

# Semantic Web Rules & Prolog



**A.I. Tools Seminar WS 06/07**

**Pascal Recktenwald**

# Übersicht

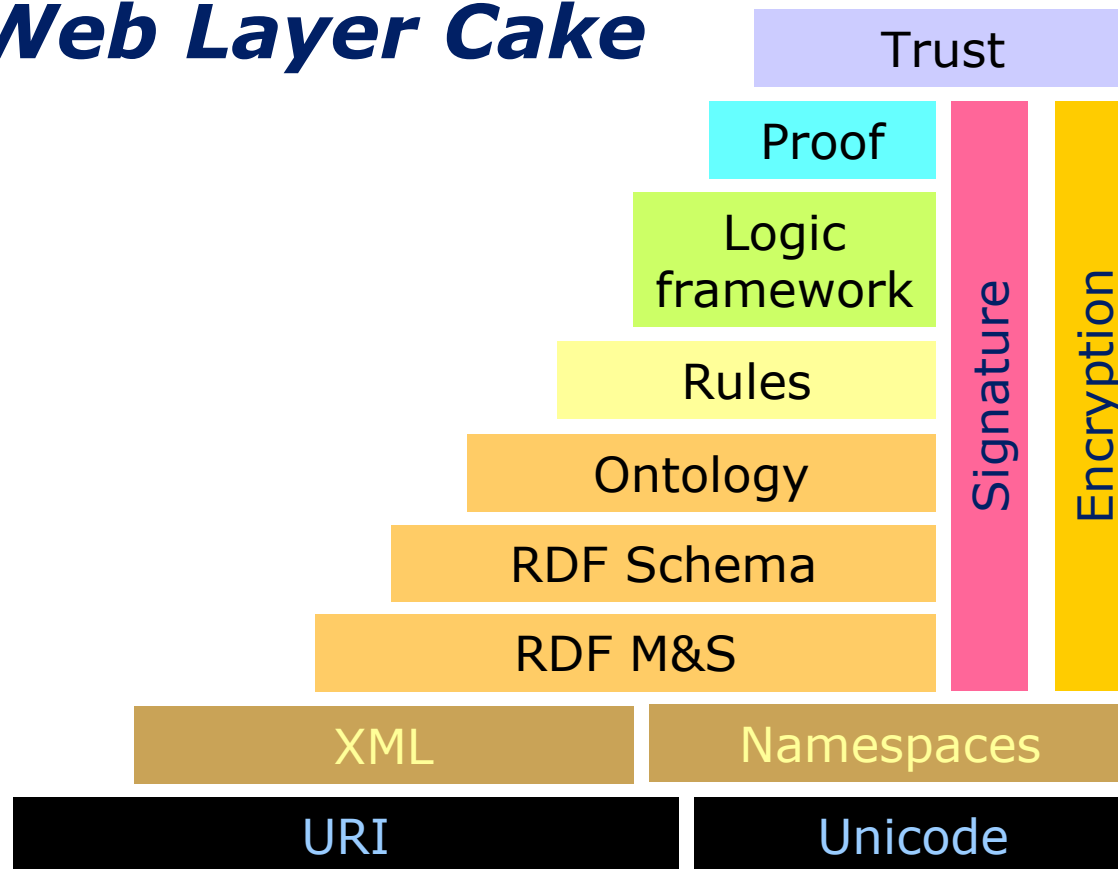


- Motivation
- Rules
- Prolog
  - Geschichte
  - Hornklauseln
  - Datenobjekte, Fakten, Anfragen, Regeln
  - Rekursion, Matching, Backtracking
  - Vordefinierte Prädikate
  - Einfluss und Anwendung
  - Demo
- Fazit

# Motivation



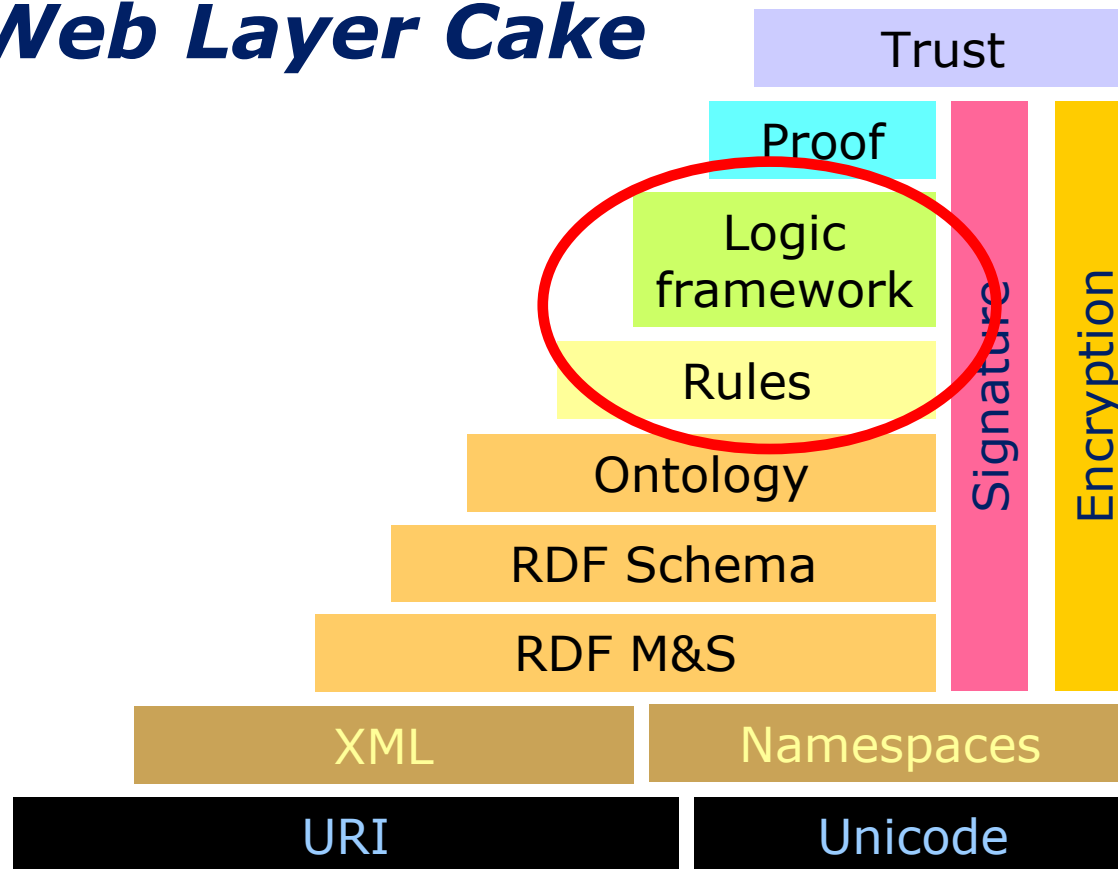
## ■ *Semantic Web Layer Cake*



# Motivation



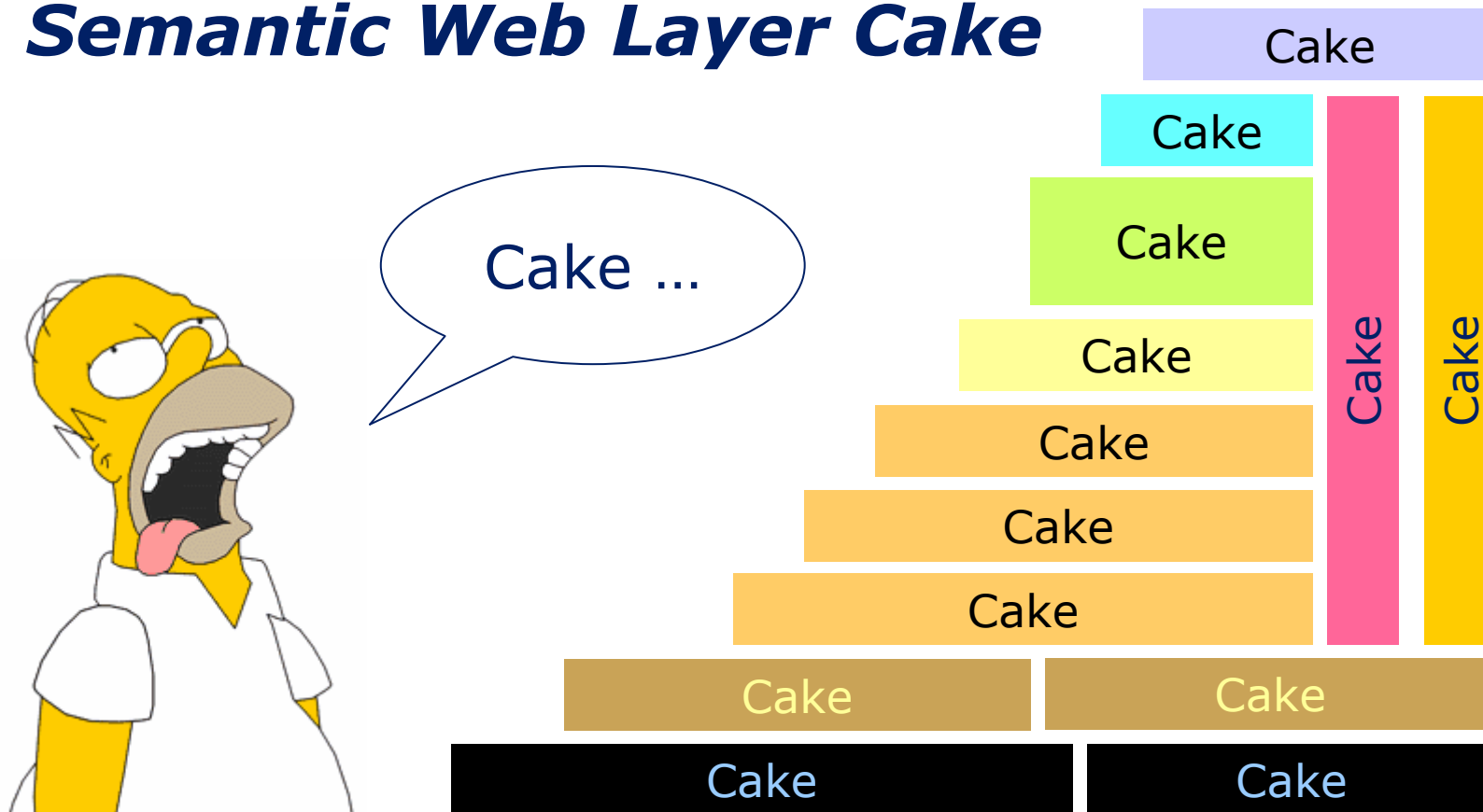
## ■ *Semantic Web Layer Cake*



# Motivation



## ■ *Semantic Web Layer Cake*



## ■ ***Ontologien***

- Implizites formulieren von Regeln
- Subsumption:  
„Frau“ ist Unterkonzept des Konzeptes „Person“

## ■ ***Regelsprachen***

- Ermöglichen neue, komplexere Konstrukte
- Regeln werden explizit deklariert und nicht implizit modelliert
- Größere Ausdrucksstärke der Ontologie
- Verschlechterung der Berechenbarkeit bzw. Entscheidbarkeit



## ■ **Beispiel**

$\forall X, Y, Z : \text{hasParent}(X, Y) \wedge \text{hasBrother}(Y, Z) \Rightarrow \text{hasUncle}(X, Z)$

