Answer Set Programming for the Semantic Web

Tutorial

Thomas Eiter, Roman Schindlauer (TU Wien)
Giovambattista Ianni (TU Wien, Univ. della Calabria)
Axel Polleres (Univ. Rey Juan Carlos, Madrid)

Supported by IST REWERSE, FWF Project P17212-N04, CICyT project TIC-2003-9001-C02.

Unit 7 – Hands-On Session

European Semantic Web Conference 2006

Unit 7: Hands-On Session

Each of the previous units was accompanied with small practical examples which you could follow over the Web-Interface to DLV.

Now: Try yourself!

Practice and combine your experiences from the different units in several exercises. Your tutor has the details!

Discover how to manipulate your online calendar ICAL/RDF data from an ASP application.

Specify an appointment matching strategy using declarative programming.

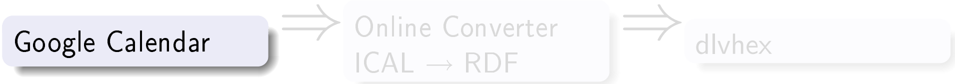
Grab a Tutor and get started!

The tutors will provide sets of new examples to be solved and can reexplain exercises from the different units. Don't hesitate to ask questions and let us know your opinions!

The Calendar Example

- The Google Calendar is available in machine readable format
  1. Ask for your team number (in range $1 \ldots 6$)
  2. Login at: `http://calendar.google.com`
  3. User: `teamX@gibbi.com`, where $X = 1 \ldots 6$
  4. Pass: `passwo`
- Feel free to make any change to your calendar!

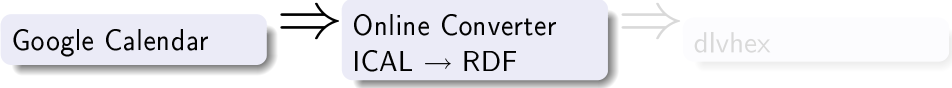
Data Flow



GOAL: Given calendar data in RDF, and a meeting day, find a suitable time slot for arranging a meeting between the six teams, under given constraints.

Fast prototype a simple program accomplishing the task, written in dlvhex

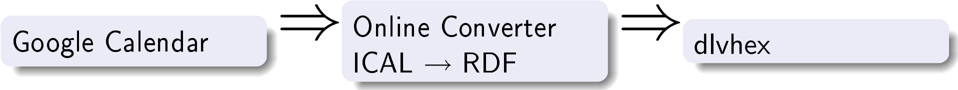
Data Flow


Google Calendar ⇒ Online Converter ICAL → RDF ⇒ dlvhex

GOAL: Given calendar data in RDF, and a meeting day, find a suitable time slot for arranging a meeting between the six teams, under given constraints.

Fast prototype a simple program accomplishing the task, written in dlvhex

Data Flow



Google Calendar ⇒ Online Converter ICAL → RDF ⇒ dlvhex

GOAL: Given calendar data in RDF, and a meeting day, find a suitable time slot for arranging a meeting between the six teams, under given constraints.

Fast prototype a simple program accomplishing the task, written in dlvhex

How to manipulate these data from a dlvhex program

- Go to example `calendar1.dlht`.
- Several building bricks available:

Fact Predicates (predefined)

1. `meetingDate("yyyy-mm-dd")`. Set this to the meeting day.
2. `calendar(teamX,URL)`. The public URL of each calendar. Comment out with % the teams you don't want to participate to the meeting.
3. `inrange("yyyy-mm-ddThh:mm:ss")`. Possible starting times: predefined to range from `08:00:00` to `19:00:00` (in slots of one hour) for the meeting day.
4. `busy(teamID,startTime,endTime,eventType)`. Time slot a given `teamID` is busy. `eventType` can be either `"OPAQUE"` (non movable appointment) or `"TRANSPARENT"` (flexible appointment).
5. `succ(time1,time2)`. It's true if `time2` is the next time slot w.r.t. `time1` (one hour later). For instance it holds `succ("2006-06-11T09:00:00","2006-06-11T10:00:00")`

Other Building Bricks

Templates

1. `overlap{p(*,*,*),q(*,*,*)}(team1,team2)`. Given two ternary predicates *p* and *q*, having extension in format `(groupID,StartTime,EndTime)` this template is true for all the couples `team1,team2` such that `team1` and `team2` have two overlapping events.

2. Example: Given facts `chosenSlot(slot,"2006-06-11T08:00:00","2006-06-11T10:00:00")`, `event(team1,"2006-06-11T09:00:00","2006-06-11T11:30:00")` and rule

   ```
   conflict(X,Y) :-
   overlap{chosenSlot(*,*,*),event(*,*,*)}(X,Y)
   ```

   , then

   `conflict(slot,team1)` is true.

Other Building Bricks

Templates

1. any{p(*)}(value). Chooses (nondeterministically) exactly one value from the values of *p*.
2. Example: Given facts slot("2006-06-11T08:00:00"), slot("2006-06-11T09:00:00") and a one rule program

   ```
   chosenSlot(X) :- any{slot(*)}(X).
   ```

   then we have two different answer sets, one containing chosenSlot("2006-06-11T09:00:00") and the other containing chosenSlot("2006-06-11T08:00:00").

Tasks to accomplish:

Task

Task 1 (easy): write a program that finds a time slot where all the participant are available.

Task

Task 2 (easy): write a program that finds a time slot where as many as possible participants are available.

Task

Task 3 (medium): write a program that finds a time slot where conflicts with opaque events are forbidden, while conflicts with transparent events are minimized.

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check ‘‘likes’’
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check ‘‘dislikes’’
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check "likes"
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check "dislikes"
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (table(T)) with certain number of chairs (nchairs(T,C)).
- Persons (person(T)) should be seated such that persons who like each other (likes(P1,P2)) are at the same table.
- Persons who dislike each other (dislikes(P1,P2)) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check "likes"
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check "dislikes"
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check "likes"
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check "dislikes"
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check "likes"
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check "dislikes"
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (table(T)) with certain number of chairs (nchairs(T,C)).
- Persons (person(T)) should be seated such that persons who like each other (likes(P1,P2)) are at the same table.
- Persons who dislike each other (dislikes(P1,P2)) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check "likes"
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check "dislikes"
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (table(T)) with certain number of chairs (nchairs(T,C)).
- Persons (person(T)) should be seated such that persons who like each other (likes(P1,P2)) are at the same table.
- Persons who dislike each other (dislikes(P1,P2)) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv

Restaurant Seating Problem

- A restaurant has tables (`table(T)`) with certain number of chairs (`nchairs(T,C)`).
- Persons (`person(T)`) should be seated such that persons who like each other (`likes(P1,P2)`) are at the same table.
- Persons who dislike each other (`dislikes(P1,P2)`) are at different tables[1].
- GOAL: find a suitable seat for everyone.

```
Guess if person P sits at table T or not
 at(P,T) v not_at(P,T) :- person(P), table(T).
Check capacity of tables
 :- table(T), chairs(T,C), not #count { P: at(P,T) } <= C.
Check seating of each person
 :- person(P), not #count{T : at(P,T)} = 1.
Check “likes”
 :- like(P1,P2), at(P1,T), not at(P2,T).
Check “dislikes”
 :- dislike(P1,P2), at(P1,T), at(P2,T).
```

[1] Example seating.dlv