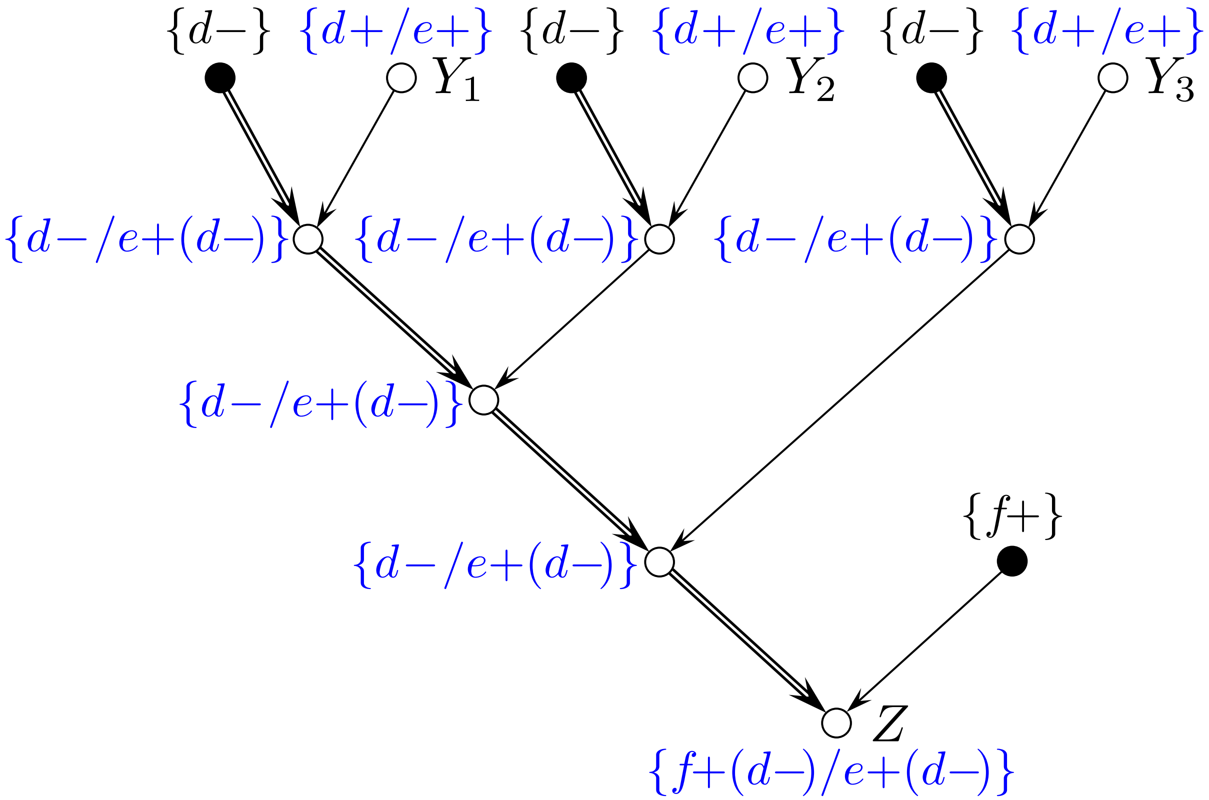
Fig_ComplexityOrLong

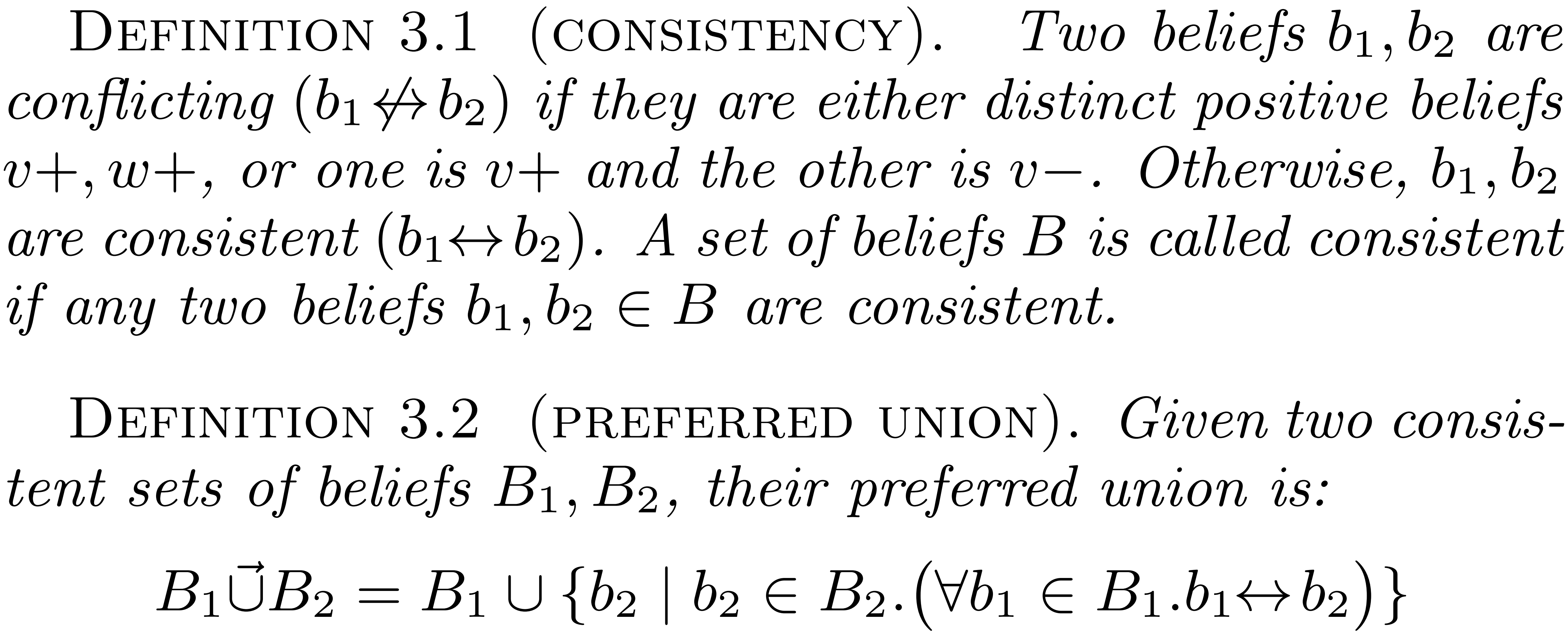
8-17-2010



Fig_ComplexityAndLong

8-17-2010





Definition 3.1 (consistency). *Two beliefs* b_1, b_2 *are conflicting* ($b_1 \not\leftrightarrow b_2$) *if they are either distinct positive beliefs* $v+, w+$, *or one is* $v+$ *and the other is* $v-$. *Otherwise,* b_1, b_2 *are consistent* ($b_1 \leftrightarrow b_2$). *A set of beliefs* B *is called consistent if any two beliefs* $b_1, b_2 \in B$ *are consistent.*

Definition 3.2 (preferred union). *Given two consistent sets of beliefs* B_1, B_2, *their preferred union is:*

$$B_1 \vec{\cup} B_2 = B_1 \cup \{b_2 \mid b_2 \in B_2 . \big(\forall b_1 \in B_1 . b_1 \leftrightarrow b_2\big)\}$$

be a consistent set of positive and/or negative beliefs. For each paradigm $\sigma \in \{\mathtt{Agnostic}, \mathtt{Eclectic}, \mathtt{Skeptic}\}$ (abbreviated by $\{\mathtt{A}, \mathtt{E}, \mathtt{S}\}$), the *normal form* $Norm_\sigma(B)$ is:

$$Norm_{\mathtt{A}}(B) = \begin{cases} \{v+\} & \text{if } \exists v+ \in B \\ B & \text{otherwise} \end{cases}$$

$$Norm_{\mathtt{E}}(B) = B$$

$$Norm_{\mathtt{S}}(B) = \begin{cases} \{v+\} \cup (\bot - \{v-\}) & \text{if } \exists v+ \in B \\ B & \text{otherwise} \end{cases}$$

The *preferred union specialized to the paradigm* σ is:

$$B_1 \vec{\cup}_\sigma B_2 = Norm_\sigma\big(Norm_\sigma(B_1) \vec{\cup} Norm_\sigma(B_2)\big) \qquad (1)$$

For example:

$$\{a-\} \vec{\cup}_{\mathtt{A}} \{b+\} = \{b+\}$$

$$\{a-\} \vec{\cup}_{\mathtt{E}} \{b+\} = \{b+, a-\}$$

$$\{a-\} \vec{\cup}_{\mathtt{S}} \{b+\} = \{b+, a-, c-, d-, \ldots\}$$

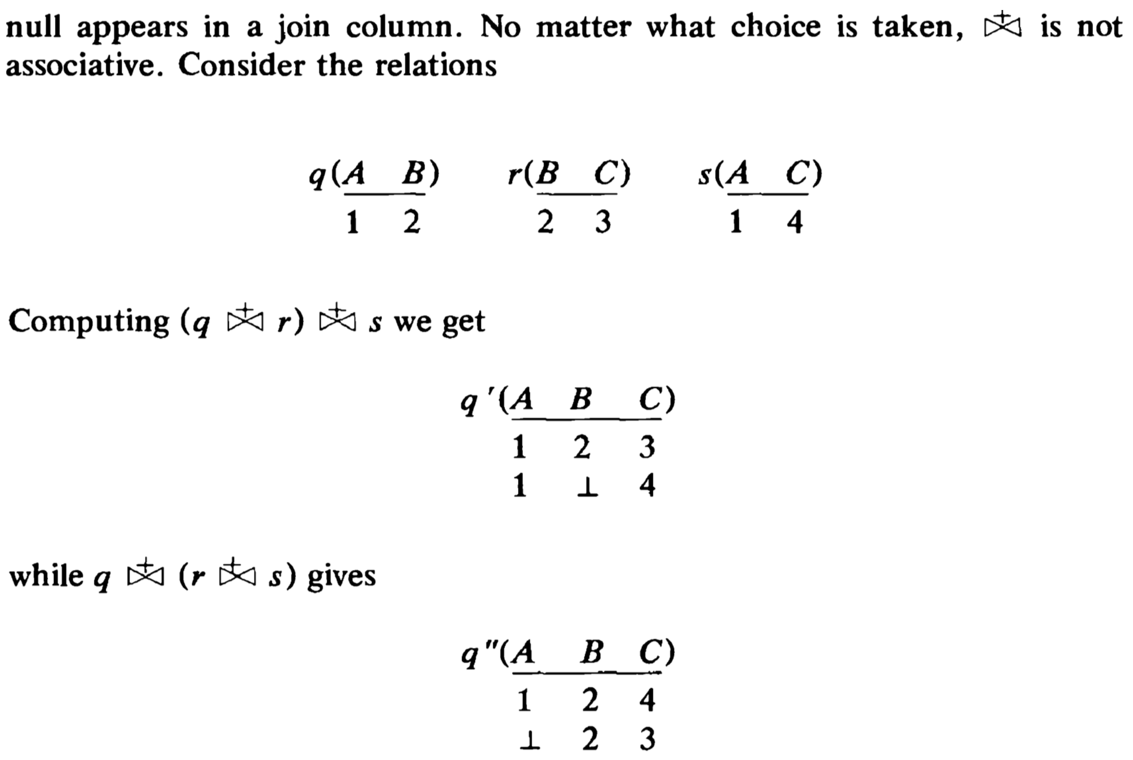
$$\{b-\} \vec{\cup}_{\mathtt{S}} \{b+\} = \bot$$

$a^- \ (a^+ \ b^+)$

$(a^- \ a^+) \ b^+$

A puzzling question is why is the **Skeptic** paradigm in PTIME, while the other two are hard. It is easy to see that the Boolean gates in Fig. 7 no longer work under **Skeptic**, but we do not consider this a satisfactory explanation. While we cannot give an ultimate cause, we point out one interesting difference. The preferred union for **Skeptic** is *associative*, while it is not associative for either **Agnostic** nor **Eclectic**. For example, consider the two expressions $B_1 = \{a-\}\vec{\cup}_\sigma\left(\{a+\}\vec{\cup}_\sigma\{b+\}\right)$, $B_2 = \left(\{a-\}\vec{\cup}_\sigma\{a+\}\right)\vec{\cup}_\sigma\{b+\}$. For **Agnostic**, we have $B_2 = \{b+\}$, for **Eclectic** $B_2 = \{a-, b+\}$, while for both $B_1 = \{a-\}$. By contrast, one can show that $\vec{\cup}_{\mathrm{S}}$ is associative. Associativity as a desirable property during data merging was pointed out in [14].

The issue of associativity



null appears in a join column. No matter what choice is taken, $\overset{+}{\bowtie}$ is not associative. Consider the relations

$q(A$	$B)$
1	2

$r(B$	$C)$
2	3

$s(A$	$C)$
1	4

Computing $(q \overset{+}{\bowtie} r) \overset{+}{\bowtie} s$ we get

$q'(A$	B	$C)$
1	2	3
1	$\perp$	4

while $q \overset{+}{\bowtie} (r \overset{+}{\bowtie} s)$ gives

$q''(A$	B	$C)$
1	2	4
$\perp$	2	3

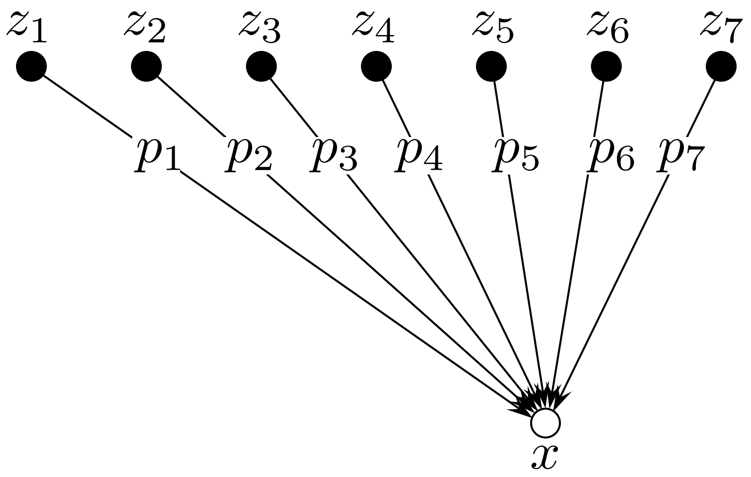
$$\{a^-\}\ \overrightarrow{\cup}_a\ \left(\{a\}\ \overrightarrow{\cup}_a\ \{b\}\right) \qquad = \{a^-\}$$

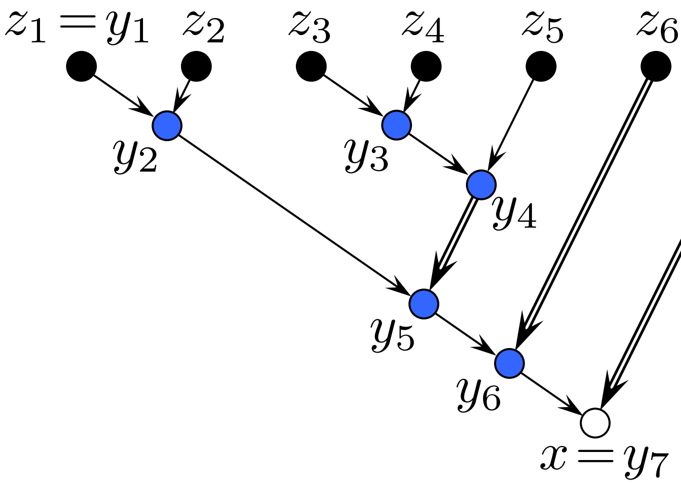
$$\left(\{a^-\}\ \overrightarrow{\cup}_a\ \{a\}\right)\ \overrightarrow{\cup}_a\ \{b\} \qquad = \{b\}$$

Source: left outer join example from p392 in "Maier. The theory of relational databases, 1983." https://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html

Source: right preferred union example from "Gatterbauer, Suciu. Conflict resolution using trust mapping. SIGMOD 2010. https://doi.org/10.1145/1807167.1807193

