CHAPTER 5
PROLOG

Presented by: Dr. R. Bentrcia
Department of Computer Science
University of M’sila

Outline

- Overview
- Prolog Basic Components
  - *Facts*
  - *Rules*
  - *Queries*
- Prolog terms
- Arity in Prolog
- Unification
- Backtracking
  - *How backtracking works in Prolog*
  - *Disadvantages of Backtracking in Prolog*
  - *Backtracking Example*
- Recursion in Prolog
  - *Recursion Example*
  - *How recursion works in Prolog*

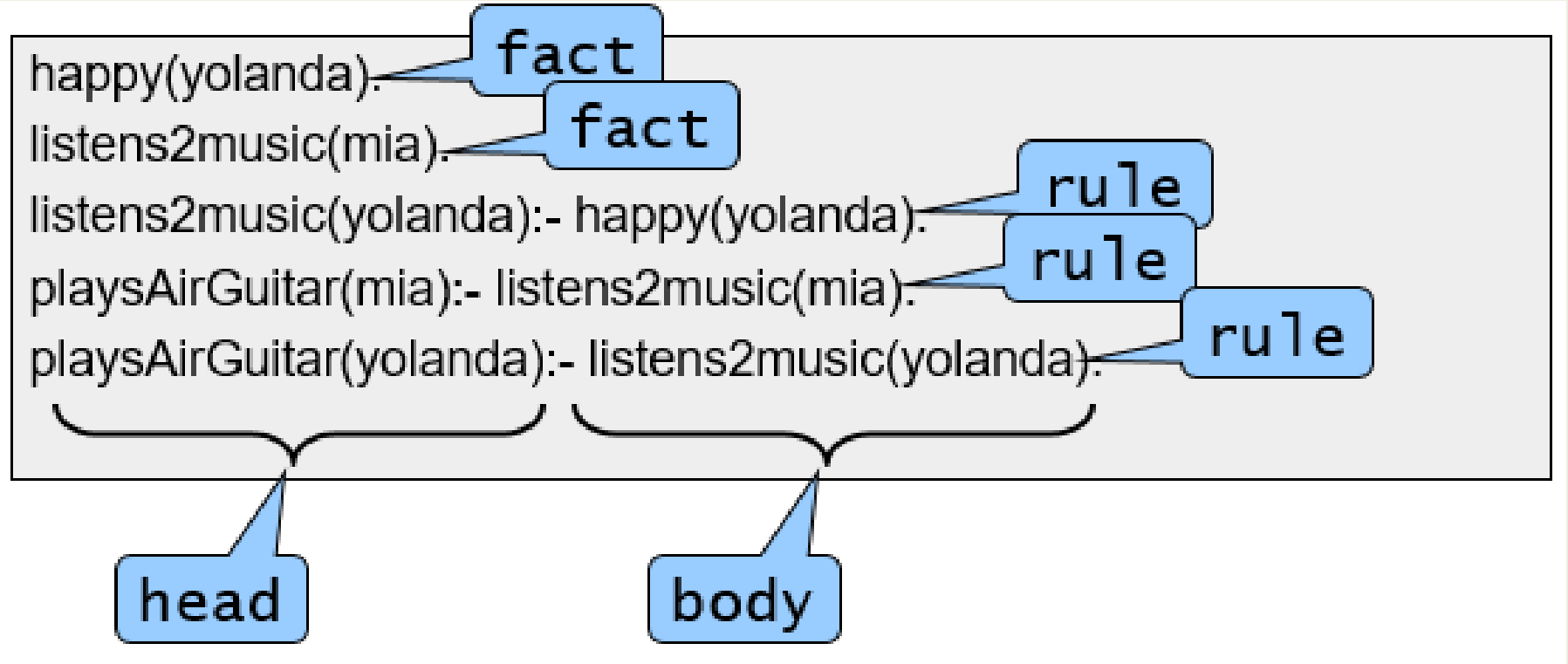
Overview

- Prolog (**Pro**gramming in **Log**ic) is one of the classical programming languages developed in 1970 for AI applications.
- It is a declarative programming language not like C and Java imperative.
  - *Imperative programming: specifying how to achieve a certain goal in a certain situation (algorithm).*
  - *Declarative programming: specifying what the situation (rules and facts) and the goal (query) are and let the Prolog interpreter derive the solution.*
- To obtain the solution, the user asks a question rather than running a program.
- When a user asks a question, then to determine the answer, the run time system searches through the database of facts and rules.