## ■ **Aktuell**

Fehlender W3C Standard für eine  
Semantic Web Regelsprache

## ■ **Vorschlag**

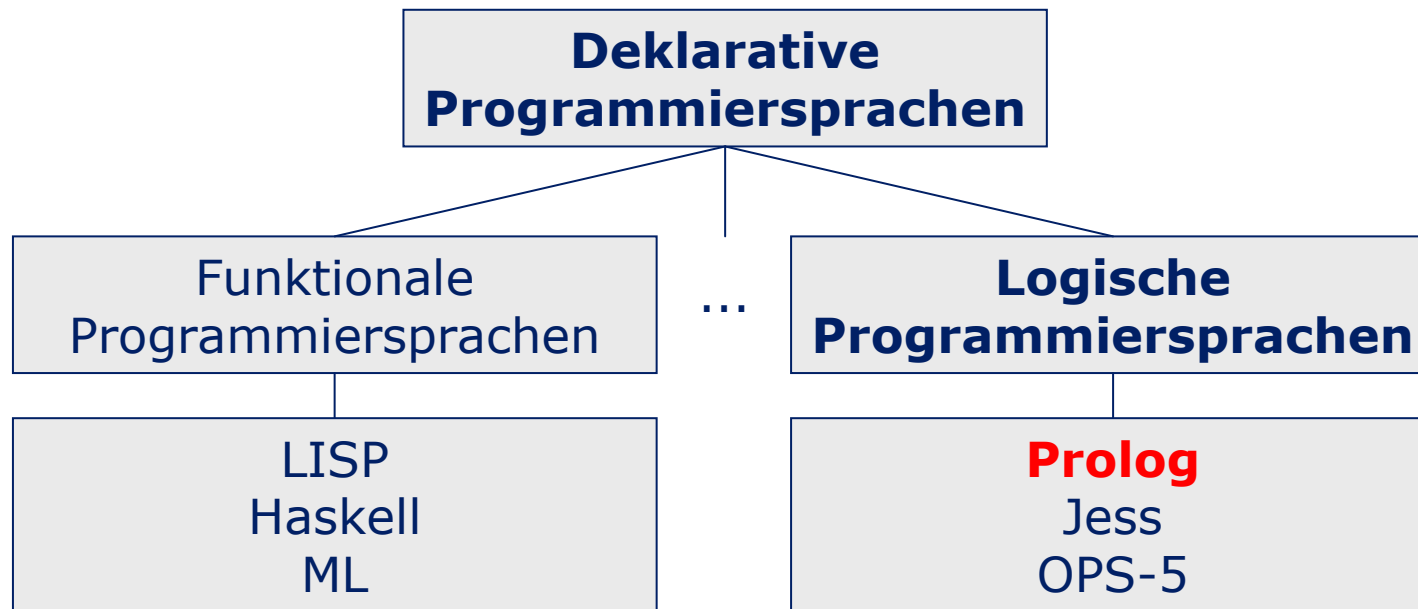
SWRL (Semantic Web Rule Language)

- OWL Lite/DL (Web Ontologie Language)
- Datalog RuleML (Rule Markup Language)
- Syntax: XML oder RDF

# Prolog Geschichte



- Prolog = „**P**rogramming in **l**ogic“
- Syntax: First Order Predicate Logic



# Prolog Geschichte



- **1972** Erster Prolog-Interpreter
  - Robert Kowalski, Edinburgh (Theorie)
  - Maarten van Emden, Edinburgh (Exp. Vorführung)
  - Alain Colmerauer, Marseilles (Implementierung)
- **1977** Erster Prolog-Compiler
  - David D.H. Warren: DEC-10 (Edinburgh) Standard
- **1983-1993** „*The Wonder Years of Sequential Prolog Implementation*“ [Peter Van Roy; 1993]
  - Verbreitung verschiedenster Prolog-Systeme
  - Von den Universitäten in die Industrie
- **1995, 2000** ISO-Prolog-Standard Teil I + II
- **Heute** SICStus-, QUINTUS-, LPA-, **SWI-Prolog**

# Prolog

## Hornklauseln



- **Prolog-Programm**  
Sammlung von Hornklauseln

$P$	: Konklusion
$\neg P_i$	: Prämisse, $i=1, \dots, n$
$\leftarrow$	: log. Implikation

- **Hornklauseln**  
Klauseln mit max. einem positiven Literal

- |                    |  |   |
|--------------------|--|---|
| □ <b>Regeln:</b>   | $\{P, \neg P_1, \neg P_2, \dots, \neg P_n\}$ | $P \leftarrow P_1 \wedge P_2 \wedge \dots \wedge P_n$ |
| □ <b>Fakten:</b>   | $\{P\}$                                      | $P$   |
| □ <b>Anfragen:</b> | $\{\neg P_1, \neg P_2, \dots, \neg P_n\}$    | $\leftarrow P_1 \wedge P_2 \wedge \dots \wedge P_n$   |

- **Hornklausel-Wissensbasis**
  - Fakten  $\cup$  Regeln  $\leftarrow$  Anfragen
  - Closed World Assumption (CWA)

# Prolog Hornklauseln



## ■ ***Prolog-Programm (Hornklauseln)***

- Einlesen und Abarbeiten aus Datei
- Interaktiv im Interpreter
- Hinzufügen/Löschen von Regeln/Fakten zur Laufzeit
- **Gegeben:** (Hornklausel-) Wissensbasis
- **Ziel:** Beantwortung von Anfragen

## ■ ***Systematische Antwortfindung durch Widerspruchsbeweis (Resolution)***

- Resultat positiv → Antwort logisch ableitbar
- Resultat negativ → Aufgrund der Datenbasis keine Antwort ableitbar (CWA)



# Prolog

## Datenobjekte (Terme)



### ■ **Konstanten**

- s\_J79, 'S. Jones', ::= (Atoms)

### **Konvention**

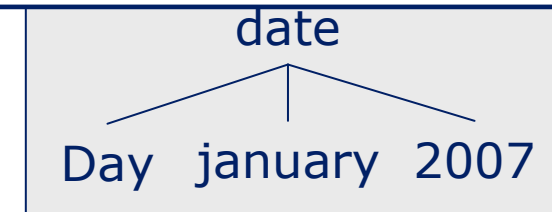
Konstanten : nur Kleinbuchstaben

Strukturen : nur Kleinbuchstaben

Variablen : mit Großbuchstabe beginnend,  
Rest Kleinbuchstaben

↑  
Funktork

← ↑ →  
Argumente



# Prolog Fakten



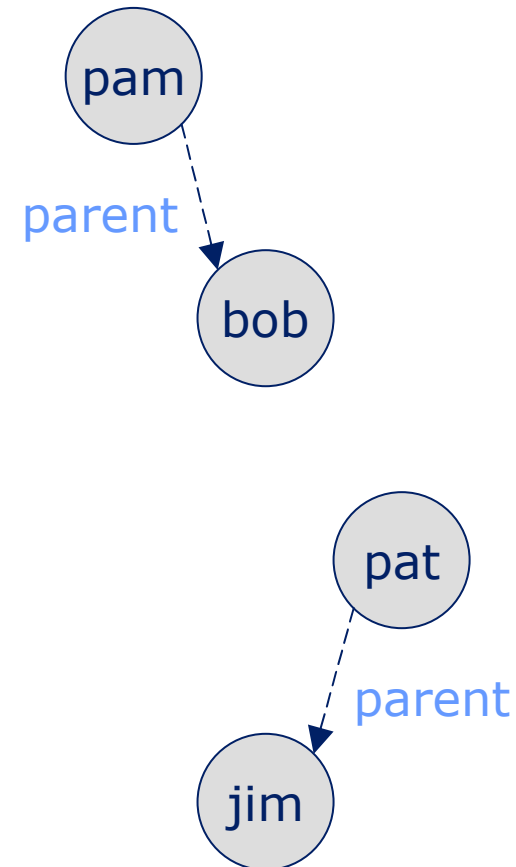
- Repräsentation von Objekten und Sachverhalten

- **Beispiel:** Familien-Relation

- „pam is a parent of bob“
- „pat is a parent of jim“

- **In Prolog:**

```
parent( pam, bob ).  
parent( tom, bob ).
```



# Prolog

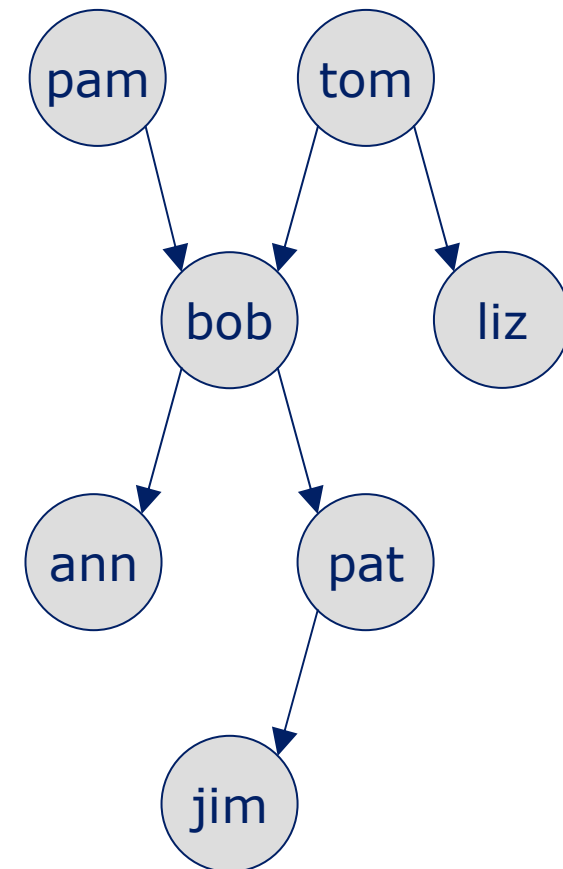
## Anfragen



### ■ *In Prolog:*

```
parent( pam, bob ).  
parent( tom, bob ).  
parent( tom, liz ).  
parent( bob, ann ).  
parent( bob, pat ).  
parent( pat, jim ).
```

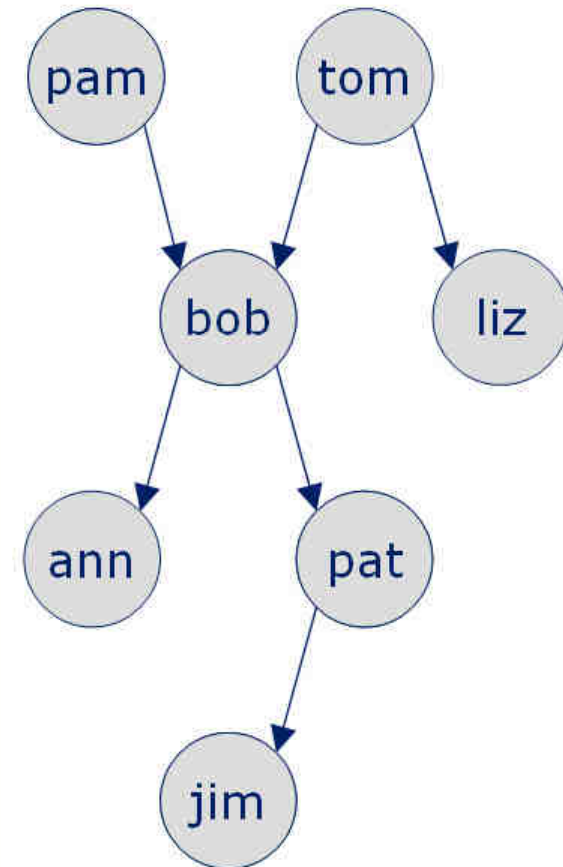
→ **Anfragen:**  
**Kurz-Demo**



```
SWI-Prolog -- j:/Präsentation/family.pl
File Edit Settings Run Debug Help
% j:/Präsentation/family.pl compiled 0.00 sec, 2,880 bytes
Welcome to SWI-Prolog (Multi-threaded, Version 5.6.20)
Copyright (c) 1990-2006 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- parent(bob,pat).
Yes
2 ?- parent(liz,pat).
No
3 ?- parent(tom,ben).
No
4 ?- parent(bob,X).
X = ann ;
X = pat ;
No
5 ?- parent(bob,_).
Yes
6 ?- parent(X,Y).
X = pam
Y = bob ;
X = tom
Y = bob
Yes
7 ?- █
```



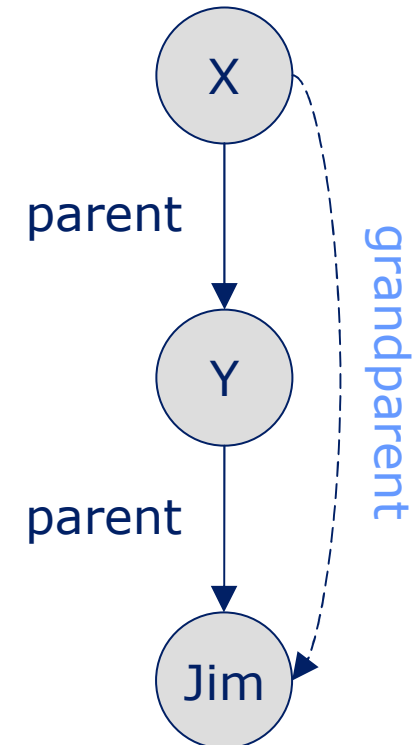
# Prolog Anfragen



- Wer ist **Großelternteil** von Jim?
  - Wer ist ein Elternteil von Jim?  
Sei dies **Y**.
  - Wer ist ein Elternteil von **Y**?  
Sei dies **X**.
- **X** ist ein Großelternteil von Jim.