Binarization example



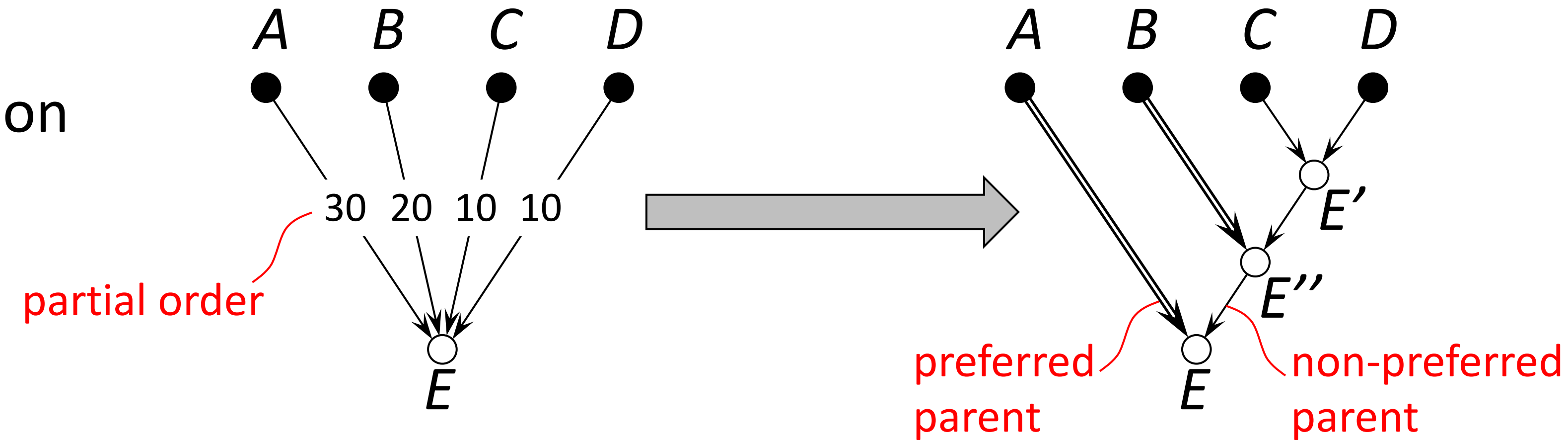


$p_1 = p_2 < p_3 = p_4 = p_5 < p_6 < p_7$

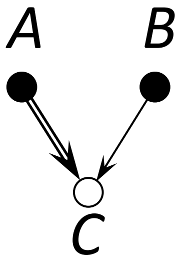
Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Logic programs with stable model semantics

Step 1:
Binarization



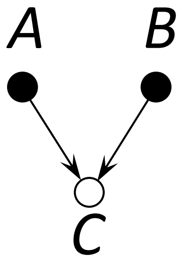
Step 2:
Logic program



1: accept all **poss** of preferred parent

```
   poss(c,X) :- poss(a,X).
block(c,b,Y) :- poss(b,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(b,Y), not block(c,b,Y).
```

2: accept **poss** from non-preferred parent, that are not conflicting with an existing value



```
block(c,a,Y) :- poss(a,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(a,Y), not block(c,a,Y).
block(c,b,Y) :- poss(b,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(b,Y), not block(c,b,Y).
```

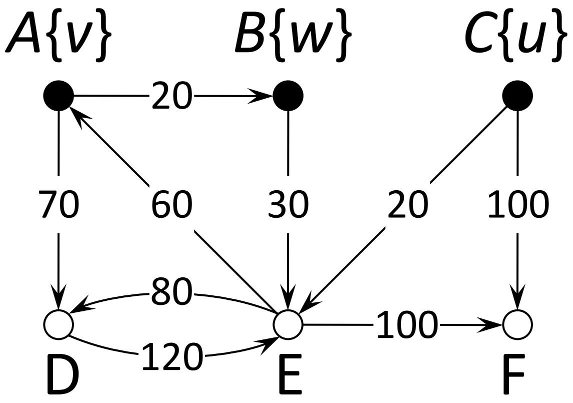
Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

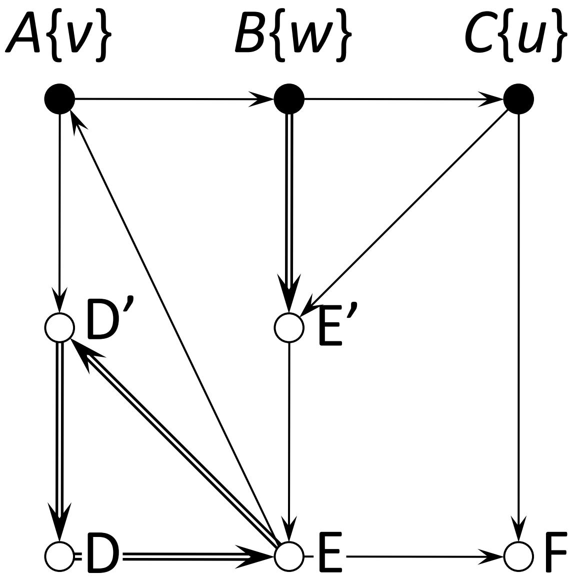
Binarization for Resolution Algorithm*

Example Trust Network (TN)
6 nodes, 9 arcs (size 15)
3 explicit beliefs: A:v, B:w, C:u

Corresponding Binary TN (BTN)
8 nodes, 12 arcs (size 20)

Size increase (N+E): ≤ 3



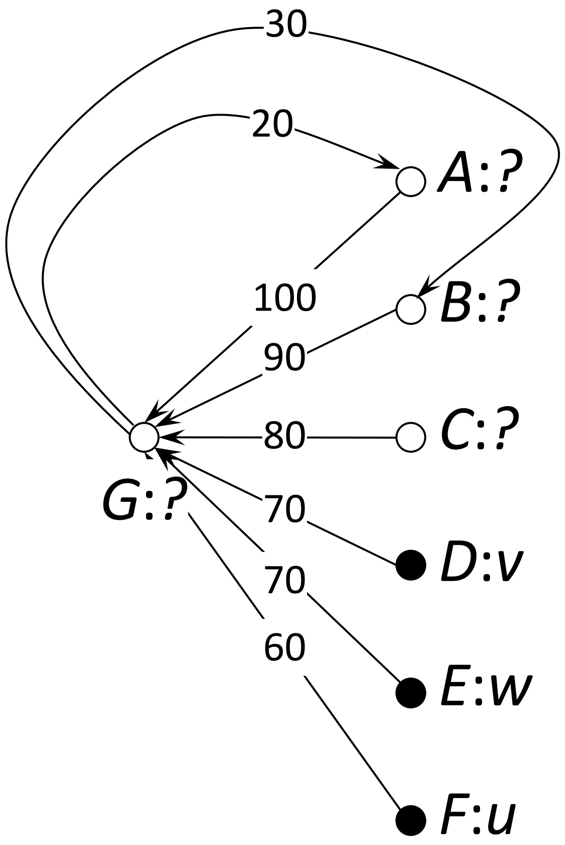


[*] Note that binarization is not necessary, but greatly simplifies the presentation

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"
- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief
- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions

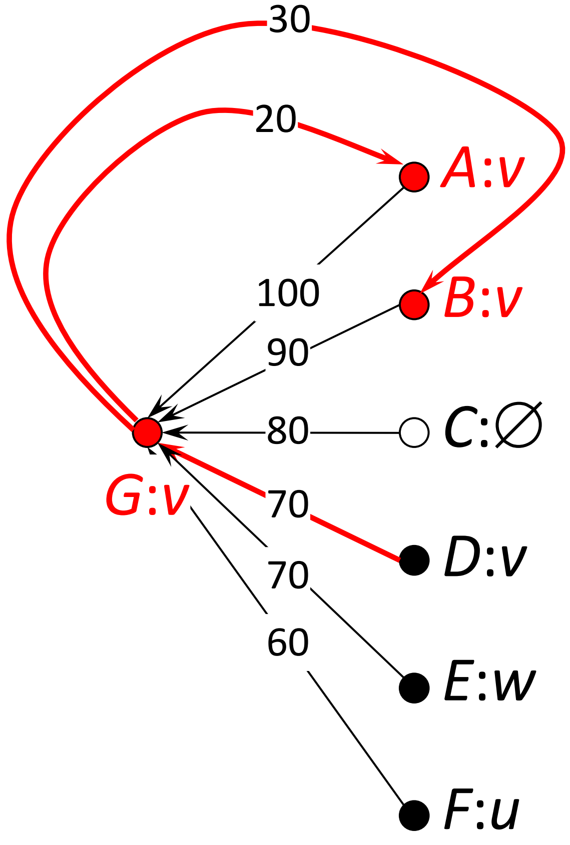


* each node with at least one ancestor with explicit belief

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"
- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief
- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions



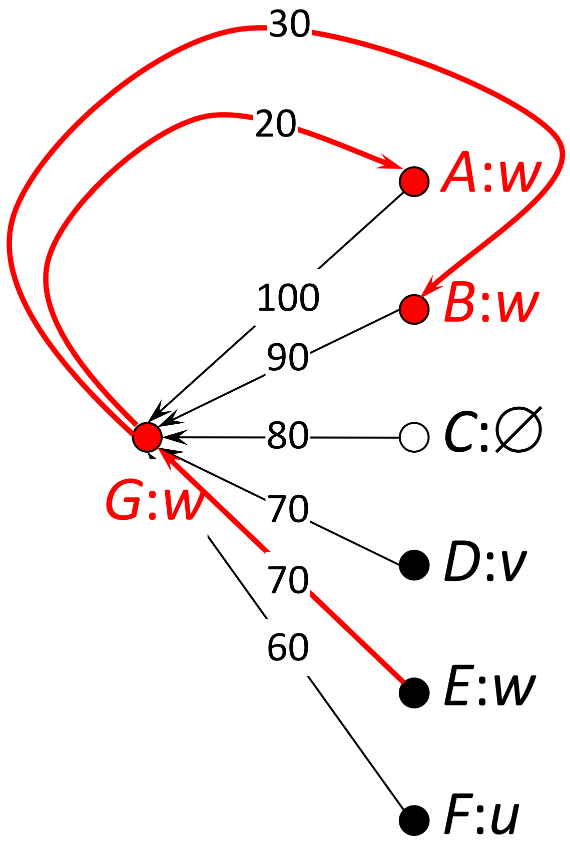
poss(*G*) = {*v*,...}

* each node with at least one ancestor with explicit belief

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"
- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief
- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions



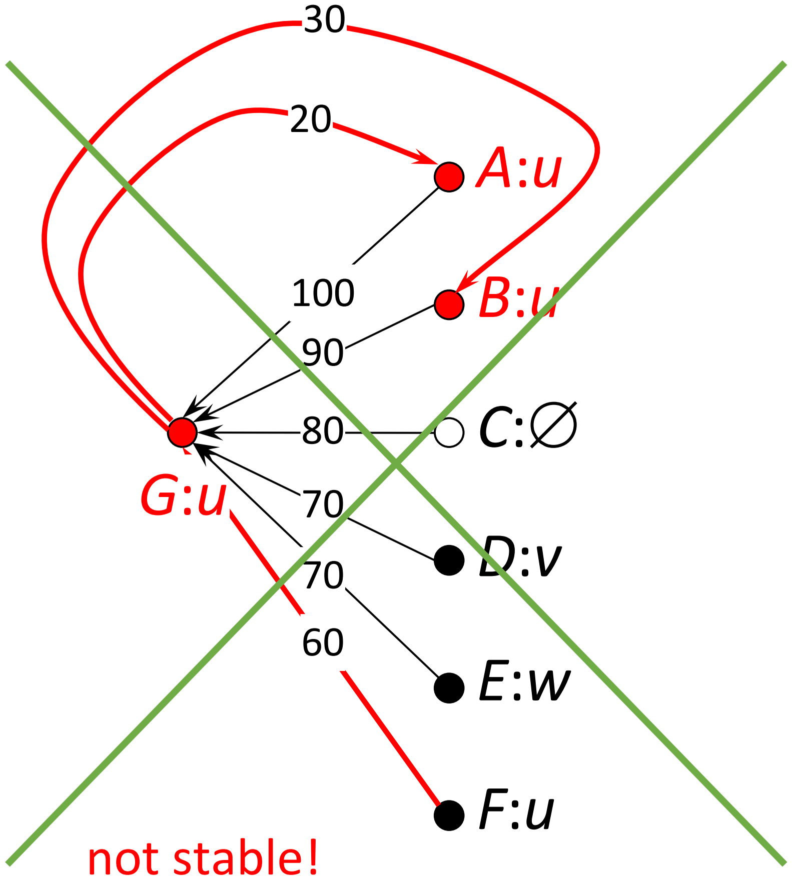
poss(*G*) = {*v*,*w*,...}

* each node with at least one ancestor with explicit belief

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"
- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief
- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions



not stable!
F→G dominated by E→G

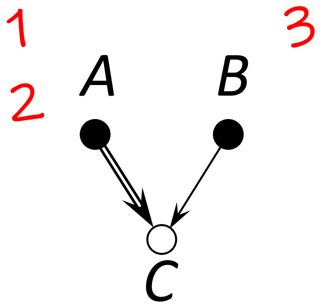
poss(*G*) = {*v*,*w*}
cert(*G*) = ∅

* each node with at least one ancestor with explicit belief

Gatterbauer, Suciu. Data Conflict Resolution Using Trust Mappings, SIGMOD 2010, https://doi.org/10.1145/1807167.1807193

exercise

Logic programs with stable model semantics



```
   poss(c,X) :- poss(a,X).
block(c,b,Y) :- poss(b,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(b,Y), not block(c,b,Y).
```

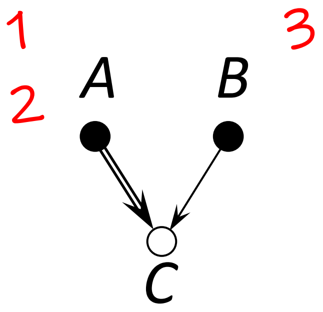
```
poss(a,1).
poss(a,2).
poss(b,3).
```

⇨

poss(c,X) ?

?

Logic programs with stable model semantics



```
    poss(c,1) :- poss(a,1)
    poss(c,2) :- poss(a,2)
    poss(c,3) :- poss(a,3)
block(c,b,3) :- poss(b,3), poss(c,1), X!=Y
block(c,b,3) :- poss(b,3), poss(c,2), X!=Y
block(c,b,3) :- poss(b,1), poss(c,3), X!=Y
                    ...
    poss(c,3) :- poss(b,3), not block(c,b,3)
    poss(c,2) :- poss(b,2), not block(c,b,2)
                    ...
```

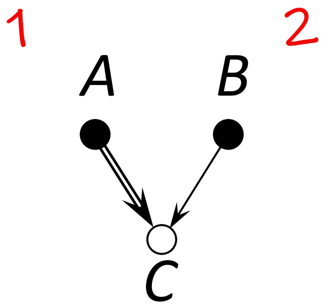
```
   poss(c,X) :- poss(a,X).
block(c,b,Y) :- poss(b,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(b,Y), not block(c,b,Y).
```

```
poss(a,1).
poss(a,2).
poss(b,3).
```

poss(c,X) ?

M={ poss(a,1), poss(a,2), poss(b,3),
poss(c,1), poss(c,2) }

Logic programs with stable model semantics



```
block(c,a,Y) :- poss(a,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(a,Y), not block(c,a,Y).
block(c,b,Y) :- poss(b,Y), poss(c,X), X!=Y.
   poss(c,Y) :- poss(b,Y), not block(c,b,Y).
```

```
poss(a,1).
poss(a,2).
```

⇨

poss(c,X) ?

?

