Databases

http://www.dcs.fmph.uniba.sk/~plachetk

/TEACHING/DB2

http://www.dcs.fmph.uniba.sk/~sturc/databazy/rldb

Tomáš Plachetka, Ján Šturc

Faculty of mathematics, physics and informatics, Comenius University, Bratislava

Summer 2023–2024

Logic

Aristotle: *There is only one logic. Those who think otherwise, think illogically*

Questions:

What is precise reasoning?

How to decide whether a statement is true or false?

How to decide whether a theory is consistent?

How to deal with inconsistency?

…

We are working with **first-order logic** (extended with grouping and aggregation). Only variables can be quantified

Datalog program perceived as a theory

B. Russel: *If we allow for one false statement (which makes a theory inconsistent), everything can be proved*

A Datalog program is an axiomatic theory. The axioms are implications which define IDB predicates (intensional database). Consequences in all the implications are positive. The EDB (extensional database) contains only positive facts. **The theory corresponding to a Datalog program is always consistent**

The only rule allowed in proofs is **modus ponens**
This makes **impossible to prove that something is false**

We have to clarify the meaning of negation. Negation is perceived differently in whole areas of life and science

Logic: science and negation

Mathematical sciences

Something is true only if it is true in the most general understanding of the theory in hand (i.e. it is true in all models of the theory)

Alternative: only what can be proved is true

Empirical sciences (physics, medicine)

Everything is assumed to be true, until someone experimentally demonstrates that it is not

Logic: science and negation

Law

Everything is true, unless a law does not state otherwise

Laws can contradict each other. The following rules resolve the contradictions:

- Lex superior derogat legi inferiori
- Lex posterior derogat legi priori
- Lex specialis derogat legi generali

Cicero: *Ius summum saepe summa est iniuria. (S*upreme law is often supreme injustice.*)*

Theories and models

A **theory** is a set of formulas (axioms)
A **model of a theory** is an interpretation for which at least the axioms of the theory hold (i.e. which does not contradict the theory)
The minimal model of a theory is a model such that none of its proper non-empty subsets is a model of the theory

In mathematical logic, a model requires a **domain** (values which can be assigned to variables, concrete functions which are assigned to functional symbols etc.)

We are interested in formulas which do not depend on the domain. **A domain independent formula depends only on the "contents of predicates"** (i.e. not on their types)

Domain independent and safe formulas

Safe formulas form a subclass of domain independent formulas, i.e. every safe formula is domain independent

It cannot be algorithmically decided whether a **formula is domain independent** or not

But there is an **algorithm** which decides whether an arbitrary **formula is safe** or not (the definition of safety requires an algorithm, which makes the decision using solely a syntax analysis)

Theories, models, databases

Datalog program together with an EDB and possibly with a query is a theory

Indeed, a Datalog program (with EDB and possibly with a query) can be rewritten into a single formula as a conjunction of implications, in conjunction with EDB, in conjunction with a query. All the variables in implications are universally quantified, except of the variables in the query (which remain free)

Datalog without negation, minimal model = fix-point

The semantics of **Datalog programs without negated subgoals** agrees with the mathematical semantics. **The minimal model is unique (it is the least model).** What holds in this minimal model is exactly what holds in all the models of the program

The minimal model is equal to the fix-point which results from naïve iteration

Datalog with negation

Datalog with negated subgoals does not always have a fix-point

Example: the program $p \leftarrow \neg p$.

has no fix-point, i.e. naïve iteration does not terminate.

Its minimal model is {p}

Datalog with negation

Datalog with negation can have several minimal models

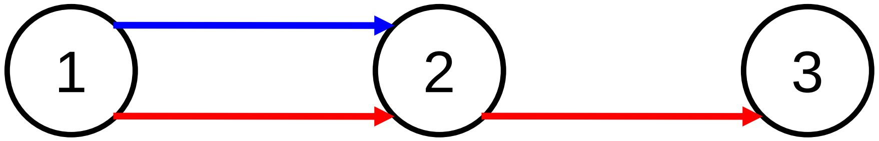
Example:

bluepath(X, Y) $\leftarrow$ blue(X, Y)

bluepath(X, Y) $\leftarrow$ blue(X, Z), bluepath(Z, Y)

redmonopol(X, Y) $\leftarrow$ red(X, Y), $\neg$ bluepath(X, Y)

