Updated 2/15/2023

Topic 1: Data models and query languages
Unit 4: Datalog
Lecture 11

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp23)

https://northeastern-datalab.github.io/cs7240/sp23/

2/14/2023

Pre-class conversations

- Last class summary
- Project ideas
- Scribe on explaining the chase procedure

- today:
  - Adding negation to Datalog "in the right way".
  - Writing a program in 4 lines of code to find 3-coloring of a graph
- Next time
  - Neha on how to write 3-coloring in 2 lines

Outline: T1-4: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Recursion in SQL [moved here from T1-U1: SQL]
  - Semantics
  - Datalog$^{\neg}$: Negation, stratification
  - Datalog$^{\pm}$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation (incl. Incremental View Maintenance)

Horn clauses and Logic Programming

Horn clauses and logic programming

A clause is a disjunction of literals.

$$\bar{a} \vee \bar{b} \vee c \vee d \qquad a \wedge b \Rightarrow c \vee d$$

$$1 \wedge a \wedge b \Rightarrow c \vee d \vee 0$$

A Horn clause has at most one positive (i.e. unnegated) literal.

?

Alfred Horn, ~1973
https://en.wikipedia.org/wiki/Alfred_Horn

Recall: $\bar{a} = \neg a = !a = {\sim}a = \text{NOT } a$

Horn clauses and logic programming

A clause is a disjunction of literals.

$$\bar{a} \lor \bar{b} \lor c \lor d \qquad\qquad a \land b \Rightarrow c \lor d$$

$$1 \land a \land b \Rightarrow c \lor d \lor 0$$

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.

$\bar{a} \vee \bar{b} \vee c \vee d$ $\quad\quad$ $a \wedge b \Rightarrow c \vee d$

$1 \wedge a \wedge b \Rightarrow c \vee d \vee 0$

Alfred Horn, ~1973
https://en.wikipedia.org/wiki/Alfred_Horn

A Horn clause has at most one positive (i.e. unnegated) literal.

$\bar{a} \vee \bar{b} \vee c$ $\quad\quad$ $a \wedge b \Rightarrow c$ $\quad$ definite clause (exactly one positive)

c $\quad\quad$ $1 \Rightarrow c$ $\quad$ unit clause (facts, unconditional knowledge, empty body)

$\bar{a} \vee \bar{b}$ $\quad\quad$ $a \wedge b \Rightarrow 0$ $\quad$ goal clause

Universal quantification (everything above was propositional)

$\neg \text{human}(X) \vee \text{mortal}(X)$

? $\quad\quad$?

Recall: $\bar{a} = \neg a = !a = {\sim}a =$ NOT a

Horn clauses and logic programming

A clause is a disjunction of literals.

$$\bar{a} \vee \bar{b} \vee c \vee d \qquad\qquad a \wedge b \Rightarrow c \vee d$$

$$1 \wedge a \wedge b \Rightarrow c \vee d \vee 0$$

A Horn clause has at most one positive (i.e. unnegated) literal.

$\bar{a} \vee \bar{b} \vee c$ $\qquad a \wedge b \Rightarrow c$ $\qquad$ 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)

$$\neg \text{human}(X) \vee \text{mortal}(X)$$

$$\forall X[\neg \text{human}(X) \vee \text{mortal}(X)] \qquad\qquad \forall X[\text{human}(X) \Rightarrow \text{mortal}(X)]$$

Recall: $\bar{a} = \neg a = !a = {\sim}a = \text{NOT } a$

Datalog grammar

$P \in$ program $= r_1.\ r_2.\ \ldots\ r_n.$

$r \in$ rule $= a_0$:- $a_1, \ldots, a_m.$

$a \in$ atom $= p(t_1, \ldots, t_k)$

$t \in$ term $= x \mid$ "c"

p = set of predicate symbols

x = set of variable symbols

c = set of constants

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 ?

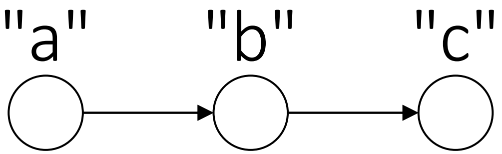
Concepts from logic programming

- 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_H
- 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

?

