Knowledge representation

lecture 5

Intro to rules and automated reasoning

Tanel Tammet

TTU

Lecture overview

Why rules

Procedural and declarative rules

Simple derivation systems for 1st order logic: resolution

Prolog as a special resolution search strategy

Queries and answers

Why rules

Derive new knowledge from existing knowledge

Learning: automatic creation of new rules

Inferences: apply rules to get new knowledge

Is the „derived knowledge“ really new or just an intrinsic consequence of data + rules? This is a philosophical question out of the scope of this course.

Rule examples

From simple to more and more complex ...

```
pub(X) => eatingplace(X)

pub(X) & openattime(X,T) => may_eat_at_time(X,T)

pub(X) & openattime(X,T) & country(X,'Britain') &
>(age(P),17) & popularity(X,S) =>

recommendscore('eating',X,T,P,0.8*poptorec(S))

...
```

Procedural & declarative

Declarative:

```
pub(X) => eatingplace(X)
```

Procedural:

```
...
d=fetchobjdata(x)
if (datahasattr_val(d,“type“,“pub“)) {
  storenewdata(x,“prop“,“eatingplace“);
}
if ....
...
```

Procedural & declarative

Procedural: database triggers

```
CREATE OR REPLACE TRIGGER tasuta_ins
BEFORE INSERT ON tasuta
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
BEGIN
declare
cursor oic is select kood from isik where kood=:new.ik;
oi oic%ROWTYPE;
cursor cat is select
  category_id,
  ...
  if ca.tasuta='Y' then
          open oic;
          fetch oic into oi;
          if oic%NOTFOUND then
             insert into isik (kood) values (:new.ik);
          end if;
          close oic;
          select ticket_sequence.nextval into t_id from dual;
          insert into ticket(
           ticket_id
           ,ik
          .....
```

Declarative: rules + engine

Declarative rules like

pub(X) => eatingplace(X)

need a

- procedural reasoning engine to actually process data + rules and add new data
- concrete syntax for data and rules which the engine understands
- optional built-in procedural functions and predicates for convenience and efficiency (arithmetics, string handling etc)

Declarative vs proc handling

Pros for proc handling:

- proc handling easy to code for specific cases: no need for special syntax, engine api, etc, easy to incorporate arbitrary libraries and program snippets

Cons for procedural handling:

- recursive/iterative handling (derivation chains) hard to program for procedural cases
- proc rules handling code hard to modify and maintain
- achieving efficiency & developing optimisations is very hard work

Declarative vs proc handling

Pros for decl handling:

- recursive/iterative handling (derivation chains)
- proc rules handling and maintenance: independcy helps
- efficiency-targeted optimisations built into engine

Cons for decl handling:

- understanding and using required syntax
- understanding and using engine api-s
- hard to add your own procedural functions/preds from other languages/toolkits

Summary: engines vs your own code like SQL engines vs writing code for processing data files

Rules are code too

Another view:

rules are code in a rule-programming-language

like different, specialised Prolog's or Datalog's

How to use a rule?

In other words, what should rule derive from data?

Common ground for almost all rule systems:

(classical 1st order logic - various limitations) +
various extensions

Why classical logic? Because it allows to derive all things which generally make sense.

pub('texas')

pub(X) => eatingplace(X)

gives eatingplace('texas') and nothing more

Applying 1st order logic

There are many different - mostly equivalent axiomatizations and rule systems for logic.

However, the practical - and sufficient - way to think is simply this:

Find all possible matches with the premisses of the rule

Each match instantiates variables

Derive an instantiated consequence of the rule

Repeat for all matches and all consequences etc etc ad infinitum

Applying 1st order logic

There are many different - mostly equivalent axiomatizations and rule systems for logic.

However, the practical - and sufficient - way to think is simply this:

Find all possible matches with the premisses of the rule

Each match instantiates variables

Derive an instantiated consequence of the rule

Repeat for all matches and all consequences etc etc ad infinitum

Conjunctive normal form

Goal: make a fact/rule set simple and uniform. Remove nested loops and existential quantifiers.

Terminology:

Atom is a propositional variable or a predicate with args like **p(X,1)**

Literal is a an atom or a negation of an atom;

Clause or disjunct is a disjunction of literals

Conjuctive normal form is a conjuction of clauses.