## ■ ***In Prolog:***

```
?- parent( Y, Jim), parent( X, Y).
```



# Prolog Anfragen



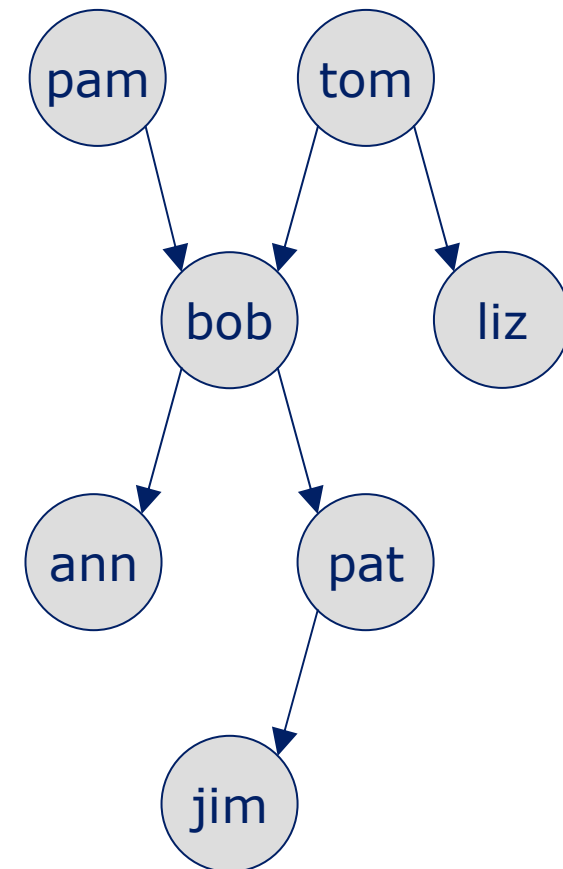
- Wer ist **Großelternteil** von Jim?
  - Wer ist ein Elternteil von Jim?  
Sei dies **Y**.
  - Wer ist ein Elternteil von **Y**?  
Sei dies **X**.
  - **X** ist ein Großelternteil von Jim.

- **In Prolog:**

```
?- parent( Y, Jim), parent( X, Y).
```

```
X = bob
```

```
Y = pat
```



# Prolog Regeln



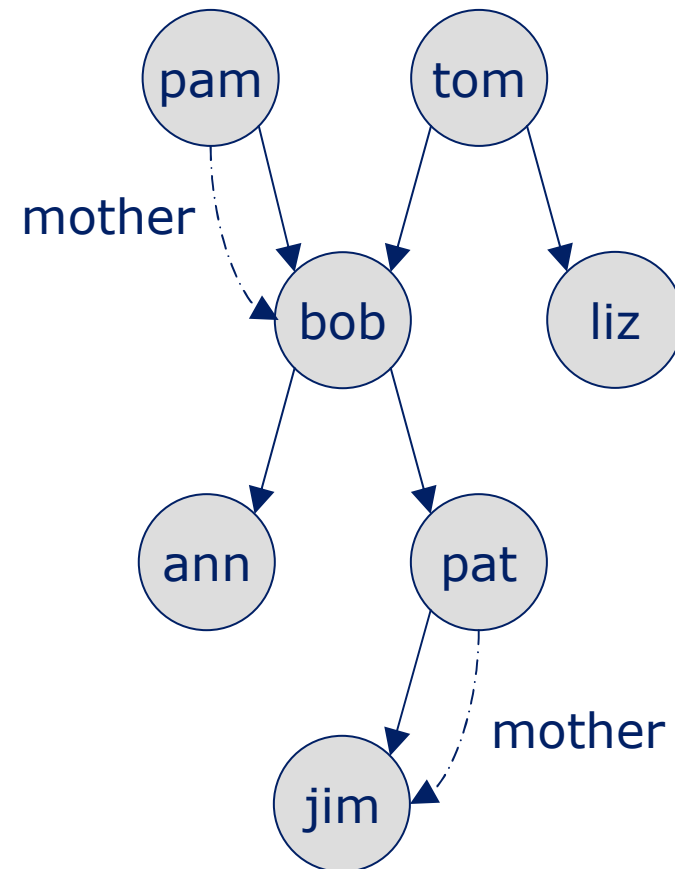
- Repräsentation von Abhängigkeiten, Kausalitäten, und Strukturbeziehungen

- **Beispiel:** Mutter-Relation

- Naiver Ansatz:

```
mother( pam, bob).  
mother( pat, jim).
```

- Eleganter:  
Allgemeine Definition von **mother**  
mit Hilfe von **parent** und **female**

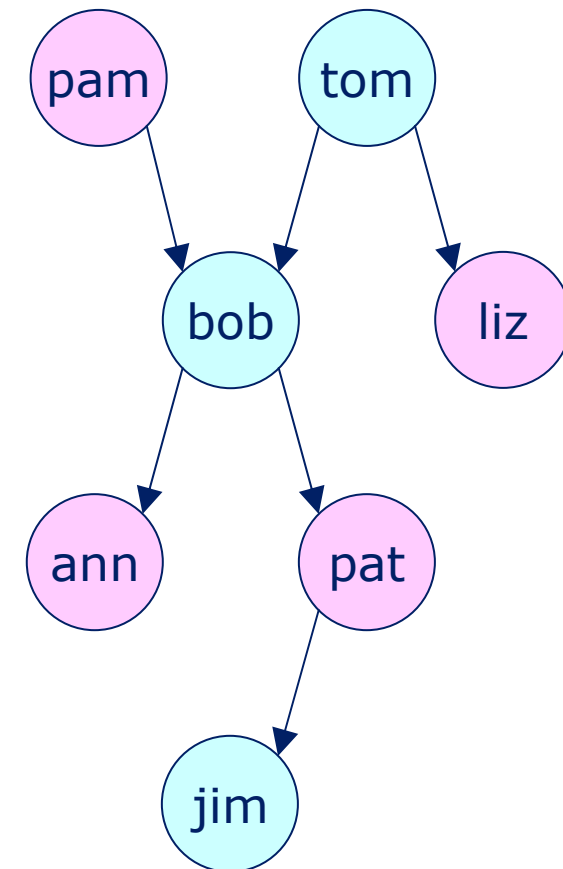


# Prolog Regeln



```
parent( pam, bob).  
parent( tom, bob).  
parent( tom, liz).  
parent( bob, ann).  
parent( bob, pat).  
parent( pat, jim).
```

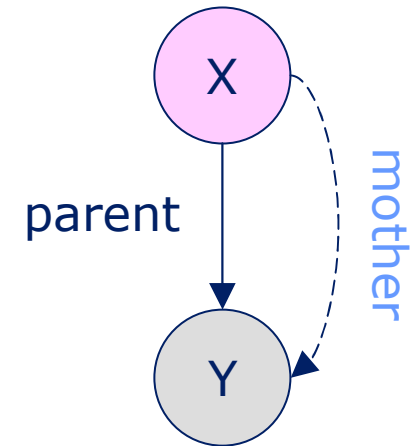
```
female( pam).  
male( tom).  
male( bob).  
female( liz).  
female( ann).  
female( pat).  
male( jim).
```



# Prolog Regeln



- Für alle  $X, Y$  gilt:  
X ist die Mutter von Y falls
  - X ein Elternteil von Y ist und
  - X weiblich ist



- ***Als Hornklausel:***

$\text{mother}(X, Y) \leftarrow \text{parent}(X, Y) \wedge \text{female}(X)$

- ***In Prolog: Regel***

$\text{mother}(X, Y) \text{ :- parent}(X, Y), \text{female}(X).$

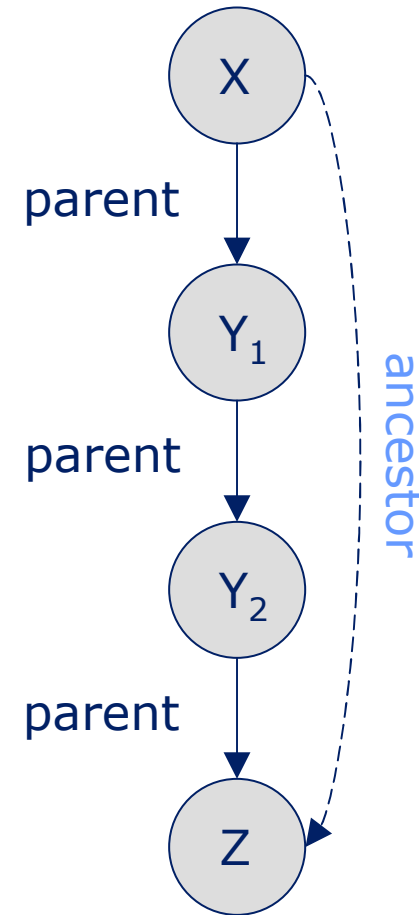
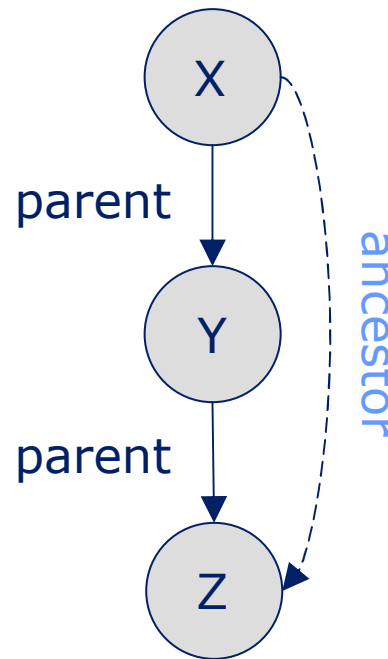
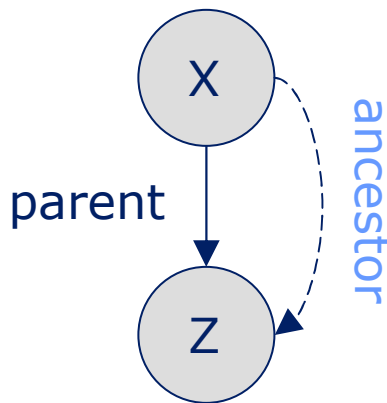
# Prolog

## Rekursion



### ■ **Beispiel:** Vorfahr-Relation

- Naiver Ansatz:
  - umständlich
  - unvollständig

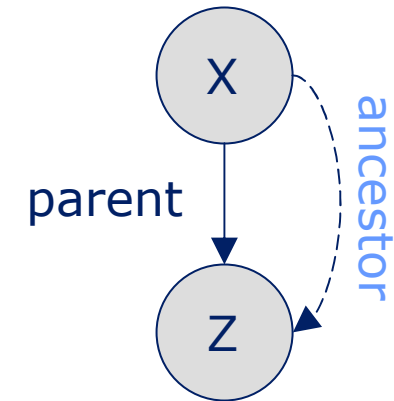


# Prolog

## Rekursion



- Für alle  $X, Z$  gilt:
  - $X$  ist ein Vorfahr von  $Z$ , falls  $X$  ein Elternteil von  $Z$  ist.
    - direkter Vorfahr

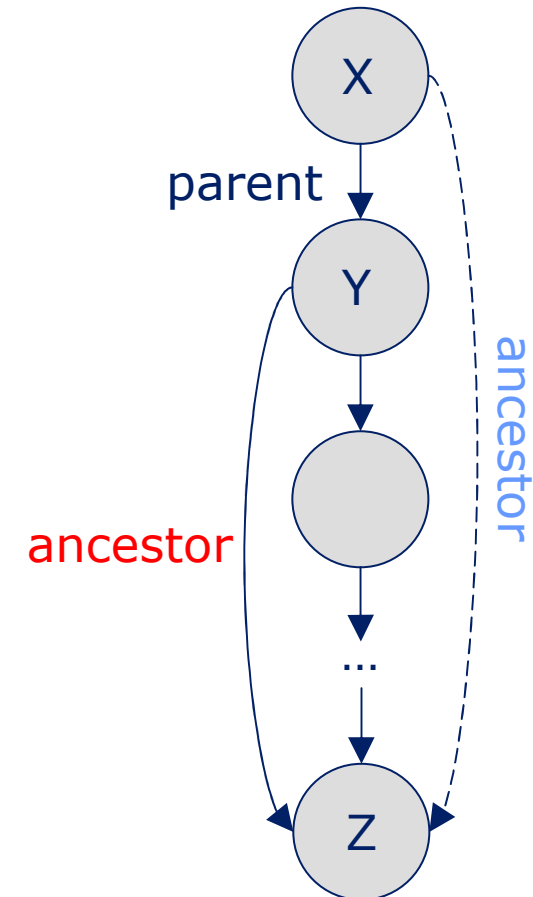


# Prolog

## Rekursion



- Für alle  $X, Z$  gilt:
  - $X$  ist ein Vorfahr von  $Z$ , falls  $X$  ein Elternteil von  $Z$  ist.
    - direkter Vorfahr
  - $X$  ist ein Vorfahr von  $Z$ , falls es ein  $Y$  gibt, sodass
    - (1)  $X$  ein Elternteil von  $Y$  und
    - (2)  $Y$  ein **Vorfahre** von  $Z$  ist.
    - indirekter Vorfahr
- Eleganter und vollständig



