Logical programming

Svarny Petr

Katedra logiky FF UK

26. dubna 2021

Overview

Logical programming

- Another declarative paradigm (,i.e., not imperative).
- Main rep. =Prolog=, but not the only one.
- Can be used for domain specific work (e.g.: Semantic web, =Oracle Constraint Definition Language=).

Basic idea

- Logic is awesome and can save us!
- Separate data and functions (again) via fixed reasoning engine (i.e., the logic).
- The program is basically the data.

Some Prolog sources

- SWI Prolog
- Learn Prolog Now
- =Metalevel Prolog=
- =MFF Prolog=
- =Visual Prolog= Someone?
- =ISO Prolog=

Core of Prolog

- First order predicate logic
- Horn clauses:
  - implication form: $u \leftarrow p \wedge q \wedge \ldots \wedge t$
  - disjunction form: $u \vee \neg p \vee \neg q \vee \ldots \vee \neg t$

Horn clauses

Definite clause

Definite clause is a Horn clause with only one positive literal.

$u \leftarrow p \wedge q \wedge \ldots \wedge t$

Unit clause

Unit clause is a Horn clause with no negative literals. Also called **Fact** if u is not a variable.

u

Goal clause

Goal clause is a Horn clause without a positive literal.

$\perp \leftarrow p \wedge q \wedge \ldots \wedge t$

Selective Linear Definite clause resolution

Given clause $u \leftarrow p$

Given fact p

Resulting goal u

i.e., to show u, show p.
The program becomes just a collection of facts resolved following the above rule.

Prolog notation

$u \leftarrow p \wedge q \wedge \ldots \wedge t$ becomes:

```
u :- p, q, ..., t.
```

Prolog Term

Term

Basic Prolog data structure of the shape name(argument,...).
Terms can be:

- atomic
- variables
- compound

Prolog Simple Terms

Atomic

Atoms i.e. strings in quotes or beginning lower-cased, a, 'A', aA

Numbers e.g., 1

Variables

Prolog variables can be either beginning with capital letters (A, Aa) or anonymous (_).

Prolog Compound Terms

Compound terms and Functors

If $T_1, T_2, \ldots$ are terms, then $F(T_1, T_2, \ldots)$ is a compound term. There the atom F is called a functor name (same syntax rules as atoms). We denote also F/N the principal functor with its arity (e.g., parent/2).

Prolog Clauses

Rule
Head :- Body.

Fact or Base Clause
Head.
i.e. the rule: Head :- true.

Goal Clause
Body of each rule is a Prolog goal.

Clause
A Prolog clause is either a Prolog fact or a Prolog rule.

Prolog Predicates

Predicate

A predicate consists of a name and zero or more arguments. The name is a Prolog atom. Each argument is an arbitrary Prolog term.

What is the difference betweeen a predicate and a functor?

Prolog Predicates

Predicate

A predicate consists of a name and zero or more arguments. The name is a Prolog atom. Each argument is an arbitrary Prolog term. I.e. a predicate is a collection of clauses. Where Pred/N is called a predicate indicator.

Predicates are semantics, functors are syntax.

Prolog basic examples

What are the following?

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
```

Prolog basic examples

The Prolog prompt, i.e. inquiry, is denoted ?−.

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
?-
```

Prolog basic examples

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
?- song_of_ice_and_fire.
```

Prolog basic examples

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
?- song_of_ice_and_fire.
yes
```

Prolog basic examples

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
?- game_of_thrones.
```

Prolog basic examples

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
?- game_of_thrones.
no
```

Prolog basic examples

```
lannister(tyrion).
stark(robb).
song_of_ice_and_fire.
?- stark(tyrion).
no
```

Prolog basic predicates

Prolog has some predicates already predefined:

```
true/0
fail/0
,/2 /* and */
;/2 /* or */
:-/2 /* turnstile, if,  */
\+/1 /* negation as failure, optionally not/1 */
```

Prolog example

```
lannister(tyrion).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
song_of_ice_and_fire.
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nebezpecny(robb) :- stark(robb).
nebezpecny(robb) :- maArmadu(robb).
```

What are functors, predicates and atoms?

