Matching and Proof Search

TK2ICM: Logic Programming (2nd Term 2018-2019)

M. Arzaki

Computing Laboratory, School of Computing
Telkom University

SoC Tel-U

February 2019

Acknowledgements

This slide is compiled using the materials in the following sources:
Books:

1. P. Blackburn, J. Bos, K. Striegnitz, *Learn Prolog Now!* (Chapter 1-6, 10,11), London: College Publications (available online), 2006. **[LPN]**
2. M. Bramer, *Logic Programming with Prolog* (Chapter 1-9), 2nd Edition, Springer, 2013. **[LPwP]**
3. I. Bratko, *Prolog Programming for Artificial Intelligence* (Chapter 1-3, 5,6,8,9), Pearson Education, 2001. (advanced reference). **[PPAI]**
4. K. H. Rosen, *Discrete Mathematics and Its Applications* (Chapter1), 7th Edition, 2012.
5. M. Ben-Ari, *Mathematical Logic for Computer Science* (Logic Programming Sections), 2nd Edition, 2000.

Lecture slides and lecture notes:

1. *Prolog Programming* by Kristina Striegnitz.
2. *Learn Prolog Now!* by Patrick Blackburn, Johan Bos, and Kristina Striegnitz.
3. *Logic Programming* at Fasilkom UI by A. A. Krisnadhi and A. Saptawijaya.
4. *Computational Logic Part 2: Logic Programming* at Fasilkom UI by L. Y. Stefanus.
5. *Logic Programming* at ILLC, University of Amsterdam by U. Endriss.
6. *Functional Programming* at Fasilkom UI by A. Azurat.
7. *Bahasa Prolog* at FPMIPA UPI by Munir.
8. Other available sources online.

Contents

Contents

Matching (or Unification)

Recall that there are three types of terms:

1. Constants. These can either be atoms (such as `vincent`) or numbers (such as 24).
2. Variables. These can either start with a capital letter (such as `Person`) or an underscore (such as `_X`).
3. Complex terms. These have the form: `functor(term_1,...,term_n)`.

Definition

Two terms **match** (or **unify**), if they are equal or if they contain variables that can be instantiated in such a way that the resulting terms are equal.

Remark

Matching in Prolog corresponds to unification in the theory of logic programming, however there are not exactly identical. We will see the differences in this lecture. Nevertheless, they share common characteristics.

Examples

- `mia` and `mia` unify (or match)
- `42` and `42` unify (or match)
- `woman(mia)` and `woman(mia)` unify (or match)

Examples

- `mia` and `mia` unify (or match)
- 42 and 42 unify (or match)
- `woman(mia)` and `woman(mia)` unify (or match)
- `vincent` and `mia` do not unify (or match)

Examples

- `mia` and `mia` unify (or match)
- `42` and `42` unify (or match)
- `woman(mia)` and `woman(mia)` unify (or match)
- `vincent` and `mia` do not unify (or match)
- `woman(mia)` and `woman(jody)` do not unify (or match)

KB4 review

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marcellus,mia).
loves(pumpkin,honey_bunny).
loves(honey_bunny,pumpkin).
```

- `mia` and `X` unify
- `woman(mia)` and `woman(X)` unify,

KB4 review

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marcellus,mia).
loves(pumpkin,honey_bunny).
loves(honey_bunny,pumpkin).
```

- `mia` and `X` unify
- `woman(mia)` and `woman(X)` unify, this can be obtained by instantiating `X` to `mia`
- `loves(vincent,X)` and `loves(X,mia)` does not unify,

KB4 review

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marcellus,mia).
loves(pumpkin,honey_bunny).
loves(honey_bunny,pumpkin).
```

- `mia` and `X` unify
- `woman(mia)` and `woman(X)` unify, this can be obtained by instantiating `X` to `mia`
- `loves(vincent,X)` and `loves(X,mia)` does not unify, it is impossible to find an instantiation of `X` that makes the two terms equal:

KB4 review

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marcellus,mia).
loves(pumpkin,honey_bunny).
loves(honey_bunny,pumpkin).
```