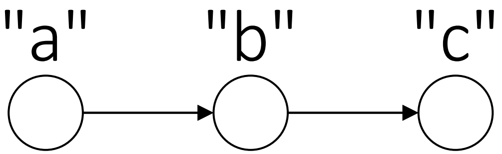
Herbrand universe U_P

?

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

?

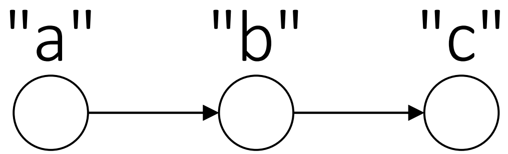
Herbrand universe U_P

{"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"}

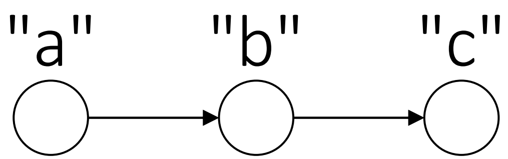
Herbrand base B_P

{ 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")

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").

Is this interpretation a model? ?

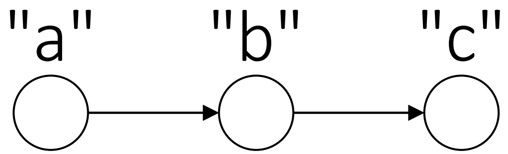
Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P

{ 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

{ 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"). }

Interpretation

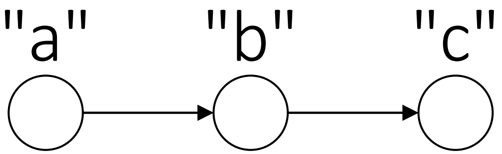
arc("a","b"). arc("b","c"). arc("b","a").
path("a","b"). path("b","c"). path("b","a").
path("a","c"). path("a","a").

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→"a", y→"a", z→"b":
path("b","b") :- arc("b","a"), path("a","b").

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").

Is this interpretation a model?

?

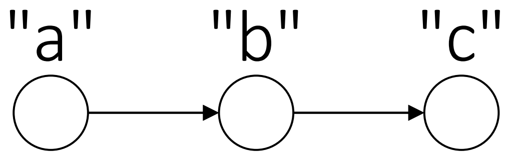
Herbrand universe U_P

{"a", "b", "c"}

Herbrand base B_P

{ 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

{ 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"). }

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").

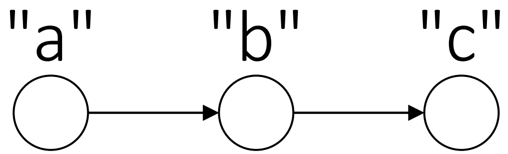
Is this interpretation a model?

Yes!

Is this model minimal?

?

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

{ 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"). }

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 interpretation a model?

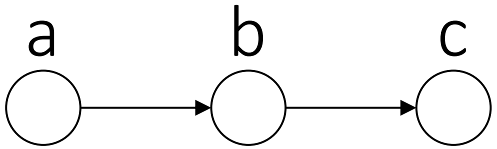
Yes!

Is this model minimal?

No! There is a properly contained model

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, 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

{ 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). }

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

Is this interpretation a model?

Yes!

Is this model minimal?

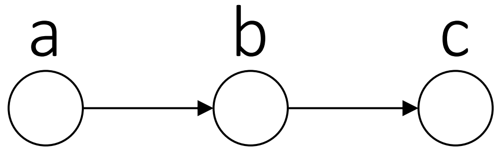
No! There is a properly contained model

arc(a,b). arc(b,c).
path(a,b). path(b,c).
path(a,c).

Evaluating ASP's with DLV ("DataLog with Disjunction")

paths.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```



paths

```
./dlv --silent paths.txt
```

prevents printing the DLV version

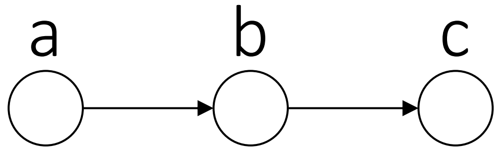
?

