Rule-Based Systems: Logic Programming

Our first rule-based system

father(peter,mary)
father(peter,john)
mother(mary,mark)
mother(jane,mary)

---

father(X,Y) AND father(Y,Z) → grandfather(X,Z)

father(X,Y) AND mother(Y,Z) → grandfather(X,Z)

mother(X,Y) AND father(Y,Z) → grandmother(X,Z)

mother(X,Y) AND mother(Y,Z) → grandmother(X,Z)

father(X,Y) AND father(X,Z) → sibling(Y,Z)

mother(X,Y) AND mother(X,Z) AND Y =/= Z → sibling(Y,Z)

The rules can be used to

- Derive all grandparent and sibling relationships (forward chaining)
- Answer questions about relationships (backward chaining)

Logic Programming

- Logic programming is the use of
  - logic as a declarative representation language
  - Backward chaining as inference rule
- Logic Programming is the basis of the programming language PROLOG

Logic Programs – A Sequence of Horn Clauses

- The sentences of a logic program are Horn clauses
  - Facts: H
  - Rules: $H \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_n$
- A Horn clause without any head H is called a query
  - Query: $\leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_n$
- Queries are not part of a logic program, they start the inference

Predicates and Literals

- Predicates are the building blocks of clauses
- Predicates have a name and arguments (parameters). Arity is the number of arguments.
- Predicates combine values which “make sense” together (are true)
- Examples:
  - person(peter)
  - married(peter, cindy)
  - appointment(knut, “AB1”, “Lecture KE”)
  - not female(knut)
- Literals are predicates and negated predicates

Exercises (1/2)

- Write as a logic program
  - john is a person
  - peter and mary are persons
  - fhnw is a university
  - john is matriculated at fhnw
  - A student is a person who is matriculated at an university.
  - Is john a student?
  - Is peter a student?

Exercises (2/2)

- Write as a logic program
  - knut is a person
  - «KEBI» is a class
  - classes are taught by teachers
  - john attends to class «KEBI»
  - students are attending to classes
  - Is john a student?
  - knut teaches «KEBI»
  - Is knut a teacher?

PROLOG

- PROLOG (= PROgramming in LOGic) is a programming language based on Horn clauses
  - Clauses are either facts or rules
  - Prolog uses „:-“ instead of „$\leftarrow$“
  - Literals in the body of a rule are separated by comma „ ,“ (the comma is equivalent to the logical AND or „$\wedge$“)
  - Each clause ends with a period „.“
- The following knowledge base consists of four facts and a rule

```
is_a_dog(pluto).
is_a_dog(snoopy).
is_a_sailor(popeye).
eats(popeye,spinach).

is_strong(popeye) :- eats(popeye,spinach).
is_strong_sailor(popeye) :- is_a_sailor(popeye),
                            eats(popeye,spinach).
```

Predicates

- All clauses whose rule headers have the same predicate symbol **and** the same arity together **define** a predicate.
- Predicates can have arbitrary arity:

```
it_rains.
eats(popeye,spinach).
eats_spinach(popeye).
```

`it_rains`/0 has arity 0.
`eats`/2 has arity 2.
`eats_spinach`/1 has arity 1.

- 0-ary facts/predicates are also called atomic facts/predicates.
- The facts `friends(popeye,pluto,garfield)` and `friends(pluto,mickey)` define two different predicates, namely `friends`/3 (arity 3) and `friends`/2 (arity 2).
- The order of the arguments is significant: `father(john,paul)` is not the same as `father(paul,john)`. We determine which argument position should stand for what, but then we have to keep it:

Terms

- The basic data structure in Prolog are **terms**. They are arguments of predicates.
- Terms are either **simple** or **compound**.
- Simple terms in Prolog are **constants** and **variables**
- The constants are **symbols** and **numbers**.
- Compound terms are either **complex terms** or **lists**.

Simple Terms: Atoms, Numbers, Variables

Atoms are strings that begin with lowercase letters and consist only of letters, numbers, and the underscore, or strings that are enclosed in quotation marks:

`popeye, dog13XYZ, my_dog, "Lea?! @", 'Homer Simpson'`

Numbers are integers or floats:

`123, 89.5, 0, -323`

Variables are strings that begin with a capital letter or an underscore and consist only of letters, numbers, and the underscore:

`X, Variable, _x, _123, Hund_123, _`

Hints:

- Terms should always be 'speaking'.
- The _ variable, which consists only of the underscore, is the anonymous variable.

Variables

```
mouse(X).
eats(popeye,spinach).
has_trained(arnold).

is_strong(popeye) :- eats(popeye,spinach).
is_strong(X) :- has_trained(X).
```

- Variables can be used in facts, rules and queries.
- Same variables stand for the same values
- The clause `mouse(x).` is a universal fact (fact with an open variable).