Updated 2/27/2024

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 13

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

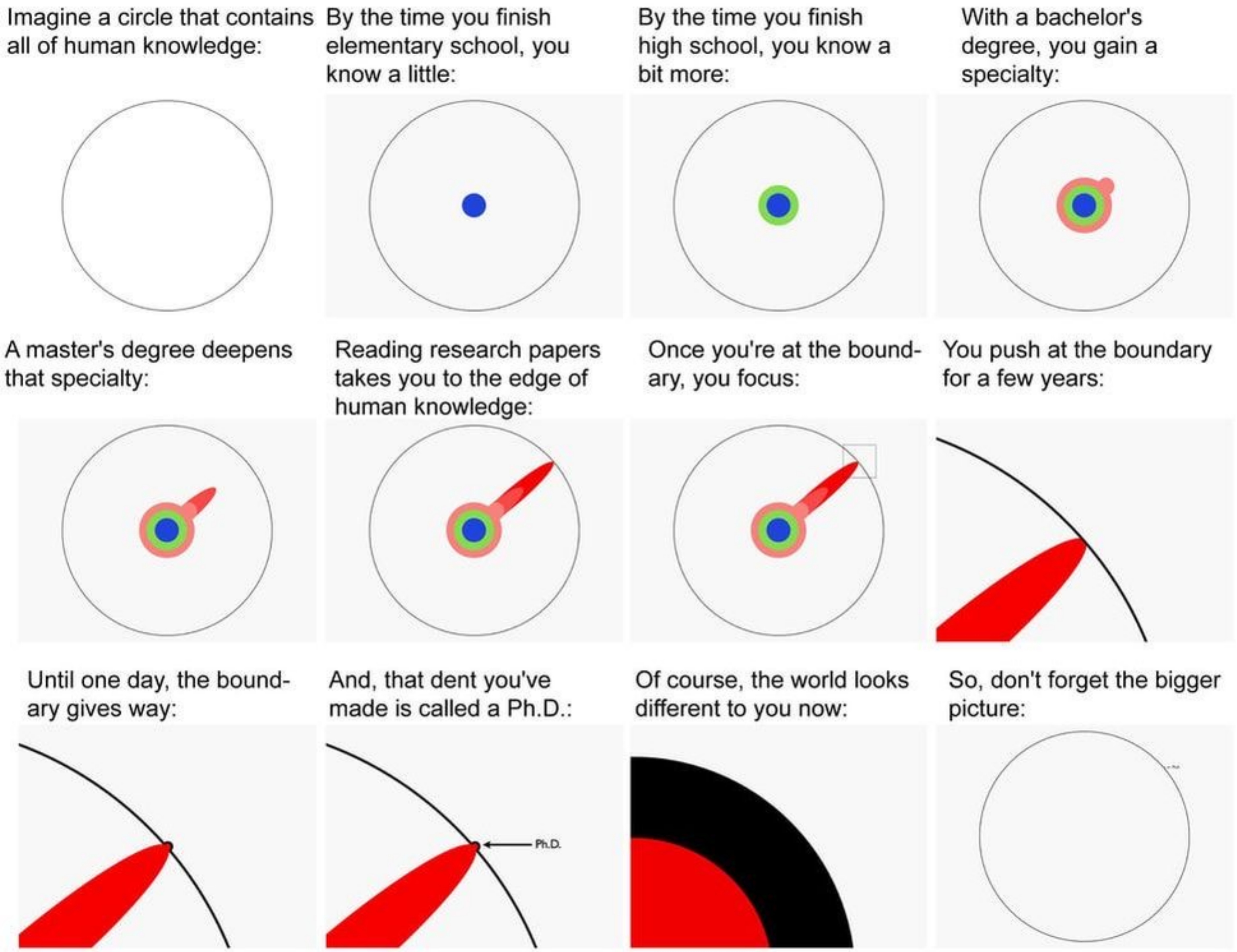
2/27/2024

Pre-class conversations

- Last class summary
- Feedback on Feedback on scribes?
- Project discussions (in class and after or via email and office hours)
- Faculty candidates (THU Feb 29, WED March 20)

- Today:
  - The power of disjunctions: Disjunctive Logic Programs
    (NP and Co-NP in the same program...)

About research (getting a PhD or finding a project topic)



Imagine a circle that contains all of human knowledge:

By the time you finish elementary school, you know a little:

By the time you finish high school, you know a bit more:

With a bachelor's degree, you gain a specialty:

A master's degree deepens that specialty:

Reading research papers takes you to the edge of human knowledge:

Once you're at the boundary, you focus:

You push at the boundary for a few years:

Until one day, the boundary gives way:

And, that dent you've made is called a Ph.D.:

Of course, the world looks different to you now:

So, don't forget the bigger picture:

The last comment: Keep pushing!

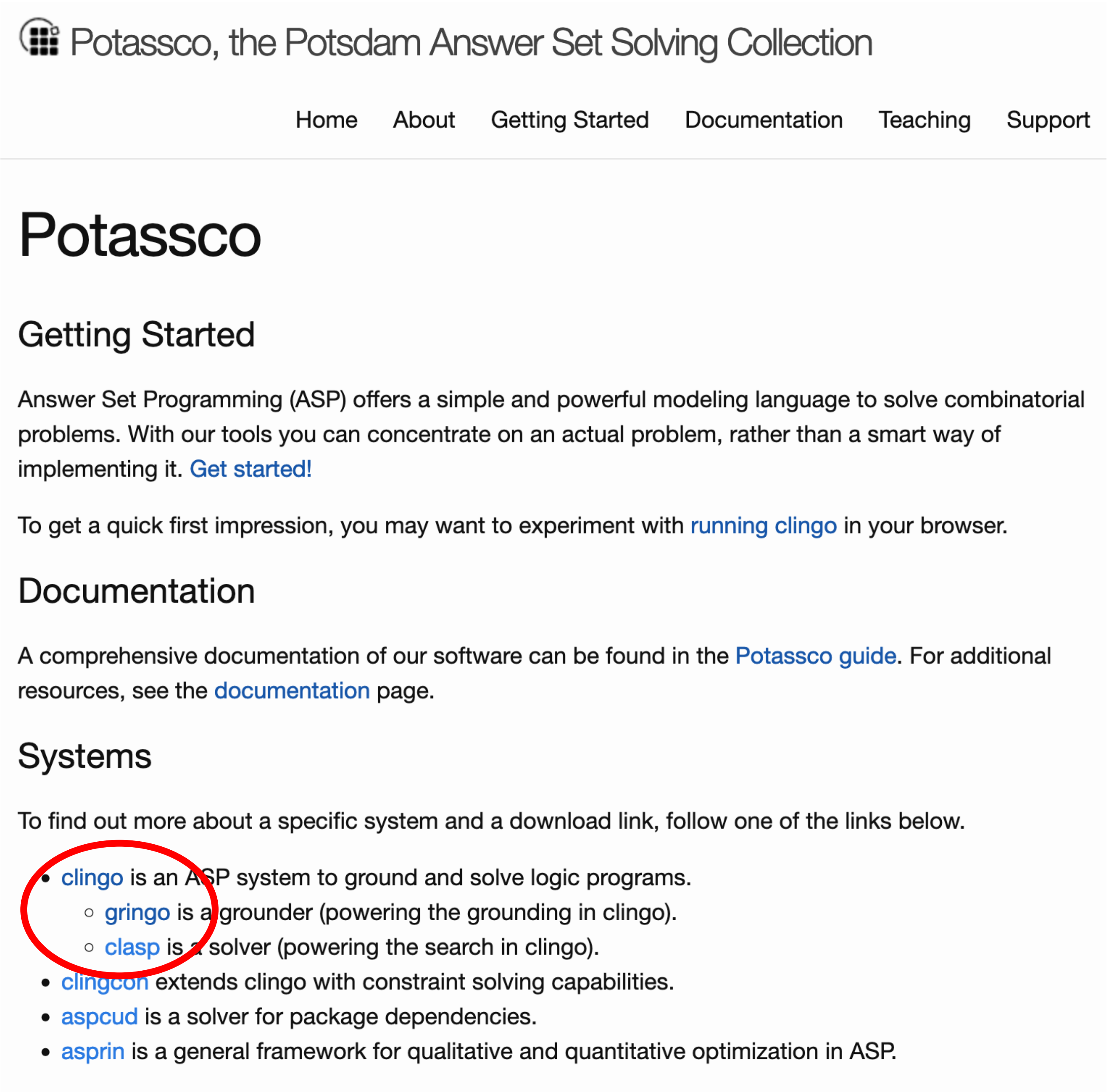
Source: https://matt.might.net/articles/phd-school-in-pictures/

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]

Disjunctive Logic Programming with Clingo/Potassco (Examples prepared together with Neha Makhija https://nehamakhija.github.io/)

Clingo, Potassco



Potassco, the Potsdam Answer Set Solving Collection

Potassco

Getting Started

Answer Set Programming (ASP) offers a simple and powerful modeling language to solve combinatorial problems. With our tools you can concentrate on an actual problem, rather than a smart way of implementing it. Get started!

To get a quick first impression, you may want to experiment with running clingo in your browser.

Documentation

A comprehensive documentation of our software can be found in the Potassco guide. For additional resources, see the documentation page.

Systems

To find out more about a specific system and a download link, follow one of the links below.

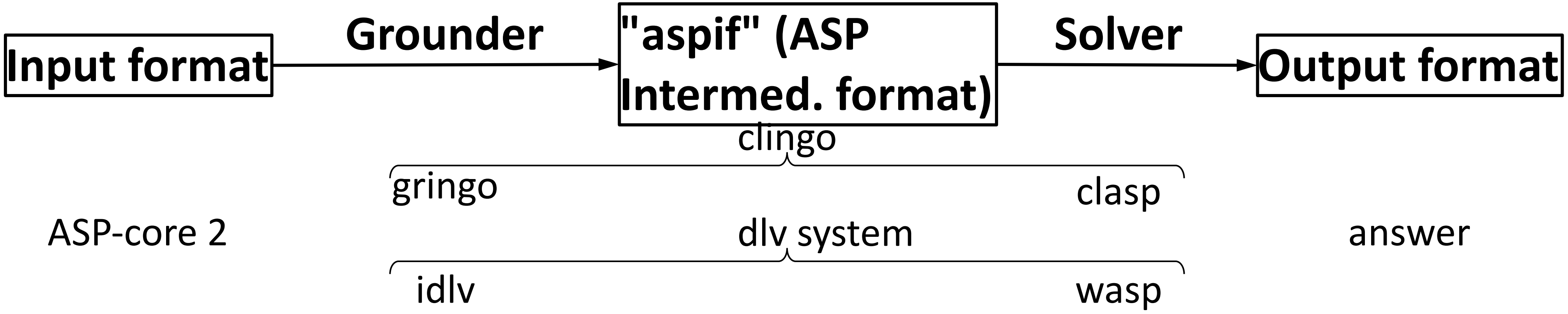
- clingo is an ASP system to ground and solve logic programs.
  - gringo is a grounder (powering the grounding in clingo).
  - clasp is a solver (powering the search in clingo).
- clingcon extends clingo with constraint solving capabilities.
- aspcud is a solver for package dependencies.
- asprin is a general framework for qualitative and quantitative optimization in ASP.

Potassco start page: https://potassco.org/
Clingo start page: https://potassco.org/clingo/
Running clingo in the browser: https://potassco.org/clingo/run/
Teaching material: https://teaching.potassco.org/
Download: https://github.com/potassco/clingo/releases/
clingo user guide: https://github.com/potassco/guide/releases/download/v2.2.0/guide.pdf

Clingo Implementation

clingo is a monolithic system that combines two steps and offers more control than using the two tools individually:

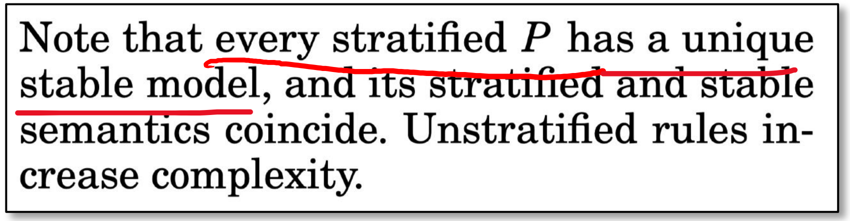
- gringo: a grounder that, given an input program with first-order variables, computes an equivalent ground (variable-free) program
- clasp: a solver that works on ground program (like other answer set solvers)
  - relies on conflict-driven nogood learning, a technique that proved very successful for SAT
  - does not rely on legacy software, such as a SAT solver or any other existing ASP solver



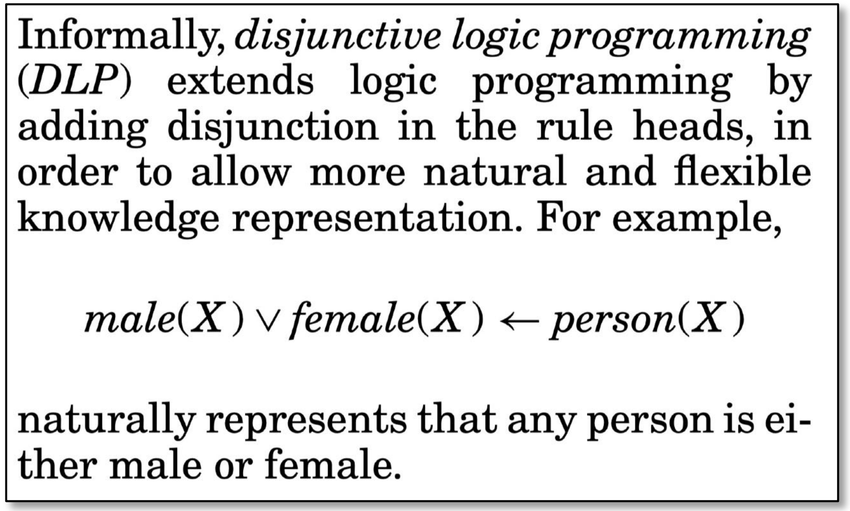
Sources: https://potassco.org/clingo/ , "ASP-Core-2 Input Language Format. Calimeri, Faber, Gebser, et al. TPLP, 2020, https://doi.org/10.1017/S1471068419000450" , "How to Build Your Own ASP-based System?!, Kaminski, Romero, Schaub, Wanko, TPLP, 2023. https://doi.org/10.1017/S1471068421000508"

Complexity and Expressive Power of Logic Programming

EVGENY DANTSIN
Roosevelt University, Chicago, IL, USA
THOMAS EITER, GEORG GOTTLOB
Vienna University of Technology, Austria
AND
ANDREI VORONKOV
University of Manchester, United Kingdom



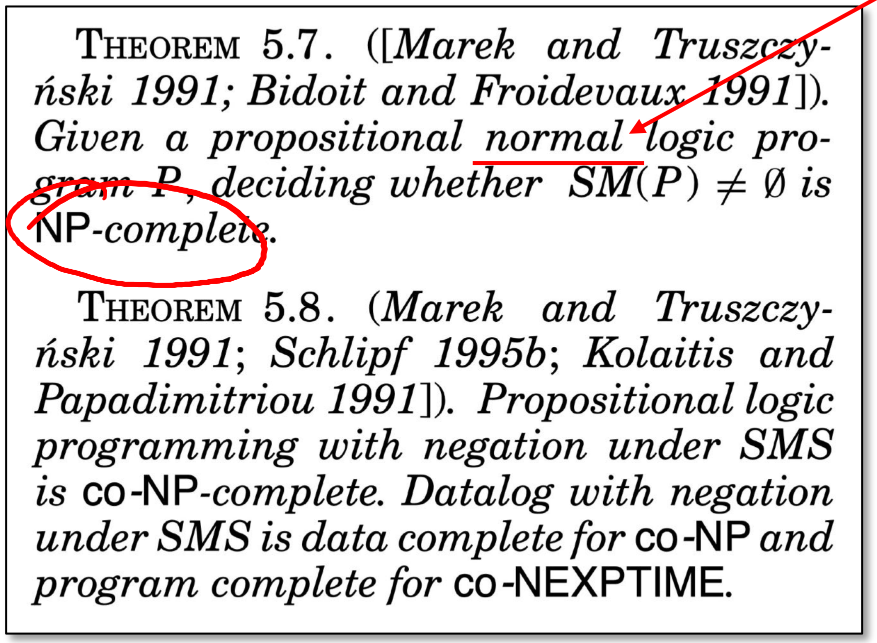
Note that every stratified P has a unique stable model, and its stratified and stable semantics coincide. Unstratified rules increase complexity.



Informally, *disjunctive logic programming* (*DLP*) extends logic programming by adding disjunction in the rule heads, in order to allow more natural and flexible knowledge representation. For example,

$$male(X) \vee female(X) \leftarrow person(X)$$

naturally represents that any person is either male or female.