EDB = {blue(1, 2), red(1, 2), red(2, 3)}



Two minimal models:

EDB $\cup$ {bluepath(1, 2), redmonopol(2, 3)} (a natural model)

EDB $\cup$ {bluepath(1, 2), bluepath(2, 3), bluepath(1, 3)}

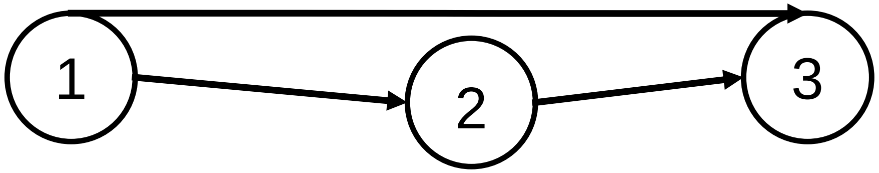
But only one fixpoint:

EDB $\cup$ {bluepath(1, 2), redmonopol(2, 3)}

Datalog with negation

For some non-stratified programs, fix-point agrees with the minimal model(s)

Example: win(X) ← move(X, Y), ¬ win(Y).
EDB = {move(1, 2), move(1, 3), move(2, 3)}



Naïve iteration terminates
The minimal model = fix-point =
EDB ∪ {win(1), win(2)}

Datalog with negation

… but it depends on the data (contents of the EDB) whether naïve iteration terminates (i.e. finds a fixpoint)

The same program: win(X) ← move(X, Y), ¬ win(Y).
EDB = {move(1, 2), move(2, 1), move(2, 3), move(3, 4), move(4, 5), move(5, 6)}

$$\begin{tikzcd} 1 \arrow[r, bend right] & 2 \arrow[l, bend right] \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \end{tikzcd}$$

Two nonempty minimal models (and fixpoints with respect to a single iteration step of naïve iteration):
EDB ∪ {win(1), win(3), win(5)}
EDB ∪ {win(2), win(3), win(5)}
Naïve iteration starting with win=∅ does not terminate

Datalog with negation: locally stratified programs

Minimal model = fix-point is guaranteed for a class of programs called **locally stratified programs** (which includes the class of stratified programs). **Local stratification depends not only on the program, but also on the EDB**

Procedural definition of local stratification:

1. Create a dependence graph of *instantiated* IDB atoms
   - Vertices are instantiated IDB atoms
   - An edge p→q exists if q occurs in an instantiated rule with head p. The edge is labeled with '¬' if q is negated in that rule
2. Strata (integer numbers representing levels) are assigned to instantiated atoms (vertices of the dependence graph) as in the global stratification
3. If the number of strata is finite, then the program is locally stratified

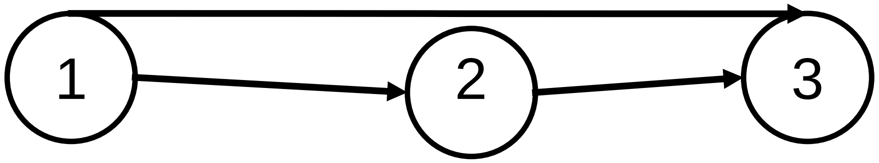
An equivalent definition of local stratification replaces Steps 2 and 3 with:

A program with an EDB is locally stratified if the dependence graph of instantiated IDB atoms does not contain a cycle with a negated edge

Locally stratified programs: minimal model = fix-point

Example: win(X) ← move(X, Y), ¬ win(Y).
EDB = {move(1, 2), move(1, 3), move(2, 3)}
The minimal model = fix-point = EDB ∪ {win(1), win(2)}

Instantiated program:
win(1) ← move(1, 2), ¬ win(2).
win(1) ← move(1, 3), ¬ win(3).
win(2) ← move(2, 3), ¬ win(3).