Complex Terms

- Compound terms consist of a **functor** and any number of **arguments**.
  - The functor is always an atom.
  - The arguments are simple or complex terms.
- Examples of complex terms:

```
eats(popeye,spinach)
friends(X,father(father(popeye)))
```

- **Note**: In the second example, `father/1` is a function symbol and not a predicate symbol
- Like predicates, complex terms also have an arity (number of arguments)

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

A Logic Programme in PROLOG Syntax

```
father(peter,mary).
father(peter,john).

mother(mary,mark).
mother(jane,mary).
```

```
grandfather(X,Z) :- father(X,Y), father(Y,Z).
grandfather(X,Z) :- father(X,Y), mother(Y,Z).

grandmother(X,Z) :- mother(X,Y), father(Y,Z).
grandmother(X,Z) :- mother(X,Y), mother(Y,Z).

sibling(Y,Z) :- father(X,Y), father(X,Z).
sibling(Y,Z) :- mother(X,Y), mother(X,Z).
```

- All Clauses with the same predicate in the head are called the definition of the predicate

Inference Procedure

Reasoning in Prolog

- Prolog's principle of automatic reasoning is based on
  - the principle of **unification** and
  - **backward chaining** with backtracking.
- To prove a target clause, Prolog tries to **unify** the clause with the facts and rule heads given in the knowledge base.
- If the query contains variables, a valid variable assignment (substitution) must be found.

Inference Procedure for Logic Programming

Let *resolvent* be the query ?- $Q_1, \ldots, Q_m$

While *resolvent* is not empty **do**

1. **Choose** a query literal Q_i from *resolvent*.
2. **Choose** a renamed[1] clause H :- $B_1, \ldots, B_n$ from P such that Q_i and H unify with an most general **unifier** σ , i.e. $Q_i\sigma = H\sigma$ (Head Unification)
3. **If** no such Q_i and clause exist, then **backtrack**
4. Remove Q_i from the *resolvent*
5. **Add** $B_1, \ldots, B_n$ to the *resolvent*
6. Add σ to σ_{all}
7. Apply substitution σ to the *resolvent* and go to 1.

If *resolvent* is empty, **return** σ_{all}, else **return** *failure*.

[1] Renaming means that the variables in the clause get new unique identifiers

Queries about Facts

```
is_a_dog(pluto).
is_a_dog(snoopy).
is_a_sailor(popeye).
eats(popeye,spinach).
```

- Inference in Prolog starts with a query. The system concludes whether the statement is true.
- Requests are made to the interpreter in the console and evaluated.
- A query about facts just checks whether the literal in in the knowledge base:

```
?- is_a_dog(pluto).


?- eats(popeye, spinach).
```

Queries with Variables

```
is_a_sailor(popeye).
eats(popeye,spinach).
likes(pluto,mickey).
likes(mickey,pluto).
likes(minnie,mickey).
likes(mickey,minnie).
```

- If a query contains variables, the interpreter tries to instantiate the variable (i.e. assing a value) in such a way that the statement becomes true
- The assignment of the variables is displayed as a response
- By entering the semicolon (or clicking "Next") the interpreter looks for more answers

```
?- eats(popeye,X).
```

```
?- likes(X,Y).
```

Unification

Head Unification

- A predicate from a query must be unifiable with the head of a clause.
  - Query: `?- mortal(socrates).`
  - Clause: `mortal(X) :- human(X).`
- Head Unification
  - predicate symbols are equal
  - Substitution: `X=socrates`
- New query: `?- human(socrates).`

Source: *Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html*

Unification

- Two expressions Q and H unify if there exists a substitution σ for any variables in the expressions so that the expressions are made identical ($Q\sigma = H\sigma$)

Unification Rules

- A constant unifies only with itself
- Two structures unify if and only if
  - they have the same (function or) predicate symbol and the same number of arguments, and
  - the corresponding arguments unify recursively
- An unbound variable unifies with anything

Substitution

- A *substitution* is a finite set of the form $\sigma = \{v_1 / t_1, \ldots, v_n / t_n\}$
  - v_i 's: distinct variables.
  - t_i 's: terms with $t_i \neq v_i$.
- Applying a substitution σ to an expression E means to replace each occurence of a variables v_i with the value t_i
- Example:

$E = p(X, Y, f(a))$
$\sigma = \{X / b, Y / Z\}$
$E\sigma = p(b, Z, f(a))$

$E = \text{father(peter,X)}$
$\sigma = \{X / \text{mary}\}$
$E\sigma = \text{father(peter,mary)}$

Unification

- In unification, two terms are compared with each other or checked whether they can be equated (unified) by a suitable variable assignment.
- Unification is a part of reasoning. However, there is also the built-in predicate =, which equates two terms.

