1

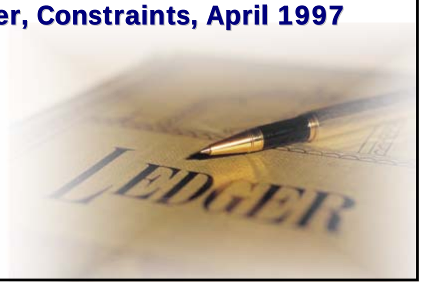
Programming with Logic and Constraints

Roman Barták
Charles University, Prague (CZ)

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

„Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.“

Eugene C. Freuder, Constraints, April 1997



Holly Grail of Programming

```
> Computer, solve the SEND, MORE, MONEY problem!

> Here you are. The solution is
  [9,5,6,7]+[1,0,8,5]=[1,0,6,5,2]
```

a Star Trek view

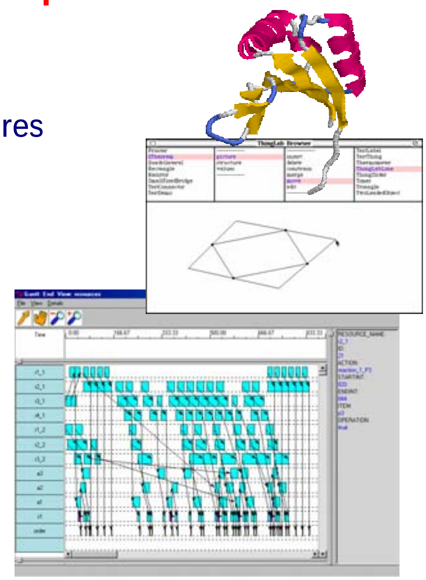
```
> Sol=[S,E,N,D,M,O,R,Y],
  domain([E,N,D,O,R,Y],0,9), domain([S,M],1,9),
            1000*S + 100*E + 10*N + D +
            1000*M + 100*O + 10*R + E #=
  10000*M + 1000*O + 100*N + 10*E + Y,
  all_different(Sol),
  labeling([ff],Sol).

> Sol = [9,5,6,7,1,0,8,2]
```

today reality

Application areas

All types of hard combinatorial problems:

- **molecular biology**
  - DNA sequencing
  - determining protein structures
- **interactive graphic**
  - web layout
- **network configuration**
- **assignment problems**
  - personal assignment
  - stand allocation
- **timetabling**
- **scheduling**
- **planning**



A bit of history

- **Procedural Interpretation of Horn Clauses** (Kowalski)
  - axiom „A if B" can be read as a procedure
    A is a procedure head
    B is a procedure body
- **Prolog** (Colmerauer)
  - Programation et Logique or Programming in Logic
  - specialised theorem prover for natural language processing
- **From unification to constraints** (Gallaire 1985, Jaffar, Lassez 1987)
  - unification is constraint solving over Herbrand universe

Course outline

- **Programming with logic** — Monday
  - **foundations of Prolog**
  - **facts, rules, and queries**
- **Extensions to pure Prolog** — Tuesday
  - **lists and arithmetic**
  - **cut, negation, and blackboard**
- **From unification to constraints** — Wednesday
  - **consistency techniques**
  - **programming filtering algorithms**
- **Programming depth-first search** — Thursday
  - **incomplete search techniques**
  - **branch and bound**
- **Modeling with constraints** — Friday
  - **modeling examples**