Prolog Basic Components

- A Prolog program is a sequence of clauses.
- **A clause** could be a rule or a fact.
- **A fact** is a predicate followed by a full stop.
- **A rule** consists of a head (a predicate) and a body followed by a full stop.
- After compilation a Prolog program is run by submitting queries to the interpreter. **A query** has the same structure as the body of a rule.

Prolog Basic Components



There are five **clauses** in this knowledge base: two facts, and three rules.
There are three **predicates** in this knowledge base: happy, listens2music, and playsAirGuitar.

Prolog Basic Components

- Prolog has logical operators:
  - *Implication* **:-**
  - *Conjunction* **,**
  - *Disjunction* **;**

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch); listens2music(butch).
```

Prolog Basic Components

- Sample of queries

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

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
no
```

Prolog Basic Components

- Using variables in queries:

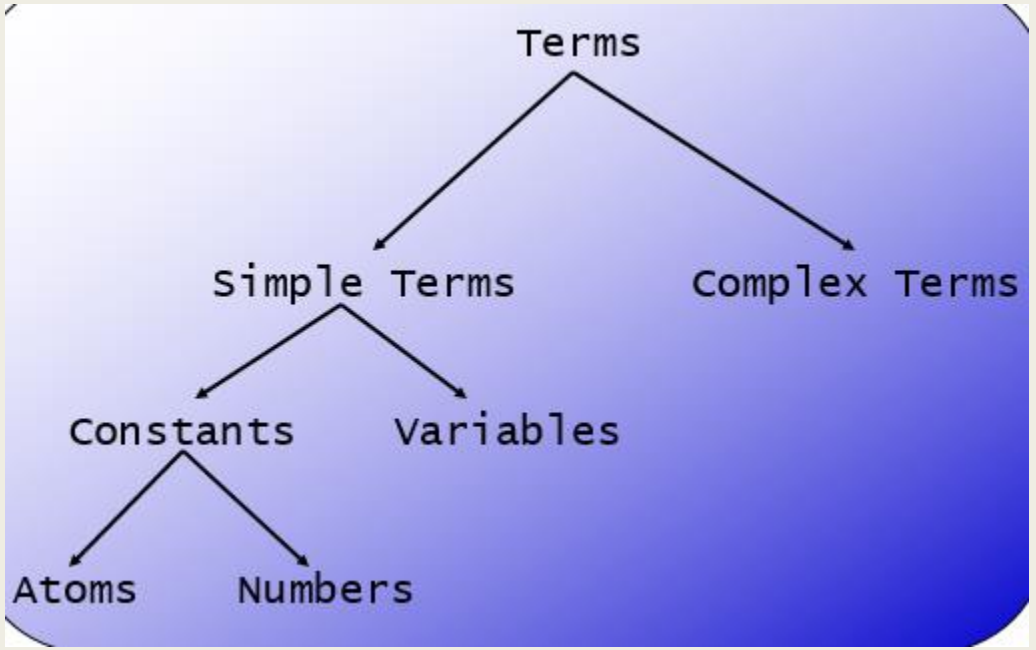
```
woman(mia).
woman(jody).
woman(yolanda).
```