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Evaluating ASP's with DLV ("DataLog with Disjunction")

paths.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```



paths

```
./dlv --silent paths.txt
```

prevents printing the DLV version

```
{path(a,b), path(b,c), path(a,c),
arc(a,b), arc(b,c)}
```

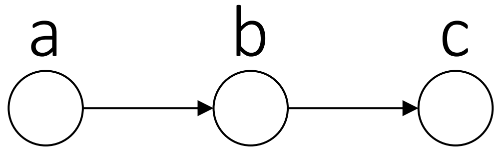
dlvsystem
SPIN-OFF OF UNIVERSITY OF CALABRIA

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv

Evaluating ASP's with DLV ("DataLog with Disjunction")

paths.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```




```
./dlv --silent paths.txt pathsquery.txt
```

contains the query

pathsquery.txt

```
path(X,Y) ?
```

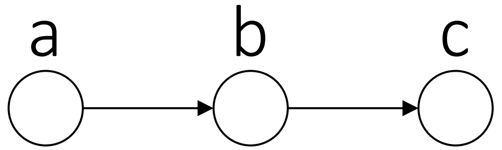
```
{path(a,b), path(b,c), path(a,c)}
```

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv

Evaluating ASP's with DLV ("DataLog with Disjunction")

paths.txt

```
arc(a,b). arc(b,c).
path(X,Y) :- arc(X,Y).
path(X,Y) :- arc(X,Z), path(Z,Y).
```



paths

```
./dlv --silent --no-facts
```

prevents printing the known facts

```
{path(a,b), path(b,c), path(a,c)}
```

```
./dlv --help
```

… to find more output options

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Back to
ASP (Answer Set Programming) and "Stable Models"

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

$\boxed{\cancel{h \text{ :- } a_1, \ldots, a_m, \neg b_1, \ldots, \neg b_n.}}$ if some $b_i \in M$

3. In remaining grounded rules, delete all negative literals

$\boxed{h \text{ :- } a_1, \ldots, a_m, \cancel{\neg b_1, \ldots, \neg b_n}.}$ if no $b_i \in M$

M is a stable model of P iff M is the least model of P^M

Semantics: "non-circular" more concisely

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 <u>least model</u> of P^M

Examples

P1:

```
a :- a.
```

M={a} Is M a stable model of P1? ?

Examples

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)

? What is a stable model?

Examples

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.

?

{ {a}, {b},
{}, {a,b} }

Examples

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.

?

{ {a}, {b}, {}, {a,b} } → ~~a :- not b.~~ → {}

Examples

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.`

?

{ {a}, {b}, {}, {a,b} }

a :- ~~not b.~~ ⟶ {a}

~~a :- not b.~~ ⟶ {}

Examples

P1: `a :- a.`

M={a} not a stable model (not minimal, derivation of "a" is based on circular reasoning)

M={} stable model

P2: `a :- not b.`

{ {a}, {b}, {}, {a,b} }

a :- ~~not b.~~ → {a}

~~a :- not b.~~ → {}

M={a}

only stable model (contrast with the earlier chess example)

P3: `a :- not a.`

{ {}, {a} }

?

Examples

P1: `a :- a.`

M={a} not a stable model (not minimal, derivation of "a" is based on circular reasoning)

M={} stable model

P2: `a :- not b.`

M={a}
only stable model

{ {a}, {b}, {}, {a,b} }

~~a :- not b.~~ → {a}

~~a :- not b.~~ → {}

P3: `a :- not a.`

{ {}, {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.

M_1={a}

M_2={b}

two stable models

a ~~:- not b~~.

~~b :- not a.~~

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.
```

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.
```

$M=\{a\}$ 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 DLV ("DataLog with Disjunction")

p4, p5

p4.txt

```
a :- not b.
b :- not a.
```

M_1={a}

M_2={b}

```
./dlv p4.txt –n 0
```

print all stable models (not just one)

```
{b}
{a}
```

p5.txt

```
a :- not b.
b :- not a.
a :- not a.
```

M={a}

```
./dlv p5.txt –n 0
```

```
{a}
```

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv

What do empty bodies or heads mean in ASP?

```
a :- b, not c.
```

Think of the head as a disjunction, body as conjunction

$0 \lor a \Leftarrow 1 \land b \land \neg c$

DLV = "DataLog with Disjunction (=V)"

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

$0 \vee a \Leftarrow 1 \wedge b \wedge \neg c$

DLV = "DataLog with Disjunction (=V)"