This program **is locally stratified for the given EDB**, because the instantiated IDB atoms can be assigned a finite number of strata:

$$\begin{tikzcd}
\text{win(1)} & \text{win(2)} \arrow[l, "\neg"'] & \text{win(3)} \arrow[l, "\neg"'] \arrow[ll, bend right, "\neg"']
\end{tikzcd}$$

stratum 3 stratum 2 stratum 1

Beyond locally stratified programs

Motivatory example: consider rules of a two-player game

EDB = {move(., .)}

move1(X) $\leftarrow$ move(X, _).

position(X) $\leftarrow$move(X, _).

position(X) $\leftarrow$ move(_, X).

loss(X) $\leftarrow$ position(X), $\neg$ move1(X). /* mate */

loss(X) $\leftarrow$ position(X), $\neg$ goodmove(X).

goodmove(X) $\leftarrow$ move(X, Y), $\neg$ win(Y).

win(X) $\leftarrow$ move(X, Y), loss(Y).

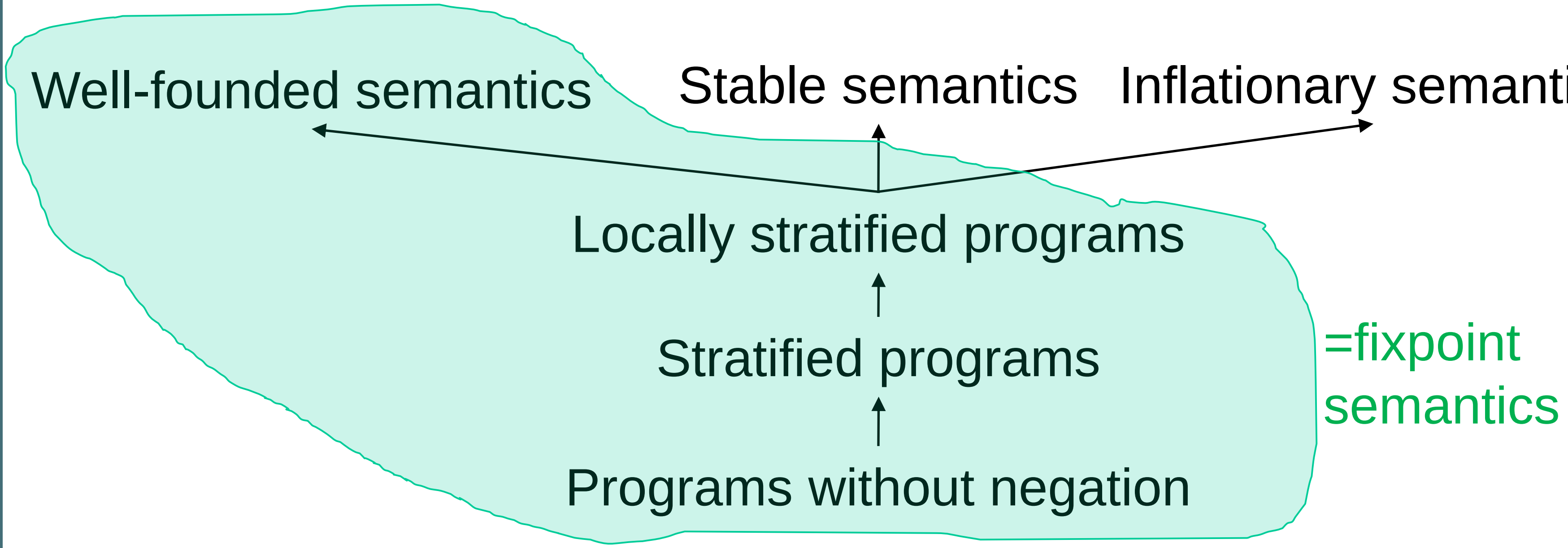
draw(X) $\leftarrow$ position(X), $\neg$ win(X), $\neg$ loss(X).