ESSLLI 2005 – Programming with Logic and Constraints

Basic concept

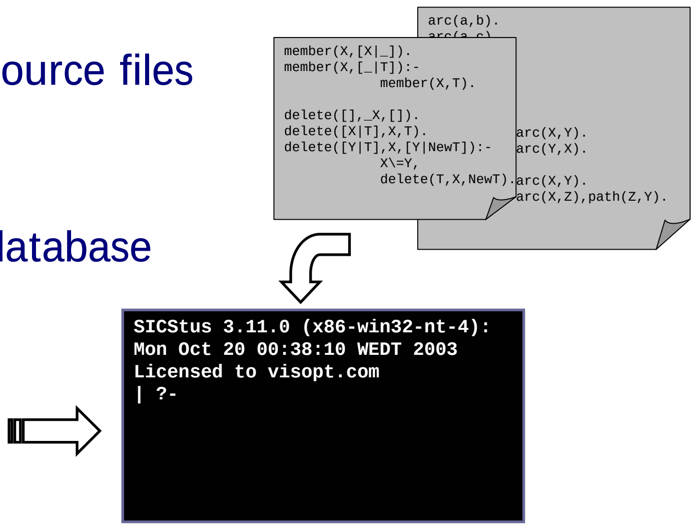
Prolog is a deductive system that finds answers to **queries** using a knowledge base consisting of **facts** and **rules**.

Where is the programming?

- writing the database of facts and rules
- Prolog interpreter deduces the answer automatically
- ↳ **declarative programming**

Prolog architecture

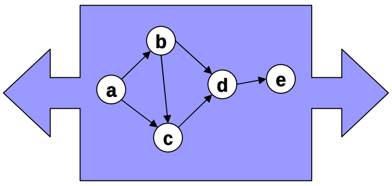
- Prolog source files
  - *.pl
- Prolog database
- Queries



Facts

Prolog facts describe basic information about the problem.

```
node(a).
node(b).
node(c).
node(d).
node(e).
```




```
arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

name

argument

more arguments separated by commas

Simple queries

It is possible to ask **queries** about the facts stored in the knowledge base:

Prolog prompt

query

answer

```
?-node(a).
yes
?-node(bla).
no
?-arc(a,c).
yes
?-arc(a,d).
no
?-path(a,d).
no
```

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

ESSLLI 2005 – Programming with Logic and Constraints

Open queries

The query may contain **variables** whose values will be found using stored facts:

```
?-node(X).
X=a ;
X=b ;
X=c ;
X=d ;
X=e ;
no
```

a request for an alternative answer

no more answers

```
?-arc(a,X).
X=b ;
X=c ;
no
```

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Compound queries

- List of facts is nothing more than a simple database.
- Is it possible to generate an answer that is not stored directly as a fact but that can be combined from several facts?

Yes. It is possible to **query over a combination of facts** from the knowledge base:

```
?-arc(a,Y),arc(Y,Z).
Y=b
Z=c ;
Y=b
Z=d ;
Y=c
Z=d ;
no
```

variables can be shared between simple open queries

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

Syntax break
Atoms vs. variables

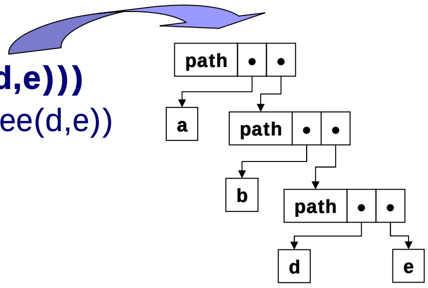
Data (and programs) are expressed using terms

- **Atoms**
  - words consisting of letters, numbers and underscores that start with a non-capital letter
    - `a, arc, john_123, …`
  - words enclosed in single quotas
    - `´Edinburgh´, …`
- **Variables**
  - words consisting of letters, numbers and underscores that start with a capital letter or underscore
    - `X, Node, _noname, …`
  - _ is an anonymous variable
    - two occurences of _ are assumed to be different variables
    - contents is not reported to the user

Syntax break
Compound terms

Compound terms express **structured information**

- atoms and variables are terms
- **functor(arg1,...,argn)** is a (compound) term, where functor is an atom and arg1, ..., argn are terms
  - arc(a,c)
  - **path(a,path(b,path(d,e)))**
  - tree(tree(a,tree(b,c)),tree(d,e))
  - arc(a,X)
  - ...



Deductive rules

- We can give a name to the query so it can be used repeatedly

  doubleArc(X,Z):-arc(X,Y),arc(Y,Z).

  - This is called a **rule**.
- After defining the rule, we can query it like the facts:

```
?-doubleArc(b,W).
W=d ;
W=e ;
no
?-doubleArc(a,W).
W=c ;
W=d ;
W=d ;
no
```

only variables from the rule head are returned to user

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


How does it work?

Deductive rules

?-doubleArc(b,W).

- find a rule whose head matches the goal and substitute variables accordingly.

  doubleArc(b,W):-arc(b,Y),arc(Y,W).
- substitute query by the body of the rule

?-arc(b,Y),arc(Y,W).

- find a matching fact (**arc(b,c)**), substitute variables, and remove the fact from the query

?-arc(c,W).

- do the same with the rest (**arc(c,d)**)

W=d ;

- Try alternative facts (**arc(b,d),arc(d,e)**)

W=e ;

no

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

Alternative rules

- It is possible to define **alternative rules** (disjunction)

```
edge(X,Y):-arc(X,Y).
edge(X,Y):-arc(Y,X).

?-edge(W,b).
W=a ;
W=c ;
W=d ;
no
```

deduced using the first rule (W=a)

deduced using the second rule (W=c, W=d)

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


How does it work?

Alternative rules

Just like before, but more alternative rules matches the query.

?-edge(W,b).

- find a rule whose head matches the goal, substitute variables accordingly, and substitute query by the body of the rule
  edge(W,b):-arc(W,b).

?-arc(W,b).

- find all solutions to a query using facts

W=a ;

- try an alternative rule for the original query
  edge(W,b):-arc(b,W).

?-arc(b,W).

- find all solutions to a query using facts

W=c ;
W=d ;
no

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

Recursive rules

- It is possible to use the rule head in its body, i.e., to use **recursion**

```
path(X,Y):-arc(X,Y).
path(X,Y):-arc(X,Z),path(Z,Y).
```

```
?-path(c,W).
W=d ;
W=e ;
no
```

W=d: deduced using the first rule and arc(c,d)

W=e: deduced using the second rule through d

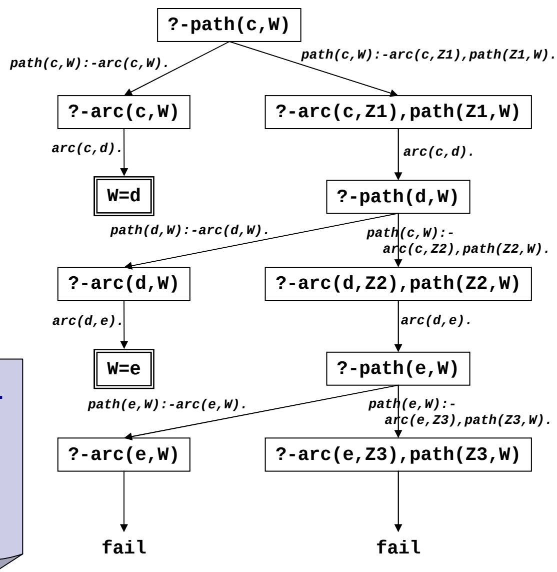
```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


How does it work?

Recursive rules

- Just like before, but the rule may be used several times.
- This is OK because each time a rule is used, its **copy with „fresh“ variables** is generated (like calling a procedure with local variables).

```
path(X,Y):-arc(X,Y).
path(X,Y):-arc(X,Z),path(Z,Y).

node(a).      arc(a,b).
node(b).      arc(a,c).
node(c).      arc(b,c).
node(d).      arc(b,d).
node(e).      arc(c,d).
              arc(d,e).
```



Prolog at glance

Prolog „program" consists of **rules** and **facts**.
Each **rule** has the structure **Head:-Body.**

- **Head** is a (compound) term
- **Body** is a query (a conjunction of terms)
  - typically Body contains all variables from Head
- rule semantics: **if Body is true then Head can be deduced**

Fact can be seen as a rule with an empty (true) body.