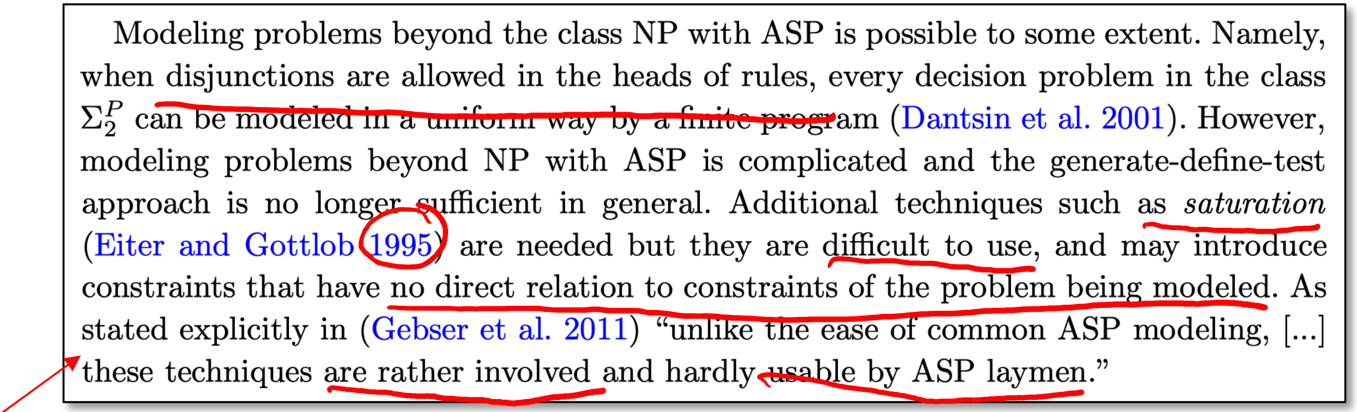
THEOREM 5.7. ([*Marek and Truszczyński 1991; Bidoit and Froidevaux 1991*]). *Given a propositional normal logic program P, deciding whether $SM(P) \neq \emptyset$ is* NP*-complete.*

THEOREM 5.8. (*Marek and Truszczyński 1991*; *Schlipf 1995b*; *Kolaitis and Papadimitriou 1991*]). *Propositional logic programming with negation under SMS is* co-NP*-complete. Datalog with negation under SMS is data complete for* co-NP *and program complete for* co-NEXPTIME.

"normal" means no disjunctions in head

Example for NP-complete problem: Boolean satisfiability problem: "given a Boolean formula, is it satisfiable" (i.e. is there an input for which the formula outputs true)?

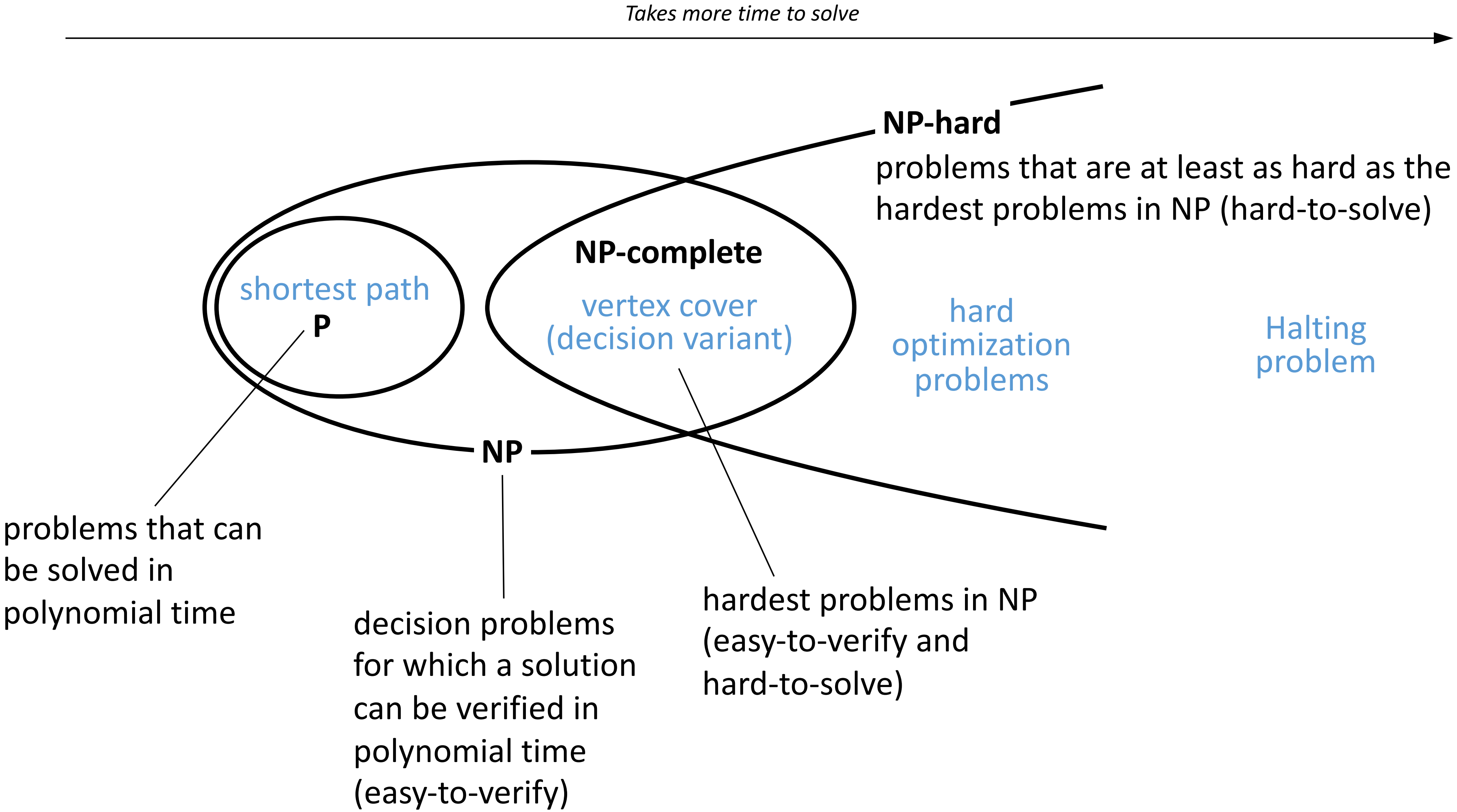
Example for co-NP problem: the complementary problem asks: "given a Boolean formula, is it unsatisfiable" (i.e. do all possible inputs to the formula output false)?



Modeling problems beyond the class NP with ASP is possible to some extent. Namely, when disjunctions are allowed in the heads of rules, every decision problem in the class Σ_2^P can be modeled in a uniform way by a finite program (Dantsin et al. 2001). However, modeling problems beyond NP with ASP is complicated and the generate-define-test approach is no longer sufficient in general. Additional techniques such as *saturation* (Eiter and Gottlob 1995) are needed but they are difficult to use, and may introduce constraints that have no direct relation to constraints of the problem being modeled. As stated explicitly in (Gebser et al. 2011) "unlike the ease of common ASP modeling, [...] these techniques are rather involved and hardly usable by ASP laymen."

Dantsin, Eiter, Gottlob, Voronkov. "Complexity and expressive power of logic programming", ACM computing survesy, 2001. https://doi.org/10.1145/502807.502810
Amendola, Ricca, Truszczynski. "Beyond NP: Quantifying over Answer Sets", TPLP, 2019. https://doi.org/10.1017/S1471068419000140
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

NP-hardness (assuming P≠NP)



Takes more time to solve

NP-hard

problems that are at least as hard as the hardest problems in NP (hard-to-solve)

NP-complete

problems that can be solved in polynomial time

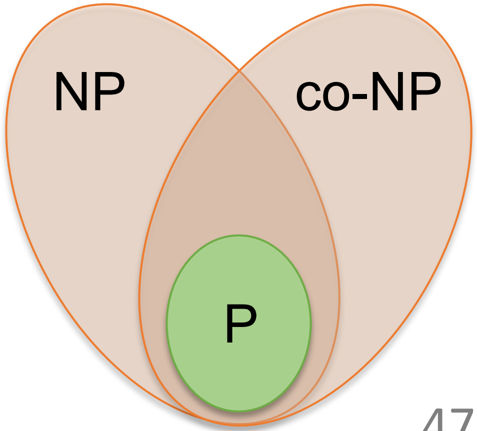
decision problems for which a solution can be verified in polynomial time (easy-to-verify)

hardest problems in NP (easy-to-verify and hard-to-solve)

NP vs. Co-NP

- NP: decision problems for which a solution can be verified in PTIME
  - SAT: Given a Boolean formula, is it satisfiable (i.e. there is an input for which the formula outputs true)?
    $\varphi = (x \vee y \vee z) \wedge (\bar{x} \vee z \vee w) \wedge (\bar{y} \vee \bar{z} \vee \bar{w})$ 3SAT (3CNF)
  - 3-colorability: Given a graph, is there an assignment of colors to nodes s.t. no edge connects same colors?
  - VC (Vertex Cover): Given a graph and a number k (as part of input), is there a VC of size k or smaller?

- Co-NP-complete: A decision problem is in co-NP if its complement is in NP.
  - $\text{Co-NP} = \{L \mid \bar{L} \in \text{NP}\}$
  - UNSAT: Given a Boolean formula, is it unsatisfiable (i.e. is it false for all choices of inputs)?
  - Tautology: Given a Boolean formula, is it a tautology (i.e. is it true for all choices of inputs)?
  - Uncolorable: Given a graph, is there <u>no assignment</u> of colors to nodes s.t. edges connect different colors?
  - "UNCOVERABLE": Given a graph and a number k, is there <u>no VC</u> of size k or smaller?



Computational Complexity of Logic Programs (LP) / ASP

a disjunctive LP with optimization statements

- Deciding whether an atom is in an optimal SM (stable model) is Δ_3^p**-complete**
- Deciding whether a set of atoms is an optimal SM is **co-NP$^{\text{NP}}$-complete**

a disjunctive LP

- Deciding whether an atoms is in a SM is **NP$^{\text{NP}}$-complete**
- Deciding whether a set of atoms is a SM of a disjunctive P is **co-NP-complete**

a normal LP with optimization statements

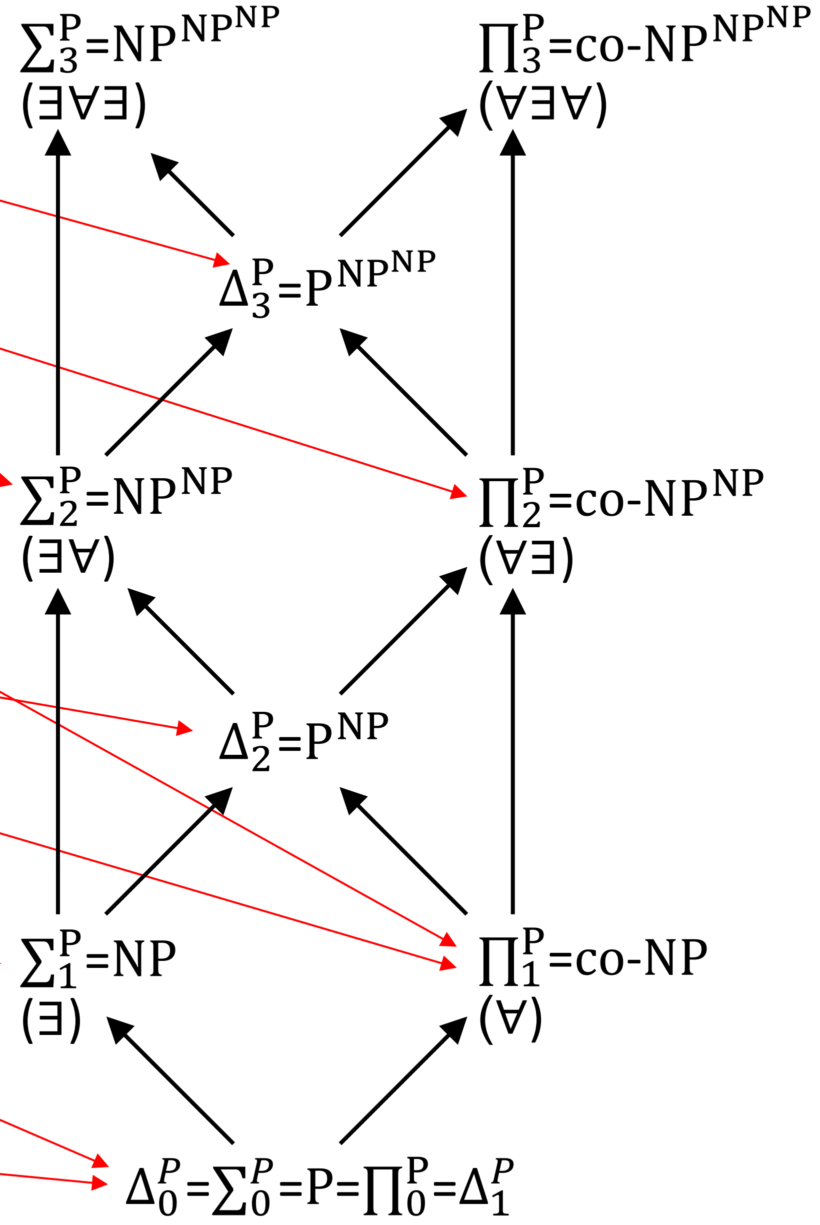
- Deciding whether an atom is in an optimal SM is Δ_2^p**-complete**
- Deciding whether a set of atoms is an optimal SM is **co-NP-complete**

a normal LP (no disjunction in head)

- Deciding whether an atom is in a SM is **NP-complete** (e.g. satisfiability)
- Deciding whether a set of atoms is a SM is **P-complete**

a positive normal LP (no negation in body)

- Deciding whether an atom is in a stable model,
- or whether a set of atoms is a stable model is **P-complete** (cf. Datalog)



Σ_3^P=$\text{NP}^{\text{NP}^{\text{NP}}}$ (∃∀∃)
Π_3^P=$\text{co-NP}^{\text{NP}^{\text{NP}}}$ (∀∃∀)
Δ_3^P=$\text{P}^{\text{NP}^{\text{NP}}}$
Σ_2^P=NP^{NP} (∃∀)
Π_2^P=co-NP^{NP} (∀∃)
Δ_2^P=P^{NP}
Σ_1^P=NP (∃)
Π_1^P=co-NP (∀)
$\Delta_0^P=\Sigma_0^P=\text{P}=\Pi_0^P=\Delta_1^P$

Created based on: Gebser, Kaminski, Kaufmann, Schaub. Answer Set Solving in Practice. Synthesis Lectures on AI and ML, 2013. https://doi.org/10.1007/978-3-031-01561-8

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

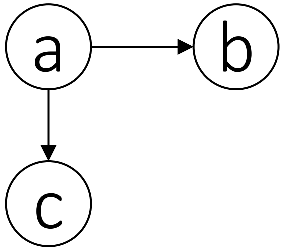
Details on Disjunctive Logic Programming

- 3-colorability
  - 3-colorability with normal or disjunctive logic programs
  - 3-uncolorability with cautious semantics
- Optimization
  - Minimal Vertex Cover with weak constraints, optimization, aggregates
  - Shortest paths with aggregation (contrast Clingo vs Souffle)
- Saturation for Disjunctive Logic Programs
  - Minimal example for the power of saturation
  - Uncolorability (program is satisfiable iff a graph is not 3-colorable)
  - Minimal Vertex Cover of a particular size without minimization

3-colorability (1/6)

```
clingo 3colorability1.txt
```

Capital letters are variables, lowercase letters and numbers are constants (notice the difference to Souffle)



3colorability1.txt

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(X,1) :- not color(X,2), not color(X,3), vertex(X).
color(X,2) :- not color(X,3), not color(X,1), vertex(X).
color(X,3) :- not color(X,1), not color(X,2), vertex(X).
:- edge(X,Y), color(X,C), color(Y,C).
```

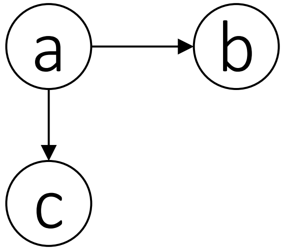
Returns a stable model if it exists. Since there is a stable model, the problem is "satisfiable".

Recall that an empty head encodes a constraint that the body can't be true. Thus no two neighbors in a valuation can share colors.