Unification Rule
Two terms are unifiable if and only if
- they are equal, or
- there is a substitution that assigns values to the variables in such a way that the two terms become equal

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Unification of Terms

- If one of the terms is a variable, then the variable can be substituted with the other term

```
?- eat spinach(popeye) = X.
        _
```

- If a variable occurs more than once in a term, the variable assignment must be compatible everywhere

```
?- X=Y, X=popeye.



?- X=popeye, X=pluto.
```

SOurce: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Unification of Complex Terms

- Complex terms match exactly when:
  1) the terms have the same functor and the same arity, **and**
  2) match all corresponding arguments match, **and**
  3) the variable assignments are compatible with each other.

```
?- food(bread,X) = food(Y,sausage).


?- meal(food(bread), drink(beer)) = meal(X,drink(Y)).


?- food(bread,X,beer) = food(Y,sausage,X)
```

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Backward Chaining

Rules and Inferences

- If the rule body is true (i.e. can be derived from the knowledge base), then the rule head is also true.
- This principle of deduction is called Modus Ponens:

$$\frac{a \rightarrow b \qquad a}{b}$$

```
b :- a.
a.
-------
b.
```

```
is_strong(popeye) :- eats(popeye,spinach).
eats(popeye,spinach).
------------------------------------------
is_strong(popeye).
```

- From the rule `is_strong(popeye) :- eats(popeye,spinach).` and the fact `eats(popeye,spinach).` the Prolog interpreter infers that `is_strong(popeye).` applies.

Rules and Queries

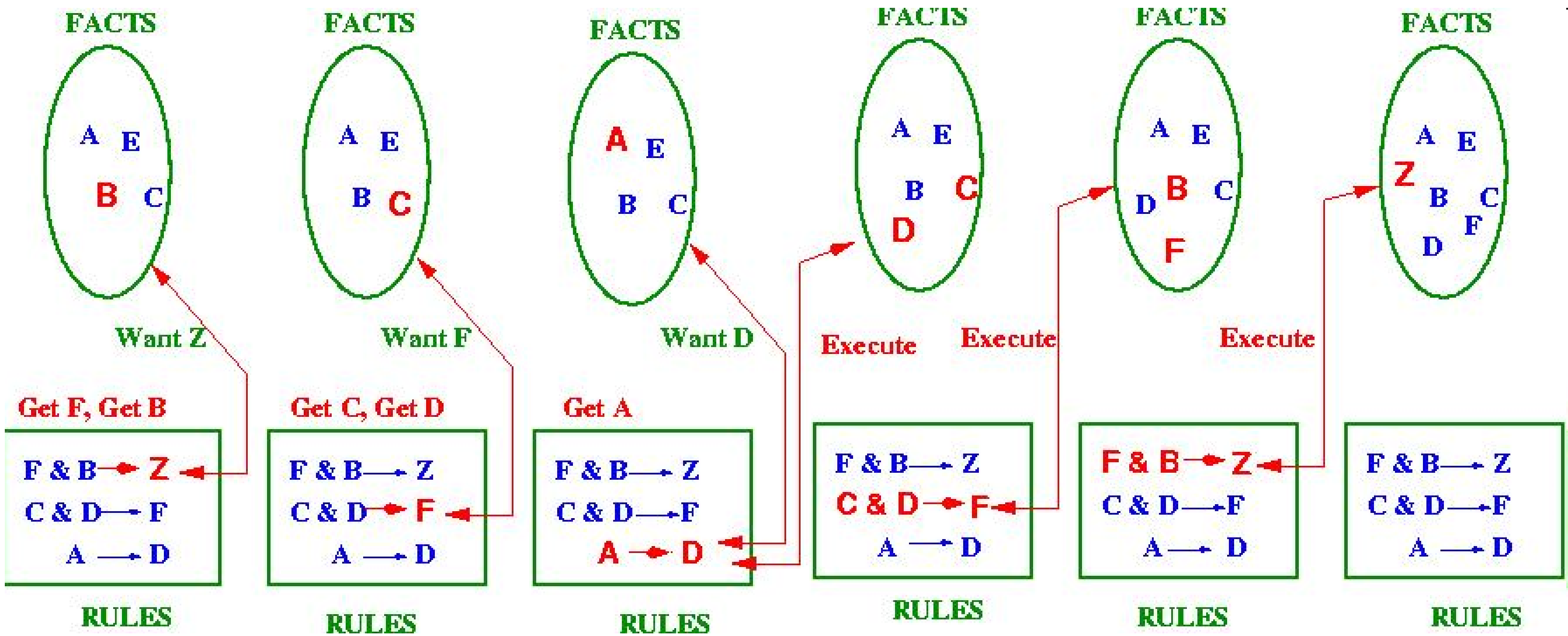
```
is_a_sailor(popeye).
eats(popeye,spinach).

is_strong(popeye) :- eats(popeye,spinach).

has_muscles(popeye) :- has_trained(popeye).
has_muscles(popeye) :- is_strong(popeye).
```