- `mia` and `X` unify
- `woman(mia)` and `woman(X)` unify, this can be obtained by instantiating `X` to `mia`
- `loves(vincent,X)` and `loves(X,mia)` does not unify, it is impossible to find an instantiation of `X` that makes the two terms equal:
    - if `X = vincent`, then we have `loves(vincent,vincent)` and `loves(vincent,mia)` which are different

KB4 review

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marcellus,mia).
loves(pumpkin,honey_bunny).
loves(honey_bunny,pumpkin).
```

- `mia` and `X` unify
- `woman(mia)` and `woman(X)` unify, this can be obtained by instantiating `X` to `mia`
- `loves(vincent,X)` and `loves(X,mia)` does not unify, it is impossible to find an instantiation of `X` that makes the two terms equal:
  - if `X = vincent`, then we have `loves(vincent,vincent)` and `loves(vincent,mia)` which are different
  - if `X = mia`, then we have `loves(vincent,mia)` and `loves(mia,mia)` which are different

How Does Prolog Match/Unify Terms?

- When Prolog unifies two terms, it performs **all the necessary instantiations**, so that the terms are equal afterwards.
- This makes unification a very powerful programming mechanism.

Matching/Unification in Prolog

- The most important operation on terms is matching.
- Matching is a process that takes two terms as input and checks whether they match.
- The operator = can be used to ask Prolog explicitly to do matching.
- Terms `date(D,M,2019)` and `date(1,feb,Y)` match as follows:
  1. `D` is instantiated to `1`
  2. `M` is instantiated to `feb`
  3. `Y` is instantiated to `2019`
- Terms `date(D,M,2019)` and `date(E,feb,Y)` match as follows:
  1. `D` is instantiated to `E`
  2. `M` is instantiated to `feb`
  3. `Y` is instantiated to `2019`

More about Matching/Unification

How many instantiations that make terms `date(D,M,2019)` and `date(E,feb,Y)` match?

1. We can make `D = 1`, `E = 1`, `M = feb`, and `Y = 2019`.
2. We can make `D = 30`, `E = 30`, `M = feb`, and `Y = 2019`.
3. We can make `D = pertama`, `E = pertama` , `M = feb`, and `Y = 2019`.
4. etc, as long as `D = E` is satisified.

The instantiation `D = E` is more general than the above three instantiations.

Matching in Prolog always produces the most general instantiation.

1. Instantiation that commits the variables to the **least possible extent**.
2. Leaving the greatest possible freedom for further instantiations if further matching is required.

Example

Test the following expression in Prolog