```
Answer: 1
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c) color(a,1)
color(b,3) color(c,3)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability (2/6)



```
clingo 3colorability2.txt
```

3colorability2.txt

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(X,1) :- not color(X,2), not color(X,3), vertex(X).
color(X,2) :- not color(X,3), not color(X,1), vertex(X).
color(X,3) :- not color(X,1), not color(X,2), vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
:- notcolored.
```

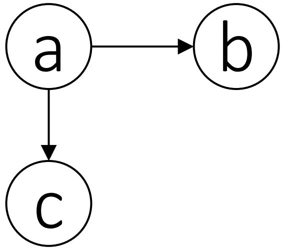
Now, if any two neighbors in a valuation share colors, then "notcolored" needs to be true.

But "notcolored" cannot be true

```
Answer: 1
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c) color(a,1)
color(b,3) color(c,3)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability (3/6)



```
clingo 3colorability3.txt
```

3colorability3.txt

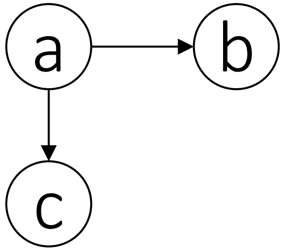
```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(X,1) :- not color(X,2), not color(X,3), vertex(X).
color(X,2) :- not color(X,3), not color(X,1), vertex(X).
color(X,3) :- not color(X,1), not color(X,2), vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
a :- notcolored, not a.
```

Another way to think about the empty header from the previous pages: if "notcolored" is true, then the body of a rule is "a :- not a", which has no stable model.

```
Answer: 1
vertex(a) vertex(b) vertex(c) edge(a,b) edge(a,c) color(a,1)
color(b,3) color(c,3)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability (4/6)



```
clingo 3colorability4.txt
```

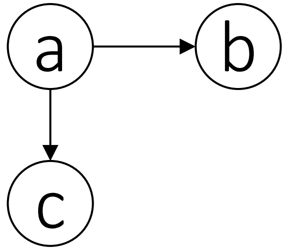
3colorability4.txt

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(X,1) :- not color(X,2), not color(X,3), vertex(X).
color(X,2) :- not color(X,3), not color(X,1), vertex(X).
color(X,3) :- not color(X,1), not color(X,2), vertex(X).
:- edge(X,Y), color(X,C), color(Y,C).
#show color/2.
```

Only show the predicate "color" with arity=2 (i.e. 2 arguments). clingo allows different predicates with same name but different arities; thus we need to include the "/2"

```
Answer: 1
color(a,1) color(b,3) color(c,3)
SATISFIABLE
```

3-colorability (5/6)



```
clingo 3colorability4.txt -n 0
```

Show all models

3colorability4.txt

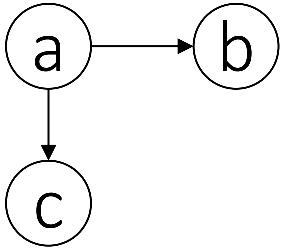
```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(X,1) :- not color(X,2), not color(X,3), vertex(X).
color(X,2) :- not color(X,3), not color(X,1), vertex(X).
color(X,3) :- not color(X,1), not color(X,2), vertex(X).
:- edge(X,Y), color(X,C), color(Y,C).
#show color/2.
```

12 possible colorings.
12 = 3 (for a) * 2 * 2 (for b and c)

```
Answer: 1
color(a,1) color(b,3) color(c,3)
Answer: 2
color(a,1) color(b,3) color(c,2)
Answer: 3
color(a,1) color(b,2) color(c,3)
Answer: 4
color(a,1) color(b,2) color(c,2)
...
Answer: 11
color(a,3) color(b,2) color(c,2)
Answer: 12
color(a,3) color(b,1) color(c,2)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability (6/6)



```
clingo 3colorability5.txt –n 0
```

3colorability5.txt

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(X,1) :- not color(X,2), not color(X,3), vertex(X).
color(X,2) :- not color(X,3), not color(X,1), vertex(X).
color(X,3) :- not color(X,1), not color(X,2), vertex(X).
:- edge(X,Y), color(X,C), color(Y,C).
#show.
#show (X,C) : color(X,C).
```

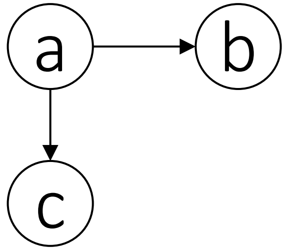
Turns off printing of all predicates by default

Conditional statement: shows (X,C) terms if color(X, C) is true

```
Answer: 1
(a,1) (b,3) (c,3)
Answer: 2
(a,1) (b,3) (c,2)
Answer: 3
(a,1) (b,2) (c,3)
Answer: 4
(a,1) (b,2) (c,2)
...
Answer: 11
(a,3) (b,2) (c,2)
Answer: 12
(a,3) (b,1) (c,2)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability: now with disjunction



```
clingo 3colorability-disjunction.txt -n 0
```

3colorability-disjunction.txt

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).

color(X,1) | color(X,2) | color(X,3) :- vertex(X).

:- edge(X,Y), color(X,C), color(Y,C).
#show.
#show (X,C) : color(X,C).
```

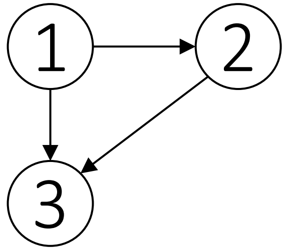
- Guess a possible color assignment of vertices. This rule does not prevent a vertex from getting assigned >1 color.
- However, a vertex having multiple colors is not part of a minimal model since it is a superset of a valid coloring.

clingo also allows ";" instead of "|" for disjunctions

```
Answer: 1
(a,1) (b,3) (c,3)
Answer: 2
(a,1) (b,3) (c,2)
Answer: 3
(a,1) (b,2) (c,3)
Answer: 4
(a,1) (b,2) (c,2)
...
Answer: 11
(a,3) (b,2) (c,2)
Answer: 12
(a,3) (b,1) (c,2)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability: Brave semantics (1/2)



```
clingo 3colorability-brave1.txt -n 0
```

3colorability-brave1.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
colored :- not notcolored.
#show.
#show yes : colored.
#show no : notcolored.
```

defines a range 1, 2, 3

We use here disjunction although not needed

If any two neighbors in a valuation share colors, then "notcolored" needs to be true. Since it is the only rule with "notcolored" in the head, "notcolored" is true iff any two neighbors share the color.

"colored" is true if "notcolored" is not.

Show "yes" if colored is true.
Show "no" if notcolored is true.

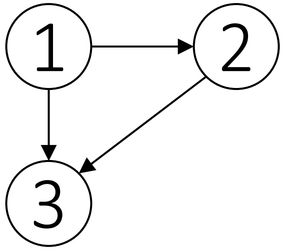
In a minimal model, notcolored and colored are not true at the same time. Thus "colored" is only true in a stable model where "notcolored" is not true and thus the color assignment is valid.

```
Answer: 1
no
Answer: 2
yes
Answer: 3
no
...
Answer: 27
no
SATISFIABLE
```

Notice 27 possible colorings. Each is either a valid coloring ("yes") or not ("no").

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability: Brave semantics (2/2)



```
clingo 3colorability-brave2.txt -e brave
```

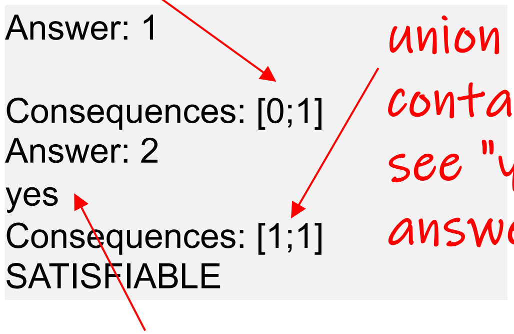
"brave" execution mode gives possible answers (union): Is there an answer set in which the query (here "yes=true") holds?

3colorability-brave2.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
colored :- not notcolored.
#show.
#show yes : colored.
```

Clingo uses multiple answer sets to converge on the final union/intersection. "Consequences [d;p]" are essentially lower and upper bounds which converge towards d=p.

(Details: There are d definite consequences and p probable consequences. For brave semantics, the value of d increases with processing of more models while in cautious semantics the value of p decreases.)



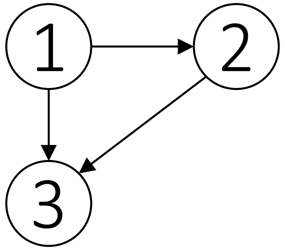
```
Answer: 1

Consequences: [0;1]
Answer: 2
yes
Consequences: [1;1]
SATISFIABLE
```

The 2nd (last) answer (after convergence) is the union of all models: it contains "colored", thus we see "yes": there is some answer that is correct.

"yes", thus there exists some model in which "colored" is true

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-uncolorability: Cautious semantics (1/3)



```
clingo 3colorability-cautious1.txt -e brave
```

3colorability-cautious1.txt

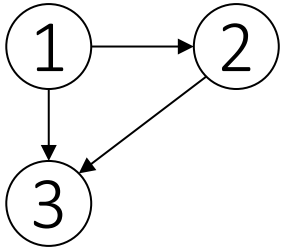
```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
colored :- not notcolored.
#show.
#show yes : notcolored.
```

Here, clingo happens to find that the first stable model it looks at has "notcolored" as true. Thus it does not need to look further: it knows that the union of the answers contains "notcolored"

Here we are asking if there is at least one stable model (one answer set) in which "notcolored" is true.

```
Answer: 1
yes
Consequences: [1;1]
SATISFIABLE
```

3-uncolorability: Cautious semantics (2/3)



```
clingo 3colorability-cautious1.txt -e cautious
```

"cautious" execution model gives certain answers (intersection): Is is true that the query holds in *all* stable models?

3colorability-cautious1.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
colored :- not notcolored.
#show.
#show yes : notcolored.
```

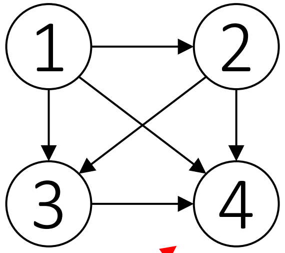
Even by looking at the 2^{nd} answer, we are done: it does not contain "notcolored" and thus the answer is no: the intersection does not contain "notcolored".

```
Answer: 1
yes
Consequences: [0;1]
Answer: 2

Consequences: [0;0]
SATISFIABLE
```

We therefore do not see "yes".

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

3-uncolorability: Cautious semantics (3/3)



```
clingo 3colorability-cautious2.txt -e cautious
```

3colorability-cautious2.txt

```
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
notcolored :- edge(X,Y), color(X,C), color(Y,C).
colored :- not notcolored.
#show.
#show yes : notcolored.
```

This new graph (a 4-clique) is not 3-colorable. Thus "notcolored" is true in all stable models, thus in all attempts to assign colors to vertices. The intersection thus contains "notcolored"

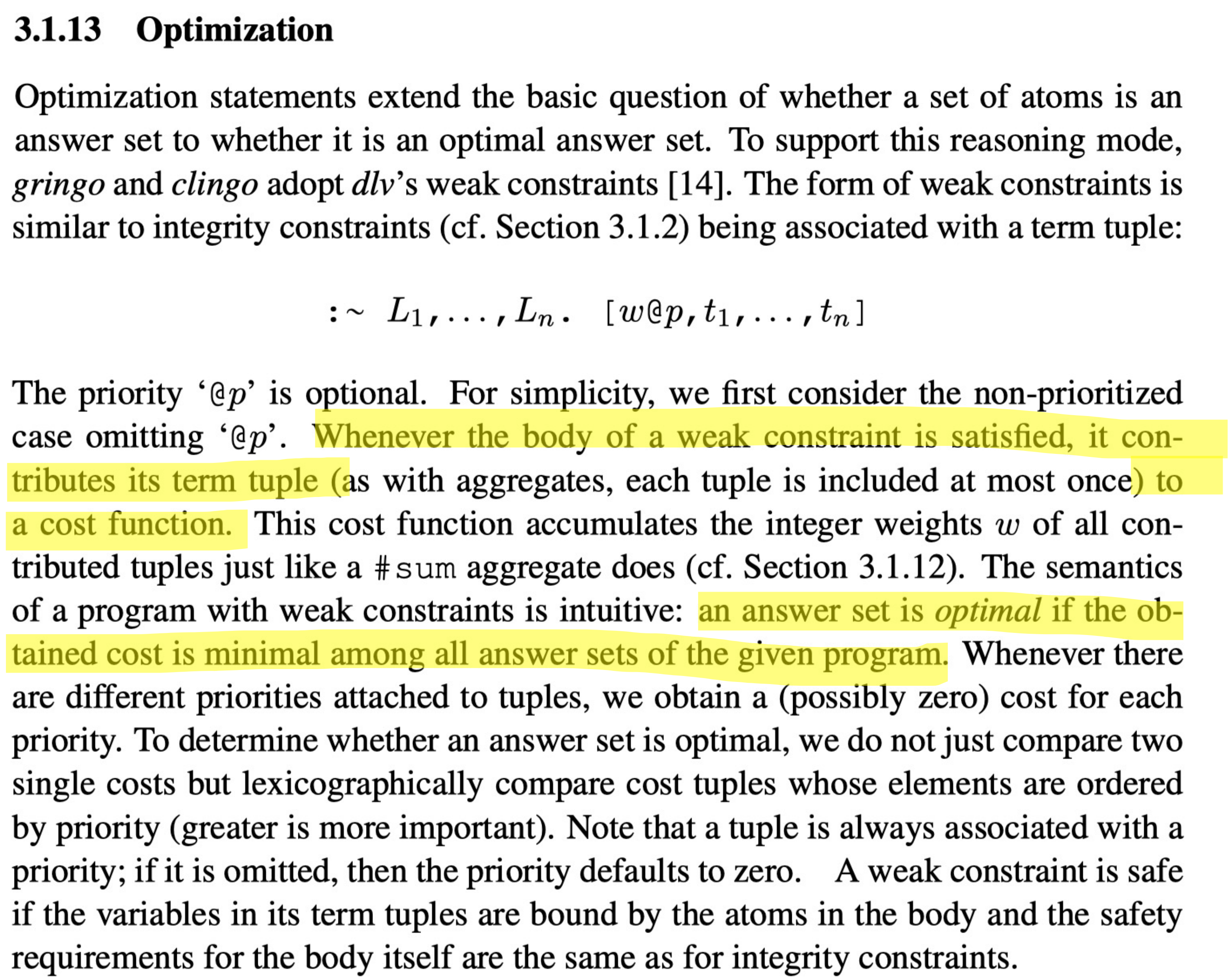
```
Answer: 1
yes
Consequences: [0;1]
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Details on Disjunctive Logic Programming

- 3-colorability
  - 3-colorability with normal or disjunctive logic programs
  - 3-uncolorability with cautious semantics
- Optimization
  - Minimal Vertex Cover with weak constraints, optimization, aggregates
  - Shortest paths with aggregation (contrast Clingo vs Souffle)
- Saturation for Disjunctive Logic Programs
  - Minimal example for the power of saturation
  - Uncolorability (program is satisfiable iff a graph is not 3-colorable)
  - Minimal Vertex Cover of a particular size without minimization

Weak constraints for optimization



3.1.13 Optimization

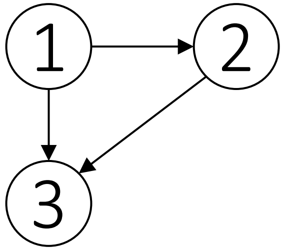
Optimization statements extend the basic question of whether a set of atoms is an answer set to whether it is an optimal answer set. To support this reasoning mode, *gringo* and *clingo* adopt *dlv*'s weak constraints [14]. The form of weak constraints is similar to integrity constraints (cf. Section 3.1.2) being associated with a term tuple:

$$:\sim\ L_1,\dots,L_n.\quad [w@p, t_1,\dots,t_n]$$

The priority '$@p$' is optional. For simplicity, we first consider the non-prioritized case omitting '$@p$'. Whenever the body of a weak constraint is satisfied, it contributes its term tuple (as with aggregates, each tuple is included at most once) to a cost function. This cost function accumulates the integer weights w of all contributed tuples just like a #sum aggregate does (cf. Section 3.1.12). The semantics of a program with weak constraints is intuitive: an answer set is *optimal* if the obtained cost is minimal among all answer sets of the given program. Whenever there are different priorities attached to tuples, we obtain a (possibly zero) cost for each priority. To determine whether an answer set is optimal, we do not just compare two single costs but lexicographically compare cost tuples whose elements are ordered by priority (greater is more important). Note that a tuple is always associated with a priority; if it is omitted, then the priority defaults to zero. A weak constraint is safe if the variables in its term tuples are bound by the atoms in the body and the safety requirements for the body itself are the same as for integrity constraints.

Source: Gebser, Kaminski, Kaufmann, Lindauer, Ostrowski, Romero, Schaub, Thiele, Wanko. Potassco user guide. version 2.2.0, 2019. https://github.com/potassco/guide/releases/
Neha Makhija. Principles of Scalable Database Management. https://northeastern-datalab.github.io/cs7240/

Minimum Vertex Cover: Optimization



```
clingo minVC-optimization.txt
```

minVC-optimization.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
cover(N,1) | cover(N,0) :- vertex (N).
:- edge(X,Y), cover(X,0), cover(Y,0).
:~ cover(X,1). [1@1, X]
#show. #show (X,C): cover(X,C).
```

We use here disjunction although not needed: every vertex "N" is in the cover (1) or not (0)

At least one endpoint of each edge needs to be in the cover, i.e. both can't be outside the cover (0)

Minimize the number of valuations for X that make "cover(X,1)" true

Body

Tail

terms (t_1, ...t_n)

priority (optional)

weight (w)

Show the nodes and whether they are in the cover (1) or not (0)

```
Answer: 1
(1,1) (2,1) (3,1)
Optimization: 3
Answer: 2
(1,1) (2,1) (3,0)
Optimization: 2
OPTIMUM FOUND
```

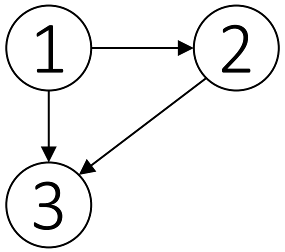
an intermediate non-optimal answer

last answer is an optimal answer

Intuitively: enforce weak constraints if possible. Minimize the number of violations.