Prolog example

```
lannister(tyrion).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
song_of_ice_and_fire.
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nebezpecny(robb) :- stark(robb).
nebezpecny(robb) :- maArmadu(robb).

?-zakerny(tyrion).
```

Prolog example

```
lannister(tyrion).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
song_of_ice_and_fire.
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nebezpecny(robb) :- stark(robb).
nebezpecny(robb) :- maArmadu(robb).

?-zakerny(tyrion).
true
```

Prolog example

```
lannister(tyrion).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
song_of_ice_and_fire.
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nebezpecny(robb) :- stark(robb).
nebezpecny(robb) :- maArmadu(robb).

?-maArmadu(robb).
```

Prolog example

```
lannister(tyrion).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
song_of_ice_and_fire.
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nebezpecny(robb) :- stark(robb).
nebezpecny(robb) :- maArmadu(robb).

?-maArmadu(robb).
fail
```

Prolog resolution

"Logically, when Prolog answers a query, it tries to find a resolution refutation of the negated query and the set of clauses that constitute the program. When a refutation is found, it means that the query is a logical consequence of the program." And this is achieved via syntactical unification.

Prolog unification

Unification

Denoted =/2.

1. Any value can be unified with itself. E.g.: mother(john) = mother(john).
2. A var with another var. Variable names then reference the same variable. E.g.: X = Y, X = 2. Y is 2 also.
3. A var with any Prolog value (instantiation of the var, full if no variables remain). E.g.: X = foo(bar, [1, 2, 3]).
4. Two expressions unify if their constituents can be unified to the same value. E.g.: mother(mary, X) = mother(Y, father(Z)). Because unifies mary=Y and X=father(Z).
5. It is legal to unify a variable recursively, so carefully! E.g.: X = foo(X, Y).

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
?- lannister(X).
```

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
?- lannister(X).
X=tyrion
```

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
?- lannister(X).
X=tyrion;
```

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
?- lannister(X).
X=tyrion;
X=joffrey
```

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
?- lannister(X).
X=tyrion;
X=joffrey;
fail
```

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nepratele(X,Y):- lannister(X),stark(Y).
?- nepratele(X,Y).
```

Unification examples

```
lannister(tyrion).
lannister(joffrey).
stark(robb).
zakerny(tyrion):-lannister(tyrion).
nebezpecny(tyrion):- zakerny(tyrion); lannister(tyr
nepratele(X,Y):- lannister(X),stark(Y).
?- nepratele(X,Y).
X = tyrion ,
Y = robb;
X = joffrey ,
Y = robb;
```

Reading a Prolog program

Declarative

Program declares what holds either unconditionally (facts) or under certain conditions (clauses).

Procedural

Invoking a predicate is similar to calling a function with specifics of Prolog (possibly unbound variables and backtracking). Very complex to read and needs tracing of the program.

For details: =Metalevel=

Prolog more built-in examples

```
write/1 /* write the provided expression, i.e. prin
nl/0 /* new line */
assert/1 /* append to the database */
asserta/1 /* prepend to the database */
+/2 /* when evaluated sum of 2 values  */
==/2 /* succeed if X, Y identical */
```

Prolog recursion

```
primy_pribuzny(X,Y) :- otec(X,Y);matka(X,Y);souroze
pribuzni(X, Y) :- primy_pribuzny(X, Y).
pribuzni(X, Y) :- primy_pribuzny(X, Z), pribuzni(Z,
pribuzni(X, Y) :- pribuzni(Y, X).
```

In general ?

Prolog recursion

```
primy_pribuzny(X,Y) :- otec(X,Y);matka(X,Y);souroze
pribuzni(X, Y) :- primy_pribuzny(X, Y).
pribuzni(X, Y) :- primy_pribuzny(X, Z), pribuzni(Z,
pribuzni(X, Y) :- pribuzni(Y, X).
```

In general:

```
predicate :- terminal.
predicate :- terminal, predicate.
```

Datastructures

Logical programming = describe relations between entities.
I.e. changing datastructures are not welcome. Relate the previous entity with the "resulting" entity.
See =Metalevel=

Pairs

Pair (-)/2

A datastructure composed of two elements. E.g., -(a,b) or also a-b.

NO strings

Basically use atoms instead of strings, if you need strings, then use lists. =Prolog Library=
Except we are so used to strings... SWI etc. have them as separate library.

List

List ./2

A datastructure composed of a Head and Tail part. E.g., .(a,b) or also [a,b].
[] is an empty list atom.

[a,b,c]=.(a,.(b,c))=.(a,.(b,.(c,[])))

Prolog List exercises

What is the Head and Tail of the following lists?

- []
- [a]
- [[a, b], c, d]

Prolog List exercises

What is the Head and Tail of the following lists?

- []
  - H = none, T = none
- [*a*]
  - H = a, T = []
- [[*a*, *b*], *c*, *d*]
  - H = [a,b], T = [c,d]

Pipe operator

Pipe operator

Separating the head from the tail in a list, i.e. .(H,T), can be done by $[H|T]$.

Uses unification to find the variables H and T.