Convert to normal form

There are six stages to the conversion:

1. Remove $\Rightarrow$
2. De Morgan’s to move negation to atomic propositions
3. Skolemizing (gets rid of $\exists$)
4. `Eliminating’ universal quantifiers
5. Distributing AND over OR
6. Arrange into clauses and maybe reorder

Convert to normal form: example

1st order formula

$\forall Y (\forall X (taller(Y,X) \mid wise(X)) \Rightarrow wise(Y))$

Simplify

$\forall Y (-\forall X (taller(Y,X) \mid wise(X)) \mid wise(Y))$

Move negations in

$\forall Y (\exists X (-taller(Y,X) \; \& \; -wise(X)) \mid wise(Y))$

Move quantifiers out

$\forall Y (\exists X ((-taller(Y,X) \; \& \; -wise(X)) \mid wise(Y)))$

Skolemize

$\exists X ((-taller(Y,X) \; \& \; -wise(X)) \mid wise(Y)) \; \gamma = \{Y\}$

$(-taller(Y,x(Y)) \; \& \; -wise(x(Y))) \mid wise(Y)$

Distribute disjunctions

$(-taller(Y,x(Y)) \mid wise(Y)) \; \& \; (-wise(x(Y)) \mid wise(Y))$

Convert to CNF

$\{ -taller(Y,x(Y)) \mid wise(Y),$

$-wise(x(Y)) \mid wise(Y) \}$

Example: facts and rules in a normal form

```
father(john,pete).
brother(pete,mark).
-brother(X,Y)  v   brother(Y,X).
-father(X,Y)     v   parent(X,Y).
-mother(X,Y)   v   parent(X,Y).
-parent(X,Y)    v   -parent(Y,Z)   v   grandparent(X,Z).
```

Note: we assume capital letters are variables

Resolution method

Simple core method for practical reasoning (logical derivations)

Used as a basis for most reasoner implementations, with numerous additions / modifications / strategies / optimisations.

Can be seen as a framework for building specialized reasoners.

Just two rules operating on a normal form:

- Generalised modus ponens (**resolution rule**)
- Limited instantiation (**factorisation rule**)

The main idea of the the resolution method: derive new clauses from given clauses potentially ad infinitum.

Typically used to show that a clause set is **contradictory**.

To prove that F is a tautology is the same as to prove that -F is contradictory.

Modus ponens:

$$\frac{A \qquad A \rightarrow B}{B}$$

Same, but with A->B as a disjunct:

$$\frac{A \qquad -A \lor B}{B}$$

Example derivation of contradiction:

$$\begin{tikzcd}
\{R, \neg P\} \arrow[dr, no head] & & \{P, Q\} \arrow[dl, no head] & & \{\neg Q\} \arrow[ddll, no head] & & \{\neg R\} \arrow[dddll, no head] \\
& \{R, Q\} \arrow[dr, no head] & & & & & \\
& & \{R\} \arrow[drr, no head] & & & & \\
& & & & \square & &
\end{tikzcd}$$

Resolution rule is a generalisation of modus ponens to arbitrary disjuncts:

$$\frac{\mathbf{A_1} \lor A_2 \lor \ldots \lor A_n \qquad \neg\mathbf{A_1} \lor B_2 \lor \ldots \lor B_m}{A_2 \lor \ldots \lor A_n \lor B_1 \lor \ldots \lor B_m}$$

How to apply the rule:

– Find a variable A_1 which is positive in one formula and negative in the ohter

– Cut off both A_1 and $\neg A_1$ and glue the rest.

Resolution rule for predicate calculus

Example (observe that P(b) is a different atom than P(X)):

$$\frac{P(b) \qquad P(X) \Rightarrow R(X)}{R(b)}$$

vars instantiated X:=b

Resolution rule for predicate calculus

Same example in normal form:

$$\frac{P(b) \qquad -P(X) \ \vee \ R(X)}{R(b) \qquad \text{vars instantiated } X{:=}b}$$

Resolution rule for predicate calculus

Example with additional ballast:

$$\frac{P(b) \lor G(s) \qquad S(Y) \qquad -P(X) \lor -S(X) \lor R(X) \lor M(X)}{R(b) \lor M(b) \lor G(s)} \quad \text{vars instantiated } X{:=}b,\ Y{:=}b$$