- To answer queries, the rules are applied backwards.
  - If the query matches a fact, the query is true
  - If the query matches the header of a rule, the body becomes the new query

```
is_strong(popeye) :- eats(popeye,spinach).
?- is_strong(popeye,spinach).
```

```
eats(popeye,spinach).
?- eats(popeye,spinach).
```

Illustrating Backward Chaining



Source: Kerber (2004), http://www.cs.bham.ac.uk/~mmk/Teaching/AI/l2.html

Two Choices in the Inference Procedure

There are two choices in the inference procedure where a decision needs to be made

- **Step 1: Selecting the Literal Qi** from the Resolvents
  - Solution in Logical Programming: **left-most goal**
- **Step 2: Choosing a clause**
  - Solution in logical programming: **top-most clause**
    - The clauses are selected in the order in which they appear.
    - **Backtracking**: If a selected clause does not succeed and there are alternative clauses, the next one is selected.

Two Choices in the Inference Procedure

There are two choices in Inference Procedure of Prolog:

- Step 1: Choice of a query literal Q_i from the resolvent
  - The inference procedure could select any literal without affecting the computation: If there exists a successful computation by choosing one literal, then there is a successful computation by choosing any other literal.
  - Prologs solution: **leftmost goal; (normally not backtracked)**
- Step 2: Choice of a clause:
  - This selection is non-deterministic. Depending on the selection
  - Affects computation: Choosing one clause might lead to success, while choosing some other might lead to failure.
  - Prolog ‘s solution: **topmost clause**
  - This means that the order of the clauses matters: clauses are selected in the order of appearance.
  - **Backtracking**: If a selected clause does not lead to success and there are alternative clauses then the next one is selected.

Within a Rule: From left to right

- Rule bodies are proven from left to right.
- Only when a proof of the literal i in a rule is found, ther literal i+1 can be proven
- Example Query:

```
?- female(X), sibling(X,Y).
```

- Example Rule

```
sister(X,Y):- female(X), sibling(X,Y).
```

- First `female(X)`, is proven and then `sibling(X,Y)`

Source: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Choosing a Clause: Top-Down

- Head Unification is performed top down.
- The interpreter searches the database from top to bottom to find suitable clauses for proof

```
eat_spinach(popeye).
has_trained(garfield).
is_strong(X) :- has_trained(X).
is_strong(X) :- eat_spinach(X).
```

- What is the first answer the the query:

```
?- is_strong(X).
```

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Search Tree for Depth-Firast Search

```
is_honest(pluto).
has_four_legs(pluto).
barks(pluto).

dog(X) :-
     is_mammal(X),
     barks(X).

is_mammal(X) :-
     has_four_legs(X).

?- dog(pluto).
```

```
?- dog(pluto).
      |  X = pluto
      v
?- is_mammal(pluto), barks(pluto).
      |
      v
?- has_four_legs(pluto), barks(pluto).
      |
      v
?- barks(pluto).
      |
      v
      ✓
```

Source: *Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html*

Search Tree for Depth-Firast Search

```
is_honest(pluto).
has_four_legs(pluto).
barks(pluto).

dog(X) :-
    is_mammal(X),
    barks(X).

is_mammal(X) :-
    has_four_legs(X).

?- dog(pluto).
```

```
[trace] 8 ?- trace, dog(pluto).
Call: (7) dog(pluto) ?
Call: (8) is_mammal(pluto) ?
Call: (9) has_four_legs(pluto) ?
Exit: (9) has_four_legs(pluto) ?
Exit: (8) is_mammal(pluto) ?
Call: (8) barks(pluto) ?
Exit: (8) barks(pluto) ?
Exit: (7) dog(pluto) ?
true.
```

Source: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

A Logic Program and Queries

```
father(peter,mary).
father(peter,john).

mother(mary,mark).
mother(jane,mary).
```

```
grandfather(X,Z) :- father(X,Y), father(Y,Z).
grandfather(X,Z) :- father(X,Y), mother(Y,Z).

grandmother(X,Z) :- mother(X,Y), father(Y,Z).
grandmother(X,Z) :- mother(X,Y), mother(Y,Z).

sibling(Y,Z) :- father(X,Y), father(X,Z).
sibling(Y,Z) :- mother(X,Y), mother(X,Z).
```

Queries :

```
?- father(peter,john).
?- father(peter,X).
?- grandfather(peter,mark).
?- grandfather(peter,mary).
?- grandfather(peter,S).
?- sibling(X,Y).
```

Adding Goal to Resolvent

- In step 5 of the Inference procedure the literals of the clause are added to the resolvent.
- Depending on whether the literals are added at the beginning or the end of the resolvent, we get two different strategies:
  - Adding the literals to the beginning of the resolvent gives **depth-first search**.
  - Adding the literals to the end of the resolvent gives **breadth-first search.**

Backtracking

Backtracking: Depth-First Search

- Backtracking can be triggered by two causes:
  - There is no futher clause for the current query predicate.
  - An alternative solution is to be calculated.