Exercise

Recursion, Lists, Pipe operators

Define the predicate *ismember*.

Exercise

Recursion, Lists, Pipe operators

Define the predicate *clen*.

```
clen(X,[X|_]).
clen(X,[_|T]) :- clen(X,T).
```

See also *member*/2.

Homework

Figure out the following predicates:

- concatenate a to a list (e.g., $[cb] \rightarrow [cba]$)
- join two lists (e.g., $[ab] + [cd] = [abcd]$)

Example of arithmetics

List length via arithmetics in Prolog:

```
len([],0).
len([_|T],N) :- len(T,X), N is X+1.
```

Instead of recursion

Use of accumulator instead of recursion:

```
accLen([_|T],A,L) :- Anew is A+1, accLen(T,Anew,L).
accLen([],A,A).
leng(List,Length) :- accLen(List,0,Length).
```

Sets

```
sidli(lannister,kingslanding).
sidli(lannister,casterlyrock).
sidli(stark,winterfell).
sidli(frey,twins).
sidli(baratheon,dragonstone).
?- bagof(X,sidli(Kdo,X),Misto). % all
?- setof(X,sidli(Kdo,X),Misto). % sorted unique
?- findall(X,sidli(Kdo,X),Misto). % only Misto
```

Simple generator

```
veta(Z):- podc(X), pric(Y), append(X,Y,Z).
podc(Z):- pojm(Z).
pric(Z):- s(X), pre(Y), append(X,Y,Z).
pric(Z):- s(Z).
pojm([tywin]).
pojm([jon]).
s([je]).
pre([vyvrhel]).
? veta(X)
```

Exercise

How to add declension?

```
veta(Z):- podc(X), pric(Y), append(X,Y,Z).
podc(Z):- pojm(Z).
pric(Z):- s(X), pre(Y), append(X,Y,Z).
pric(Z):- s(Z).
pojm([tywin]).
pojm([jon]).
s([je]).
pre([vyvrhel]).
? veta(X)
```

Definite Clause Grammar

```
veta(X,Z):- podc(X,Y), pric(Y,Z).
```

Grammar with syntactic sugar:

```
veta --> podc, pric.
podc --> pojm.
pric --> s, pre.
pric --> s.
pojm --> [tywin].
pojm --> [jon].
s --> [je].
pre --> [vyvrhel].
? veta(X,[]).
```

Exercise

What does this generate?

```
ab --> [].
ab --> l, ab, r.
l --> [a].
r --> [b].
```

Exercise

$a^n b^n$

```
ab --> [].
ab --> l, ab, r.
l --> [a].
r --> [b].
```

Cut

Cut

!/0 is the cut predicate, always succeeds and prevents the return to previous goals.

```
nepratele(X,Y):- lannister(X),stark(Y).
```

vs.

```
nepratele(X,Y):- lannister(X),!,stark(Y).
```

Cut

```
p :-  a , b .
p :-  c .
```

$$p \iff (a \wedge b) \vee c$$

vs.

```
p :-  a , ! , b .
p :-  c .
```

$$p \iff (a \wedge b) \vee (\neg a \wedge c)$$

Prolog workings

```
p(1).
p(2) :- !.
p(3).
?- p(X).
?- p(X),p(Y).
?- p(X),!,p(Y).
```

Glogical programming

Logical programming is still relevant!
Main motivation is to replace SQL.
Comes from =Datalog=, i.e. kind of Prolog.
For details: =Google Logica=
See also the =Tutorial=

Replacing what?

Replacing ... =SQL=.
Language for data manipulation and retrieval with COBOL based natural language-like commands.

SQL basic

```
SELECT * FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT * FROM Customers
ORDER BY Country;

SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

SQL Joins

=Joins=

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;
```

SQL basic

```
MagicNumber(x: 2);
MagicNumber(x: 3);
MagicNumber(x: 5);
```

vs.

```
SELECT 2 AS x
UNION ALL
SELECT 3 AS x
UNION ALL
SELECT 5 AS x;
```

SQL basic

```
MagicNumber(x:) :-
  x in [2, 3, 5];
```

Glogical programming

- Facts aka Tables
- Rules aka Queries
- JSON-like structures
- Conjunction aka Join

See also the =Tutorial=

Description logic

Non-classical logic for reasoning about properties.