Resolution rule for predicate calculus

Example with three premises

$$\frac{P(b) \qquad S(Y) \qquad P(X)\ \&\ S(X)\ \text{=>}\ R(X)}{R(b) \qquad \text{vars instantiated } X{:=}b,\ Y{:=}b}$$

is the same as two steps of resolution:

$$\dfrac{\dfrac{P(b) \qquad -P(X)\ \text{v}\ -S(X)\ \ \text{v}\ R(X)}{-S(b)\ \ \text{v}\ R(b) \qquad S(Y)}}{R(b)}$$

Full rule with unification for predicate calculus:

$$\frac{\mathbf{A_1} \lor A_2 \lor \ldots \lor A_n \qquad \neg\mathbf{A_1} \lor B_2 \lor \ldots \lor B_m}{(A_2 \lor \ldots \lor A_n \lor B_1 \lor \ldots \lor B_m)s}$$

Where s = unify(A1,B1) :

unify(A,B) calculates *the most general unifier* of A and B: it is the minimal instantiation s making As=Bs

Unification examples:

P(X,a,Y),
P(Z,U,Z)) — Gives {X:=Z, U:=a, Y:=Z }

p(X, f(cat))
p(f(Y) ,f(Y))
p(f(Z), T)

Gives
{X:=f(cat),Y:=cat,T:=f(cat),Z:=cat}

p(X, f(cat)),
p(f(Y), f(Y)),
p(f(dog), Z)}

Fails

p(f(Y), f(Y))
p(f(Z), Z)

Fails (occur check)

Factorization: eliminate duplicates

$$\frac{\mathbf{A_1} \vee \mathbf{A_1} \vee A_2 \vee \ldots \vee A_n}{\mathbf{A_1} \vee A_2 \vee \ldots \vee A_n}$$

Factorization for predicate calculus

Example:

$$\frac{P(X)\ \text{v}\ P(a)}{P(a)}$$

Rule for gluing together two literals in the same disjunct using the minimal unifier.

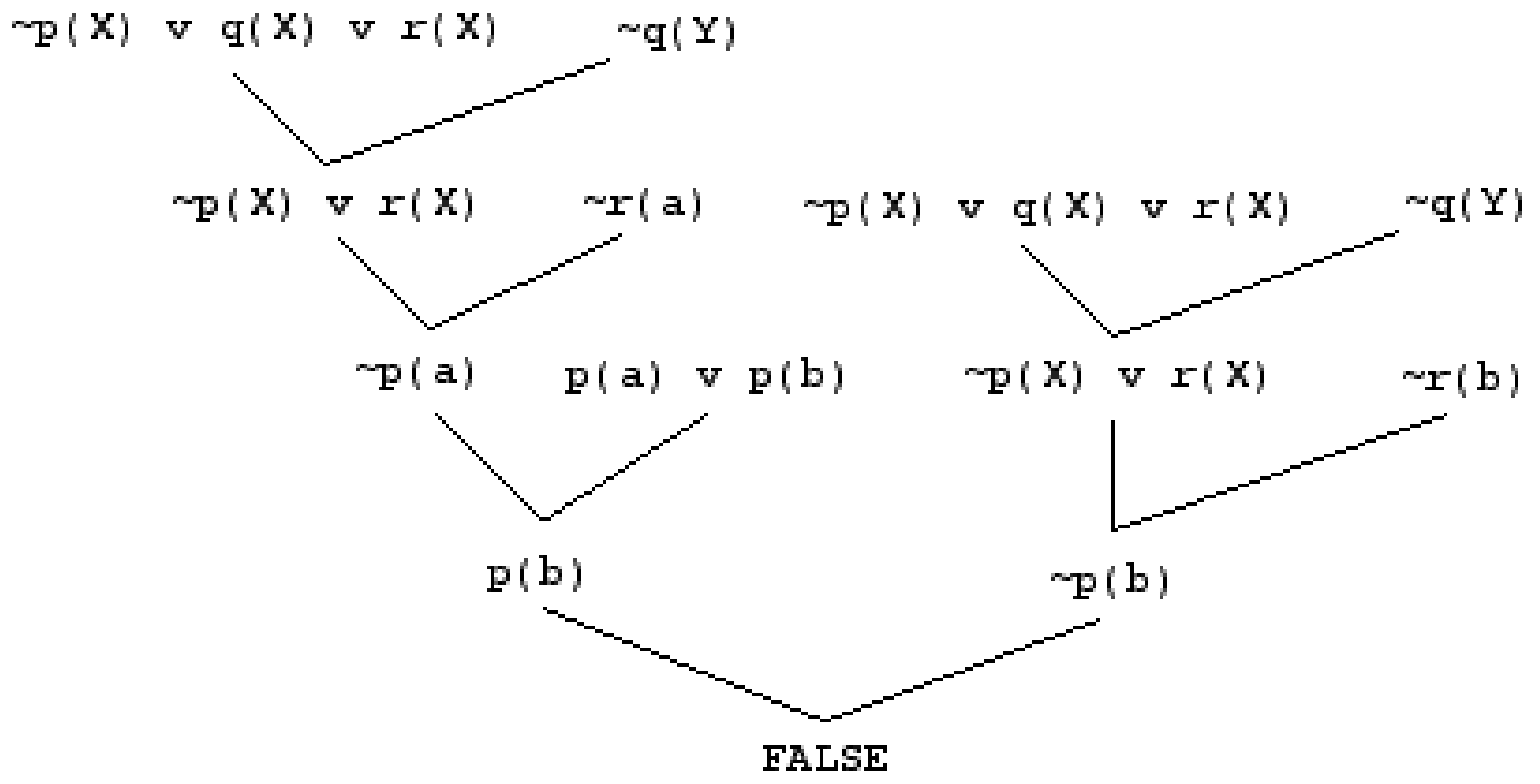
$$\frac{\begin{array}{l} A1 \text{ v } A2 \text{ v } \dots \text{ v } An \\ \text{and } s = \text{unify}(A1,A2) \end{array}}{(A2 \text{ v } \dots \text{ v } An)\ s}$$