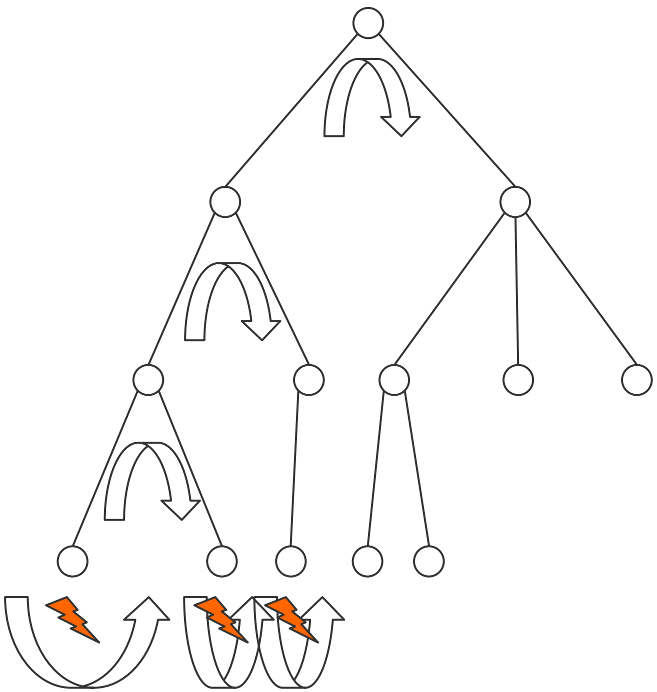
- In any case, the interpreter goes back to the last branch in the proof tree, where alternatives were still open (depth-first).

```
eat_spinach(popeye).
has_trained(garfield).
is_strong(X) :- has_trained(X).
is_strong(X) :- eat_spinach(X).
```

```
?- is_strong(popeye).
?- is_strong(X).
```

Source: *Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html*

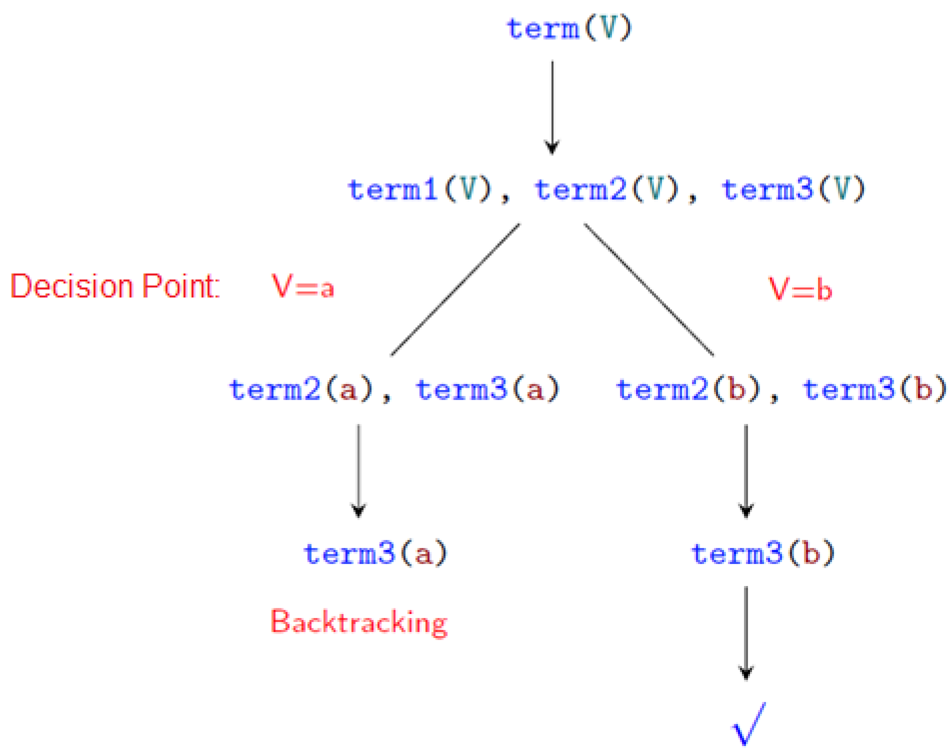
Backtracking



- Record any decision (**choose**) and its alternative
- If backtracking, then go back to the last decision and try another option
- When backtracking then roll back to the former situation (esp. for *resolvent* and σ_{all})

Search Tree: Decision Points

```
term1(a).
term1(b).

term2(a).
term2(b).

term3(b).

term(X) :-
    term1(X),
    term2(X),
    term3(X).

?- term(X).
```



Source: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Prolog ‘s Solution: Summary

- Choice of a query literal:
  → **leftmost** literal first
- Choice of a clause
  → **Topmost clause first - with backtracking**
- Adding new goal to the resolvent
  → **At the beginning**.