The definitions of predicates win(.), loss(.) and draw(.) are cyclic. It is perhaps possible to make the program locally stratified for an arbitrary game, but this requires a deeper understanding of each specific game. It is better to handle an arbitrary game with no human assistance. This requires to a more general semantics

Beyond locally stratified programs

Even if a program is not locally stratified, answers to queries to such a program can be computed with respect to some model other than a minimal model

The choice for the semantics includes **stable models, the well-founded model, the inflationary model**. The crucial concept in the construction of these models is **Gelfond-Lifschitz transformation**



Gelfond-Lifschitz Transformation (GLT)

Gelfond-Lifschitz transformation transforms a Datalog program into a Datalog program with no negated IDB subgoals, with respect to a partial interpretation M

Hence, **it also transforms M to an interpretation GLT(M) with respect to the program and the interpretation M**

Definition: **GLT(M) w.r.t. a partial interpretation M and a program S** is the *minimal model* of a program consisting (only) of:

1. **rules of S with no IDB predicates in their bodies**;

2. rules

$\boldsymbol{H \leftarrow P_1, \ldots, P_n}$ for which there exists a clause in S

$H \leftarrow P_1,\ \ldots,\ P_n,\ \neg N_1,\ \ldots,\ \neg N_k$

such that $N_1, \ldots, N_k \notin M$ (i.e., none of $N_1, \ldots, N_k$ belongs to M).

Gelfond-Lifschitz Transformation (GLT)

Given a program P and a partial interpretation M (in which the EDB and true built-in atoms are true). GLT(M) can then be computed as a sequence of the following steps:

1. **Instantiate** the rules of P in all possible ways
2. **Delete all the rules** with a goal of the form "¬ <atom>", where the IDB <atom> is in M
3. **Delete all the goals** of the form "¬ <atom>", where the IDB <atom> is not in M
4. Use the remaining rules to compute the fixpoint (i.e., to **infer all the IDB atoms from the reduced program and the EDB**)

The set of IDB atoms inferred in step 4 united with the EDB atoms and with true built-in atoms is the resulting interpretation GLT(M)

Gelfond-Lifschitz Transformation (GLT)

Example: program win(X) $\leftarrow$ move(X, Y), $\neg$ win(Y).

EDB = {move(1, 2), move(1, 3), move(2, 3)}

A partial interpretation: M = EDB $\cup$ {win(1), win(2)}

1. Instantiation (we can immediately omit rules in which move(., .) is not in EDB, they will be deleted anyway in Step 2):

win(1) $\leftarrow$ move(1, 2), $\neg$ win(2). r1

win(1) $\leftarrow$ move(1, 3), $\neg$ win(3). r2

win(2) $\leftarrow$ move(2, 3), $\neg$ win(3). r3

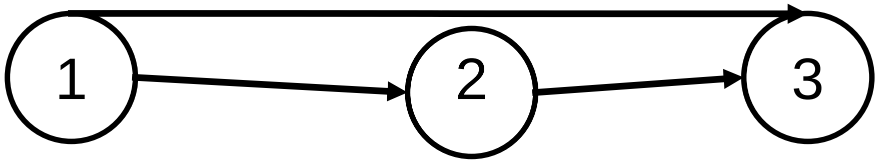
2. Delete r1, because win(2) $\in$ M

3. Delete $\neg$ win(3) from r2 and r3, because win(3) $\notin$ M. Delete true EDB goals move(1, 3) from r2 and move(2, 3) from r3

4. What remains is

win(1). r2'

win(2). r3'



GLT(M) = EDB $\cup$ {win(1), win(2)}

Gelfond-Lifschitz Transformation (GLT)

Example: program

$p \leftarrow q, \neg r, \neg s.$ r1

$r \leftarrow q.$ r2

$q \leftarrow \neg s.$ r3

A partial interpretation: $M = \emptyset$. EDB = $\emptyset$.

Step 1: nothing to instantiate

Step 2: nothing

Step 3: delete subgoals $\neg r$ and $\neg s$, because $r, s \notin M$

Step 4: What remains is

$p \leftarrow q.$ r1’

$r \leftarrow q.$ r2’

$q.$ r3’

GLT(M) = {p, q, r}

Gelfond-Lifschitz Transformation (GLT)

Example: program

$p \leftarrow q, \neg r, \neg s.$ r1

$r \leftarrow q.$ r2

$q \leftarrow r.$ r3

A partial interpretation: $M = \varnothing$. EDB = $\varnothing$.