# Prolog

## Rekursion



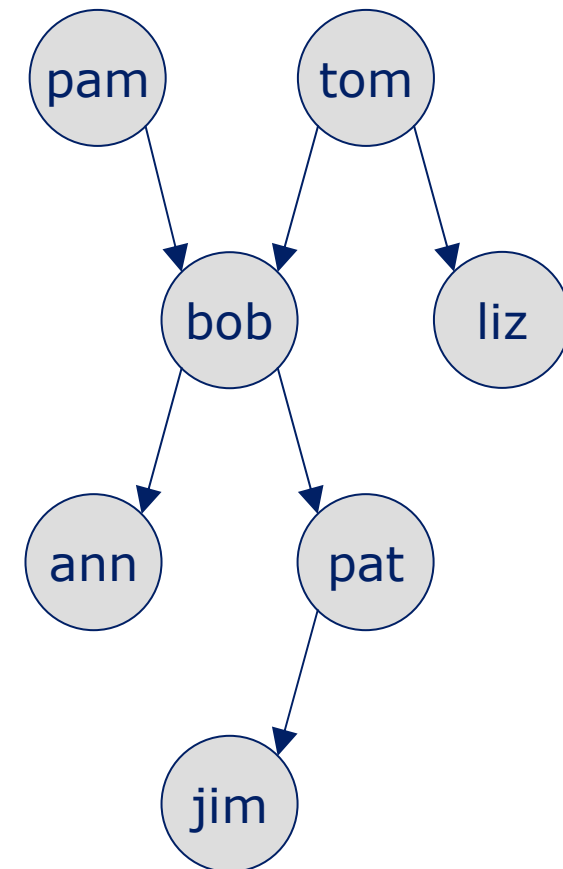
### ■ *In Prolog:*

```
ancestor( X, Z) :-      % direkt
    parent( X, Z).
ancestor( X, Z) :-      % indirekt
    parent( X, Y),
    ancestor( Y, Z).
```

### ■ *Anfrage:*

```
?- ancestor(pam, X).
```

```
X = bob
X = ann
X = pat
X = jim
```



# Prolog Matching



- 2 Terme matchen
  - wenn sie identisch sind
  - wenn die Variablen zu Objekten instantiiert werden können, sodass die beiden Terme nach der Substitution der Variablen durch diese Objekte identisch sind
- **Beispiele:**
  - `date(D, M, 2001)` und `date(D1, may, Y1)` matchen
  - `date(1, M, Y)` und `date(2, may, 2001)` matchen nicht
  - `date(D, M, 2001)` und `d(D, M, 2001)` matchen nicht
- Prolog berechnet die allgemeinste Instantiierung  
→ Vergleichbar mit Unifikation in der Logik

# Prolog Matching



## ■ **Anwenden einer Regel**

- Erfüllen von einem oder mehreren Zielen
- Zeigen, dass das Ziel aus den Fakten und Regeln folgt

## ■ **Anfrage:** ?- mother(pam, bob).

- keine Fakten über die Relation „mother“ vorhanden
- Regel: mother(X, Y) :- parent(X, Y), female(X)
- Matching: X = pam und Y = bob
- mother(pam, bob) :- parent(pam, bob), female(pam)

## ■ **Neues Ziel:** parent(pam, bob), female(pam)

- beide Fakten erfüllt
- Prolog antwortet mit „yes“

# Prolog

## Backtracking



**Anfrage:**      ?- ancestor( tom, pat).

- Tiefensuche nach Regeln,  
deren Kopf das Ziel matchen
- Einzige Relevante Klauseln:  
Regeln der ancestor Relation

```
ancestor( X, Z) :- % ar1
    parent( X, Z).
ancestor( X, Z) :- % ar2
    parent( X, Y),
    ancestor( Y, Z).
```

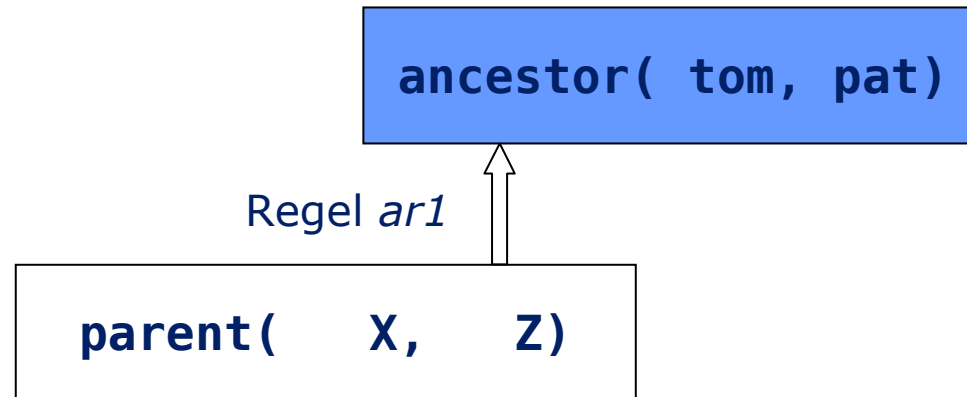
# Prolog Backtracking



```
ancestor( tom, pat)
```

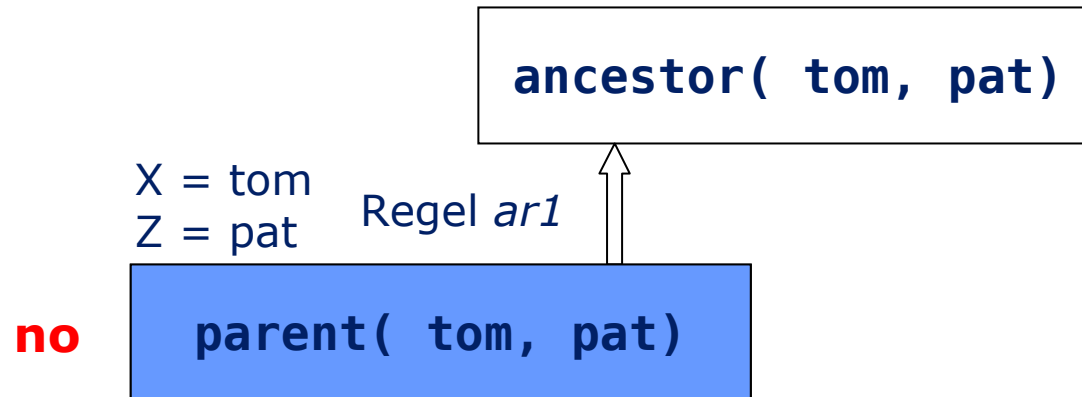
```
ancestor( X, Z) :- % ar1  
    parent( X, Z).  
ancestor( X, Z) :- % ar2  
    parent( X, Y),  
    ancestor( Y, Z).
```

# Prolog Backtracking



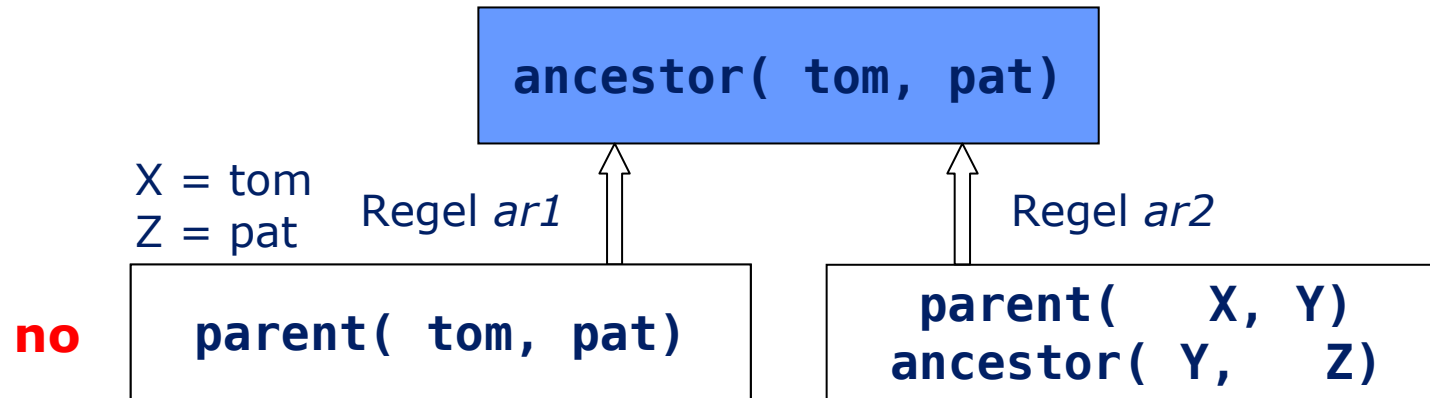
```
ancestor( X, Z) :- % ar1  
    parent( X, Z).  
ancestor( X, Z) :- % ar2  
    parent( X, Y),  
    ancestor( Y, Z).
```

# Prolog Backtracking



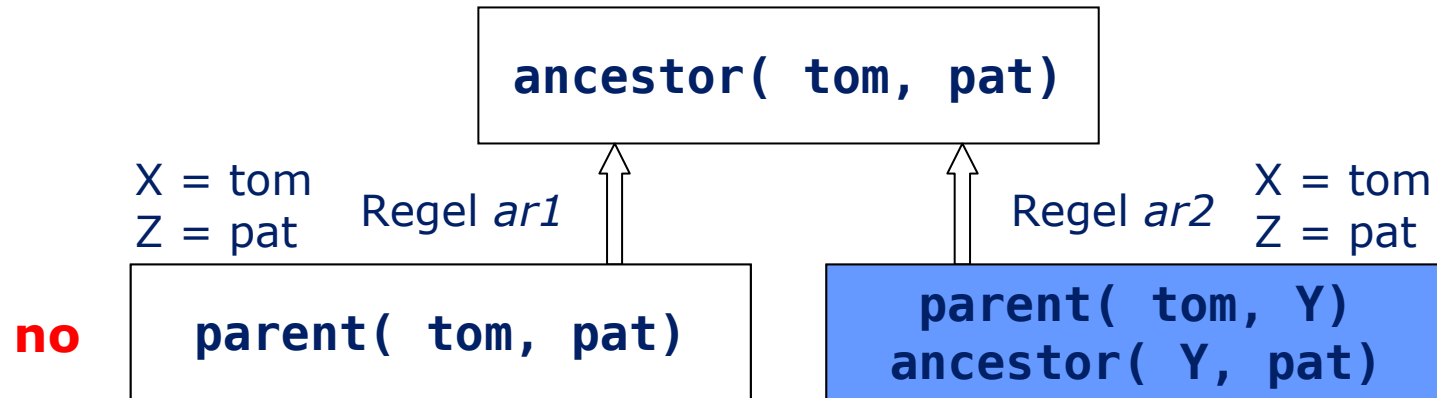
```
ancestor( X, Z) :- % ar1
    parent( X, Z).
ancestor( X, Z) :- % ar2
    parent( X, Y),
    ancestor( Y, Z).
```

