Matteo Palmonari

matteo.palmonari@disco.unimib.it

Reasoning for Datalog and Beyond with DLV

*Hands-on Session: Exercises **with Solutions***

INSID&S Lab

***In**teraction and **S**emantics for*

***I**nnovation with **D**ata **&** **S**ervices*

Datalog & NMR Hands-on Session

*Answer Set Programming with the DLV system
*Datalog-based Reasoning with ASP/DLV
*Common-sense Reasoning with ASP/DLV

Structure of the Hands-on Session

- Answer Set Programming with the DLV system
  - Short presentation and links to online material
- Reasoning with Definite Logic Programs
  - The level that you are required to master (pure Datalog)
- Reasoning with more sophisticated Programs (Disjunctive and Common-sense Reasoning)
  - You are not “required“ to get here (= you won’t be *asked to know* how to write programs to handle these inferences). But I think it’s really interesting and you may have fun trying some examples. It could also be objective of a course project.
  - How to: more intuitive approach with the DLV tutorial (see next slides); uploaded papers for theoretical background
- Discussion in video-call

Datalog & Answer Set Programming

- We use a tool for ASP to exercise with Datalog
  - ASP is a generalization of (pure Datalog = definite programs = programs with definite clauses)
- Resources:
  - Main & download page for DLV
    - http://www.dlvsystem.com/dlv/
    - New versions exist (DLV2) – for this exercises I suggest using the versions on this page
  - Tutorial
    - http://www.dlvsystem.com/html/The_DLV_Tutorial.html
    - Excellent tutorial: intuitive presentation but very precise and full of (executable) examples
  - User manual
    - http://www.dlvsystem.com/html/DLV_User_Manual.html

Datalog & Answer Set Programming

- Command line tool: we use it in the simplest mode
  - Write programs in .txt files
  - Execute these files (w. parameters)
  - Results printed on the screen
    - Very easy to make experiments and understand: you modify the .txt, save and run again the command; more advanced extensions to read files from DBs / RDF / etc exist.
- We'll cover a few of the DLV functionalities
  - … but the tutorial is great if you want to explore more

Getting started

- Invoke DLV (check use manual, Ch.1 and 6)

```
$ DLV
DLV [build BEN/May 23 2004   gcc 2.95.4 20011002 (Debian prerelease)]

usage: DLV {FRONTEND} {OPTIONS} [filename [filename [...]]]

Specify -help for more detailed usage information.
```

- Example

```
dlv.bin tweety2.txt madonna.txt
```

Syntax Example: Program + Query

Tweety2.txt

```
fly(X) :- bird(X), not -fly(X).
-fly(X) :- penguin(X).
bird(X) :- penguin(X).
bird(tweety).
penguin(skippy).
```

Tweety2-query.txt

```
fly(X)?
```

We need to specify a **reasoning mode** (brave | cautious)

- dlv.bin Tweety2.txt Tweety2-query.txt

- Non-ground queries are only supported with brave and cautious reasoning.

- dlv.bin tweety2.txt Tweety2-query.txt –brave

- tweety

Brave vs. Cautious Reasoning

- Brave
  - If in **some** answer set
- Cautious
  - If in **all** the answer sets

Tweety3.txt

```
fly(X) :- bird(X), not -fly(X).
-fly(X) :- penguin(X).
bird(X) :- penguin(X).
bird(tweety).
bird(skippy).
penguin(skippy) v -penguin(skippy).
```

Tweety2-query.txt

```
fly(X)?
```

```
dlv.bin Tweety3.txt -silent

{fly(tweety), bird(tweety), bird(skippy), -penguin(skippy), fly(skippy)}
{fly(tweety), bird(tweety), bird(skippy), penguin(skippy), -fly(skippy)}
```

Remark!