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

$0 \vee a \Leftarrow 1 \wedge b \wedge \neg c$

DLV = "DataLog with Disjunction (=V)"

Empty body:

```
a.
```

$a \Leftarrow 1$

Empty body describes a fact:
"a" needs to be true.
Also in Datalog

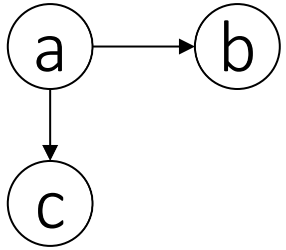
Empty head:

```
:- b, not c.
```

$0 \Leftarrow b \wedge \neg c$

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



Q: For a graph (V, E) assign each vertex a color in {1, 2, 3} such that no adjacent vertices have the same color.

?

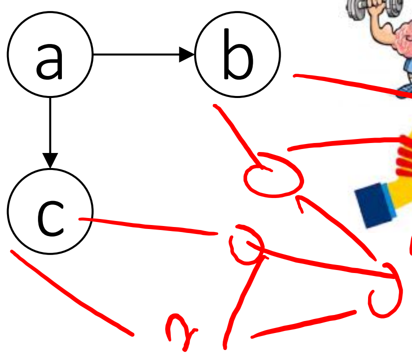
Convention in ASP: Capital letters are variables, lower case letters constants

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

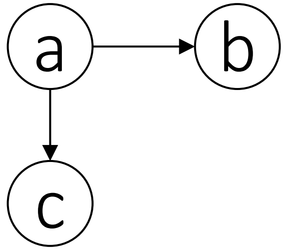
Convention in ASP: Capital letters are variables, lower case letters constants

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).
```

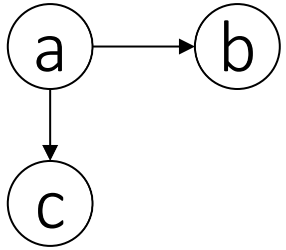
Convention in ASP: Capital letters are variables, lower case letters constants

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).
```

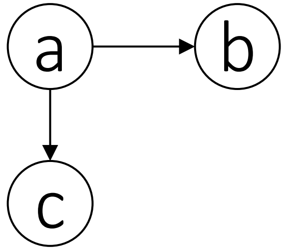
Convention in ASP: Capital letters are variables, lower case letters constants

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).
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).
```

IDB

constraint

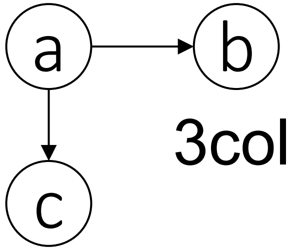
Convention in ASP: Capital letters are variables, lower case letters constants

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



3colorability

```
./dlv 3colorability.txt --silent --no-facts
```

3colorability.txt

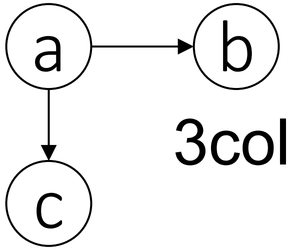
```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
:- edge(V,U), color(V,C), color(U,C).
```

```
{color(a,1), color(b,3), color(c,3)}
```

dlvsystem SPIN-OFF OF UNIVERSITY OF CALABRIA

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

3-colorability



3colorability

```
./dlv 3colorability.txt --silent --no-facts -n 0
```

print all stable models (not just one)

3colorability.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).
```

```
{color(a,1), color(b,3), color(c,3)}
{color(a,1), color(c,2), color(b,3)}
{color(a,2), color(b,3), color(c,3)}
{color(b,1), color(a,2), color(c,3)}
{color(b,1), color(c,1), color(a,2)}
{color(c,1), color(a,2), color(b,3)}
{color(a,1), color(b,2), color(c,3)}
{color(a,1), color(b,2), color(c,2)}
{color(b,2), color(c,2), color(a,3)}
{color(c,1), color(b,2), color(a,3)}
{color(b,1), color(c,1), color(a,3)}
{color(b,1), color(c,2), color(a,3)}
```

dlvsystem
SPIN-OFF OF UNIVERSITY OF CALABRIA

DLV available for download at: https://dlv.demacs.unical.it/
DLV example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/dlv
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/