Knowledge representation

lecture 1: intro and overview

T. Tammet, TUT

KR and reasoning blocks

Knowledge representation

Background and basics.

Programming and databases. SQL: meaning and representation of facts.

Ontologies. RDF, RDFa, RDFS, OWL and friends. HTML annotations.

Natural language and restricted natural language

Reasoning

Propositional solvers

First order logic solvers

SMT solvers

Logics for uncertain knowledge

Declarative and procedural knowledge

Declarative and procedural?

In psychology:

Declarative knowledge is knowing "that" (e.g., that Washington D.C. is the capital of America), as opposed to **procedura**l knowledge is knowing "how" (e.g., how to drive a car).

Declarative and procedural?

In programming:

- Declarative representation: data and rules written in an easily processable simple format.
- Procedural representation: data and rules written as an executable program.
- A strict distinction is - obviously - impossible

Example: configuration

- Option 1: directly in the code

```
if (!strcmp(client_id,”Mycompany”) ) ....
```

- Option 2: macro defs

```
#define OUR_ID “Mycompany”

....

if (!strcmp(client_id,OUR_ID) ) ....
```

Example: configuration

- Option 3: configuration text file

client_id: Mycompany

....

if (!strcmp(client_id,get_conf(client_id))) ....

- Option 4: configuration xml file
- Option 5: configuration table in the database

....

Layers of declarative data

- First layer: **plain facts** like

  company_id: Mycompany

  limit1: 64

- Second layer: **rules** like

  if(X1<123 || X1>44 || X2<65) then result=12

Layers in psychology

- Episodic knowledge: memory for "episodes" (i.e., the context of where, when, who with etc); usually measured by accuracy measures, has autobiographical reference.
- Semantic knowledge: Memory for knowledge of the world, facts, meaning of words, etc. (e.g., knowing that the first month of the year is April (alphabetically) but January (chronologically).

Observations in programming

- Attempting to make everything “configurable” leads us to using just another programming language for expressing configurable data and rules!
- Ordinary programming languages are also seen as “declarative”, at least by compilers.

Where does declarativeness help?

- Same configurable information (like “Mycompany”) has to be used by different programs/different languages
  - several “main programs”
  - administrator interfaces, database tools, ...
- We can derive new information (facts, rules) from given facts/rules.
  - additional limits/conditions not given explicitly
  - convert imported data, important special cases, ...
- We can learn new information automatically

Transformation (in psychology)!

- declarative knowledge converted to procedural
- learning to drive a car, play tennis, ...
- “For example, when I was learning to play tennis, I learned all about the rules of the game, where to come into contact with the ball on my racket, how to make the ball go where I wanted to by the follow through, and how to position my body for a backhand stroke. This is a set of factual information. Putting those facts into practice helped me gain the skills to transform a series of declarative knowledge into procedural knowledge. The skills I acquired couldn't be learned simply by being told. I gained the skills only after actively putting them into practice and being monitored by a coach who was constantly providing feedback. “

Transformation (in programming)!

declarative knowledge converted to procedural

goal: automation and speed

- Rough understanding, notes and spec of the solution written down as an actual program code.
- Source code compiled into machine code.

Examples from robotics etc

- Robotics: planning vs reactive architectures
- Lenat´s Cyc vs Brooks's robotic insects
- Planning & reaction combo in DARPA races
- Roboswarm: combining declarative and procedural knowledge

Viewing relational databases as logic

Presentation plan

- refresher of 1st order predicate logic
- meaning of data in databases
- tables as predicates: straightforward encoding
- queries and joins as rules
- special objects and features: multiple rows, null
- encoding structures in db-s
- db keys as a way to encode functions

Pred calculus example 1: prolog

Data:

```
father(jan,pete).

father(jan,martin).

father(martin,matt).

father(frank,mary).

mother(mary,mike).
```