It makes a difference only for programs with possibly more answer sets
(not relevant for pure datalog = definite logic programs)

Brave vs. Cautious Reasoning

- Brave
  - If in **some** answer set
- Cautious
  - If in **all** the answer sets

Tweety3.txt

```
fly(X) :- bird(X), not -fly(X).
-fly(X) :- penguin(X).
bird(X) :- penguin(X).
bird(tweety).
bird(skippy).
penguin(skippy) :- not -penguin(skippy).
-penguin(skippy) :- not penguin(skippy).
```

Tweety2-query.txt

```
fly(X)?
```

```
dlv.bin _______.txt -silent

{fly(tweety), bird(tweety), bird(skippy), -penguin(skippy), fly(skippy)}
{fly(tweety), bird(tweety), bird(skippy), penguin(skippy), -fly(skippy)}
```

Remark!

It makes a difference only for programs with possibly more answer sets
(not relevant for pure datalog = definite logic programs)

Brave vs. Cautious Reasoning

- Brave
  - If in **some** answer set
- Cautious
  - If in **all** the answer sets

Tweety3.txt

```
fly(X) :- bird(X), not -fly(X).
-fly(X) :- penguin(X).
bird(X) :- penguin(X).
bird(tweety).
bird(skippy).
penguin(skippy) v -penguin(skippy).
```

Tweety2-query.txt

```
fly(X)?
```

```
dlv.bin Tweety3.txt -silent

{fly(tweety), bird(tweety), bird(skippy), -penguin(skippy), fly(skippy)}
{fly(tweety), bird(tweety), bird(skippy), penguin(skippy), -fly(skippy)}


dlv.bin Tweety3.txt Tweety2-query.txt -silent -brave

tweety
skippy
```

Brave vs. **Cautious** Reasoning

- Brave
  - If in **some** answer set
- Cautious
  - If in **all** the answer sets

Tweety3.txt

```
fly(X) :- bird(X), not -fly(X).
-fly(X) :- penguin(X).
bird(X) :- penguin(X).
bird(tweety).
bird(skippy).
penguin(skippy) v -penguin(skippy).
```

Tweety2-query.txt

```
fly(X)?
```

```
dlv.bin Tweety3.txt -silent

{fly(tweety), bird(tweety), bird(skippy), -penguin(skippy), fly(skippy)}
{fly(tweety), bird(tweety), bird(skippy), penguin(skippy), -fly(skippy)}


dlv.bin Tweety3.txt Tweety2-query.txt -silent -cautious

tweety
```

Parameters & Front-ends

- Check Ch. 6 of Online Manual
  - http://www.dlvsystem.com/html/DLV_User_Manual.html#AEN30

- Parameters
  - Several options, e.g., filter on certain predicates, provide support for inferences, and many other more complicated options ☺