Unifier

- A substitution σ is a *unifier* of expressions E and F iff

$$E\sigma = F\sigma$$

- Example: Let E and F be two expressions:
  - $E = f(x, b, g(z))$,
  - $F = f(f(y), y, g(u))$.

  Then $\sigma = \{x / f(b), y / b, z / u\}$ is a unifier of E and F:
  - $E\sigma = f(f(b), b, g(u))$,
  - $F\sigma = f(f(b), b, g(u)$

- A unifier σ of E and F is *most general* iff is more general than any other unifier of E and F, i.e. for any other unifier ρ there exists a unifier τ such that $\rho = \tau \circ \sigma$

Recursion

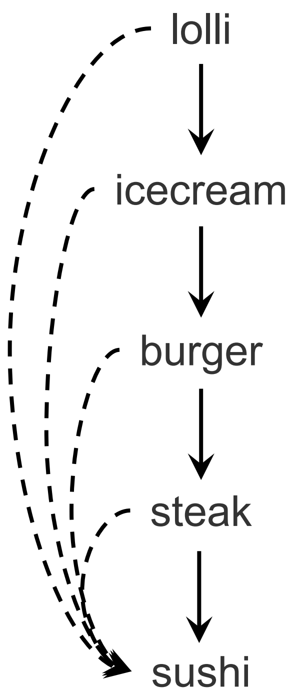
Recursion

```
likes(pluto,mickey).
likes(mickey,pluto).
likes(minnie,mickey).
likes(mickey,minnie).
```

```
likes(pluto,mickey).
likes(minnie,mickey).
likes(X,Y) :- likes(Y,X).
```

- In the knowledge base we see that pluto likes mickey and als mickey likes pluto. The same for minnie and mickey.
- Assume that likes is a inverse predicate. How can we avoid to write all facts. Assume want to write only one likes fact for a couple and get the inverse by inference.

Task



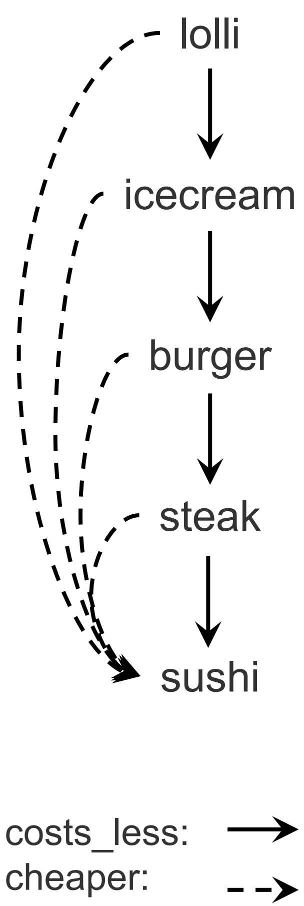
costs_less: →
cheaper: - ->

- Given these facts:

```
costs_less(lolli,icecream).
costs_less(icecream,burger).
costs_less(burger,steak).
costs_less(steak,sushi).
```

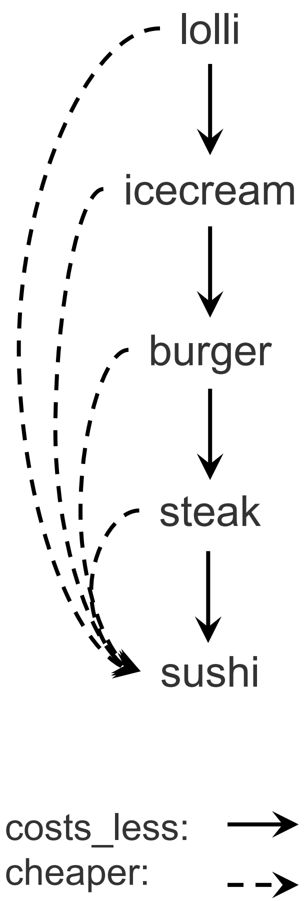
Write rules for cheaper/2. such that cheaper(X,Y) is true, if X costs less than Y.

Nicht-rekursive Definition




```
costs_less(lolli,icecream).
costs_less(icecream,burger).
costs_less(burger,steak).
costs_less(steak,sushi).
```

Solution: Recursive Predicate




```
costs_less(lolli,icecream).
costs_less(icecream,burger).
costs_less(burger,steak).
costs_less(steak,sushi).
```

Recursive Predicates

- A predicate is defined recursively when the predicate in the rule head is called in one of the defining rules.
- The basic idea is to reduce a common task to a simpler task of the same class (loops).
- Recursion makes it possible to write compact predicate definitions and avoid redundancy.