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no
```

Prolog Terms

- There are **terms** of four kinds: atoms, numbers, variables, and compound terms.
- **Atoms**: Atoms are usually strings made up of lower- and uppercase letters, digits, and the underscore, starting with a lowercase letter. b, abcXYZ, x_123, ‘aaa’
- **Numbers**: All Prolog implementations have an integer type: a sequence of digits, optionally preceded by a - (minus). Some also support floats. 145, -10, 18,2
- **Variables**: Variables are strings of letters, digits, and the underscore, starting with a capital letter or an underscore. X, Elephant, _4711
- **Compound/complex terms**: Compound terms are made up of a functor (a Prolog atom) and a number of arguments (Prolog terms, i.e., atoms, numbers, variables, or other compound terms) enclosed in parentheses and separated by commas. playsAirGuitar(jody), father(X, Y)

Prolog Terms



Source: P Blackburn, J Bos & K Striegnitz, learn prolog now, 2001

Arity in Prolog

- We can define two predicates with the same functor but with different arity.
- Prolog would treat this as two different predicates.
- It is indicated with the suffix "/" followed by a number.
- Example about the previous knowledge base:
  - *happy/1*
  - *listens2music/1*
  - *playsAirGuitar/1*

Unification

- Prolog uses the unification technique (matching technique).
- In unification, one or more variables being given value to make the two call terms identical. This process is called binding the variables to values.
- Unification rules according to (Blackburn et al., 2001):
  - *If T1 and T2 are constants, then*
    T1 and T2 unify if they are the same atom, or the same number.
  - *If T1 is a variable and T2 is any type of term, then T1 and T2 unify, and T1 is instantiated to T2. (and vice versa)*
  - *If T1 and T2 are complex terms then they unify if:*
    - They have the same functor and arity, and
    - All their corresponding arguments unify, and
    - The variable instantiations are compatible.

Unification

- Examples:
  - *parent(A, B, C)*
  - *parent(kevin, henry, 30)*
- It will succeed with A, B, and C variables bound to kevin, henry, and 30, respectively.
  - *pred2(A, A, male)*
  - *pred2(canada, cat, X)*
- The variable A cannot unify with canada and cat, so it will fail.

Unification

- Prolog queries:

?- mia = X.

X=mia

yes

?- X=mia, X=vincent.

no

?- father(X) = X.

X=father(father(father(...))))

yes

k(s(g),Y) = k(X,t(k)).

X=s(g)

Y=t(k)

yes

?- k(s(g),t(k)) = k(X,t(Y)).

X=s(g)

Y=k

yes

?-

Backtracking

- Backtracking refers to the process of exploring alternative choices when trying to satisfy a query or reach a goal.
- It allows the Prolog system to search for multiple solutions, find the best solution, or explore different branches of a search tree until a satisfactory result is found.
- Backtracking allows Prolog to handle complex decision-making tasks where decisions depend on multiple criteria, conditions, and rules.

How backtracking works in Prolog

- **Goal Satisfaction:** When you pose a query or goal in Prolog, the Prolog system attempts to satisfy that goal by searching for a combination of rules, facts, and data that match the query.
- **Rule Evaluation:** Prolog evaluates rules and predicates in a depth-first manner. It starts with the first available choice, executes it, and proceeds to the next subgoal or condition within the rule.
- **Unification:** During rule evaluation, Prolog uses a process called unification to match the query or goal with the available facts and rules. Unification is the process of finding values for variables such that the query and rule match.

How backtracking works in Prolog

- **Backtracking:** If a subgoal or condition within a rule fails to be satisfied, Prolog automatically backtracks to the most recent choice point. This means it abandons the current choice and explores other available alternatives.