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Minimum Vertex Cover: Optimization



```
clingo minVC-optimization.txt
```

minVC-optimization.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
cover(N,1) | cover(N,0) :- vertex (N).
:- edge(X,Y), cover(X,0), cover(Y,0).
:~ cover(X,1). [1@1, X]
#show. #show (X,C): cover(X,C).
```

Body

Tail

terms $(t_1, \ldots t_n)$

priority (optional)

weight (w)

SEMANTICS OF WEAK CONSTRAINTS:

For any program **P** and answer set **A**, *weak*(**P**,**A**) is the set of all unique tails of weak constraints in ground(**P**) whose body is satisfied by **A**

Goal is to minimize $\sum_{(t_1,\ldots t_n)\in \mathbf{weak}(\mathbf{P},\mathbf{A})} w$

Higher priority levels are more important

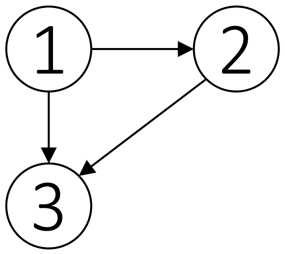
```
Answer: 1
(1,1) (2,1) (3,1)
Optimization: 3
Answer: 2
(1,1) (2,1) (3,0)
Optimization: 2
OPTIMUM FOUND
```

an intermediate non-optimal answer

last answer is an optimal answer

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Minimum Vertex Cover: Optimization



```
clingo minVC-aggregation.txt
```

minVC-aggregation.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
cover(N,1) | cover(N,0) :- vertex (N).
:- edge(X,Y), cover(X,0), cover(Y,0).
#minimize {1@1, X : cover(X,1)}.
#show. #show (X,C): cover(X,C).
```

Minimize the number of valuations for X that make "cover(X,1)" true

Body

terms (t_1, ...t_n)

priority (optional)

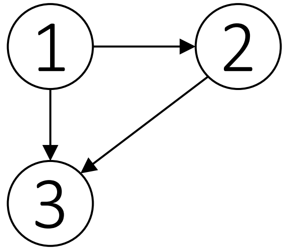
weight (w)

```
Answer: 1
(1,1) (2,1) (3,1)
Optimization: 3
Answer: 2
(1,1) (2,1) (3,0)
Optimization: 2
OPTIMUM FOUND
```

same answer

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Minimum Vertex Cover: Aggregate / Decision



```
clingo minVC-decision1.txt -n 0
```

Show all models

minVC-decision1.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
cover(N,1) | cover(N,0) :- vertex (N).
:- edge(X,Y), cover(X,0), cover(Y,0).
:- #count{X : cover(X, 1)} > 2.

#show. #show (X,C): cover(X,C).
```

Aggregate Atom

Counts values X that make "cover(X,1)" true

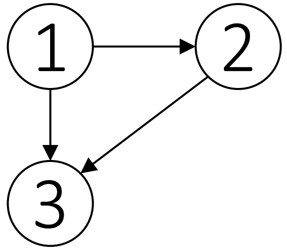
Check if there is some valid cover with 2 or fewer vertices covered

The size of the cover cannot be > 2

```
Answer: 1
(1,1) (2,1) (3,0)
Answer: 2
(1,1) (2,0) (3,1)
Answer: 3
(1,0) (2,1) (3,1)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Minimum Vertex Cover: Aggregate / Decision



```
clingo minVC–decision1.txt –n 0
```

minVC-decision1.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
cover(N,1) | cover(N,0) :- vertex (N).
:- edge(X,Y), cover(X,0), cover(Y,0).
:- #count{X : cover(X, 1)} > 2.

#show. #show (X,C): cover(X,C).
```

Aggregate Atom

Counts values X that make "cover(X,1)" true

SEMANTICS OF AGGREGATES:

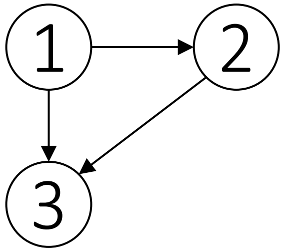
An aggregate atom occurring in a rule body takes the form $l\ \alpha\{t_1 : L_1; \dots; t_n : L_n\}\ u$ where

- α is an aggregate function,
- $t_1 : L_i$ aggregate t_1 when L_i holds
- l, u are optional lower and upper bounds

```
Answer: 1
(1,1) (2,1) (3,0)
Answer: 2
(1,1) (2,0) (3,1)
Answer: 3
(1,0) (2,1) (3,1)
SATISFIABLE
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Minimum Vertex Cover: Aggregate / Decision



```
clingo minVC-decision2.txt -n 0
```

minVC-decision2.txt

```
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
cover(N,1) | cover(N,0) :- vertex (N).
:- edge(X,Y), cover(X,0), cover(Y,0).
solution :- #count{X: cover(X, 1)} <= 2.
:- not solution.



#show. #show (X,C): cover(X,C).
```

Check if there is some valid cover with 2 or fewer nodes covered

If the size of the cover is <= 2, then it is a solution.

And "solution" cannot be false (otherwise it true would imply false)

```
Answer: 1
(1,1) (2,1) (3,0)
Answer: 2
(1,1) (2,0) (3,1)
Answer: 3
(1,0) (2,1) (3,1)
SATISFIABLE
```

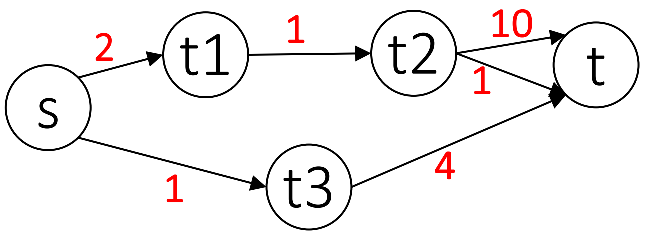
Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Details on Disjunctive Logic Programming

- 3-colorability
  - 3-colorability with normal or disjunctive logic programs
  - 3-uncolorability with cautious semantics
- Optimization
  - Minimal Vertex Cover with weak constraints, optimization, aggregates
  - Shortest paths with aggregation (contrast Clingo vs Souffle)
- Saturation for Disjunctive Logic Programs
  - Minimal example for the power of saturation
  - Uncolorability (program is satisfiable iff a graph is not 3-colorable)
  - Minimal Vertex Cover of a particular size without minimization

Shortest Path via Aggregation

weights of edges



```
clingo shortestpath1.txt
```

shortestpath1.txt

```
edge(s,v1,2). edge(v1,v2,1). edge(v2,t,1).
edge(v2,t,10). edge(s,v3,1). edge(v3,t,4).


path(X,Y,W) :- edge(X,Y,W).
path(X,Z,W1+W2) :- path(X,Y,W1), path(Y,Z,W2).


minpath(X,Y,C) :- path(X,Y,_), C=#min{W: path(X,Y,W)}.
#show. #show W: minpath(s,t,W).
```

Aggregate Atom

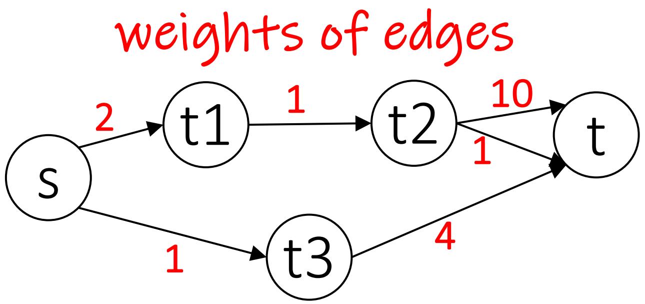
For all possible values X,Y grounded by "path(X,Y,_)", find the minimum weight W, call it C and store it in minpath(X,Y,C)

```
Answer: 1
4
SATISFIABLE
```

The length of the shortest path

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Shortest Path via Aggregation



```
clingo shortestpath2.txt
```

shortestpath2.txt

```
edge(s,v1,2). edge(v1,v2,1). edge(v2,t,1).
edge(v2,t,10). edge(s,v3,1). edge(v3,t,4).


path(X,Y,W) :- edge(X,Y,W).
path(X,Z,W1+W2) :- path(X,Y,W1), path(Y,Z,W2).


minpath(C) :- C=#min{W: path(s,t,W)}.
#show. #show W: minpath(W).
```

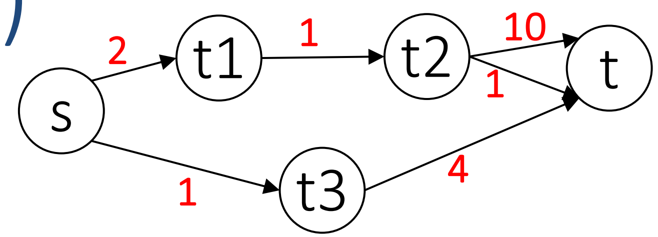
For all possible values W grounded by "path(s,t,W)", find the minimum weight W, call it C and store it in minpath(C)

```
Answer: 1
4
SATISFIABLE
```

The length of the shortest path

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Shortest Path via Aggregation (Souffle)



weights of edges

```
souffle shortestpath.dl
```

shortestpath.dl

```
.decl edge(x: symbol, y: symbol, wt:number)
.input edge
.decl path(x: symbol, y: symbol, wt:number)
path(x,y,w) :- edge(x,y,w).
path(x,z,w1+w2) :- path(x,y,w1), path(y,z,w2).
.decl minpath(w:number)
minpath(c) :- c = min w:{path("s","t",w)}.
.output minpath
```

edge.facts

```
s   v1  2
v1  v2  1
v2  t   1
v2  t   10
s   v3  1
v3  t   4
```

minpath.csv

```
4
```

Recall that in souffle, constants are indicated by quotation marks

Answer in minSTpath

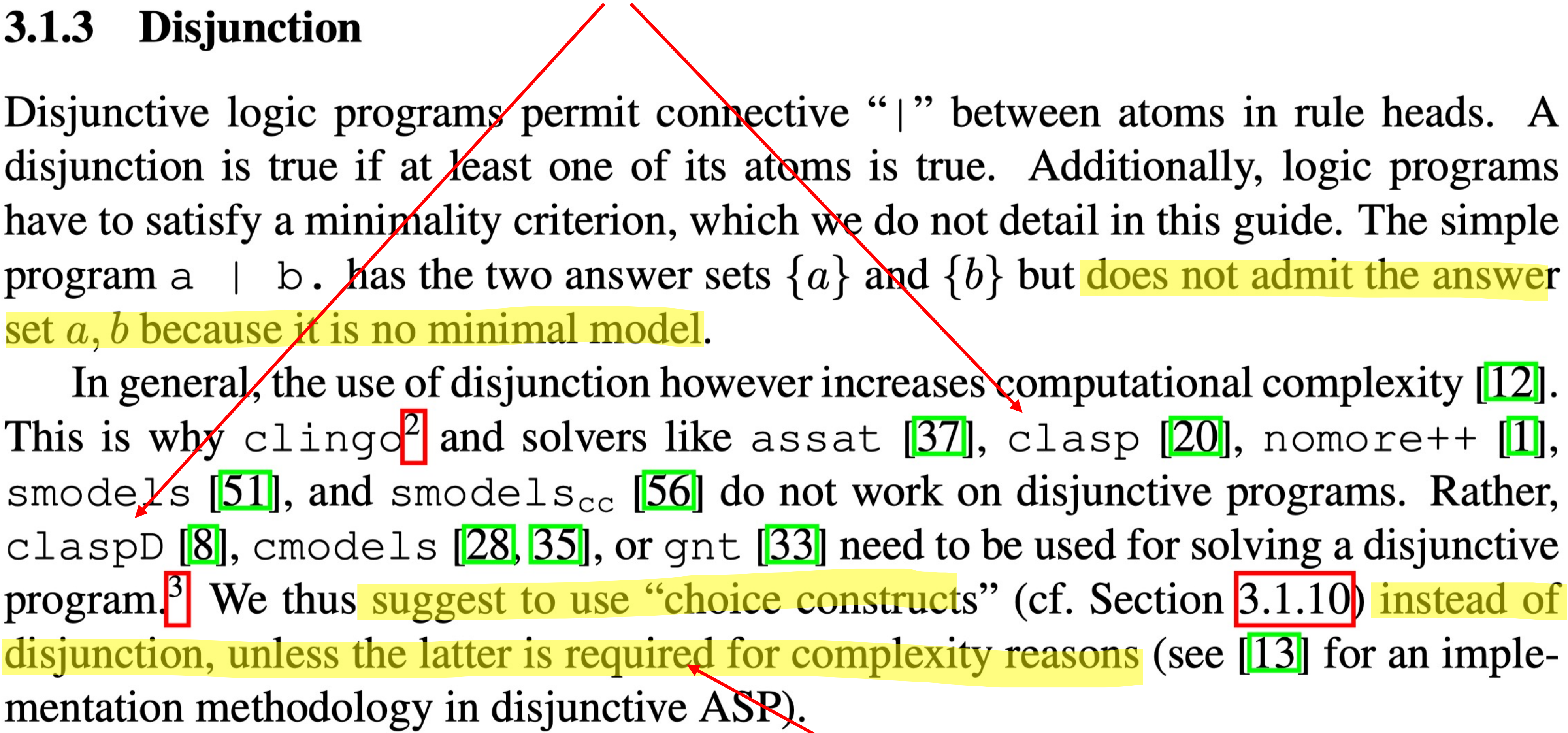
Souffle example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/souffle

Details on Disjunctive Logic Programming

- 3-colorability
  - 3-colorability with normal or disjunctive logic programs
  - 3-uncolorability with cautious semantics
- Optimization
  - Minimal Vertex Cover with weak constraints, optimization, aggregates
  - Shortest paths with aggregation (contrast Clingo vs Souffle)
- Saturation for Disjunctive Logic Programs
  - Minimal example for the power of saturation
  - Uncolorability (program is satisfiable iff a graph is not 3-colorable)
  - Minimal Vertex Cover of a particular size without minimization

Use Disjunction only if needed

clasp and claspD have been united into clasp



3.1.3 Disjunction

Disjunctive logic programs permit connective "|" between atoms in rule heads. A disjunction is true if at least one of its atoms is true. Additionally, logic programs have to satisfy a minimality criterion, which we do not detail in this guide. The simple program `a | b.` has the two answer sets $\{a\}$ and $\{b\}$ but does not admit the answer set a, b because it is no minimal model.

In general, the use of disjunction however increases computational complexity [12]. This is why `clingo`[2] and solvers like `assat` [37], `clasp` [20], `nomore++` [1], `smodels` [51], and `smodels`$_{cc}$ [56] do not work on disjunctive programs. Rather, `claspD` [8], `cmodels` [28, 35], or `gnt` [33] need to be used for solving a disjunctive program.[3] We thus suggest to use "choice constructs" (cf. Section 3.1.10) instead of disjunction, unless the latter is required for complexity reasons (see [13] for an implementation methodology in disjunctive ASP).

It is is possible that modern solvers can detect head-cycle free disjunctions and internally "shift" the heads to normal logic programs.