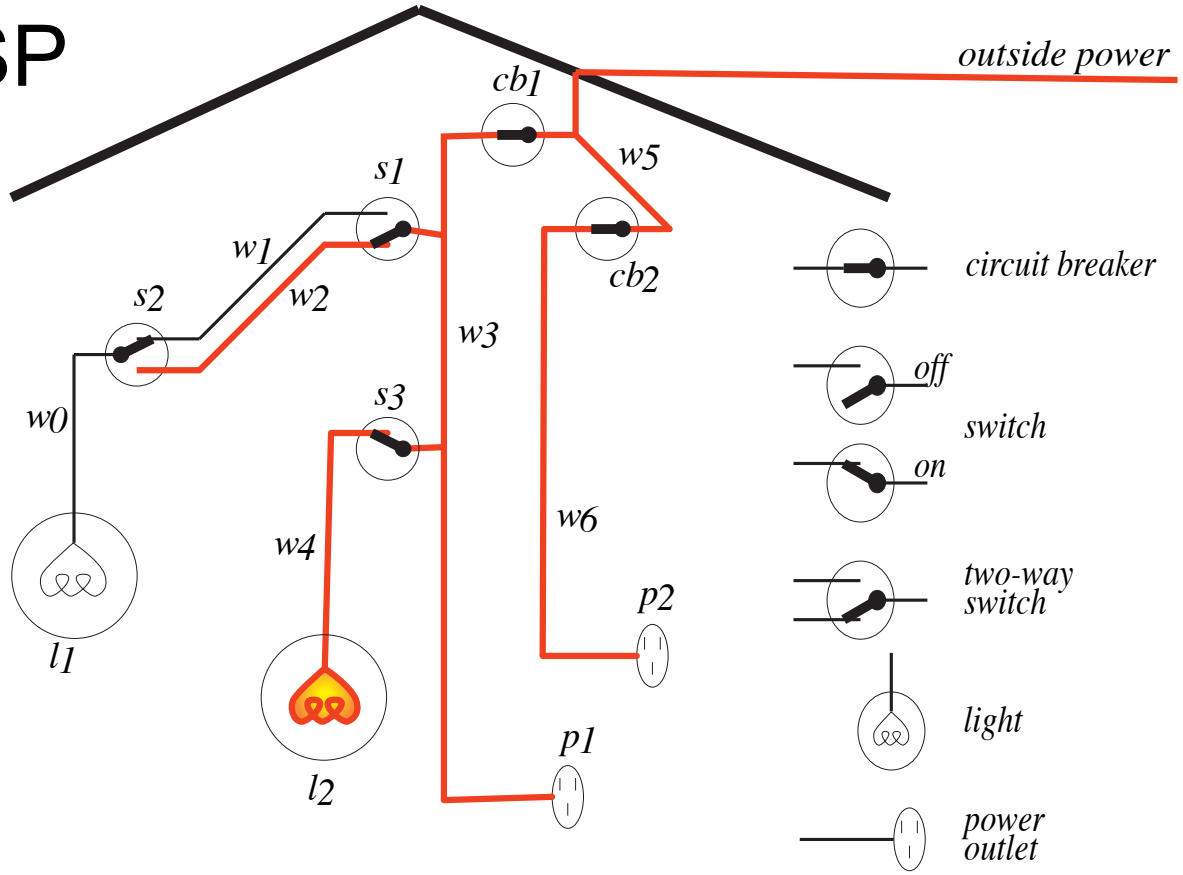
- Front-ends (+ options)
  - Front-end is for special inference paradigms that boil down to ASP
  - Diagnosis (Reiter's + Abductive)
  - Planning

Exercises with the Textbook's Electrical Engineering Example

Textbook's Electrical Environment Example - 1

Think + use the solution

- Formalize the electrical environment KB example in ASP
  - You may take the textbook formalization
- Queries:
  - What is up?
  - What is connected to w3?



Suggestion: make x versions of one file for the KB (e.g., ElectricPowerX.txt, where X is the version); make different files for each query (e.g., ElectricPower-Query1-Up.txt)

Textbook's Electrical Environment Example - 1 - Solution

ElectricPower1.txt

```
light(l1).
light(l2).

up(s3).
up(s2).
down(s1).

ok(cb2).
ok(cb1).
ok(l2).
ok(l1).

connected_to(w0,w1) :- up(s2).
connected_to(w0,w2) :- down(s2).
connected_to(w1,w3) :- up(s1).
connected_to(w2,w3) :- down(s1).
connected_to(w4,w3) :- up(s3).
connected_to(p1,w3).
```

ElectricPower-Query1-up.txt

```
up(X)?
```

ElectricPower-Query2-connectedw0.txt

```
connected_to(Y,w3)?
```

Suggestion: make x versions of one file for the KB (e.g., ElectricPowerX.txt, where X is the version); make different files for each query (e.g., ElectricPower-Query1-Up.txt)

Textbook's Electrical Environment Example - 2

- Formalize the following rules:
  - `Lit(L)`: true if the light L is lit
    - L is *lit* if L: is a *light*, is *ok*, is *live*
  - `live(C)`: true if there is power coming into C
    - C is *live* if it is *connected to* some Z that is also *live*
    - Remark: this is a recursive definition
- Make sure that from the KB you can infer
  - *l2* is lit, *l1* is not lit
    - **Remark**: formalization in previous solutions is not complete; debug the KB adding the knowledge required to support this inference

Textbook's Electrical Environment Example - 3 – Default Reasoning

- Main idea of this exercise…
- **Default reasoning:**
  - We usually don't need to specify what is *ok* (= not broken);
  - We usually assume that things are not broken unless we have explicit evidence that they are broken (or we find it via some sort of diagnosis)
  - We assume that things are *ok* by default, but we are ready to retract inferences made upon this assumption if there is evidence that something is actually broken

Textbook's Electrical Environment Example - 3 – Default Reasoning

- Remove (or comment) from the KB all literals with predicate `ok(L)`: is *l2* still lit?

