Lecture 4: Prolog

COMP24412: Symbolic AI

Martin Riener

School of Computer Science, University of Manchester, UK

February 2019

Overview

Outline

1. Introduction: What is Prolog?
2. Prolog queries
3. Syntax of Prolog programs
4. Unification

Prolog

- Programmation en logique - "Programming in logic"
- Declarative programming language:
  describe solution, not how to get there
- Based on automated theorem proving in FOL (SLD Resolution)
  easier to reason about
- Super-set of Datalog
- Turing-complete

History

- $\sim$1972: Colmerauer and Roussel define language, first implementation
- 1977: Warren writes first compiler (DEC-10 Prolog)
- 1983: Warren abstract machine (WAM)
- 1995: Becomes ISO/IEC 13211-1 standard
- 2000: Latest standard so far ISO/IEC 13211-2

Common Implementations

- Ciao Prolog
- Eclipse
- GNU Prolog
- IF Prolog
- Sicstus Prolog*
- SWI Prolog*
- XSB Prolog
- YAP Prolog

* available during exercise classes

Fields of use

- Prototyping
- Constraint Solving, Logistics
- Parsing, Natural Language Processing
- Search Problems with non-deterministic decisions

Use in Industry

- Query Engine of IBM Watson

  `https://www.theregister.co.uk/2009/04/27/ibm_watson_jeopardy?page=2`

- Clarissa (NASA): speech guided navigations through maintainence procedures on ISS

  `https://ti.arc.nasa.gov/tech/cas/user-centered-technologies/clarissa`

- $\sim 1/3$ of flight bookings in Europe handled by a Prolog system

  `https://www.sics.se/projects/sicstus-prolog-leading-prolog-technology`

Anatomy of a Prolog Program

- Program = Facts + Rules
- Query: "Is this fact derivable from the program?"
- Queries may contain variables
- Answer substitution:
  "Which variable assignments are necessary to derive the query?"
- Queries often have multiple answers!

Outline

1. Introduction: What is Prolog?
2. Prolog queries
3. Syntax of Prolog programs
4. Unification

Prolog terms

- Constant: basic object
- Variable: can be replaced by another term
- Predicate:
  - Rules define predicates
  - Never appear inside another term
  - There is no return value!

Prolog terms

- Predefined predicates:
  - `X = Y`: true if LHS and RHS are equal
    There is no assignment!
  - `dif(X,Y)`: true if LHS and RHS are different (not ISO)
- Function:
  - always appears inside a predicate
  - comparable to datastructures
  - There is no return value!

Prolog terms

```
band_song_date(rihanna, Song,date(Y,M,D))
```

- rihanna : Constant term
  Starts with a lower-case letter
- 'Twist and shout': Quoted constant term
- Song : Variable
  Starts with an upper-case letter
- _: Anonymous variable
  We will not be informed about assignments of this variable
- band_song_date/3 : Predicate (of arity 3) Starts with a lower-case letter
- date/3: function (of arity 3)

Queries

- Suppose we have a database of bands and their songs
- "Is Rihanna's Diamonds in the database?"

```
?- band_song(rihanna, diamonds).
false.
```

Queries

- "Do we know about any song by Rihanna?"

```
?- band_song(rihanna, Song).
false.
```

Queries

- "Is there anything in the database?"

```
?- band_song(Band, Song).
Band = beatles,
Song = 'While␣my␣guitar␣gently␣weeps' ;
Band = beatles,
Song = 'Twist␣and␣shout' ;
Band = beatles,
Song = 'Love␣me␣do'
% ....
```

Queries

- "There's surely more than The Beatles?"

```
?- dif(Band, beatles), band_song(Band, Song).
Band = 'Isley␣Brothers',
Song = 'Twist␣and␣shout' ;
Band = iggy,
Song = 'The␣passenger' ;
Band = banshees,
Song = 'The␣passenger' .
% ....
```

Queries

- "Which versions of The Passenger are there?"

```
?- Song = 'The␣passenger', band_song(Band, Song).
Song = 'The␣passenger',
Band = iggy ;
Song = 'The␣passenger',
Band = banshees ;
Song = 'The␣passenger',
Band = bauhaus.
```

Outline

Anatomy of a query

```
 ?- dif(Band, beatles), band_song(Band, Song).
Band = 'Isley Brothers',
Song = 'Twist and shout';
Band = iggy,
Song = 'The passenger' a
```

- ?- Query prompt
- , Conjunction of queries
- . End of query
- Band = 'Isley Brothers', Song = 'Twist and shout': Answer substitution
- ; User input (next answer)
- a User input (abort)

Turning a query into a rule

- Query

```
?- dif(Band, beatles), band_song(Band, Song).
```

- Rule

```
nobeatles_song(Band, Song) :-
  dif(Band, beatles),
  band_song(Band, Song).
```

Anatomy of a rule

```
head(X,Y,Z) :-
  goal1(X,A),
  goal2(Y,B),
  goal3(A,B,Z).
```

- "Derive head **if** goal1 **and** goal2 **and** goal3 are derivable."
- Predicate logic formula:
  $\forall X, Y, Z, A, B.$
  $goal1(X, A) \wedge goal2(Y, B) \wedge goal3(A, B, Z)$
  $\rightarrow head(X, Y, Z)$

Facts

A fact is always true:

```
coldplace(siberia) :-
  true.
```

Easier to write:

```
coldplace(siberia).
```

Outline

Substitutions

Substitution:

- Maps finitely many variables to terms
- All other variables are mapped to themselves
- Apply $\sigma = \{$`X=car, Y=house`$\}$ to `owns(lucia,X)` and obtain `owns(lucia,car)`
- Apply σ to `owns(lucia,Z)` and obtain `owns(lucia,Z)`
- Substitutions can be composed:
  $\tau = \{$`pair(X,Y)`$\}$
  $\tau\sigma = \{$ `pair(car,house)` $\}$

Unification

```
?- X=1, X=2.
false.
```

Why?