Source: Gebser, Kaminski, Kaufmann, Ostrowski, Schaub, Thiele. A user's guide to gringo, clasp, clingo, and iclingo. version 3.x. 2010.

Horn clauses and logic programming

A clause is a disjunction of literals.

$\bar{a} \lor \bar{b} \lor c \lor d \qquad a \land b \Rightarrow c \lor d$

$$1 \land a \land b \Rightarrow c \lor d \lor 0$$

A Horn clause has at most one positive (i.e. unnegated) literal.

$\bar{a} \lor b \lor c \qquad a \land \bar{b} \Rightarrow c$

$\bar{a} \lor b \lor c \qquad a \land \bar{c} \Rightarrow b$

$\bar{a} \lor b \lor c \qquad a \Rightarrow b \lor c$

Those express the same models and minimal models. However, for a model in which both a and b are true, the non-disjunctive version does not include the rules in the reduct because the body is not true!

Recall: $\bar{a} = \neg a = !a = \sim a =$ NOT a

Disjunctive logic programming

Datalog

```
b :- a.
c :- a.
```

If a is true, then both b and c need to be true too

$b \land c \Leftarrow a$

Datalog with negation and stable model semantics, or disjunction in head

```
b :- a, not c.
c :- a, not b.
```

If a is true, then either b or c need to be true (both can be true only if there are other rules)

$b \lor c \Leftarrow a$

```
b | c :- a.
```

If a is true, then at least b or c need to be true:

$b \lor c \Leftarrow a$

When disjunctions add expressiveness (1/2)

```
clingo saturation1.txt -n 0
```

saturation1.txt

```
a :- not b.
b :- not a.
```

→ ?

saturation2.txt

```
a | b :-.
```

→ ?

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

When disjunctions add expressiveness (1/2)

```
clingo saturation1.txt –n 0
```

saturation1.txt

```
a :- not b.
b :- not a.
```

```
Solving...
Answer: 1
b
Answer: 2
a
SATISFIABLE

Models       : 2
```

saturation2.txt

```
a | b :-.
```

```
Solving...
Answer: 1
b
Answer: 2
a
SATISFIABLE

Models       : 2
```

{{a}, {b}}

both have the same two SMs {a} and {b}. {a,b} would also be a model, but is not minimal, thus not a SM

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

When disjunctions add expressiveness (1/2)

```
clingo saturation1.txt –n 0
```

saturation1.txt

```
a :- not b.
b :- not a.
```

```
Solving...
Answer: 1
b
Answer: 2
a
SATISFIABLE

Models       : 2
```

saturation2.txt

```
a | b :-.
```

```
Solving...
Answer: 1
b
Answer: 2
a
SATISFIABLE

Models       : 2
```

{{a}, {b}}

reduct w.r.t {a}

a :- ~~not b.~~

~~b :- not a.~~

→ {a}

reduct w.r.t {a}

```
a | b :-.
```

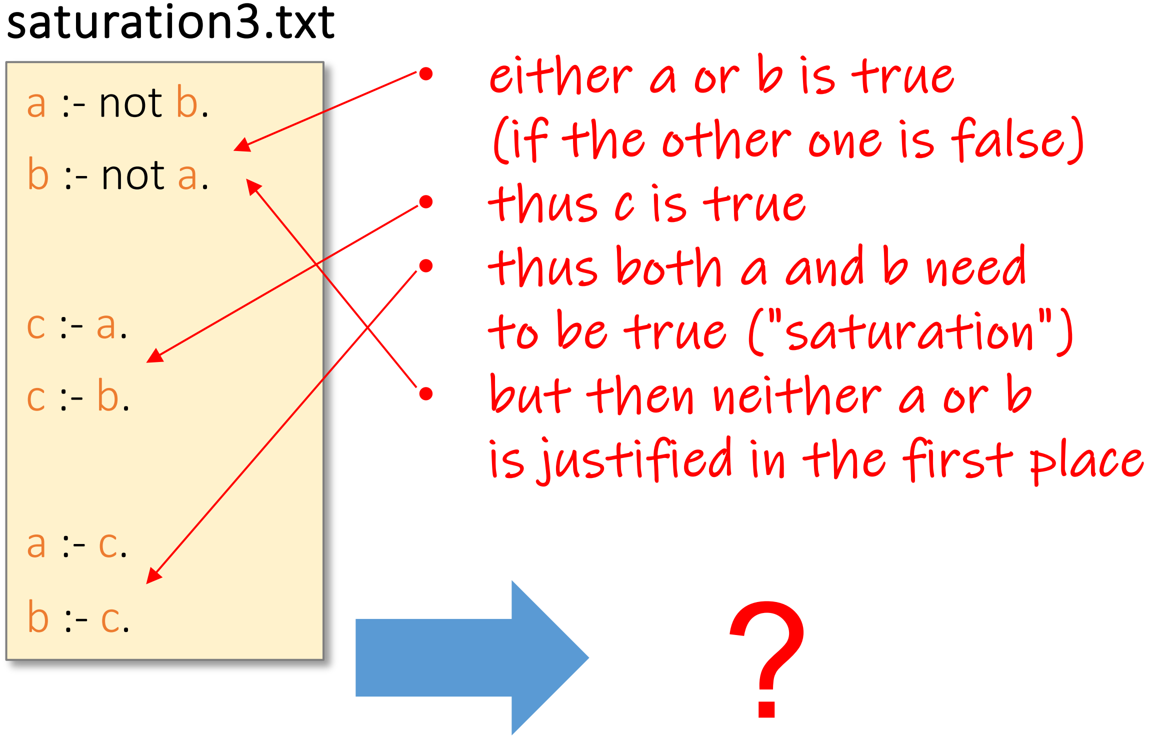
→ {a} or {b}

both have the same two SMs {a} and {b}. {a,b} would also be a model, but is not minimal, thus not a SM

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

When disjunctions add expressiveness (2/2)

```
clingo saturation1.txt -n 0
```

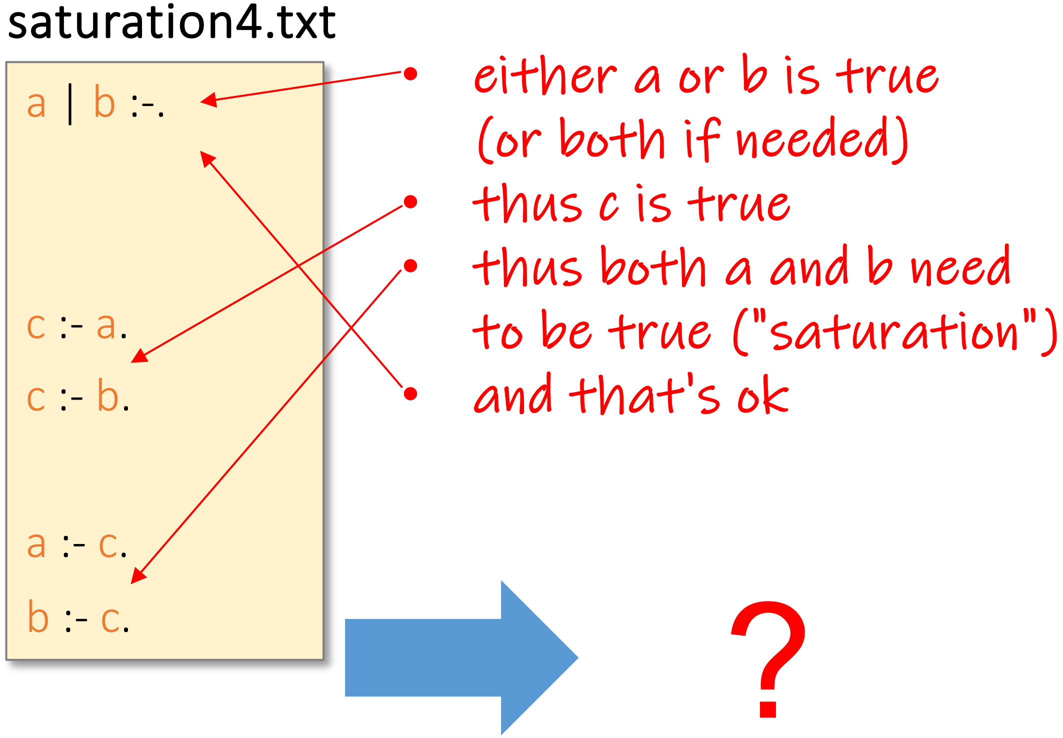


saturation3.txt

```
a :- not b.
b :- not a.

c :- a.
c :- b.

a :- c.
b :- c.
```

- either a or b is true (if the other one is false)
- thus c is true
- thus both a and b need to be true ("saturation")
- but then neither a or b is justified in the first place

?



saturation4.txt

```
a | b :-.

c :- a.
c :- b.

a :- c.
b :- c.
```

- either a or b is true (or both if needed)
- thus c is true
- thus both a and b need to be true ("saturation")
- and that's ok

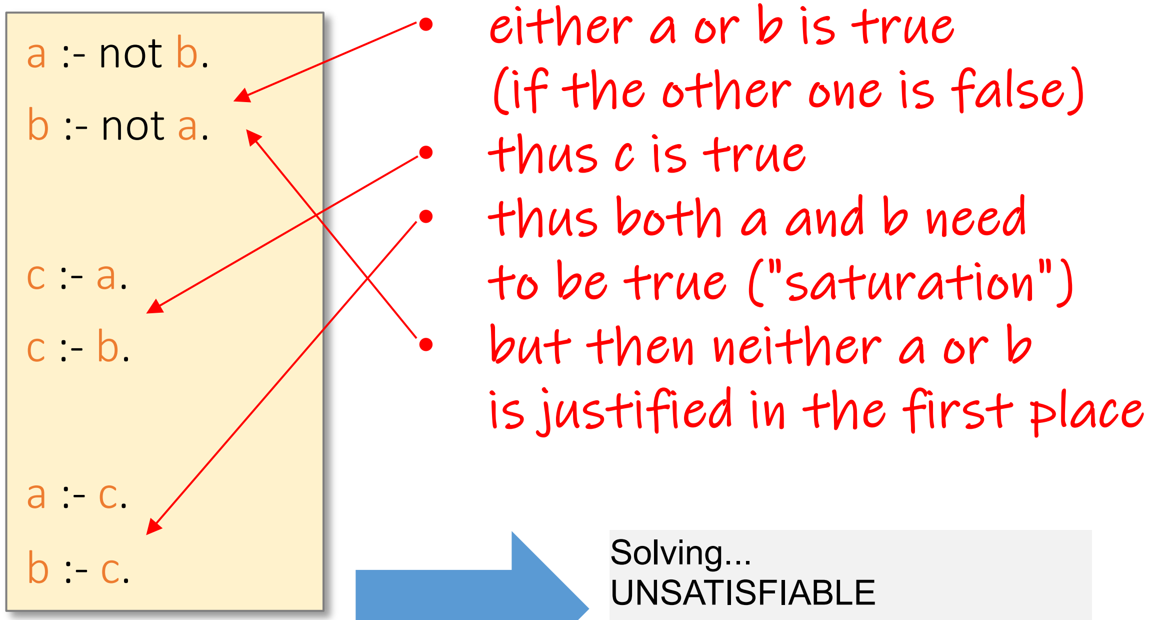
?

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

When disjunctions add expressiveness (2/2)

```
clingo saturation1.txt -n 0
```

saturation3.txt



```
a :- not b.
b :- not a.

c :- a.
c :- b.

a :- c.
b :- c.
```

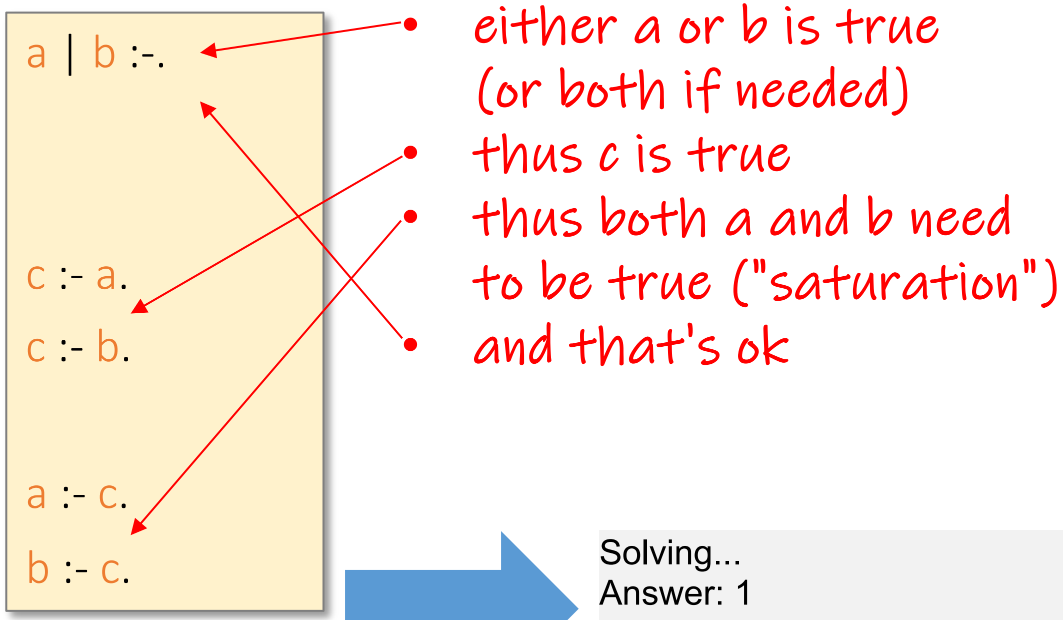
- either a or b is true (if the other one is false)
- thus c is true
- thus both a and b need to be true ("saturation")
- but then neither a or b is justified in the first place

```
Solving...
UNSATISFIABLE

Models       : 0
```

{}

has no SM (stable model)

saturation4.txt



```
a | b :-.



c :- a.
c :- b.

a :- c.
b :- c.
```

- either a or b is true (or both if needed)
- thus c is true
- thus both a and b need to be true ("saturation")
- and that's ok

```
Solving...
Answer: 1
a b c
SATISFIABLE

Models       : 1
```

{a,b,c}

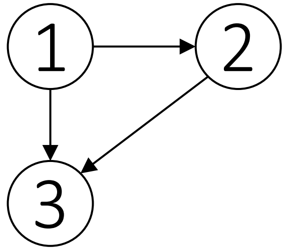
has 1 SM that includes both a and b

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Details on Disjunctive Logic Programming

- 3-colorability
  - 3-colorability with normal or disjunctive logic programs
  - 3-uncolorability with cautious semantics
- Optimization
  - Minimal Vertex Cover with weak constraints, optimization, aggregates
  - Shortest paths with aggregation (contrast Clingo vs Souffle)
- Saturation for Disjunctive Logic Programs
  - Minimal example for the power of saturation
  - Uncolorability (program is satisfiable iff a graph is not 3-colorable)
  - Minimal Vertex Cover of a particular size without minimization

3-uncolorability: via disjunctive LP



```
clingo 3uncolorability2.txt -n 0
```

3uncolorability2.txt

```
% Facts
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
% Guess
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
% Check desired property (of being "uncolored")
uncolored :- edge(X,Y), color(X,C), color(Y,C).
% Saturate if desired property holds
color(X,1..3) :- uncolored, vertex(X).
```

"notcolored" is true iff any two neighbors share the color.