Basic saturation procedure

```
While a refutation has not been found:

    Copy two clauses from the set
    Generate all logical consequences, e.g., by resolution
    Put logical consequences into the set
```

Derivation example



ANL loop for saturation

```
Let CanBeUsed = {}

Let ToBeUsed = Input clauses

While Refutation not found && ToBeUsed not empty:

    Select the ChosenClause from ToBeUsed
    Move the ChosenClause to CanBeUsed
    Infer all possible clauses using the ChosenClause and other clauses
          from CanBeUsed.
    Add the inferred clauses to ToBeUsed
```

ANL loop for saturation

Depending on how the ChosenClause is selected from the ToBeUsed set, different search strategies can be implemented.

- **Depth first search**
  Select a most recently created resolvant as the ChosenClause.
  Does not guarantee a complete search (could get into an infinite loop)
  Does not guarantee finding the shortest refutation.

- **Breadth first search**
  Select a least recently created resolvant as the ChosenClause.
  Will find the shortest refutation.
  Implements a ply-by-ply search.

- **Best first search**
  Select the 'best' clause as the ChosenClause. the best possible literals.
  The notion of "best" is determined by a heuristic function

Saturation may not terminate

In case the clause set is not contradictory (i.e. empty clause cannot be derived) the saturation method may run forever

Simple example:

$\neg P(x) \lor P(f(x))$
$P(y) \lor \neg P(f(y))$

Will give $\neg P(x) \vee \neg P(f(f(x))$, $\neg P(x) \vee \neg P(f(f(f(x)))$, ... etc

However, for these clauses saturation will stop quickly:

$\neg P(x) \lor P(a)$
$P(y) \lor \neg P(b)$

Query: contradiction search

Observe that

Facts & Rules => Query

is a tautology iff

Facts & Rules & -Query

is a contradiction

Since -(Facts & Rules => Query)   =   -(-Facts v -Rules v Query)

Answer mechanism

Prolog query ? R(X) means adding
-R(X) | Answer(X)
to facts and then searching for contradiction:

-R(X) | Answer(X)
P(a)
-P(X) | R(X)

-P(X) | Answer(X)

Answer(a)

Prolog

Uses a highly specialised (and incomplete!) strategy
of resolution for horn clauses: rules with max one
literal in the consequent
(max one positive literal in a disjunct)

Derivation direction always from positive (right) side of
the rule, giving instantiated antecedent as a result:

P(a)
P(X) => R(X)
-R(a) (query)
----
derives -P(a), and then finds contradiction with P(a)

Does not derive R(a).