# Prolog Backtracking



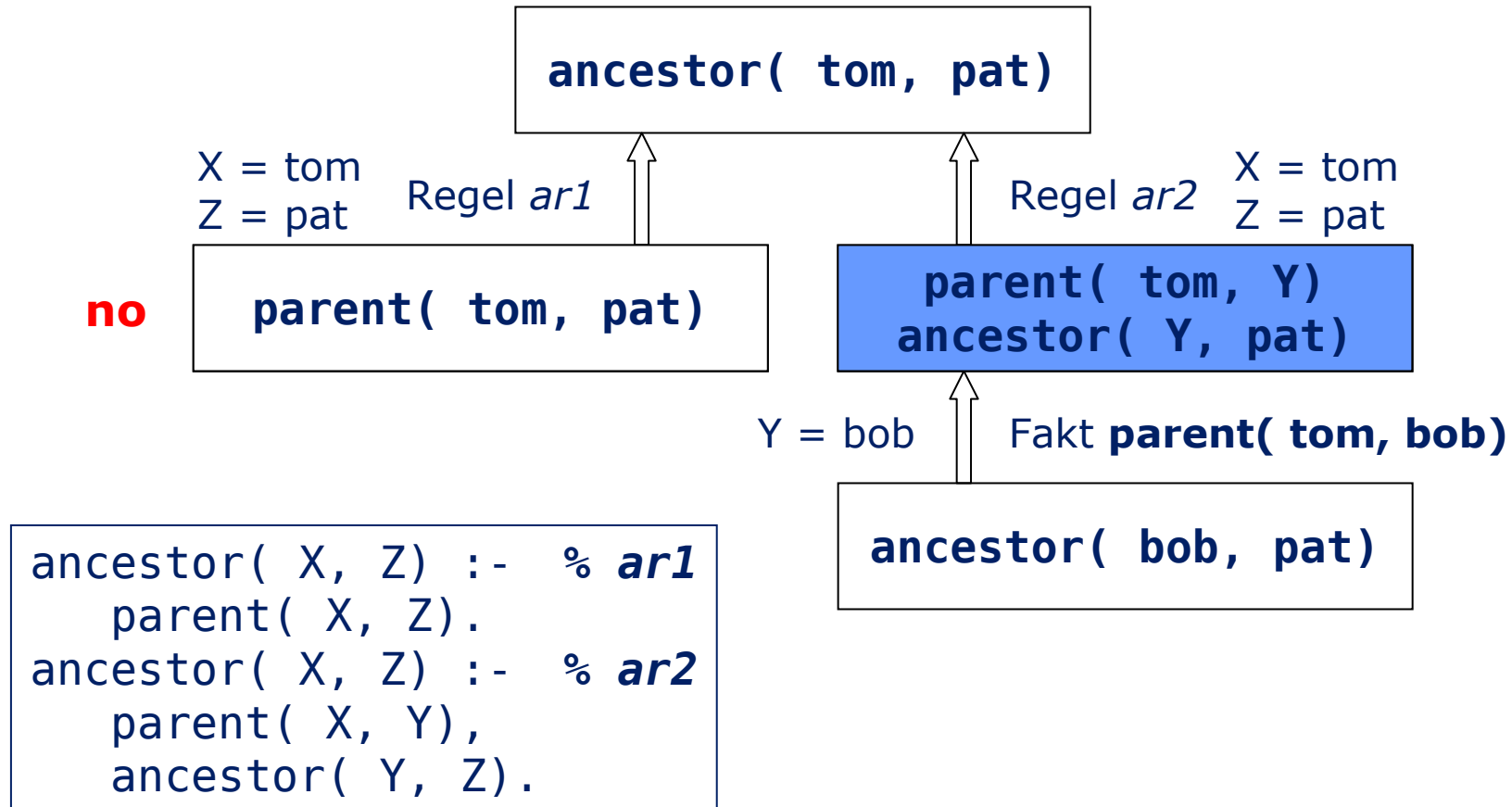
```
ancestor( X, Z ) :- % ar1
    parent( X, Z ).
ancestor( X, Z ) :- % ar2
    parent( X, Y ),
    ancestor( Y, Z ).
```

# Prolog Backtracking

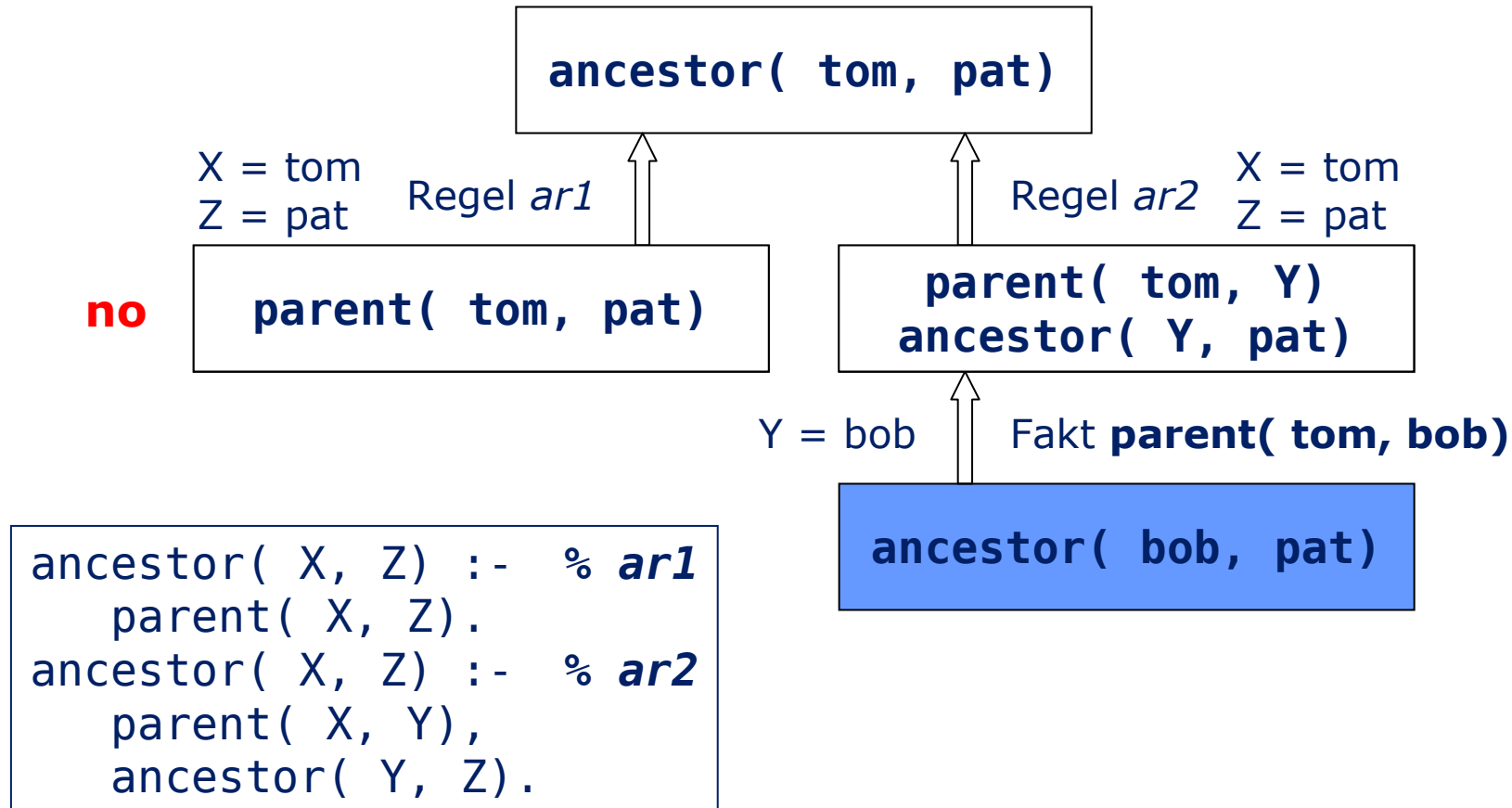


```
ancestor( X, Z ) :- % ar1
    parent( X, Z ).
ancestor( X, Z ) :- % ar2
    parent( X, Y ),
    ancestor( Y, Z ).
```