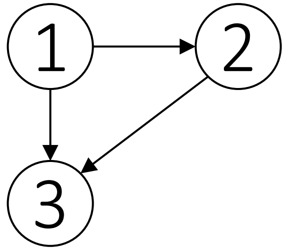
If "notcolored" is true then "saturate" all vertices with all colors. This will never be a minimal SM if there is at least one valid coloring

There are 6 possible colorings in which notcolored is not made true. Thus "notcolored" is never included.

```
Answer: 1
vertex(1) vertex(2) vertex(3) edge(1,2) edge(1,3) edge(2,3) color(1,3) color(2,2) color(3,1)
Answer: 2
vertex(1) vertex(2) vertex(3) edge(1,2) edge(1,3) edge(2,3) color(1,2) color(2,3) color(3,1)
Answer: 3
vertex(1) vertex(2) vertex(3) edge(1,2) edge(1,3) edge(2,3) color(1,3) color(2,1) color(3,2)
Answer: 4
vertex(1) vertex(2) vertex(3) edge(1,2) edge(1,3) edge(2,3) color(1,2) color(2,1) color(3,3)
Answer: 5
vertex(1) vertex(2) vertex(3) edge(1,2) edge(1,3) edge(2,3) color(1,1) color(2,3) color(3,2)
Answer: 6
vertex(1) vertex(2) vertex(3) edge(1,2) edge(1,3) edge(2,3) color(1,1) color(2,2) color(3,3)
SATISFIABLE

Models      : 6
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-uncolorability: via disjunctive LP



```
clingo 3uncolorability3.txt -n 0
```

3uncolorability3.txt

```
% Facts
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
% Guess
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
% Check desired property (of being "uncolored")
uncolored :- edge(X,Y), color(X,C), color(Y,C).
% Saturate if desired property holds
color(X,1..3) :- uncolored, vertex(X).
#show. #show yes : uncolored.
```

"notcolored" is true iff any two neighbors share the color.

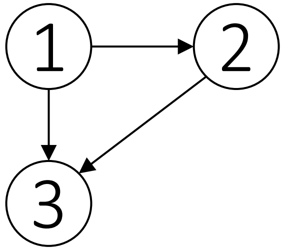
If "notcolored" is true then "saturate" all vertices with all colors. This will never be a minimal SM if there is at least one valid coloring

There are 6 possible colorings in which notcolored is not made true. Thus "notcolored" is never included.

```
Answer: 1

Answer: 2

Answer: 3

Answer: 4

Answer: 5

Answer: 6

SATISFIABLE

Models       : 6
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-uncolorability: via disjunctive LP



```
clingo 3uncolorability3.txt
```

3uncolorability3.txt

```
% Facts
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
% Guess
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
% Check desired property (of being "uncolored")
uncolored :- edge(X,Y), color(X,C), color(Y,C).
% Saturate if desired property holds
color(X,1..3) :- uncolored, vertex(X).
#show. #show yes : uncolored.
```

"notcolored" is true iff any two neighbors share the color.

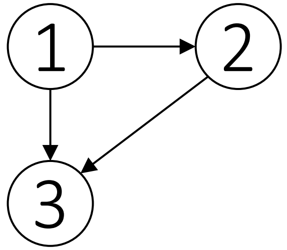
If "notcolored" is true then "saturate" all vertices with all colors. This will never be a minimal SM if there is at least one valid coloring

There are 6 possible colorings in which notcolored is not made true. Thus "notcolored" is never included.

```
Solving...
Answer: 1

SATISFIABLE

Models       : 1+
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-uncolorability: via disjunctive LP



```
clingo 3uncolorability6.txt
```

3uncolorability6.txt

```
% Facts
vertex(1..3). edge(1,2). edge(1,3). edge(2,3).
% Guess
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
% Check desired property (of being "uncolored")
uncolored :- edge(X,Y), color(X,C), color(Y,C).
% Saturate if desired property holds
color(X,1..3) :- uncolored, vertex(X).
% Additionally require desired property
:- not uncolored.
```

"notcolored" is true iff any two neighbors share the color.

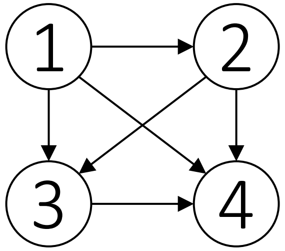
If "notcolored" is true then "saturate" all vertices with all colors. This will never be a minimal SM if there is at least one valid coloring

Additionally require the desired property "uncolored" to be true as additional constraint (recall this rule does not make it true, it needs to be derivable in the reduct)

```
Solving...
UNSATISFIABLE

Models       : 0
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

3-uncolorability: (non-existence of coloring)



```
clingo 3uncolorability1.txt
```

3uncolorability1.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
% Check desired property (of being "uncolored")
uncolored :- edge(X,Y), color(X,C), color(Y,C).
% Saturate if desired property holds
color(X,1..3) :- uncolored, vertex(X).
% Additionally require desired property
:- not uncolored.
```

"notcolored" is true iff any two neighbors share the color.

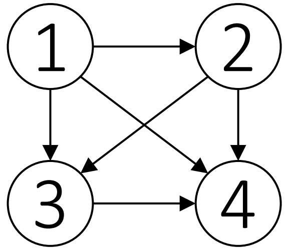
If "notcolored" is true then "saturate" all vertices with all colors. This will never be a minimal SM if there is at least one valid coloring

There is no possible coloring and "notcoloring" is always true. Thus there is only one "saturated" SM that also contains "notcolored" (which is also required)

```
Answer: 1
vertex(1) vertex(2) vertex(3) vertex(4) edge(3,4) edge(1,2)
edge(1,3) edge(1,4) edge(2,3) edge(2,4) color(1,1) color(1,2)
color(1,3) color(2,1) color(2,2) color(2,3) color(3,1) color(3,2)
color(3,3) color(4,1) color(4,2) color(4,3) notcolored
SATISFIABLE

Models       : 1
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

3-colorability: (existence of coloring)



```
clingo 3colorability6.txt
```

3colorability6.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess
color(X,1) | color(X,2) | color(X,3) :- vertex(X).
% Check undesired property (of being "uncolored")
uncolored :- edge(X,Y), color(X,C), color(Y,C).



% Additionally disallow undesired property
:- uncolored.
```

"notcolored" is true iff any two neighbors share the color.

```
Solving...
UNSATISFIABLE

Models       : 0
```

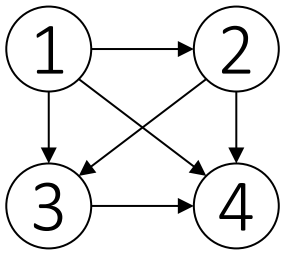
Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Details on Disjunctive Logic Programming

- 3-colorability
  - 3-colorability with normal or disjunctive logic programs
  - 3-uncolorability with cautious semantics
- Optimization
  - Minimal Vertex Cover with weak constraints, optimization, aggregates
  - Shortest paths with aggregation (contrast Clingo vs Souffle)
- Saturation for Disjunctive Logic Programs
  - Minimal example for the power of saturation
  - Uncolorability (program is satisfiable iff a graph is not 3-colorable)
  - Minimal Vertex Cover of a particular size without minimization

existence of VC = 3

```
clingo minVC-existence2.txt
```

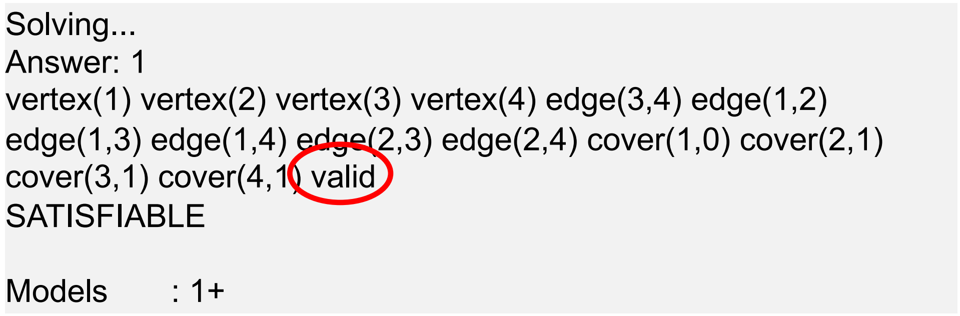


minVC-existence2.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess solutions
cover (X,1) | cover (X,0) :- vertex(X).

% Check and enforce properties
:- edge(X,Y), cover (X,0), cover (Y,0).
valid :- #count{X: cover (X,1)} = 3.
:- not valid.
```

Guess a solution (expressiveness of disjunctive rule is not required here)

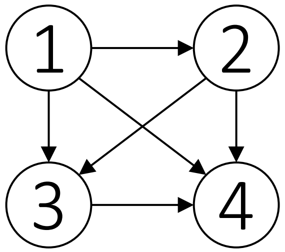
The valid solution needs to be a cover and have 3



```
Solving...
Answer: 1
vertex(1) vertex(2) vertex(3) vertex(4) edge(3,4) edge(1,2)
edge(1,3) edge(1,4) edge(2,3) edge(2,4) cover(1,0) cover(2,1)
cover(3,1) cover(4,1) valid
SATISFIABLE

Models       : 1+
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

non-existence of VC < 3



```
clingo minVC-nonexistence2.txt
```

minVC-nonexistence2.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess solutions

cover_all(X,1) | cover_all(X,0) :- vertex(X).
% Check and enforce properties




invalid :- edge(X,Y), cover_all(X,0), cover_all(Y,0).
invalid :- #count{X: cover_all(X,1)} >= 3.
:- not invalid.
% Additionally saturate if desired property holds
cover_all(X,0..1) :- invalid, vertex(X).
```

Guess all cover candidates with disjunction (here disjunction is needed as we use it with saturation later below)

All cover candidates must be invalid (not a cover or >= 3)

Saturate all other cover candidates if invalid

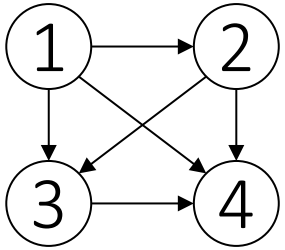
```
Answer: 1
vertex(1) vertex(2) vertex(3) vertex(4) edge(3,4) edge(1,2)
edge(1,3) edge(1,4) edge(2,3) edge(2,4) cover_all(1,1)
cover_all(1,0) cover_all(2,1) cover_all(2,0) cover_all(3,1)
cover_all(3,0) cover_all(4,1) cover_all(4,0) invalid
SATISFIABLE

Models       : 1
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

minVC = 3 (exists 3 and not exists <3)

```
clingo minVC-existsandnot1.txt
```



minVC-existsandnot1.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess solutions
cover (X,1) | cover (X,0) :- vertex(X).
cover_all(X,1) | cover_all(X,0) :- vertex(X).
% Check and enforce properties
:- edge(X,Y), cover (X,0), cover (Y,0).
valid :- #count{X: cover (X,1)} = 3.
:- not valid.
invalid :- edge(X,Y), cover_all(X,0), cover_all(Y,0).
invalid :- #count{X: cover_all(X,1)} >= 3.
:- not invalid.
% Additionally saturate if desired property holds
cover_all(X,0..1) :- invalid, vertex(X).
```

Guess a valid solution (disjunction is not required here)

We want all other cover candidates to not be better (disjunction is required here)

The valid solution needs to be a cover and have 3

All other cover candidates must be invalid (not a cover or >= 3)

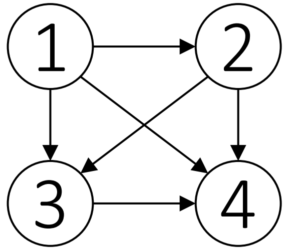
Saturate all other cover candidates if invalid

```
Answer: 1
vertex(1) vertex(2) vertex(3) vertex(4) edge(3,4) edge(1,2)
edge(1,3) edge(1,4) edge(2,3) edge(2,4) cover_all(1,1)
cover_all(1,0) cover_all(2,1) cover_all(2,0) cover_all(3,1)
cover_all(3,0) cover_all(4,1) cover_all(4,0) invalid cover(1,0)
cover(2,1) cover(3,1) cover(4,1) valid
SATISFIABLE

Models       : 1+
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

minVC = K (exists K and not exists <K)

```
clingo minVC-existsandnot2.txt
```



minVC-existsandnot2.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess solutions
cover (X,1) | cover (X,0) :- vertex(X).
cover_all(X,1) | cover_all(X,0) :- vertex(X).
% Check and enforce properties
:- edge(X,Y), cover (X,0), cover (Y,0).
minvc(K) :- #count{X: cover (X,1)} = K.

invalid :- edge(X,Y), cover_all(X,0), cover_all(Y,0).
invalid :- #count{X: cover_all(X,1)} >= K, minvc(K).
:- not invalid.
% Additionally saturate if desired property holds
cover_all(X,0..1) :- invalid, vertex(X).
```

Guess a valid solution (disjunction is not required here)

We want all other cover candidates to not be better (disjunction is required here)

The valid solution needs to be a cover and have some size K

All other cover candidates must be invalid (not a cover or >= K)

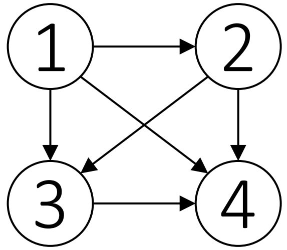
Saturate all other cover candidates if invalid

```
Answer: 1
vertex(1) vertex(2) vertex(3) vertex(4) edge(3,4) edge(1,2)
edge(1,3) edge(1,4) edge(2,3) edge(2,4) cover_all(1,1)
cover_all(1,0) cover_all(2,1) cover_all(2,0) cover_all(3,1)
cover_all(3,0) cover_all(4,1) cover_all(4,0) cover(1,0) cover(2,1)
cover(3,1) cover(4,1) minvc(3) invalid
SATISFIABLE

Models       : 1+
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

minVC = K (exists K and not exists <K)

```
clingo minVC-existsandnot3.txt
```



minVC-existsandnot3.txt

```
% Facts
vertex(1..4). edge(1,2..4). edge(2,3..4). edge(3,4).
% Guess solutions
cover (X,1) | cover (X,0) :- vertex(X).
cover_all(X,1) | cover_all(X,0) :- vertex(X).
% Check and enforce properties
:- edge(X,Y), cover (X,0), cover (Y,0).
minvc(K) :- #count{X: cover (X,1)} = K.

invalid :- edge(X,Y), cover_all(X,0), cover_all(Y,0).
invalid :- #count{X: cover_all(X,1)} >= K, minvc(K).
:- not invalid.
% Additionally saturate if desired property holds
cover_all(X,0..1) :- invalid, vertex(X).


#show. #show K: minvc(K).
```

Guess a valid solution (disjunction is not required here)

We want all other cover candidates to not be better (disjunction is required here)

The valid solution needs to be a cover and have some size K

All other cover candidates must be invalid (not a cover or >= K)

Saturate all other cover candidates if invalid

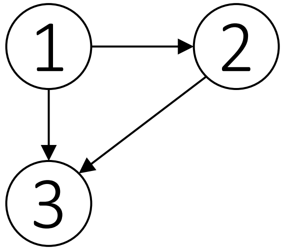
Only show the single entry K in "minvc(K)"

```
Answer: 1
3
SATISFIABLE

Models       : 1+
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

minVC = K (exists K and not exists <K)

```
clingo minVC-existsandnot4.txt
```



minVC-existsandnot4.txt

```
% Facts
vertex(1..3). edge(1,2..3). edge(2,3).
% Guess solutions
cover (X,1) | cover (X,0) :- vertex(X).
cover_all(X,1) | cover_all(X,0) :- vertex(X).
% Check and enforce properties
:- edge(X,Y), cover (X,0), cover (Y,0).
minvc(K) :- #count{X: cover (X,1)} = K.

invalid :- edge(X,Y), cover_all(X,0), cover_all(Y,0).
invalid :- #count{X: cover_all(X,1)} >= K, minvc(K).
:- not invalid.
% Additionally saturate if desired property holds
cover_all(X,0..1) :- invalid, vertex(X).


#show. #show K: minvc(K).
```

Guess a valid solution (disjunction is not required here)

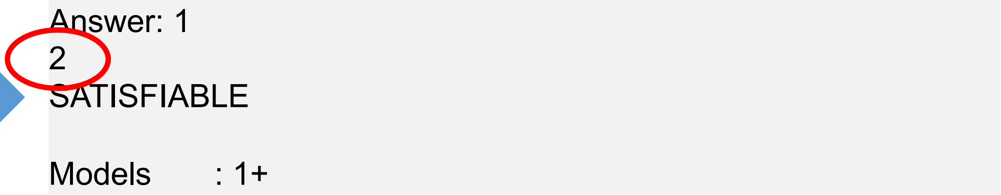
We want all other cover candidates to not be better (disjunction is required here)

The valid solution needs to be a cover and have some size K

All other cover candidates must be invalid (not a cover or >= K)

Saturate all other cover candidates if invalid

Only show the single entry K in "minvc(K)"



```
Answer: 1
2
SATISFIABLE

Models       : 1+
```

Clingo example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/clingo

Outline: T1-4: Datalog & ASP

- Datalog
- Answer Set Programming
  - Intro to Rules with Negation
  - Horn clauses and Logic Programming
  - Stable model semantics
  - An application and surprising complexity result
  - The power of Disjunctions
  - [A surprising application: automating hardness proofs: moved to T2-U4: Reverse Data Management]