Step 1: nothing to instantiate

Step 2: nothing

Step 3: delete subgoals $\neg$ r and $\neg$ s, because r, s $\notin$ M

Step 4: What remains is

$p \leftarrow q.$ r1’

$r \leftarrow q.$ r2’

$q \leftarrow r.$ r3’

GLT(M) = $\varnothing$

Gelfond-Lifschitz Transformation (GLT)

Example: program

$p \leftarrow q, \neg r, \neg s.$ r1

$r \leftarrow q.$ r2

A partial interpretation: $M = \{q\}$. EDB = $\emptyset$.

Step 1: nothing to instantiate

Step 2: nothing

Step 3: delete subgoals $\neg r$ and $\neg s$, because $r, s \notin M$

Step 4: What remains is

$p \leftarrow q.$ r1'

$r \leftarrow q.$ r2'

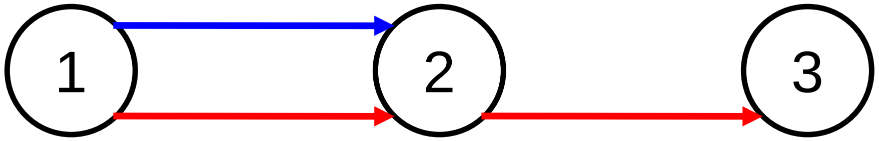
GLT(M) = $\emptyset$

Although $q \in M$, the program does not **force** q to be true (indeed, there is no rule which implies q)

Stable models

A model M is **stable** if M = GLT(M).
In other words, a stable model is a model which is invariant to GLT

Example:
bluepath(X, Y) ← blue(X, Y)
bluepath(X, Y) ← blue(X, Z), bluepath(Z, Y)
redmonopol(X, Y) ← red(X, Y), ¬ bluepath(X, Y)
EDB = {blue(1, 2), red(1, 2), red(2, 3)}



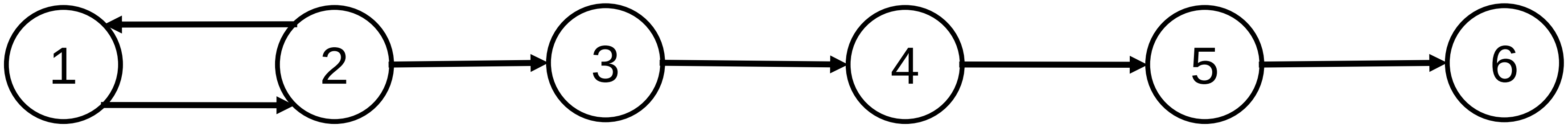
Stable model:
EDB ∪ {bluepath(1, 2), redmonopol(2, 3)} (a natural model)

This looks good so far. But...

Stable models

Example: win(X) ← move(X, Y), ¬ win(Y).

EDB = {move(1, 2), move(2, 1), move(2, 3), move(3, 4), move(4, 5), move(5, 6)}



Two stable models:

EDB ∪ {win(1), win(3), win(5)}

EDB ∪ {win(2), win(3), win(5)}

Generally, there can be several stable models for a program with an EDB (it can also happen that there is no stable model). This non-uniqueness causes problems, because a query ?-win(X) could compute {1, 3, 5} as well as {2, 3, 5}

Well-founded model

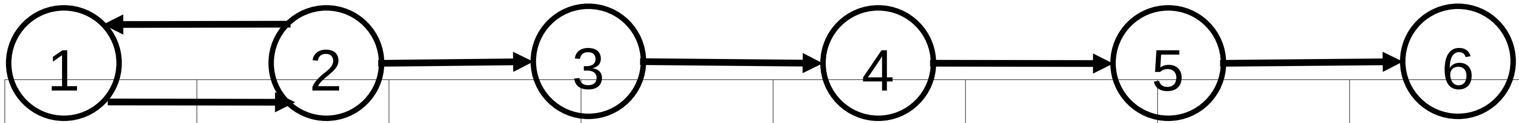
The **well-founded model** is unique. It is a (minimal stable) **3-valued model**. An IDB atom is either **true**, **false** or **unknown**.