- **Choice Points:** Prolog maintains a record of choice points, which are points in the execution where alternative choices or solutions are possible. When backtracking occurs, Prolog revisits these choice points and explores different branches of the search tree.
- **Multiple Solutions:** Prolog continues to backtrack and explore alternative choices and branches until it finds a solution that satisfies the query. If a solution is found, it is reported to the user. Prolog can continue searching for additional solutions if they exist.

Disadvantages of Backtracking in Prolog

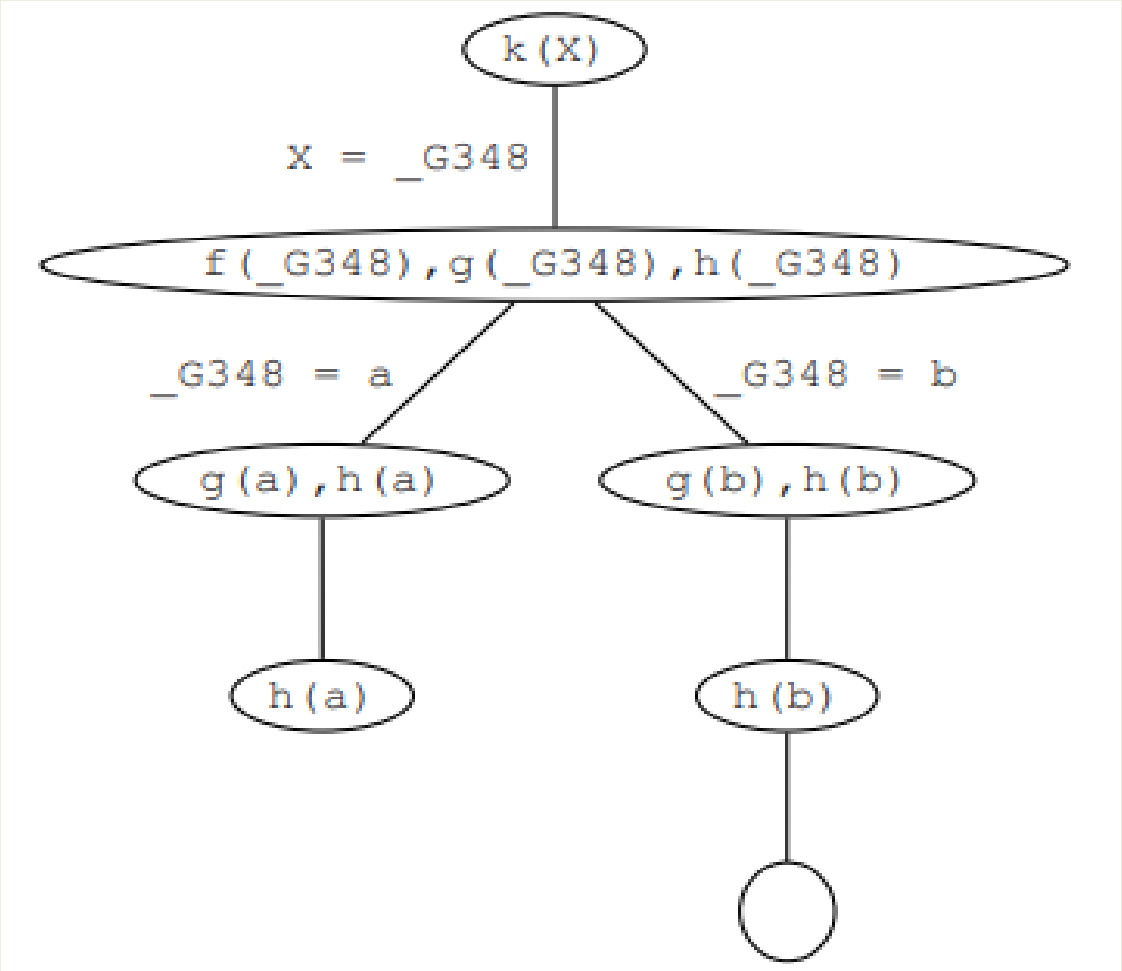
- **Inefficient Performance:** Backtracking can be computationally expensive, especially in cases where the search space is vast or the problem involves an extensive number of choices.
- **Exponential Time Complexity:** Some problems that require exhaustive search and backtracking may exhibit exponential time complexity. This means that the execution time grows rapidly as the problem size increases.
- **Memory Consumption:** Backtracking can consume significant memory, especially when dealing with deep recursion or large search trees.
- **Potential for Infinite Loops:** If not properly constrained, recursive predicates in Prolog can lead to infinite loops during backtracking.

Backtracking Example

- The graphical representation of backtracking is a tree structure called a search tree.
- The nodes of the tree are the goals that have to be satisfied at a certain point during the search.
- The edges we keep track of the variable instantiations that are made when the current goal is match to a fact or the head of a rule in the knowledge base.
- Leave nodes which still contain unsatisfied goals are point where Prolog failed, because it made a wrong decision somewhere along the path.
- Leave nodes with an empty goal list, correspond to a possible solution.

Backtracking Example

- Suppose we are working with the following knowledge base:
  - *f(a).*
  - *f(b).*
  - *g(a).*
  - *g(b).*
  - *h(b).*
  - *k(X) :- f(X), g(X), h(X).*
- Suppose we then pose the query:
  - *k(X).*



Recursion in Prolog

- A predicate is recursively defined if one or more rules in its definition refers to itself.
- When writing a recursive predicate, it should always have at least two clauses: a base clause (the clause that stops the recursion at some point), and one that contains the recursion.

Recursion Example

- Consider the following knowledge base:
  - *is_digesting(X,Y) :- just_ate(X,Y).*
  - *is_digesting(X,Y) :- just_ate(X,Z), is_digesting(Z,Y).*
  - *just_ate(mosquito,blood(john)).*
  - *just_ate(frog,mosquito).*
  - *just_ate(stork,frog).*
- We pose the query:

  ?- is_digesting(stork,mosquito).

How recursion works in Prolog

- By unifying X with stork and Y with mosquito it obtains the following goal: just_ate(stork,mosquito) but the knowledge base doesn’t contain the information.
- Prolog tries to use the second rule. By unifying X with stork and Y with mosquito it obtains the following goals: just_ate(stork,Z), is_digesting(Z,mosquito).
- And there is such a value for Z, namely frog. It is immediate that just_ate(stork,frog). will succeed, for this fact is listed in the knowledge base.
- And deducing is_digesting(frog,mosquito). is almost as simple, for the first clause of is_digesting/2 reduces this goal to deducing just_ate(frog,mosquito). and this is a fact listed in the knowledge base.

References

- F. W. Clocksin and C. S. Mellish. Programming in Prolog. 5th edition, Springer -Verlag, 2003.
- I. Bratko. Prolog Programming for Artificial Intelligence. 4th edition, Addison Wesley Publishers, 2012
- Examples and figures are taken from: Patrick Blackburn, Johan Bos & Kristina Striegnitz, Learn Prolog Now, 2001
- Ulle Endriss, An Introduction to Prolog Programming, 2021
- https://piembsystech.com/backtracking-in-prolog-language/