Rule-Based Systems: Logic Programming

Remember: 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) → 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 “makes sense” together (are true)
- Examples:
  - person(peter)
  - married(peter, cindy)
  - Appointment(1.3.2016, holger, “AB1”, “Lecture KE”)
  - not female(holger)
- Literals are predicates and negated predicates

Variables, Constants, and Data

- Data are classical data which you know from programming languages
  - 31
  - “Lecture KE”
  - 21.3.2015
- Constants are values
  - peter
  - cindy
- Differences between constants and data
  - data: rich operations (+, -, ...) and comparisons (=, <, >, <=, …)
  - constants: only identity (=), but very quick!
- Variables are placeholders for constants or data
  - likes(holger, X)

Exercises (1/2)

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

Exercises (2/2)

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

PROLOG

- PROLOG (= PROgramming in LOGic) is a programming language based on Horn clauses
- Syntax:
  - Prolog uses „:-“ instead of „←“
  - Literals in the body are separated by comma „ ,“
    (the comma is equivalent to the logical AND or „∧“)
  - Each clause ends with a period „.“
  - Variables are either
    - strings starting with capital letter: X, Person
    - strings starting with a underline: _x, _person

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

Reasoning in Logic Programming

- INPUT:
  - A logic programme *P and*
  - a query Q (?- Q_1, Q_2, …, Q_m)
- INFERENCE: Backward Chaining
- OUTPUT:
  - If the query Q does *not* contain variables the answer is
    - yes if Q can be deduced from P
    - no, if Q cannot be deduced from P
  - If the query Q does contain variables the answer is
    - A substition σ for the variables in Q such $Q\sigma$ can be deduced from *P*
    - no, if there is no substitution σ such that $Q\sigma$ can be deduced from Q

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).
```

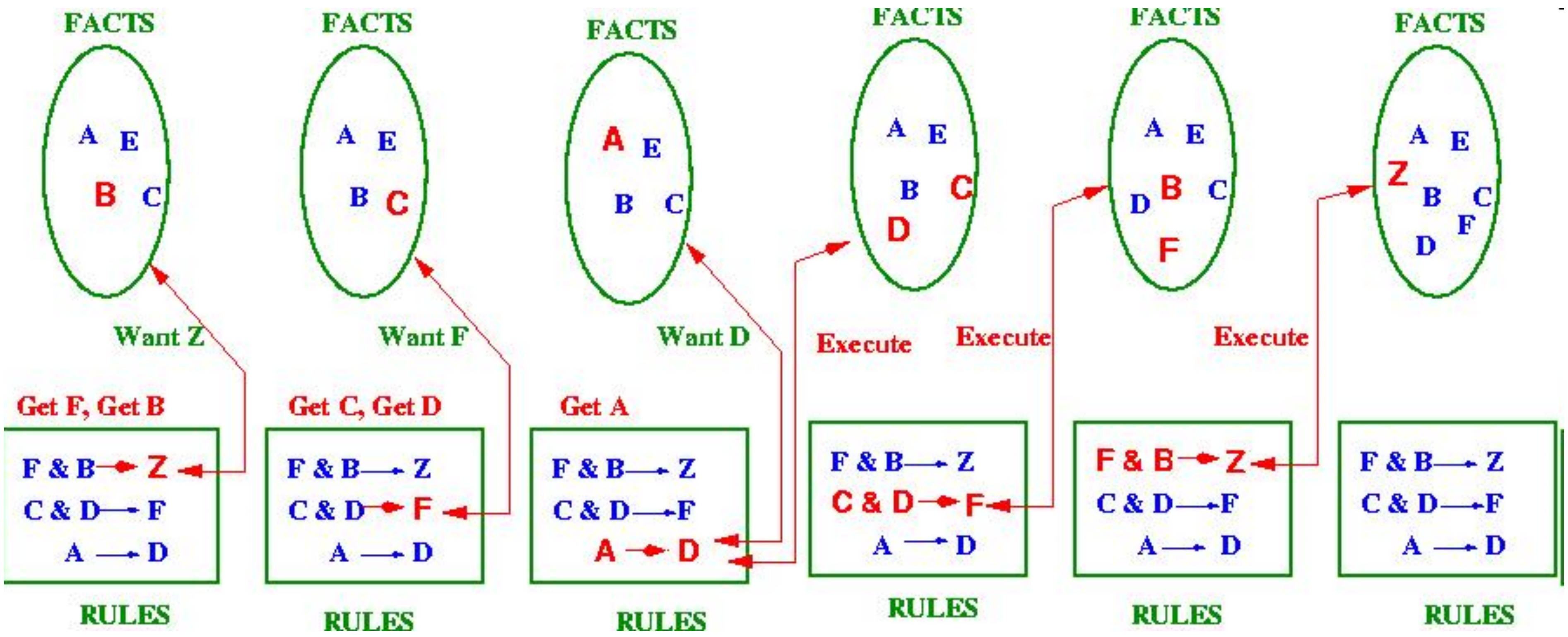
Substitution

- A *substitution* is a finite set of the form $\sigma = \{v_1 / t_1, . . . , v_n / t_n\}$
  - 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 = father(peter,X)
σ = {X / mary}
Eσ = father(peter,mary)

Illustrating Backward Chaining



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

Inference Procedure for Logic Programming

Let *resolvent* be the query ?- Q_1, …, 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. $Q_i\sigma = H\sigma$
3. **If** no such Q_i and clause exist, then **exit**
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.
8. If *resolvent* is empty, **return** σ_{all}, else **return** *failure*.

---

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

Corrected Inference Procedure for Logic Programming

Let *resolvent* be the query ?- Q_1, …, Q_m

if *resolvent* is not empty **then**

1. **Choose** a query literal Q_i from *resolvent*.
2. **Choose** a renamed[1] clause *H :- B_1, …, B_n* from *P* such that Q_i and *H* unify with an most general **unifier** σ , i.e. $Q_i\sigma = H\sigma$
3. **If** no (more) such clause exist, then **return failure**
4. Remove Q_i from the resolvent
5. **Add** *B_1, …, B_n* to the resolvent
6. Add σ to σ_{all}
7. Apply substitution σ to the *resolvent* and call procedure recursively.
8. If *recursive call return sucessfully*, **return success with** σ_{all}, else backtrack, i.e. choose other alternatives in step 1 and 2.

else

return success with σ_{all}

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

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**
- 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.

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.**

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**.

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

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 $\rho = \tau \circ \sigma$

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 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 $n \geq 0$
  - 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 procedure
- 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).
```

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.

Built-in Arithmetic

- 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.

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