Rules:

```
grandfather(X,Z) :- father(X,Y), father(Y,Z).

grandfather(X,Z) :- father(X,Y), mother(Y,Z).
```

Queries:

```
? father(martin,mike)

? father(jan,X)

? grandfather(X,mike)
```

Pred calculus example 2

- brother(jim,pete)
- Forall X,Y (bother(X,Y) => brother(Y,X))
- Forall X,Y (brother(X,Y) => man(X)).
- query: man(X)
- answers: man(jim), man(pete)

Pred calculus with functions

Data:

```
father(pete)=jan.

father(martin)=jan.

father(matt)=martin.

father(mary)=frank.

mother(mike)=mary.
```

Rules:

```
grandfather(X,Z) <- father(Y)=X & father(Z)=Y

alternative: grandfather(father(father(X)),X).

grandfather(X,Z) <- father(Y)=X & mother(Z)=Y

alternative: grandfather(father(mother(X)),X).
```

Words in logic have no meaning

Data:

```
foo(p1)=j.

foo(m1)=j).

foo(m2)=m1.

foo(m3)=f.

bar(m4)=m3.
```

Rules:

```
grm(X,Z) <- foo(Y)=X & foo(Z)=Y

alternative: grm(foo(foo(X)),X).

grm(X,Z) <- foo(Y)=X & bar(Z)=Y

alternative: grm(foo(bar(X)),X).
```

Meaning of words?

- Relations give meaning to words
- What about = ?? father(john)=pete then just replace .
- Three basic rules:
  - e(X,X)
  - e(X,Y) -> e(Y,X)
  - e(X,Y) & e(Y,Z) -> e(X,Z)
- Examples: parallel lines, >=, relative

Meaning of words?

- Substitution rules of equality:
  - e(X,Y) & father(X,Z) -> father(Y,Z)
  - e(X,Y) & father(Z,X) -> father(Z,Y)
  - e(X,Y) & e(father(X),Z) -> e(father(Y),Z)
  - Etc for all predicates and functions we have

Built-in procedures and theories for relations and functions

- Arithmetics + * etc: procedural attachments (procedural data representation)
- String, list, date, file etc etc proc attachments
- Special built-in theories like Presburger arithm

- Solvers focusing on procedural attachments and built-in theories are called **SMT** solvers: **solvers modulo theories**

Relational db table and logic

Client table:

id	name	balance
1	john	100
2	pete	-200

Logic:

client(1,john,100)

client(2,pete,-200)

Queries

```
select
client.name, client.balance
from client
where  balance<0;
```

client(I,N,B) & B<0 -> answer(N,B)

? answer(X,Y)

Join

client table:

id	name	balance
1	john	100
2	pete	-200

cars table:

id	model	owner
1	ford	1
2	opel	2
3	saab	2

```
select client.name, cars.model from client, cars
where client.id=cars.owner
```

client(I,N,B) & cars(J,M,I) -> ans(N,M)

Representing complex structures in relational databases

+(*(1.9, 2.5),3)

Term:

id	op	a1	t1	a2	t2
1	*	2	D	1	D
2	+	1	T	3	I

Data:

	varchar	type
1	“2.5”	F
2	“1.9”	F
3	“3”	I

Special relational db gadgets

These things require extra care when encoding in logic:

- null value
- keys
- multiple rows
- closed world

null values

- **null** values in two different locations are never equal!
- client(1,jaan,null) and car(1,opel,null) then null in client is not equal to null in car
- null represents an existentially quantified var:
  - Exists X. client(1,jaan,X).
  - Exists X. car(1,opel,X).

Keys

Client table:

id	name	balance
1	jan	100
2	pete	-200

ALTER TABLE client ADD PRIMARY KEY (id)

Would mean in logic:

Client.name(1)=jan

Client.name(2)=pete

Client.balance(1)=100

….