```
date(D,M,2019) = date(E,feb,Y), date(D,M,2019) = date(9,M,Y).
```

Matching/Unification in Prolog: Precise Definiton

Definition (matching/unification in Prolog)

1. If `term1` and `term2` are constants, then `term1` and `term2` match if and only if they are the same atom, or the same number.

Matching/Unification in Prolog: Precise Definiton

Definition (matching/unification in Prolog)

1. If `term1` and `term2` are constants, then `term1` and `term2` match if and only if they are the same atom, or the same number.
2. If `term1` is a variable and `term2` is any type of term, then `term1` and `term2` match, and `term1` is instantiated to `term2`. Similarly, if `term2` is a variable and `term1` is any type of term, then `term1` and `term2` match, and `term2` is instantiated to `term1`.
3. If `term1` and `term2` are complex terms, then they match if and only if:

Matching/Unification in Prolog: Precise Definiton

Definition (matching/unification in Prolog)

1. If `term1` and `term2` are constants, then `term1` and `term2` match if and only if they are the same atom, or the same number.
2. If `term1` is a variable and `term2` is any type of term, then `term1` and `term2` match, and `term1` is instantiated to `term2`. Similarly, if `term2` is a variable and `term1` is any type of term, then `term1` and `term2` match, and `term2` is instantiated to `term1`.
3. If `term1` and `term2` are complex terms, then they match if and only if:
    1. They have the same functor and arity.

Matching/Unification in Prolog: Precise Definiton

Definition (matching/unification in Prolog)

1. If `term1` and `term2` are constants, then `term1` and `term2` match if and only if they are the same atom, or the same number.
2. If `term1` is a variable and `term2` is any type of term, then `term1` and `term2` match, and `term1` is instantiated to `term2`. Similarly, if `term2` is a variable and `term1` is any type of term, then `term1` and `term2` match, and `term2` is instantiated to `term1`.
3. If `term1` and `term2` are complex terms, then they match if and only if:
    1. They have the same functor and arity.
    2. All their corresponding arguments match.

Matching/Unification in Prolog: Precise Definiton

Definition (matching/unification in Prolog)

1. If `term1` and `term2` are constants, then `term1` and `term2` match if and only if they are the same atom, or the same number.
2. If `term1` is a variable and `term2` is any type of term, then `term1` and `term2` match, and `term1` is instantiated to `term2`. Similarly, if `term2` is a variable and `term1` is any type of term, then `term1` and `term2` match, and `term2` is instantiated to `term1`.
3. If `term1` and `term2` are complex terms, then they match if and only if:
    1. They have the same functor and arity.
    2. All their corresponding arguments match.
    3. The variable instantiations are compatible.

Prolog Unification: =/2 Predicate

Exercise

Check whether the following terms unify or not and give the reason:

1. `mia = vincent`
2. `vincent = vincent`
3. `mia = X`
4. `X = mia, X = vincent`

`X = mia, X = vincent` do not unify because after working through the first goal, Prolog has instantiated `X` with `mia`, so that it cannot unify it with `vincent` anymore. Hence the second goal fails.

Exercise

Check whether the following terms unify or not and give the reason:

1. `date(D,M,2019) = date(D1,feb,Y),`
   `date(D,M,2019) = date(1,M,Y).`
2. `k(Z,f(X,b,Z)) = k(h(X),f(g(a),Y,Z)).`
3. `date(1,feb) = date(Day, Month, Year)`
4. `k(s(g),Y) = k(X,t(k)).`
5. `k(s(g),t(k)) = k(X,t(Y)).`
6. `loves(X,X) = loves(marsellus,mia).`

Unification in Theory of Logic Programming: Martelli-Montanari Algorithm

Suppose f and g are functors, each s_i and t_i are terms.

1. If $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$, then replace this equation with $s_1 = t_1$, $s_2 = t_2$, …, $s_n = t_n$.
2. If $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_m)$ with $n \neq m$, then halt with failure.
3. If $f(s_1, \ldots, s_n) = g(t_1, \ldots, t_n)$ with $f \neq g$, then halt with failure.
4. If $x = x$, then delete the equation.
5. If $t = x$ and t is not a variable then replace the equation with $x = t$.
6. If $x = t$ and x does not occcur in t and x occurs elsewhere, then instantiate every x to t on all other equations.
7. If $x = t$ and x occurs in t and $t \neq x$, then halt with failure.

Matching in Prolog vs. Unification in Theory of LP

Matching does not take into account the "occur check" problem. this means omitting the occur check in action (6) and dropping action (7).

Exercise

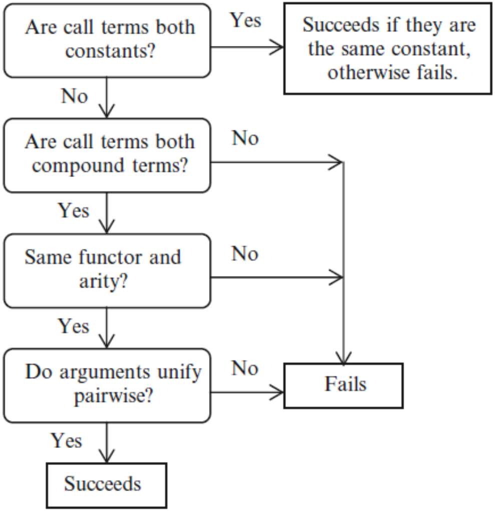
Test the following expression:

1. `k(Z,f(X,b,Z)) = k(h(X),f(g(a),Y,Z)).`
2. `date(26,sept) = date(Day, Month, Year).`
3. `X = f(X).`

Call Terms

- Every goal must be a Prolog term, **but not any kind of term**.
- It may only be an atom or a compound term, not a number, variable, list or any other type of term provided by some particular implementation of Prolog.
- This restricted type of term is **called a call term**.

Matching Algorithm’s Flowchart


Are call terms both constants?
Yes
Succeeds if they are the same constant, otherwise fails.
No
Are call terms both compound terms?
No
Yes
Same functor and arity?
No
Yes
Do arguments unify pairwise?
No
Fails
Yes
Succeeds

Contents

Programming with Matching

Sometimes we can write useful programs simply by using complex terms to define interesting concepts.

Problem

Given two points $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ that are respectively represented as `point(X1,Y1)` and `point(X2,Y2)` in Prolog. How do we reason that the line segment of these two points forms a vertical line or a horizontal line.

Remark

The line segments of $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ forms:

- a vertical line iff $x_1 = x_2$
- a horizontal line iff $y_1 = y_2$

Representing Vertical and Horizontal Line with Matching

The following solution is due to Ivan Bratko:

```
% to denote vertical line
vertical(line(point(X,_),point(X,_))).
% to denote horizontal line
vertical(line(point(_,Y),point(_,Y))).
```

Exercise

Check the output of the following queries:

1. `vertical(line(point(1,1),point(1,3))).`
2. `vertical(line(point(1,1),point(3,2))).`
3. `horizontal(line(point(1,1),point(1,Y))).`
4. `horizontal(line(point(2,3),Point)).`

Contents

Proof Search and Search Tree

- Now that we know about unification, we are in a position to learn how Prolog searches a knowledge base to see if a query is satisfied.
- In other words: we are ready to learn about <u>proof search</u> and <u>search trees</u>.

Proof Search

```
wizard(harry).
wizard(ron).
wizard(hermione).
muggle(uncle_vernon).
muggle(aunt_petunia).
chases(crookshanks,scabbars).
```

Suppose we pose the query `?- wizard(X).`

Proof Search

```
wizard(harry).
wizard(ron).
wizard(hermione).
muggle(uncle_vernon).
muggle(aunt_petunia).
chases(crookshanks,scabbars).
```

Suppose we pose the query `?- wizard(X).`

- Prolog checks for facts that match the query. (There are three.)

Proof Search

```
wizard(harry).
wizard(ron).
wizard(hermione).
muggle(uncle_vernon).
muggle(aunt_petunia).
chases(crookshanks,scabbars).
```

Suppose we pose the query `?- wizard(X).`

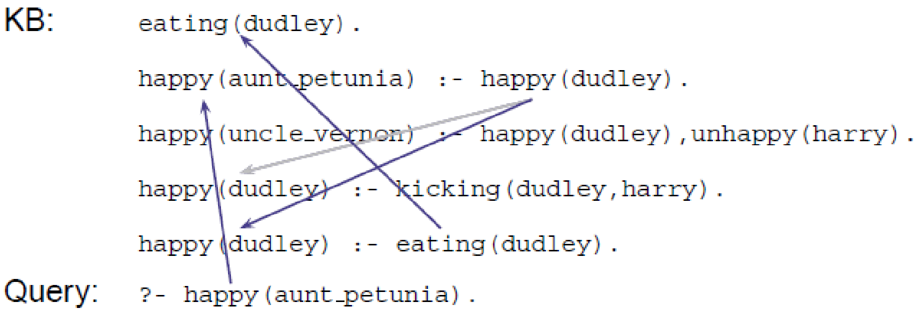
- Prolog checks for facts that match the query. (There are three.)
- Prolog starts from the top of the knowledge base and, therefore, finds `wizard(harry)` first.

Proof Search

```
wizard(harry).
wizard(ron).
wizard(hermione).
muggle(uncle_vernon).
muggle(aunt_petunia).
chases(crookshanks,scabbars).
```

Suppose we pose the query `?- wizard(X).`

- Prolog checks for facts that match the query. (There are three.)
- Prolog starts from the top of the knowledge base and, therefore, finds `wizard(harry)` first.
- Typing [tab] forces Prolog to check whether there are other possibilities