A procedural definition of the well-founded model uses so-called **alternating-fixpoint** obtained by a reiteration of GLT, starting with an empty partial model. In odd iteration rounds, the set of true (instantiated) IDB atoms increases. In odd rounds, the set of true IDB atoms decreases. For a finite set of IDB atoms, it is guaranteed that after a finite number of iterations this process converges, i.e. the values of all the IDB atoms in two subsequent even rounds or two subsequent odd rounds are equal. In all further iterations, the truth value of some IDB atoms does not change, these are true or false respectively in the well-founded model. The remaining IDB atoms are unknown

Well-founded model

Example: win(X) ← move(X, Y), ¬ win(Y).

EDB = {move(1, 2), move(2, 1), move(2, 3), move(3, 4), move(4, 5), move(5, 6)}


$$\begin{tikzcd} 1 \arrow[r, bend right] & 2 \arrow[l, bend right] \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \end{tikzcd}$$

	It. 0	It. 1	It. 2	It. 3	It. 4	It. 5	Truth value
win(1)	F	T	F	T	F	T	unknown
win(2)	F	T	F	T	F	T	unknown
win(3)	F	T	F	T	T	T	true
win(4)	F	T	F	F	F	F	false
win(5)	F	T	T	T	T	T	true
win(6)	F	F	F	F	F	F	false

The truth values in iterations 3 and 5 are the same. Iteration 6 would thus result in the same truth values as iteration 4, etc.

The query ?- win(X) returns {3, 5} with respect to WFM

Well-founded model

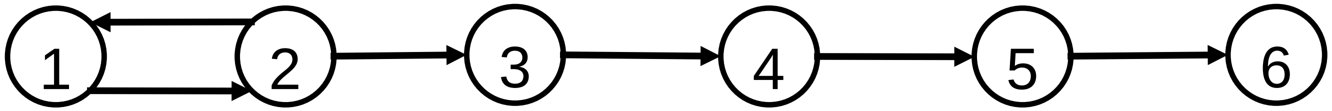
Important properties of the well founded model:

- **If an instantiated IDB atom is true in the well-founded model, then it is true in all stable models**
- **If an instantiated IDB atom is false in the well-founded model, then it is false in all stable models**
- **If no instantiated IDB atom is unknown in the well-founded model, then there is exactly one stable model which equals the well-founded model**

The distinction between false and unknown atoms is not important in answering database queries. Only the truths appear in results

Alternative for semantics: Inflationary model

A procedural definition of the **inflationary model** resembles the alternating fix-point construction of the well-founded model. But when an IDB atom once becomes true, it remains true forever


$$\begin{tikzcd} 1 \arrow[r, bend right] & 2 \arrow[l, bend right] \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \end{tikzcd}$$

Example: win(X) $\leftarrow$ move(X, Y), $\neg$ win(Y).

EDB = {move(1, 2), move(2, 1), move(2, 3), move(3, 4), move(4, 5), move(5, 6)}

win(1)	F	T	T	true
win(2)	F	T	T	true
win(3)	F	T	T	true
win(4)	F	T	T	true
win(5)	F	T	T	true
win(6)	F	F	F	false

The query ?- win(X) returns {1, 2, 3, 4, 5}

Other alternatives for semantics

Negation as inconsistency (Gabbay, Sergot). Idea: for a given Datalog program P and a partial model M, a formula ϕ holds if ϕ is consistent with $P \cup M$. This approach requires storing also the negative facts in the database

Disjunctive Datalog (e.g. DLV project, http://www.dlvsystem.com). Disjunctive Datalog extends usual Datalog rules with disjunction in the heads of the rules. Moreover, the atoms in the heads can be negated. This allows e.g. for the definition of the *law of excluded third*, $p \vee \neg p$ (which cannot be expressed in usual Datalog)

Similar choices for the semantics find applications e.g. in automatic theorem proving and semantic web