Multiple rows

payments table without a primary key

client	sum
1	100
1	100
2	50

two identical facts in logic mean a single fact:

payments(1,100) …. payments(1,100)

Open versus closed world

- Classical logic is **open**:
  - We know N facts. We do NOT say that only these N facts hold: maybe there are M more facts which are true but which we do not know.

- In databases we normally assume that world is **closed**: only these facts hold which we know . For example we assume that all the payments performed are in our database. This allows us to use **aggregate functions** like sum, avg, ...

Schemaless databases and RDF

The relational databases have been a standard way to store and query data for decades

Implementations are complex and polished

SQL is everywhere

Alternatives to relational databases

A varied landscape of technologies,
part of which are under a NoSQL umbrella name:

- Network databases
- Graph databases
- Document databases
- Key-value databases
- Object-relational mapping
- Main memory databases
- XML databases
- RDF, Sparql, semantic web
- Google Bigtable and MapReduce framework

Triplets: an obvious idea to Implement schemaless databases

Each row with N cols is represented as N rows of three columns, called sometimes as

- Row/Object id Column name Value
- Object Property Value
- Subject Predicate Object

Similar to key-value

Object Property Value

can be combined to

Object:Property Value

Schema-less is often inevitable

Read data from numerous sources, aggregate in our own database:

- We have no control over foreign data
- Our understanding of foreign data changes
- Our data sources change

There is one „schemaless“ standard, but beware

There is a wide range of schemaless databases, but most of them are basically API-s or have proprietary query languages: no real standards.

However, there is one standard **- RDF (resource description framework)** – which is not really loved.

- Developed and pushed by W3C
- Cornerstone of the semantic web project
- Large number of systems supporting
- A lot of tools

RDF: triple not really a triple

Object Property **Value** **Valuetype**

With valuetype normally being either:

- One of xml schema datatypes
- Global id: URI
- Local id

RDF: some restrictions

Object Property Value Valuetype

Object, Property, Valuetype: URI-s

Value: URI or literal value

Many representation syntaxes

- RDF/XML
- RDFa
- N3
- N-triples
- Turtle
- ....

Example in Turtle syntax

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

How to add metadata to a row?

Like timestamp, changer, row id, status etc etc?

Horrible answer: reification

The ugly head of reification

We have

personid:12 salary 20000

Want to add timestamp and entering person?

The reification way

From

personid:12 salary 2000 + timestamp etc

To

datarow:10001	subject	personid:12
datarow:10001	predicate	salary
datarow:10001	object	2000
datarow:10001	timestamp	2009-10-20 13:45
datarow:10001	modifier	personid:345

From the relational db ...

One row, N cols in the relational db

First, get N rows of four cols in RDF

Second, get (N*3)+X rows of four cols after reification

N → 12*N

Problem with containers

RDF provides a *container vocabulary* consisting of three predefined types (together with some associated predefined properties).

A *container* is a resource that contains things. The contained things are called *members*. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers:

rdf:Bag

rdf:Seq

rdf:Alt

Problem with containers

Containers are a fake:

- Containers have no real semantics in RDF
- Container semantics would make calc hard.

Problem with local id-s

Different object id-s:

- Global URI-s.
  - These are fine.
- Local “blank nodes”.
  - Their semantics/use in the RDF spec is broken: creates unnecessary problems.

Storage of RDF in a relational db

Predicate, subject, valuetype URI-s:

- keep a separate table for unique strings
- use numeric string id-s in pred,subject,valuetype

Storage of rdf in a relational db

Storing value? Can be int, float, string, URI, ...

Several ways, all bad:

- Encode everything as a string
- Encode everything as a number
- Use several columns for different (main) types

Sparql query language example

```
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
   ?country   a   type:LandlockedCountries ;
   rdfs:label   ?country_name ;
   prop:populationEstimate  ?population .
 FILTER
   (?population > 15000000 &&
     langMatches(lang(?country_name), "EN")) .
} ORDER BY DESC(?population)
```