```
KB:     eating(dudley).
        happy(aunt_petunia) :- happy(dudley).
        happy(uncle_vernon) :- happy(dudley),unhappy(harry).
        happy(dudley) :- kicking(dudley,harry).
        happy(dudley) :- eating(dudley).
Query:  ?- happy(aunt_petunia).
```

Prolog works as follows:

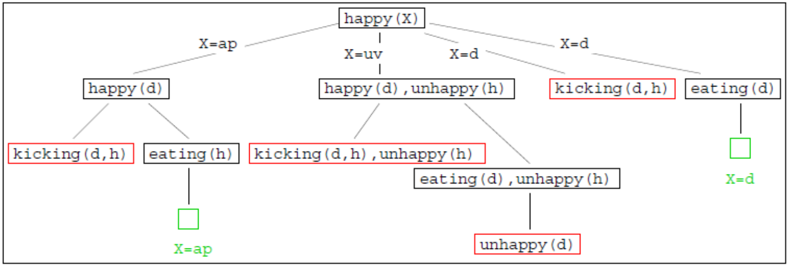
- Check for a fact or a rule's head that match the query.
- If Prolog finds a fact, it is done.
- If Prolog finds a rule, it tries to prove all goals specified in the body of the rule.

Search Tree

Search tree is a graphical representation of a proof search.

KB:
```
eating(dudley).
happy(aunt_petunia):-happy(dudley).
happy(uncle_vernon):-happy(dudley),unhappy(harry).
happy(dudley):-kicking(dudley,harry).
happy(dudley):-eating(dudley).
```

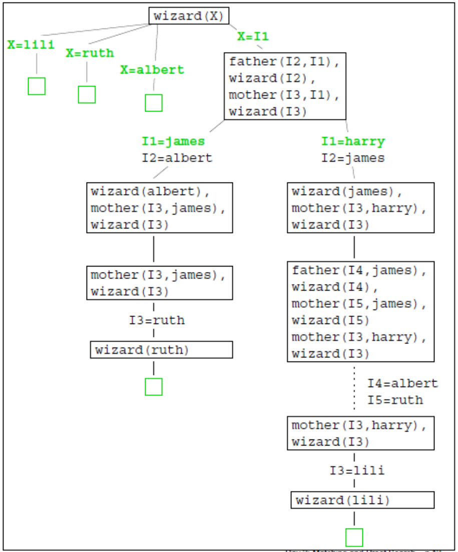
Query:
```
?- happy(X).
```



Example 5

```
father(albert,james).
father(james,harry).
mother(ruth,james).
mother(lili,harry).