Query is a conjunction of terms: Q = **Q1,Q2,…,Qn.**

- **Find a rule** whose head matches goal Q1.
  - If there are more rules then introduce a choice point and use the first rule.
  - If no rule exists then backtrack to the last choice point and use an alternative rule there.
- **Use the rule body** to substitute Q1.
  - For facts (Body=true), the goal Q1 disappears.
- **Repeat until empty query** is obtained.

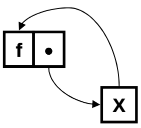
Prolog technology

Prolog = Unification + Backtracking

- **Unification** (matching)
  - to select an appropriate rule
  - to compose an answer substitution
  - How?
    - make the terms syntactically identical by applying a substitution
- **Backtracking** (depth-first search)
  - to explore alternatives
  - How?
    - resolve the first goal (from left) in a query
    - apply the first applicable rule (from top)

Unification

- a basic mechanism for **information passing**
- syntactic equality of terms via substitution of terms to variables
- **?-X=f(a).** **-> X/f(a)**
- **?-f(X,a)=f(g(b),Y).** **-> X/g(b), Y/a**
- **?-f(X,b,g(a))=f(a,Y,g(X)).** **-> X/a, Y/b**
- **?-X=f(X).** **-> infinite term**
  - **occurs check** can forbid such structures
  - but cyclic structures might be very useful for modeling pointer structures



Selecting rules

- **Unification is used for rule selection.**

?-path(f(a),G).

- rule: path(X,Y):-arc(X,Y).
- **do unification**: **X=f(a),Y=G**

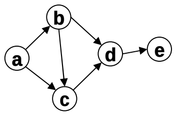
?-arc(f(a),G).

- rule (fact): arc(a,b).
- **do unification**: f(a)=a, G=b **-> fail**
- rule (fact): arc(a,c).
- **do unification**: f(a)=a, G=c **-> fail**
- ...

Computing results

- **Unification is used for answer composition.**

```
path(X,Y,path(X,Y)):-
  arc(X,Y).
path(X,Y,path(X,PathZY)):-
  arc(X,Z),
  path(Z,Y,PathZY).

?-path(a,d,P).
P=path(a,path(b,d));
P=path(a,path(b,path(c,d))) ;
P=path(a,path(c,d)) ;
no
```




```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Information passing

- How to obtain the result?
- **Accumulator**
  - Accumulate partial results in a parameter of the procedure.
  - Requires additional parameter with initialization.
- **Composition of substitutions**
  - Compute the result from partial results to be computed later.
  - Specific to Prolog and substitutions.

Accumulator

Symbolic addition of unary represented numbers
(`0, s(0), s(s(0)), …`).

Result is **accumulated** in a parameter of the procedure.

```
plus(0,X,X).
plus(s(X),Y,Z):-plus(X,s(Y),Z).
```

accumulator

```
?-plus(s(s(s(0))),       s(0)    ,Sum).
?-plus(  s(s(0)) ,     s(s(0))   ,Sum).
?-plus(    s(0)   ,   s(s(s(0))) ,Sum).
?-plus(      0    ,s(s(s(s(0)))),Sum).
```


Composition

Symbolic addition of unary represented numbers.

Result is a **composition of substitutions** that will be computed later.

```
plus2(0,X,X).
plus2(s(X),Y,s(Z)):-plus2(X,Y,Z).
```

argument for composing the result

```
?-plus2(s(s(s(0))),s(0),S1). %S1=s(S2)
?-plus2(  s(s(0)) ,s(0),S2). %S2=s(S3)
?-plus2(    s(0)  ,s(0),S3). %S3=s(S4)
?-plus2(      0   ,s(0),S4). %S4=s(0)
```

ESSLLI 2005 – Programming with Logic and Constraints

Homework

- Propose a simple **genealogy database**:
  - facts
    - **`man, woman, parent, …`**
  - rules
    - **`father, mother, son, daughter, grandparent, uncle, aunt, siblings, descendant, …`**

- For example **solution** look at
  `http://kti.mff.cuni.cz/bartak/prolog/genealogy.html`