Declarative und procedural Interpretation of a Knowledge Base

Declarative Interpretation

- Declarative meaning is the meaning that is 'meant' or 'expressed' when reading the knowledge base as a set of logical statements.
- The declarative meaning of a prolog program can be defined as the set of all statements that can be logically derived from the knowledge base

Procedural Interpretation

- Procedural interpretation is the meaning that comes from what Prolog 'does' with a knowledge base.
- The procedural meaning of a Prolog program can be defined as the set of all queries (statements) for which the Prolog interpreter finds a variable assignment that results in the output `true`

Source: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Procedural Interpretation of Recursive Predicates

```
cheaper(X,Y):- costs_less(X,Y).
cheaper(X,Y):- costs_less(X,Z),
               cheaper(Z,Y).
```

- First rule: To prove that X is cheaper than Y, it is enough to prove that X costs less than Y.
- Second rule: To prove that X is cheaper than Y, this problem can be broken down into two sub-problems. We are looking for a Z so that X costs less than Z (subproblem 1) and that Z is cheaper than Y (subproblem 2).

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Definieren harmloser rekursiver Prädikate

- Recursive predicates always require at least two clauses:
  - a recursive clause
  - an anchor or exit clause.
- The anchor clause should always precede the recursive clause (otherwise there is a risk of an infinite loop).
- In the rule body of the recursive clause, it often makes sense to put the recursive call at the end.

anchor clause

```
cheaper(X,Y):-
        costs_less(X,Y).
cheaper(X,Y):-
        costs_less(X,Z),
        cheaper(Z,Y).
```

Objectives: Recursive predicates,
- that do not lead to endless loops,
- who schedule as early as possible,
- which can be invoked with open variables.

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog

Prozedurale und deklarative Bedeutung

- As a reminder, Prolog works its way
  - through the knowledge base from top to bottom,
  - within the clauses from left to right.

- How does the order affect the procedural behavior of the predicate?

```
parent(john,peter).
parent(mary,john).
parent(susan,mary).
```

```
ancestor1(X,Y):- parent(X,Y).
ancestor1(X,Z):- parent(X,Y),
                 ancestor1(Y,Z).
```

```
ancestor3(X,Y):- parent(X,Y).
ancestor3(X,Z):- ancestor3(Y,Z),
                 parent(X,Y).
```

```
ancestor2(X,Z):- parent(X,Y),
                 ancestor2(Y,Z).
ancestor2(X,Y):- parent(X,Y).
```

```
ancestor4(X,Z):- ancestor4(Y,Z),
                 parent(X,Y).
ancestor4(X,Y):- parent(X,Y).
```

Quelle: Wiebke Petersen, Grundkurs Prolog, HHU Düsseldorf, https://user.phil.hhu.de/~petersen/WiSe2324_Prolog/WiSe2324_Prolog.html

Multiple Answers to a Query

- The inference procedure of Prolog computes one solution.
- The user can force the system to compute the next solution by typing a „;“ (typing „;“ is interpreted by the system as a fail and thus backtracking is started to compute an alternative solution)
- Example:

```
father(peter,mary).
father(peter,john).
father(peter,paul).
```

```
sibling(Y,Z) :- father(X,Y), father(X,Z).
sibling(Y,Z) :- mother(X,Y), mother(X,Z).
```

```
?- sibling(X,Y).
X=mary, Y=mary;
X=mary, Y=john;
X=mary, Y=paul;
X=john, Y=mary
```

Negation and Cut

Negation as Failure

- Prolog allows a form of negation that is called negation as failure
- A negated query

  not Q

  is considered proved if the system fails to prove Q
- Thus, the clause

  alive(X) :- not dead(X)

  can be read as „Everyone is alive if not provably dead“

Declarative Reading vs Procedural Reading

- Logic Program: Finite set of clauses.
  - H :- B_1, …, B_n $\quad n \geq 0$
  - Example:
    - mortal(X) :- human(X).
- **Declarative reading:**
  - *H* is implied by the conjunction of the B_i's.
  - Example: If someone is human then he/she is mortal.
- **Procedural reading (backward chaining):**
  - To answer the query ?-*H*, answer the conjunctive query ?-B_1, …, B_n
  - Example: To prove that someone is mortal, prove whether he/she is a human
- All clauses with the same head predicate are
  - A definition (in declarative reading)
  - A procedure (in procedural reading)

The Cut Operator

- Under procedural reading, a logic program consists of a set of procedures
- Each procedure consists of a sequence of alternatives
- The inference procedure of Prolog computes all possible alternatives for a query
- The cut operator (written as „!“) prevents backtracking. It is a special literal that is always true but that stops all other alternatives from being applied.

```
sibling(Y,Z) :- father(X,Y), !, father(X,Z).
sibling(Y,Z) :- mother(X,Y), mother(X,Z).
```