- Add a default rule that specify that something is *ok* if it is not known to be <u>not *ok*</u>

Textbook's Electrical Environment Example - 3 – Default Reasoning - Plus

- Ok, now try to add some facts to prevent *l2* from being *lit*
  - Suggestion: add facts that state that something is *not ok* (what?) … run DLV and check which conclusions are retracted ... Check which piece of evidence removes *lit(l2)* from the set of conclusions.
- Default reasoning is **non-monotonic**:
  - In monotonic reasoning (e.g., FOL), the set of conclusions always increases monotonically as the set of premises increases
  - In non-monotonic reasoning (e.g., ASP), the set of conclusions may decrease even when the set of premises increases
  - We often refer to *belief revision* as to the process of retracting conclusions derived from a KB as more evidence is added to the KB

Exercises with a Simple Music KB Definite Logic Programs

Small Music KB Example

Think + use the solution + change it

- Let's build a small KB in the music domain
- Let's define the following unary predicates
  - `artist(X)`: X is an artist
  - `band(X)`: X is a band
  - `singer(X)`: X is a singer
  - `musician(X)`: X is a musician
  - `music_work(X)`: X is a music work (a single, an LP)
  - `genre(X)`: X is a genre

Small Music KB Example

Think + use the solution + change it

- Let's build a small KB in the music domain
- Let's define the following binary predicates
  - `artist_of(X,Y)`: X is the artist of the Y music work
  - `member_of(X,Y)`: X is member of the band Y
  - `music_genre(X,Y)`: Y is the genre of the music work X
  - `subgenre(X,Y)`: X is subgenre of Y
  - `plays_genre(X,Y)`: X plays music of genre Y

Music KB: 1 – Facts

Think + use the solution + change it

- Let's build a small KB in the music domain
- Let's populate the KB with some facts
  - 2+ artists (1 must be a band), and 2+ works for each artist (each student its own artists…)
  - The members of the band
  - Define a subgenre relation that specify at least three levels of specification
    - (e.g., Rock → Soft Rock → British Rock) – you can keep a small number of branches, e.g., those covering the described works
  - Add the music genre of each work (possibly, 2+ for at least one work)

Music KB: 1 – Facts – Ex. solution #1 (rule1)

```
% band and members
singer(james_brown).
band(jbs).
musician(maceo_parker).
member_of(james_brown,jbs).
member_of(fred_wesley,jbs).
member_of(maceo_parker,jbs).
member_of(bootsy_collins,jbs).
member_of(bobby_bird,jbs).

% singer
singer(anderson_paak).
musician(anderson_paak).

% songs - albums
music_work(the_grunt).
music_work(escape_ism).

% artists of songs
artist_of(jbs,the_grunt).
artist_of(jbs,escape_ism).
```

files: music-fact1.txt music-rules1.txt

- Fed up with writing all the facts? Let's use some rule to complete the KB
  - Place rules in a separate file, e.g., music-rules.txt, invoke DLV on both files
  - Use the "–nofacts" option to print only consequences and not the facts used as premises
  - X is a music_work if some artist is artist of it

```
% band and members from member_of
music_work(X) :- artist_of(_,X).
```

"_" is a unnamed variable: useful to simplify writing when the variable is not bound by anything else

Music KB: 1 – Facts – Ex. solution #2 (rule1)

```
% genres
genre(black_music).
genre(funk).
genre(jazz_funk).
genre(raw_funk).
genre(rap).
genre(trap).

% subgenres
subgenre(funk,black_music).
subgenre(rap,black_music).
subgenre(raw_funk,funk).
subgenre(jazz_funk,funk).
subgenre(trap,rap).

% music works and genre
music_genre(the_grunt,raw_funk).
music_genre(escapeism,jazz_funk).
music_genre(milk_n_honey,trap).
music_genre(tints,funk).
music_genre(tints,rap).
```