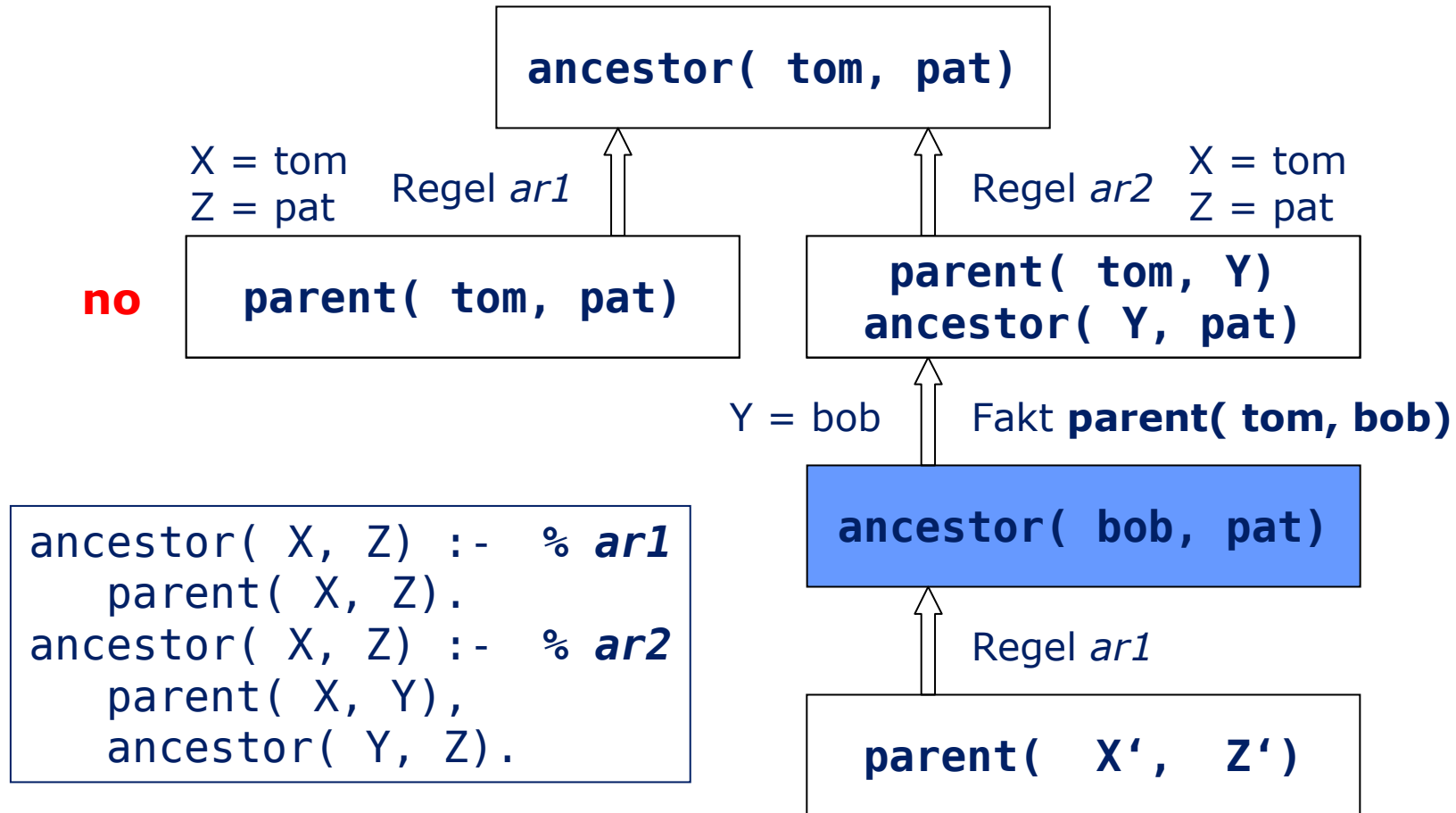
# Prolog Backtracking



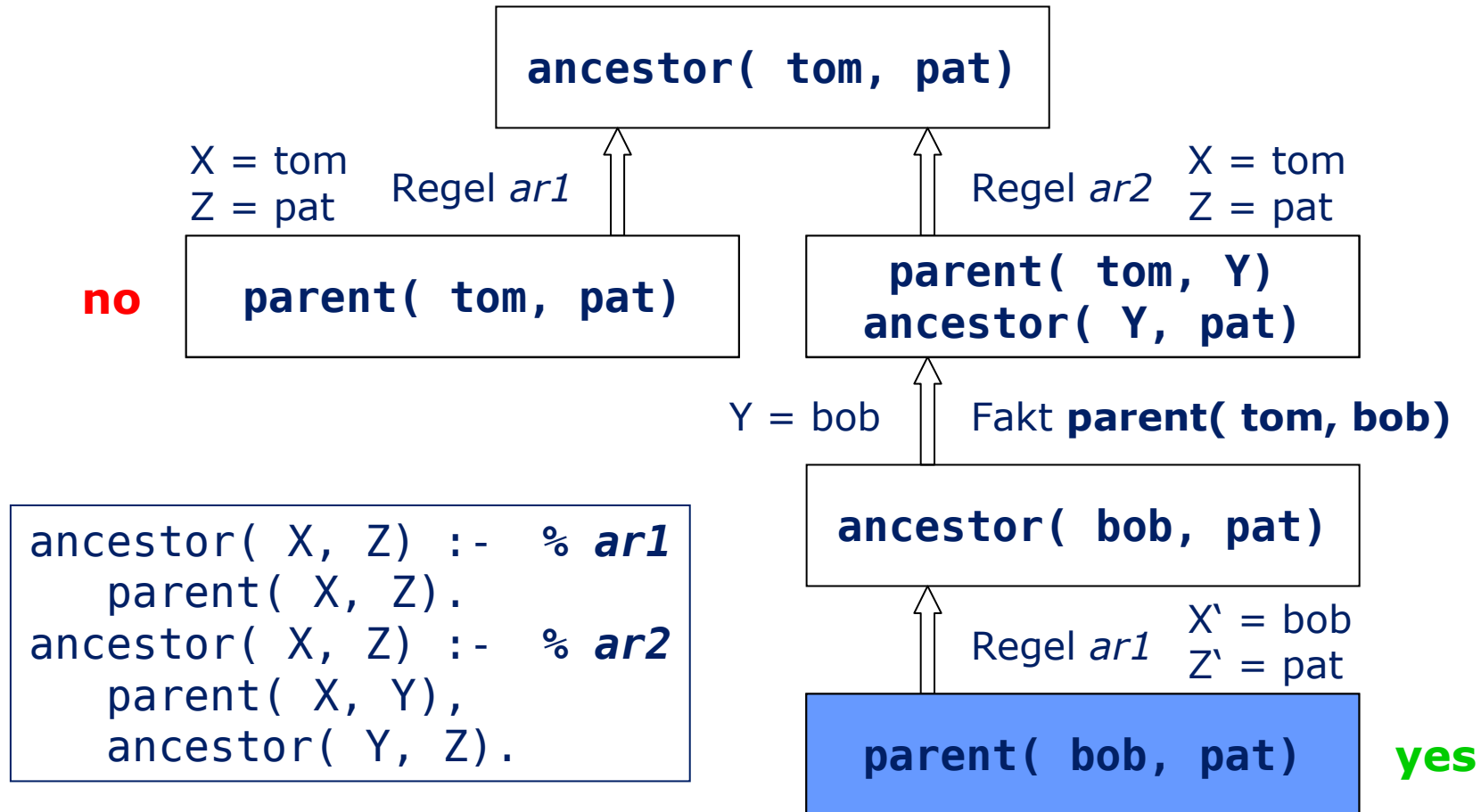
# Prolog Backtracking



# Prolog Backtracking



# Prolog Backtracking



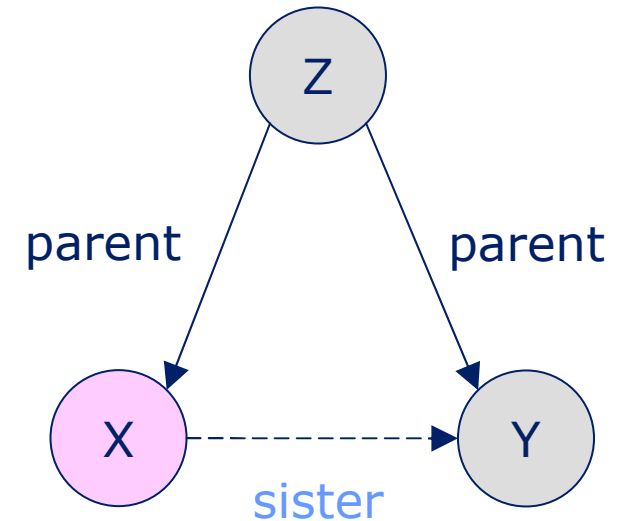
# Prolog

## Vordefinierte Prädikate



- Für alle  $X, Y$  gilt:  $X$  ist eine Schwester von  $Y$  falls gilt:

- (1)  $X$  und  $Y$  haben ein gemeinsames Elternteil  $Z$  und
- (2)  $X$  ist weiblich



- ***In Prolog:***

```
sister( X, Y) :-      % X is a sister of Y if
  parent( Z, X),      % X and Y have the
  parent( Z, Y),      % same parent and
  female( X).         % X is female
```

# Prolog

## Vordefinierte Prädikate



### ■ **Anfrage:**

```
?- sister( X, pat).  
X = ann  
X = pat
```

### ■ **Lösung:** vordefiniertes Prädikat **different**

```
sister( X, Y) :-      % X is a sister of Y if  
    parent( Z, X),    % X and Y have the  
    parent( Z, Y),    % same parent and  
    female( X),       % X is female and  
    different( X, Y). % X and Y are different
```

### ■ Weitere vordefinierte Prädikate und Funktionen: u.a. Arithmetik, Steuerung, Listen, I/O

# Prolog Arithmetik



## ■ Prädikate für die grundlegende Arithmetik:

- + : Addition
- - : Subtraktion
- \* : Multiplikation
- / : Gleitkomma Division
- \*\* : Potenz
- // : Ganzzahl Division
- mod : Modulo

## ■ **Beispiel:**

$?- X = 1 + 2.$   
 $X = 1 + 2$  →  $?- X = +(1, 2).$

## **Lösung: is**

$?- X \text{ is } 1 + 2.$   
 $X = 3$

# Prolog

## Arithmetik



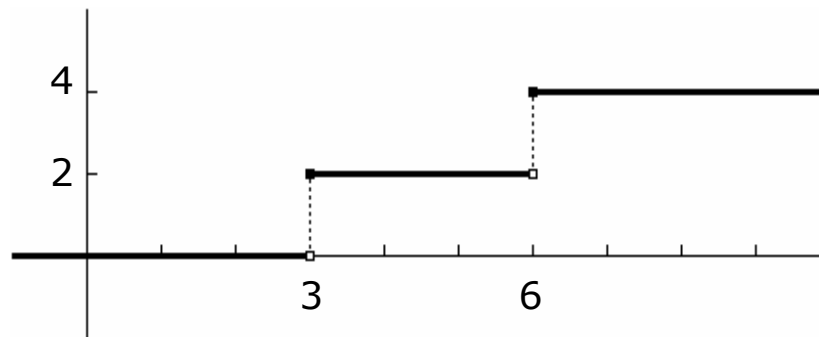
- Prädikate für den Wertevergleich:
  - $>$  : größer
  - $<$  : kleiner
  - $>=$  : größer-gleich
  - $=<$  : kleiner-gleich
  - $\neq$  : Ungleichheit
  - $:=$  : Gleichheit
  - Vorsicht: „ $=$ “ bedeutet matchen von Objekten!

# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*



$f(1, Y)$   
 $2 < Y$

### ■ *In Prolog:* $f(x, y)$

$f(X, 0) \text{ :- } X < 3.$   
 $f(X, 2) \text{ :- } 3 \leq X, X < 6.$   
 $f(X, 4) \text{ :- } 6 \leq X.$

### ■ *Anfrage:*

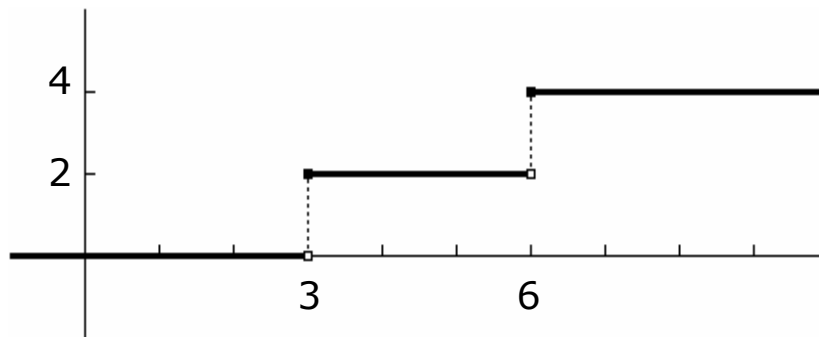
$?- f(1, Y), Y < 2.$

# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

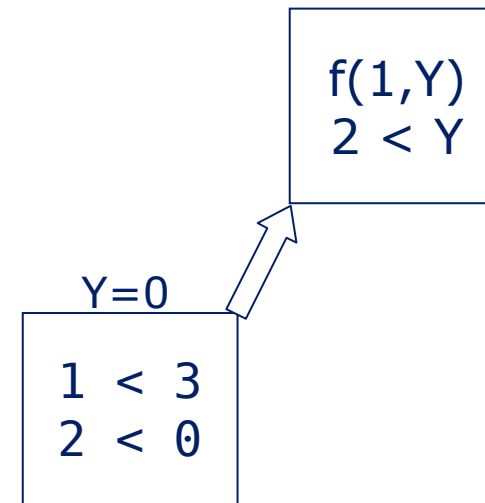


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3.  
f(X,2) :- 3 =< X, X < 6.  
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```

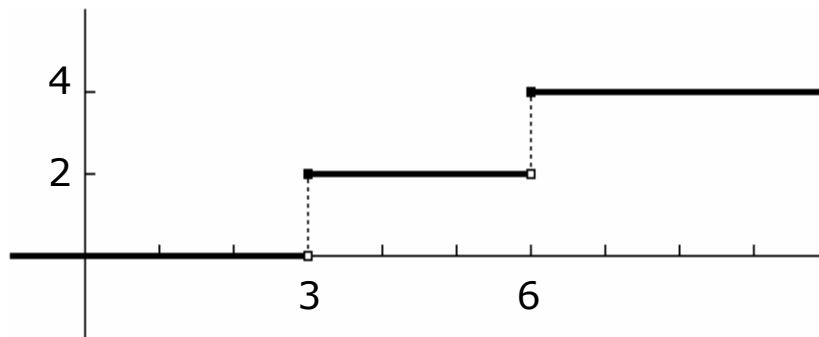


# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

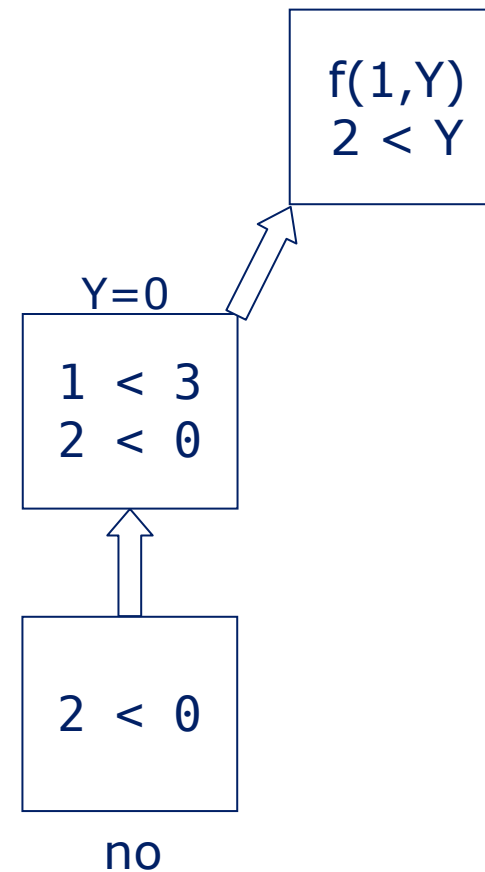


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3.  
f(X,2) :- 3 =< X, X < 6.  
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```