FOL	**DL**	**OWL**
constant	individual	individual
unary predicate	concept	class
binary predicate	role	property

Description logic - language

Symbol	**Description**
$\top$	top concept
$\bot$	bottom, empty concept
$C \sqcap D$	intersection/conjunction of concepts (and)
$C \sqcup D$	union/disjunction of concepts (or)
$\neg C$	negation of concepts
$\forall R.C$	universal restriction, all R-successors are in C
$\exists R.C$	existential restriction, an R-successors exists in C
$C \sqsubseteq D$	concept inclusion, all C are D
$C \equiv D$	C is equivalent to D
$C \doteq D$	definition, C is defined to be equal to D
$a : C$	a is a C
$(a, b) : R$	a is R-related to b

DL variants

Many language variants, marked based on their properties.

$\mathcal{ALC}$ "Attribute language"

- Atomic negation
- Concept intersection
- Universal restriction
- Limited existential quantification
- Complex concept negation

$\mathcal{FL}$ "Frame based language"

- Concept intersection
- Universal restrictions
- Limited existential quantification
- Role restrictions

DL variants

Why so many variants?

DL variants

Different rules mean different complexity
Different complexity means different decidability
DLs are just fragments of general first order logic
See =ESSLLI 2018=

Elements of a language - Common concepts

- Domain specific languages
- Boolean constructors (negation, conjunction, disjunction)
- Role restrictions (existential and value restriction)
- Complex concepts are created from atomic and other complex concepts
- Based also on the chosen logic

Elements of a language - Atomic concepts

Atomic concept names
$C = A_1, ..., A_n$, subsets of the domain, i.e., basic classes of the domain
E.g.: student, teacher, admin
Special concepts are: Top $\top$ (whole domain), Bottom $\bot$ (empty)

Elements of a language - Relationships

Atomic role names

$R = r_1, ..., r_n$, powersets in the domain, i.e.: basic relations between concepts

E.g.: employedBy, supervisedBy, ...

Elements of a language - Relationships

Individual names

$I = a_1, ..., a_n$, members of the domain, i.e.: names of the objects, instantiations from the domain

E.g.: John, Jane, ...

Knowledge bases

Intentional knowledge

Definition of concepts, by subsumption statements
"TBox", $\mathcal{T}$, (terminology)

Extensional knowledge

Instantiation of concepts and roles, the database, by assertions
"ABox", $\mathcal{A}$, (assertion box), i.e. : expressions

Knowledge base (ontology)

TBox + ABox, $< \mathcal{T}, \mathcal{A} >$

Example

TBox:

- $Woman \equiv Female \sqcap Person$
- $Man \equiv Male \sqcap Person$
- $\top \equiv Woman \sqcup Man$
- $Parent \equiv \exists hasChild.\top$

ABox:

- anne:Woman, pete:Person
- mary:Woman
- (pete, anne): hasChild
- (mary, anne): hasChild

We can study $\mathcal{T} \models$, $\mathcal{A} \models$, $< \mathcal{T}, \mathcal{A} > \models$
Visualization is possible throught Venn diagrams and graphs.

Ontology

Ontologia, philosophical term, “nature of being and reality”
Technical ontologies = rigorous description of a domain:

- its concepts/classes
- objects/instantiations
- and their relationships
- with a shared conceptualisation (i.e. what ideas/concepts are used)

Formal ontologies = captured by a formal system
E.g.: Medical classification, biological taxonomy and gene ontology,...
Basis of the Semantic Web project
Automated reasoning with many tools (e.g., =Protege= or see =list=)

OWL

=Web Ontology Language=, W3C project for the Semantic Web

Split into sublanguages based on the need/use (Lite $\subset$ DL $\subset$ Full)

OWL ontology is an Resource Description Framework (RDF) graph

Can be also translated to RDF Schema, metadata models (can be used with query languages as SPARQL) (XML-like annotation)

See tutorial for example =here=, especially =examples=

Semantic web

=Semantic web=, i.e. a web of data
Idea (was) to annotate data so that inferences are possible
E.g. Google data sheets were based on ontologies/semantic web technology
Some problems:

- finding appropriate ontologies,
- annotating data,
- data usually hand-annotated, i.e. unreliable.

SUMO

Suggested Upper Merged Ontology
Formal ontology that is free
Any kind of reasoning/domain
E.g.: Ebay to reason about products

Oracle Configurator Constraint Definition Language

=OCCDL= is an example of an internal tool of automated reasoning
Used for server etc. setups in the Oracle shop
Clients can select their server config
Some configs are not possible (component interactions)
Deciding that by hand would be impossible
The properties of components are modeled in the CDL
A reasoner decides based on the model in the shop if the config is possible

CDL example

```
CONTRIBUTE Frame.Width - 2*Frame.Border + 2*0.5
TO Glass.Width;
CONTRIBUTE Frame.Height- 2*Frame.Border + 2*0.5
TO Glass.Height;
```