- Fed up with writing all the facts? Let's use some rule to complete the KB
  - Place rules in a separate file, e.g., music-rules.txt, invoke DLV on both files
  - Use the "–nofacts" option to print only consequences and not the facts used as premises
  - X is a music_work if some artist is artist of it

```
% band and members from member_of
music_work(X) :- artist_of(_,X).
```

"_" is a unnamed variable: useful to simplify writing when the variable is not bound by anything else

files: music-fact1.txt music-rules1.txt

Music KB: 2 – Axiomatization#1

- Infer genres from relations
  - We don't want to specify that X is a genre explicitly;
    - **Comment** in the fact program all the explicit facts with predicate `genre(X)`
  - We want to infer that X is a genre by using the relations that mention genres:
    - Add to the rules' program the following rules:
      - X is a genre if it is (subgenre of something or the supergenre of something) – [disjunction in the body?]
      - X is a genre if it is the music genre of something

Music KB: 2 – Axiomatization#2

- Inheritance with unary predicates
  - Unary predicates represent classes of individuals
  - Express logical relationships among, `artist(X)` , `singer(X)` , `musician(X)` and `band(X)` ; e.g. :
    - if X is a singer X is a musician and an artist
      - (even if you don't agree); but try to write the rule and then comment it or remove it [conjunction in the head]
    - X is an *artist* if X is a *singer* or a *musician* or a *band*

Music KB: 2 – Axiomatization#3

- Transitivity of relations
  - Introduce a new relation `transitive_subgenre(X,Y)` that represents the transitive closure of the subgenre relation
- Inverse relations
  - Introduce a relation `recorded_by(X,Y)` as the inverse of the `artist_of(X,Y)` relation
- Infer artists' genre from music genres
  - Introduce a relation `play_genre(X,Y)` that is true if an artist has music works of a certain genre, including all super-genres

TIP: you can use queries to focus on certain inferences (see also next exercise)

Music KB: 2 – Axiomatization#3 + Queries

- Query the KB

(queries with constants are based on my KB; try to adapt them to yours)

  - Find transitive subgenre of ...
  - Find all the instances of the transitive subgenre relation
  - Find what is recorded by whom
  - Find which genre is played by whom (and/or filter by specific genres)

- Try a conjunctive query; in my example:
  - Who does play music of trap and funk genres?

Exercises with a Simple Music KB NMR with Logic Programs

Music KB: 3 –Programs with NAF #1

- For simplicity from now on we'll use shared examples – from my KB (you can adapt it to your KB if you want)
- Adding default rules
  - If a music work is not associated with any genre, it is pop
    - Test:
      - Add artist_of(madonna,la_isla_bonita) to the KB
      - Infer music_genre(la_isla_bonita,pop)
      - Find which music genres are associated with (all) the music works
      - Question: how is this propagating to artists and their genres?
  - If an artist is not a musician or a band, he is a singer
- Revising inferences when more information is available
  - What if we know that *la isla bonita* is *soft funk*?
  - What if we know that *elton_john* is *a musician*?

Exercises with a Simple Music KB Disjunctive Logic Programs

Music KB: 3 – Disjuntive Programs #1

- For simplicity from now on we'll use shared examples – from my KB (you can adapt it to your KB if you want)
- Adding disjunction
  - An artist is a singer or a musician or a band
  - Let's introduce an artist *madonna* (we don't know if she's a singer, a musician, or a band)

- Removing possible answer set when more information is available
  - Ok, we know madonna it's not a band; how many answer set do we have now?
  - Ok, we know that madonna is both a singer and a musician (this may not be true ☺ in the real world)

Music KB: 3 – ALL - Solutions

- You can find my solutions in:
  - KB – facts: music-facts3.txt
  - KB – rules: music-rules3.txt
  - Queries: music-queries.txt