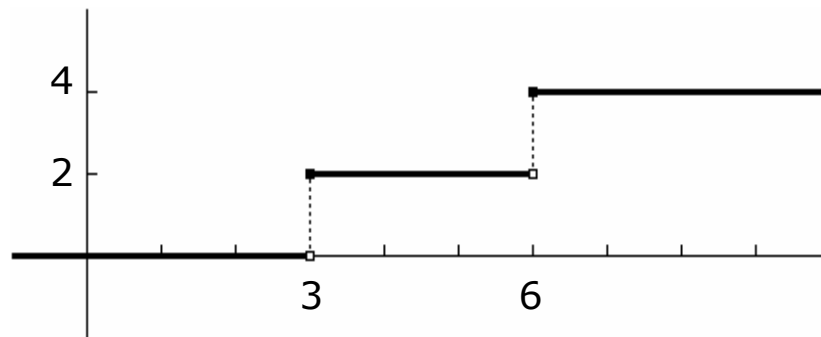


# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

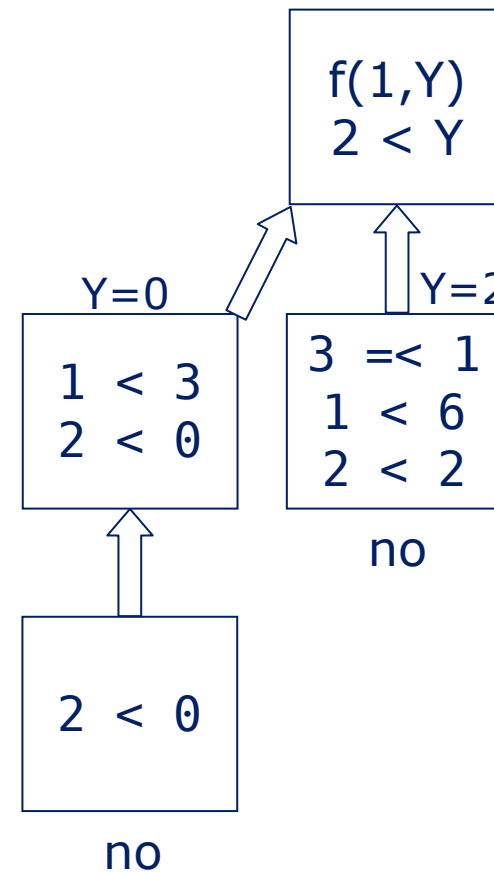


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3.  
f(X,2) :- 3 =< X, X < 6.  
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```

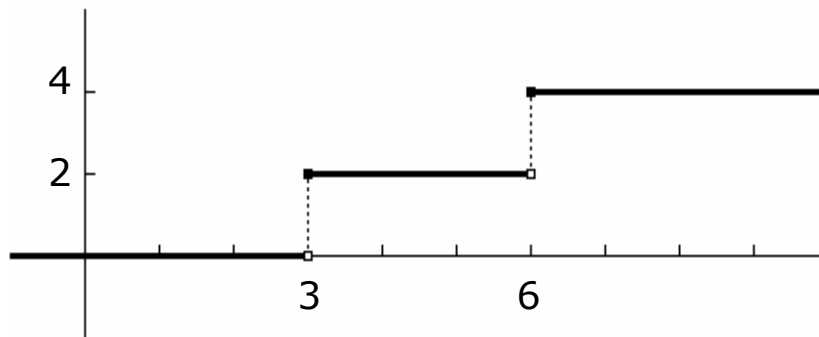


# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

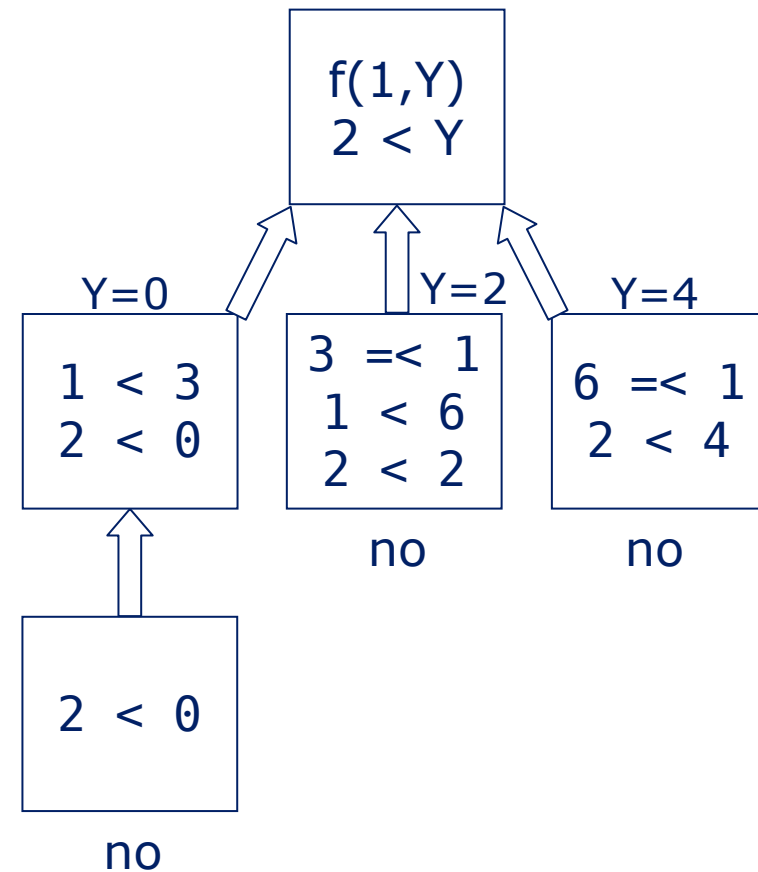


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3.
f(X,2) :- 3 =< X, X < 6.
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```

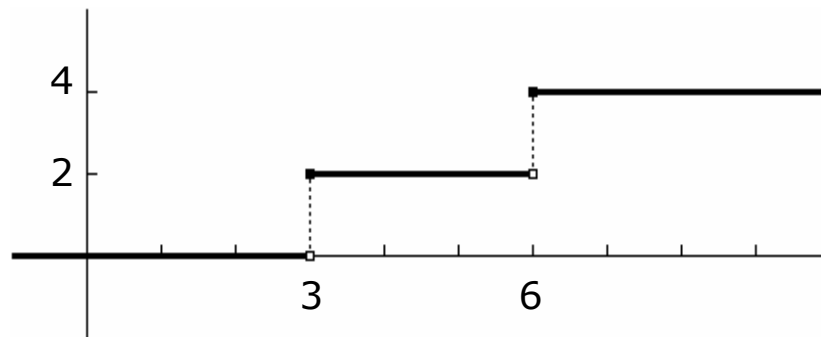


# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*



f(1,Y)  
2 < Y

### ■ *In Prolog:* f(x, y)

```
f(X,0) :- X < 3, !.  
f(X,2) :- 3 =< X, X < 6, !.  
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```

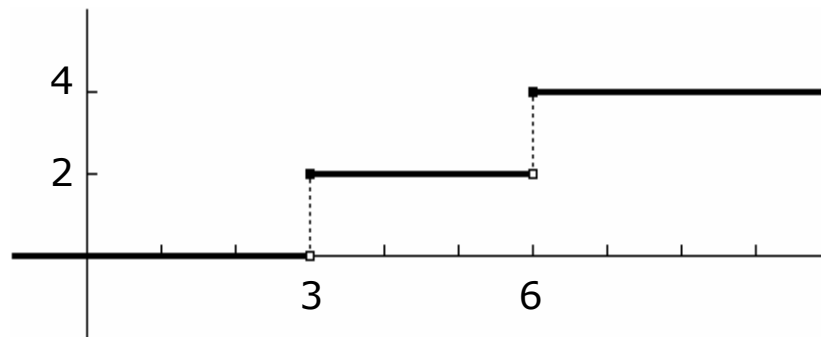
**Cut: !**

# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

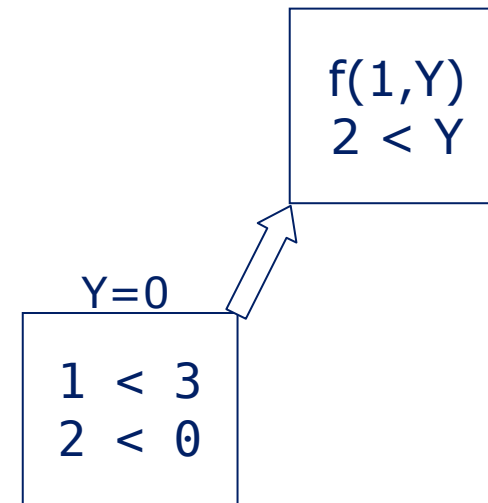


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3, !.  
f(X,2) :- 3 =< X, X < 6, !.  
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```



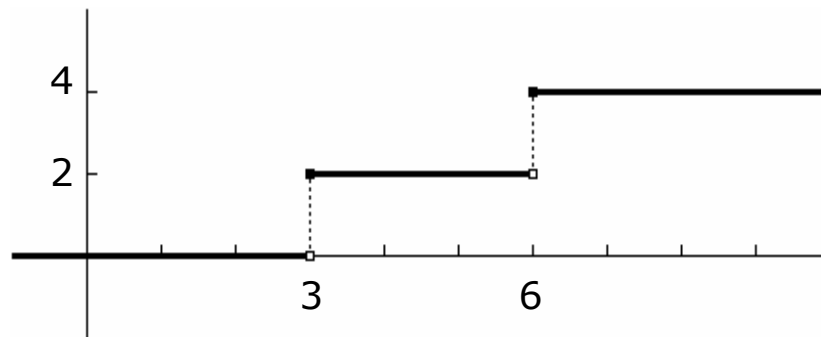
**Cut: !**

# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

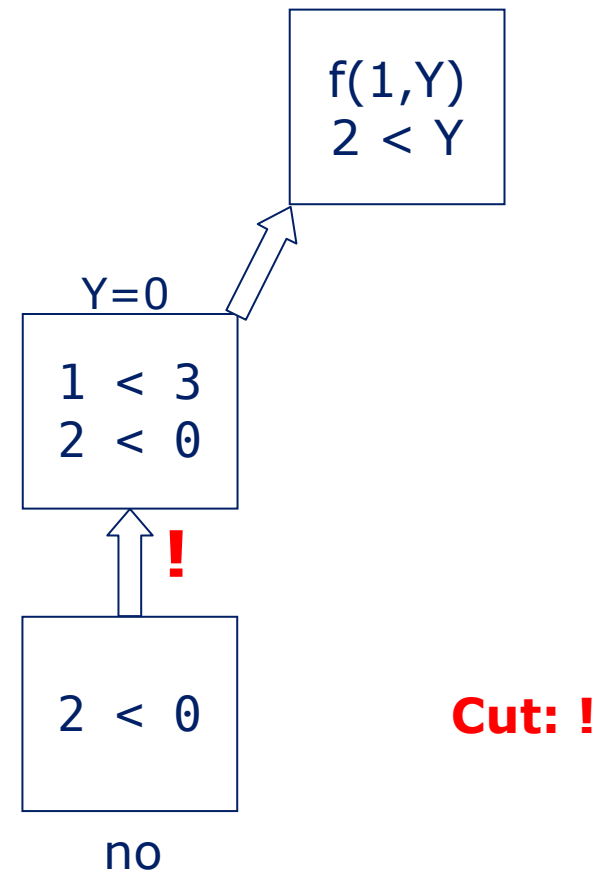


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X, X < 6, !.
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```

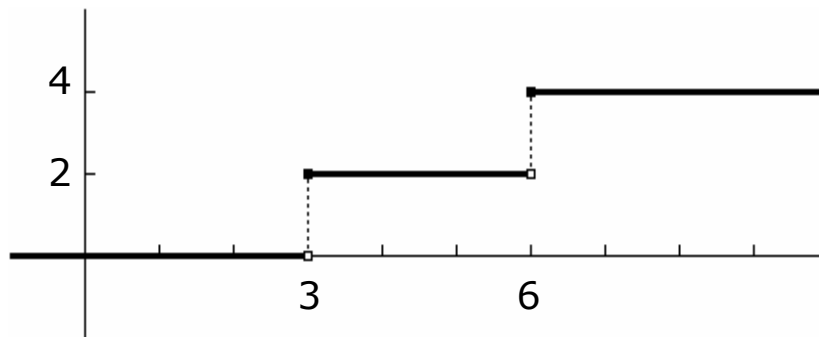


# Prolog

## Steuerung: Cut Operator



### ■ *Beispiel:*

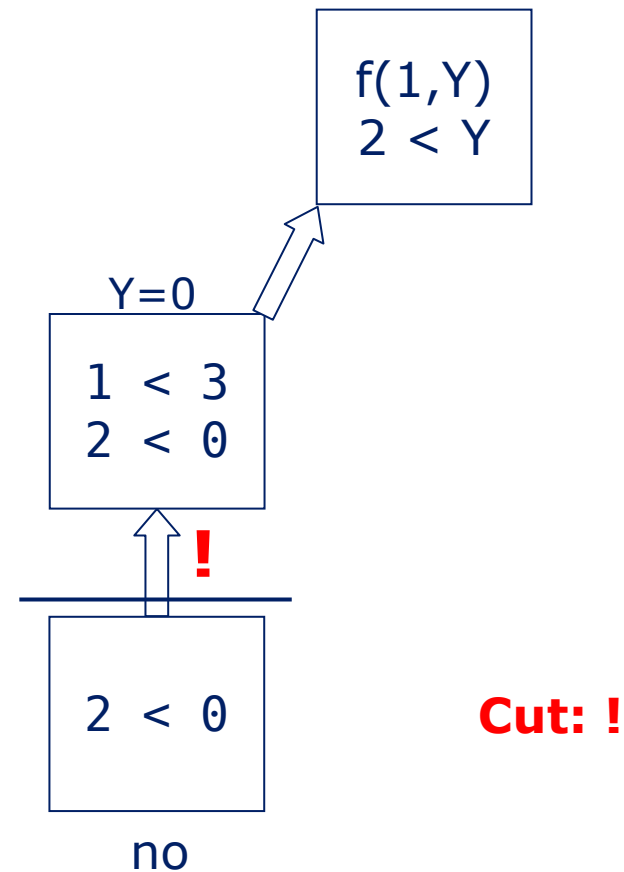


### ■ *In Prolog:* $f(x, y)$

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 =< X, X < 6, !.
f(X,4) :- 6 =< X.
```