Horn clause is a disjunct with max one positive literal.

-A v -B v C	is Horn
-A v -B	is Horn
-A v -B v C v D	not Horn
A v B	not Horn

Unit resolution is a search strategy of resoluton where at least one of arguments must be unit (a single literal).

Theorem: unit resolution is complete for Horn clause sets.

Subsumption:

A subset S of a disjunct D is said to subsume D.

Examples:

A subsumes –B v A

A v B subsumes C v B v –R v A

Theorem: when a derived disjunct N subsumes another disjunct C, then throwing away C preserves completeness

Complexity of unit resolution for the Horn case

Observe that every unit resolution rule application to Horn clauses creates a shorter disjunct which subsumes a long argument.

Example:

$$\frac{A \qquad -A \lor -B \lor C}{-B \lor C}$$

Robbins algebras are boolean: Mccune, 1997

In 1933, E. V. Huntington presented the following basis for Boolean algebra:

$x + y = y + x.$

$(x + y) + z = x + (y + z).$

$n(n(x) + y) + n(n(x) + n(y)) = x.$

Shortly thereafter, Herbert Robbins conjectured that the Huntington equation can be replaced with a simpler : $n(n(x + y) + n(x + n(y))) = x$. Robbins and Huntington could not find a proof, and the problem was later studied by Tarski and his students.

The successful search took about 8 days on an RS/6000 processor and used about 30 megabytes of memory.

```
2 (wt=7) [] -(n(x + y) = n(x)).

3 (wt=13) [] n(n(n(x) + y) + n(x + y)) = y.

5 (wt=18) [para(3,3)] n(n(n(x + y) + n(x) + y) + y) = n(x + y).

6 (wt=19) [para(3,3)] n(n(n(n(x) + y) + x + y) + y) = n(n(x) + y).

24 (wt=21) [para(6,3)] n(n(n(n(x) + y) + x + y + y) + n(n(x) + y)) = y.

47 (wt=29) [para(24,3)] n(n(n(n(n(x) + y) + x + y + y) + n(n(x) + y) + z) + n(y + z)) = z.

48 (wt=27) [para(24,3)] n(n(n(n(x) + y) + n(n(x) + y) + x + y + y) + y) = n(n(x) + y).

146 (wt=29) [para(48,3)] n(n(n(n(x) + y) + n(n(x) + y) + x + y + y + y) + n(n(x) + y)) = y.

250 (wt=34) [para(47,3)] n(n(n(n(n(x) + y) + x + y + y) + n(n(x) + y) + n(y + z) + z) + z) = n(y + z).

996 (wt=42) [para(250,3)] n(n(n(n(n(n(x) + y) + x + y + y) + n(n(x) + y) + n(y + z) + z) + z + u) + n(n(y + z) + u)) = u.

16379 (wt=21) [para(5,996),demod([3])] n(n(n(n(x) + x) + x + x + x) + x) = n(n(x) + x).

16387 (wt=29) [para(16379,3)] n(n(n(n(n(x) + x) + x + x + x) + x + y) + n(n(n(x) + x) + y)) = y.

16388 (wt=23) [para(16379,3)] n(n(n(n(x) + x) + x + x + x + x) + n(n(x) + x)) = x.

16393 (wt=29) [para(16388,3)] n(n(n(n(x) + x) + n(n(x) + x) + x + x + x + x) + x) = n(n(x) + x).

16426 (wt=37) [para(16393,3)] n(n(n(n(n(x) + x) + n(n(x) + x) + x + x + x + x) + x + y) + n(n(n(x) + x) + y)) = y.

17547 (wt=60) [para(146,16387)] n(n(n(n(n(x) + x) + n(n(x) + x) + x + x + x + x) + n(n(n(x) + x) + x + x + x) + x) + x) = n(n(n(x) + x) + n(n(x)
+ x) + x + x + x + x).

17666 (wt=33) [para(24,16426),demod([17547])] n(n(n(x) + x) + n(n(x) + x) + x + x + x + x) = n(n(n(x) + x) + x + x + x).
```