Unification

```
?- X=1, X=2.
false.
```

Why?

- assign $X = 1$:
  $1{=}1,\ 1{=}2.$

Unification

```
?- X=1, X=2.
false.
```

Why?

- assign $X = 1$:
  $1{=}1,\ 1{=}2.$
- assign $X = 2$:
  $1{=}2,\ 2{=}2.$

Are these terms unifiable?

- `contains(X, milk) = contains(capuccino, Y)`

Are these terms unifiable?

- `contains(X, milk) = contains(capuccino, Y)`
  yes
- `contains(X, house) = contains(house, X)`

Are these terms unifiable?

- `contains(X, milk) = contains(capuccino, Y)`
  yes
- `contains(X, house) = contains(house, X)`
  yes
- `contains(X, milk) = contains(capuccino, X)`

Are these terms unifiable?

- `contains(X, milk) = contains(capuccino, Y)`
  yes
- `contains(X, house) = contains(house, X)`
  yes
- `contains(X, milk) = contains(capuccino, X)`
  no
- `climate(X) = climate(Y)`
  yes

Unification Problem

Unification Problem

Given a set of term equalities $\mathtt{s}_1 = \mathtt{t}_1, \ldots, \mathtt{s}_n = \mathtt{t}_n$, is there a unifying substitution σ such that for each equation s=t, sσ and tσ are the same terms?
If yes, which one?

Unification Problem

Unification Problem

Given a set of term equalities $\mathtt{s}_1 = \mathtt{t}_1, \ldots, \mathtt{s}_n = \mathtt{t}_n$, is there a unifying substitution σ such that for each equation s=t, sσ and tσ are the same terms?
If yes, which one?

Unifiers for the problems before:

- `X=cappucino, Y=milk`
- `X=house, Y=X`
- not unifiable
- X=Y

Unification Rules

Transformation rules:

- trivial: $t = t, P$
  delete $t = t$, solve P
- orient: $t = X, P$
  move variable to LHS: $X = t, P$
- decomposition: $f(s1, \ldots, sn) = f(t1, \ldots, tn), P$
  solve $s1 = t1, \ldots, sn = tn, P$
- variable elimination: $X = t, P$
  replace all occurrences of X in P with t, solve P

Unification Rules

Solved form:

- P contains only equations $X = a, Y = b, \ldots$
- No Transformation rule can be applied

Failure cases:

- name clash (constants): $c = d$
- name clash (functions): $f(A, B) = g(X, Y)$
- occurs check: $X = f(X)$
  (X occurs nested inside RHS term)

Unification: examples

- $contains(X, milk) = contains(capuccino, Y)$
  - Decompose: $X = capuccino, milk = Y$
  - Orient: $X = capuccino, Y = milk$
  - Solved!

Unification: examples

- $contains(X, house) = contains(house, X)$
  - Decompose: $X = house, house = X$
  - Eliminate X: $house = house$
  - Remove trivial
  - Solved!

Unification: examples

- $contains(X, milk) = contains(capuccino, X)$
  - Decompose: $X = capuccino, milk = X$
  - Eliminate X: $milk = capuccino$
  - Constant clash
  - Failure!

Most general unifiers

Problem: $f(X) = f(Y)$ has infinitely many unifiers:

```
X=a, Y=a
X=b, Y=b
X=c, Y=c
...
X=f(a), Y=f(a)
X=g(a), Y=g(a)
...
X=Y
```

Most general unifiers

More general substitutions

Let σ and τ be substitutions. If there exists a non-trivial substitution λ such that $\sigma\lambda = \tau$ then σ is *more general* than τ.

Most general unifiers

More general substitutions

Let σ and τ be substitutions. If there exists a non-trivial substitution λ such that $\sigma\lambda = \tau$ then σ is *more general* than τ.

Most general unifier

A unifier is a *most general* substitution if there is no other unifier that is more general.

Most general unifiers

More general substitutions

Let σ and τ be substitutions. If there exists a non-trivial substitution λ such that $\sigma\lambda = \tau$ then σ is *more general* than τ.

Most general unifier

A unifier is a *most general* substitution if there is no other unifier that is more general.

- {X=Y} is more general than $X = a$ (take $\lambda =$ {Y=a})
- {X=b} neither more nor less general than {X=a}

Most general unifiers

More general substitutions

Let σ and τ be substitutions. If there exists a non-trivial substitution λ such that $\sigma\lambda = \tau$ then σ is *more general* than τ.

Most general unifier

A unifier is a *most general* substitution if there is no other unifier that is more general.

Applying the unification algorithm presented, we always obtain the most general unifier (up to renaming of variables).

Summary

- Prolog is a Turing complete, logic based programming language
- Queries to Prolog program yields a sequence of answer substitutions
- Answers are found by backward chaining, trying the rules in order of appearance
- Suitable rules to apply are found via unification
- Functions allow the expression of arbitrary large terms, e.g. lists

That's all for today!