Overview

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

- Clarissa (NASA): speech guided navigations through maintainence procedures on ISS
  https://ti.arc.nasa.gov/tech/cas/user-centered-technologies/clarissa

- ~ 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

```prolog
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)
Suppose we have a database of bands and their songs

“Is Rihanna’s Diamonds in the database?”

?- band_song(rihanna, diamonds).
false.
“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' 
% ....
“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'.
% ....
“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

1. Introduction: What is Prolog?

2. Prolog queries

3. Syntax of Prolog programs

4. Unification
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**

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

- **Rule**

```prolog
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:
  \[ \forall X, Y, Z, A, B. \]
  \[ goal1(X, A) \land goal2(Y, B) \land goal3(A, B, Z) \]
  \[ \rightarrow head(X, Y, Z) \]
Facts

A fact is always true:

coldplace(siberia) :-
  true.

Easier to write:

coldplace(siberia).
1. Introduction: What is Prolog?

2. Prolog queries

3. Syntax of Prolog programs

4. Unification
Substitutions

Substitution:

- Maps finitely many variables to terms
- All other variables are mapped to themselves
- Apply $\sigma = \{X=\text{car}, \ Y=\text{house}\}$ to $\text{owns}(\text{lucia}, X)$ and obtain $\text{owns}(\text{lucia}, \text{car})$
- Apply $\sigma$ to $\text{owns}(\text{lucia}, Z)$ and obtain $\text{owns}(\text{lucia}, Z)$
- Substitutions can be composed:
  $\tau = \{\text{pair}(X,Y)\}$
  $\tau\sigma = \{\ \text{pair}(\text{car}, \text{house}) \ \}$
?- X=1, X=2.
false.

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

Why?

- assign X = 1:
  1=1, 1=2.
?- 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)
  - yes
- contains(X, house) = contains(house, X)
  - yes
- contains(X, milk) = contains(capuccino, X)
  - no
- climate(X) = climate(Y)
  - yes
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)
  no
Are these terms unifiable?

- \( \text{contains}(X, \text{milk}) = \text{contains}(\text{capuccino}, Y) \)  
  yes

- \( \text{contains}(X, \text{house}) = \text{contains}(\text{house}, X) \)  
  yes

- \( \text{contains}(X, \text{milk}) = \text{contains}(\text{capuccino}, X) \)  
  no

- \( \text{climate}(X) = \text{climate}(Y) \)  
  yes
Unification Problem

Given a set of term equalities $s_1 = t_1, \ldots, s_n = t_n$, is there a unifying substitution $\sigma$ such that for each equation $s = t$, $s\sigma$ and $t\sigma$ are the same terms?
If yes, which one?
Unification Problem

Given a set of term equalities $s_1 = t_1, \ldots, s_n = t_n$, is there a unifying substitution $\sigma$ such that for each equation $s=t$, $s\sigma$ and $t\sigma$ are the same terms?
If yes, which one?

Unifiers for the problems before:

- $X=\text{cappucino}, Y=\text{milk}$
- $X=\text{house}, Y=X$
- 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:** \( f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n), P \)
  - solve \( s_1 = t_1, \ldots, s_n = t_n, P \)

- **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

\[ \text{contains}(X, \text{milk}) = \text{contains}(\text{capuccino}, Y) \]

- Decompose: \( X = \text{capuccino}, \text{milk} = Y \)
- Orient: \( X = \text{capuccino}, Y = \text{milk} \)
- Solved!
Unification: examples

- \( contains(X, house) = contains(house, X) \)
  - Decompose: \( X = house, house = X \)
  - Eliminate \( X \): \( house = house \)
  - Remove trivial
  - Solved!
Unification: examples

- \( \text{contains}(X, \text{milk}) = \text{contains}(\text{capuccino}, X) \)
  - Decompose: \( X = \text{capuccino}, \text{milk} = X \)
  - Eliminate \( X \): \( \text{milk} = \text{capuccino} \)
  - Constant clash
  - Failure!
Most general unifiers

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

- \( X=a, \ Y=a \)
- \( X=b, \ Y=b \)
- \( X=c, \ Y=c \)
- ... 
- \( X=f(a), \ Y=f(a) \)
- \( X=g(a), \ Y=g(a) \)
- ... 
- \( X=Y \)
More general substitutions

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

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

Most general unifier

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

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

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 $\{X=a\}$
More general substitutions

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

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!