wizard(lili).
wizard(ruth).
wizard(albert).
wizard(X) :-
            father(Y,X),
            wizard(Y),
            mother(Z,X),
            wizard(Z).
```


wizard(X)
X=lili
X=ruth
X=albert
X=I1
father(I2,I1), wizard(I2), mother(I3,I1), wizard(I3)
I1=james
I2=albert
I1=harry
I2=james
wizard(albert), mother(I3,james), wizard(I3)
wizard(james), mother(I3,harry), wizard(I3)
mother(I3,james), wizard(I3)
I3=ruth
wizard(ruth)
father(I4,james), wizard(I4), mother(I5,james), wizard(I5) mother(I3,harry), wizard(I3)
I4=albert
I5=ruth
mother(I3,harry), wizard(I3)
I3=lili
wizard(lili)
Day 2: Matching and Proof Search – p.12

Contents

Goals and Subgoals

Suppose we have a rule of the form: <head>:- t1,t2,...,tn, where n ≥ 1.

- The head can also be viewed as a *goal*.
- Each of the term t1, t2, ..., tn in the body can also be viewed as a *subgoal*.
- In order to achieve the goal <head>, it is necessary to achieve all subgoals t1, t2, ..., tn.

Goal Evaluation in Prolog

- Prolog searches through the database from top to bottom examining those clauses whose have heads with the same functor and arity as the goal.

Goal Evaluation in Prolog

- Prolog searches through the database from top to bottom examining those clauses whose have heads with the same functor and arity as the goal.
    - If there are none the goal fails.

Goal Evaluation in Prolog

- Prolog searches through the database from top to bottom examining those clauses whose have heads with the same functor and arity as the goal.
    - If there are none the goal fails.
    - If it yields a unification, the outcome depends on whether the clause is a rule or a fact.

Goal Evaluation in Prolog

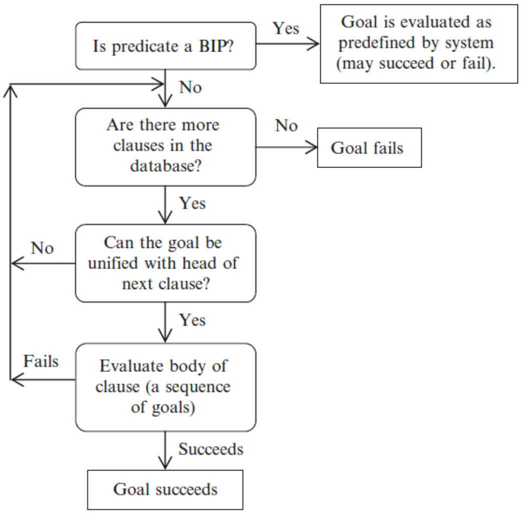
- Prolog searches through the database from top to bottom examining those clauses whose have heads with the same functor and arity as the goal.
  - If there are none the goal fails.
  - If it yields a unification, the outcome depends on whether the clause is a rule or a fact.
- If the clause is a fact, the goal succeeds immediately.

Goal Evaluation in Prolog

- Prolog searches through the database from top to bottom examining those clauses whose have heads with the same functor and arity as the goal.
  - If there are none the goal fails.
  - If it yields a unification, the outcome depends on whether the clause is a rule or a fact.
- If the clause is a fact, the goal succeeds immediately.
- If the clause is a rule, Prolog evaluates the subgoals (*the goals in the body*) one by one, **from left to right**.

Goal Evaluation in Prolog

- Prolog searches through the database from top to bottom examining those clauses whose have heads with the same functor and arity as the goal.
  - If there are none the goal fails.
  - If it yields a unification, the outcome depends on whether the clause is a rule or a fact.
- If the clause is a fact, the goal succeeds immediately.
- If the clause is a rule, Prolog evaluates the subgoals (*the goals in the body*) one by one, **from left to right**.
- If all the subgoals succeeds, the original goal succeeds (assuming that the body does not contain ; operator).
- Note: if the goal is a BIP (built-in predicate) it will evaluated by the system.


Is predicate a BIP?
Yes
Goal is evaluated as predefined by system (may succeed or fail).
No
Are there more clauses in the database?
No
Goal fails
Yes
Can the goal be unified with head of next clause?
No
Yes
Evaluate body of clause (a sequence of goals)
Fails
Succeeds
Goal succeeds

Contents

Exercise: Sentence Generation

In this exercise we use the application of matching in Prolog to generate a sentence. Here is a tiny lexicon and mini grammar with only one rule which defines a sentence as consisting of five words: an article, a noun, a verb, and again an article and a noun.

```
word(article,a)  word(article,every).
word(noun,criminal).  word(noun,'beef burger').
word(verb,eats).  word(verb,likes).
```

Your task is to define a predicate `sentences/0` that returns any possible sentence which is grammatically correct, i.e., it complies to the following grammatical rules:

$$\langle sentence\rangle ::= \ \langle article\rangle \mid \langle noun\rangle \mid \langle verb\rangle \mid \langle article\rangle \mid \langle noun\rangle$$
$$\langle article\rangle ::= \ a \mid every$$
$$\langle noun\rangle ::= \ criminal \mid beef\ burger$$
$$\langle verb\rangle ::= \ eats \mid likes$$

Every word must be separated by a space.

Input/output examples:

```
sentences. returns
a criminal eats a criminal
true ;
a criminal eats a beef burger
true ;
a criminal eats every criminal
true ;
a criminal eats every beef burger
true ;
a criminal likes a criminal
true ;
⋮
```