Application of the Cut

- The query ?-risk(X) gives X=high

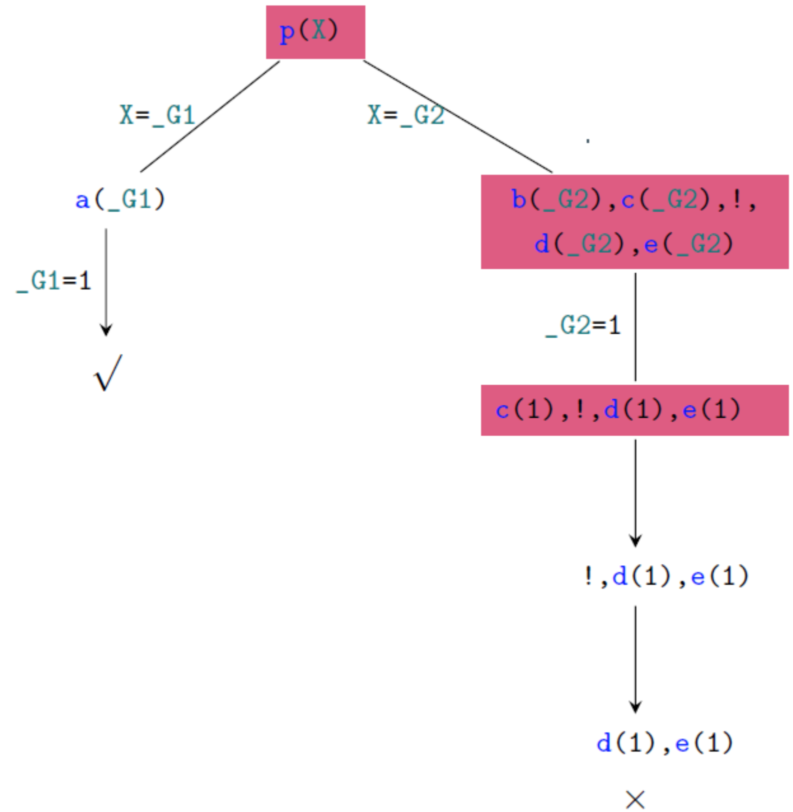
```
return(4).

risk(high) :-  return(X), X < 5.
risk(low).
```

- When asking for another solution is also give X=low, which is wrong.
- How this be avoided?

Example with Cut

```
p(X) :- a(X).

p(X) :- b(X),
        c(X),!,
        d(X),
        e(X).

p(X) :- f(X).


a(1).
b(1).
b(2).
c(1).
c(2).
d(2).
e(2).
f(3).
```

```
?-p(X).
X=1;
false.
```



Using the Cut

- Reasons to use the Cut
  - Efficiency: Cropping the search space
  - Shorter Programs
  - Enforcing determinism.
  - Modeling of defaults.

Use the cut only if there is an obvious advantage.

- Downsides of the cut
  - The cut destroys the declarativity of prologue programs.
  - The interpretation of a predicate definition with cuts is usually only possible if the order of the proof steps is taken into account.

Defining Negation as Failure with the Cut Operator

- The cut operator can be used to define negation as failure

```
not(Q) :- Q, !, fail.
not(Q).
```

- If ?- Q can be proved then the query not(Q) fails.
- If Q cannot be proved, the second clause is applied which always succeeds.
- If Q can be proved the second clause must not be applied. This is assured by the cut: If Q can be proved, then the cut prevents backtracking.

Arithmetics

Built-in Arithmetic: The Operator is/2

- In Prolog there is a set of built-in functions for arithmetics. To apply these function there exists a special predicate „**`is`**“:

  `X is Y` is true when X is equal to the value of Y.

- Built-in functions include: +, –, *, /, //, mod, (// performs integer division)
  - Using these functions we can compute a value for terms involving numbers.
- Example:
  - **`?- X is 7+1`**.

  Will give the answer **`X = 8`**
- The **`is`** Predicate works as follows:
  - First evaluate the right-hand argument (after the „is“)
  - The result is is unified with the left-hand argument.
  - The values of all the variables on the right-hand side of is must be known for evaluation to succeed.

How to use is/2

- The operator forces the second argument to be evaluated immediately. Therefore, the second argument must be an evaluable arithmetic expression
- If the second argument cannot be evaluated, Prolog aborts with an error message

```
?- 3+5 is X.

?- X is 4+Y.

?- X is a.
```

Comparison

The comparison operators < (smaller), =<, (less than or equal to), > (greater), >= (greater than or equal to), =:= (equal), and =\= (unequal) force the immediate evaluation of both arguments

```
?- 1+4 < 3*5.


?- 1+7 =< 3*2.


?- 1+3 =:= 2*2.


?- 1+3 =\= 2*3.


?- X < 3.
```

Comparison

Equality:

Pred	Description	Variable Substitution	Arithmetic Computation
=	unifiable	yes	no
is	is value of	first	second
=:=	same value	no	yes
==	identical	no	no

Other Comparisons:

X>Y	The value of X is greater than the value of Y
X>=Y	The value of X is greater than or equal to the value of Y
X<Y	The value of X is less than the value of Y
X=<Y	The value of X is less than or equal to the value of Y
X=\=Y	The values of X and Y are unequal