### ■ *Anfrage:*

```
?- f(1,Y), Y < 2.
```



# Prolog

## Einfluss und Anwendung



### ■ ***Beeinflussung vieler Entwicklungen***

- 5th Generation Computer
- Deduktive Datenbanken
- Constraint Logic Programmierung

### ■ ***Anwendungsgebiete***

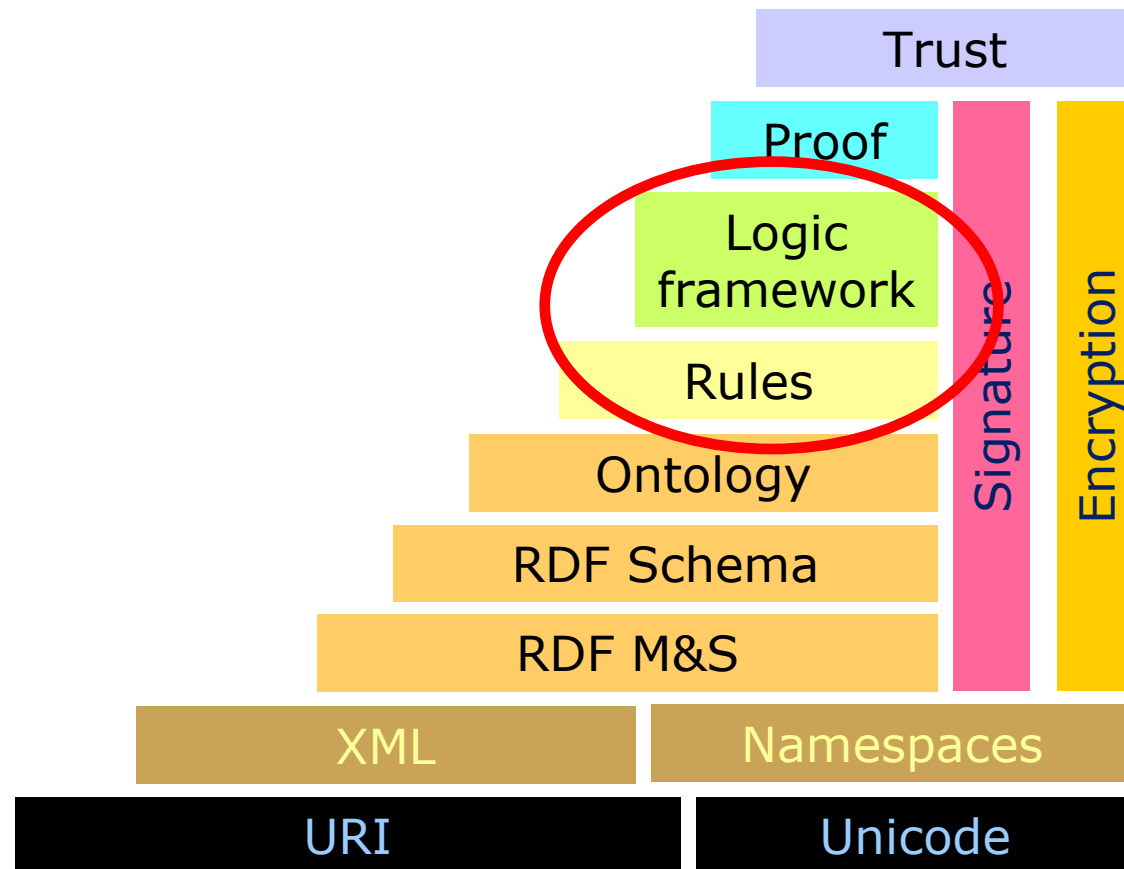
- Künstliche Intelligenz
- Computerlinguistik
- Expertensysteme
- Automatisierte Beweise

### ■ ***Kommerzielle Anwendung***

- z.B. Systemmanagement von Geschäftsanwendungen
- IBM: „Tivoli Enterprise Console“ (BIM-Prolog)

# Prolog

## Anwendung: Semantic Web



# Prolog

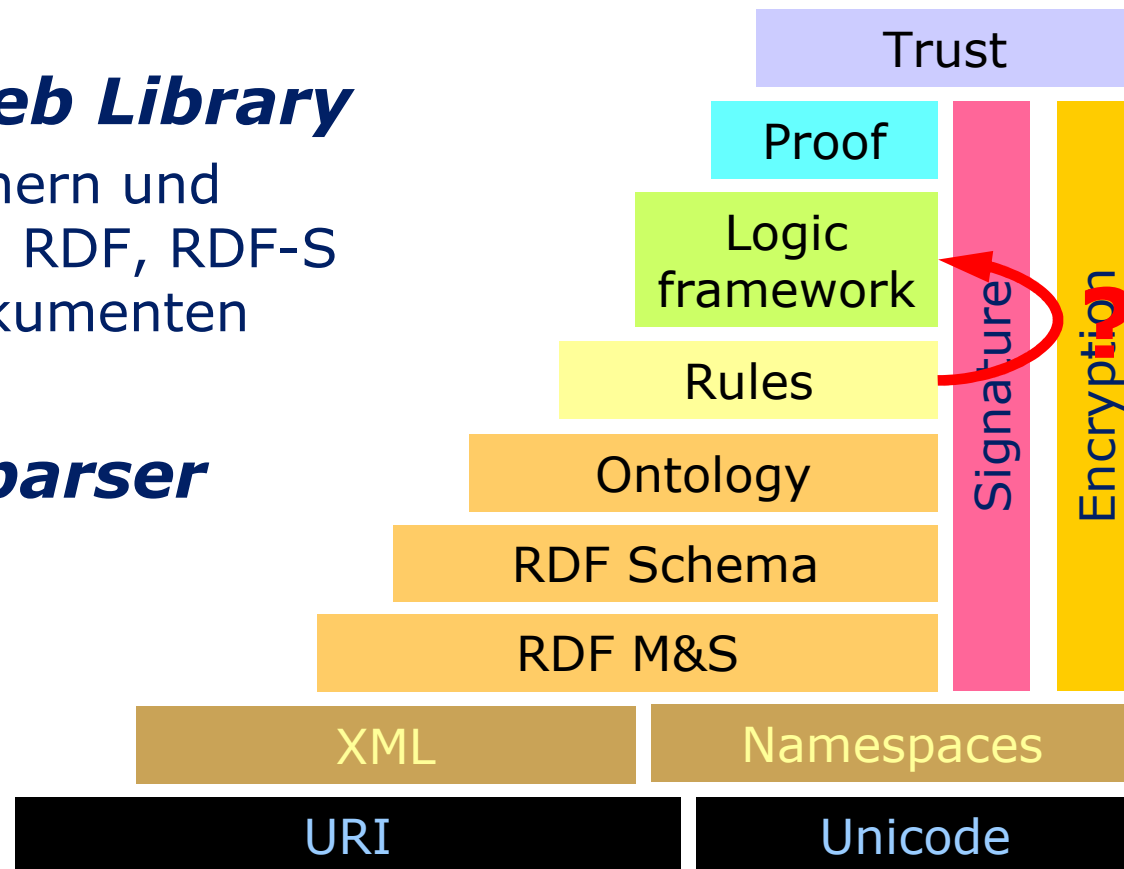
## Anwendung: Semantic Web



- **SWI-Prolog**  
**Semantic Web Library**

→ Lesen, Speichern und Anfragen von RDF, RDF-S und OWL Dokumenten

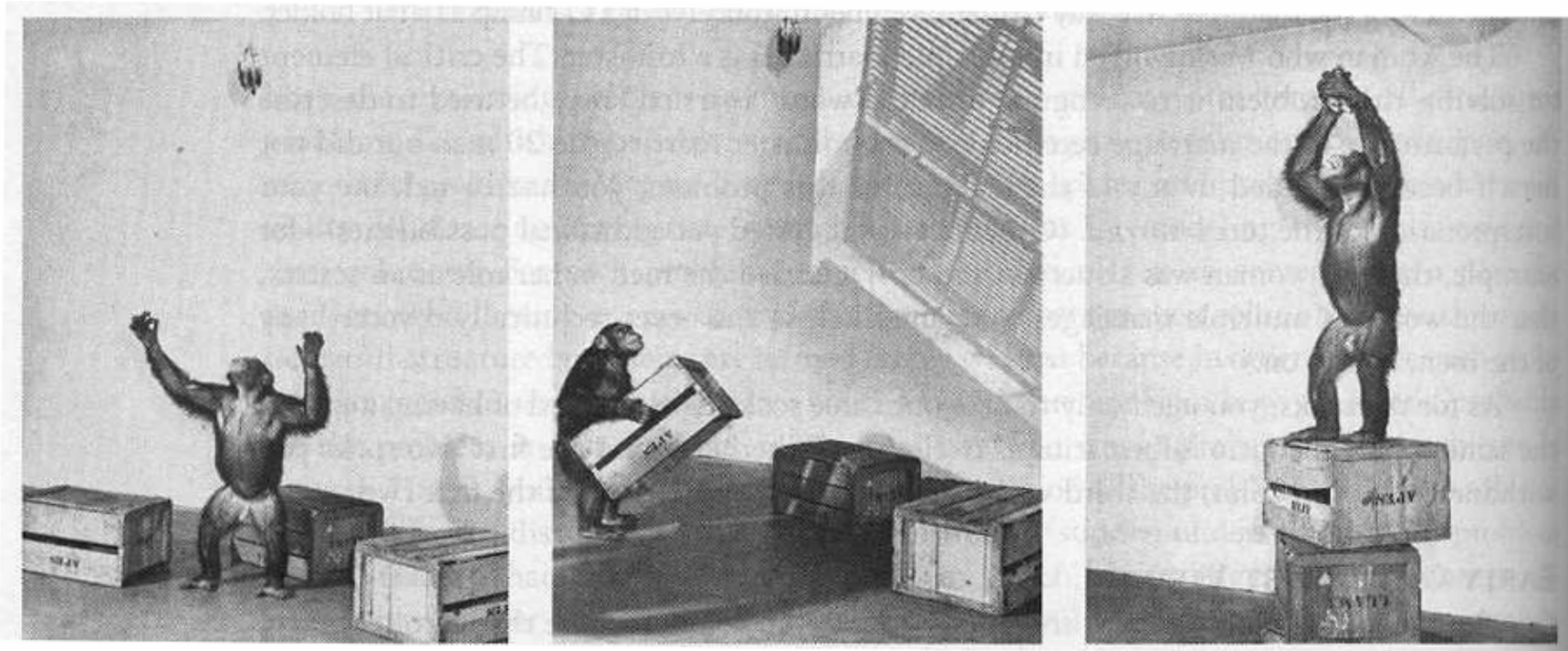
- **SWI-Prolog**  
**SGML/XML parser**



# Prolog Demo



## ■ *Demo: Monkey Banana Problem*





## ■ Rules

- Kein aktueller Standard für Regelsprachen
- Großes Forschungsgebiet
- Ansatz: SWRL

## ■ Prolog

- Mechanismen: Baum-basierte Datenstrukturierung, Matching, Backtracking
- Speziell geeignet für Probleme, die (strukturierte) Datenobjekte und deren Beziehungen behandeln
- Mächtige Sprache für Künstliche Intelligenz und nicht-numerische Programmierung
- Mögliches Logik Framework für das Semantic Web durch Semantic Web Library/XML Parser für SWI Prolog

# Semantic Web Rules & Prolog



***Fragen?***