Kapitel 6:
Prädikatenlogik - Hornlogik - PROLOG

Quellen:

- für ein free download
  http://www.swi.psy.uva.nl/projects/SWI-Prolog/
- für ein anderes, inklusive **vielen** Demos etc.
  http://www.visual-prolog.com
- ein kurzes Tutorial:
  http://cbl.leeds.ac.uk/~tamsin/prologtutorial/
- ein langes Tutorial mit vielen Beispielen:
  http://www.intranet.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html
- ein Klassiker:
  Ivan Bratko: PROLOG - Programmierung für KI, Addison-Wesley
- für die logischen Grundlagen wie immer
  Socher-Ambrosius & Heinsohn

Wo wir sind ...

Einleitung
- Einführung KI ✓
- Einführung Expertensysteme ✓

(Schwache) Problemlösemethoden
- Heuristische Suche ✓
- Constraints ✓

Wissensrepräsentation und Inferenz
- Aussagenlogik ✓
- Rückwärtsregeln: Prolog
- Vorwärtsregeln: OPS 5
- Semantische Netze
- Frames
- ...

Nochmal zurück zur Aussagenlogik

Beispiel:

- Aussage A: ***Martin ist ein Informatiker.***
- Aussage B: ***Jeder Informatiker kann programmieren.***
- Aussage C: ***Martin kann programmieren.***

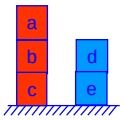
Kann ich C aus A und B folgern?

Ein Kernproblem der Aussagenlogik

- Ich kann keine Aussagen über ganze Klassen von Objekten machen und diese sinnvoll zum Schlußfolgern über Individuen nutzen.
- Dieser Mechanismus (abstrakte Aussagen und spezifische Instantiierungen) ist aber ein wesentliches Element von Wissen!
- Grundproblem: die Modellierungsmöglichkeiten der Ausagenlogik (komplette Aussagen als Ganzes, ohne die beteiligten Objekte näher beschreiben zu können), sind zu grob und unstrukturiert, um die „Welt“ angemessen zu beschreiben.
- Ziel: einen Apparat entwickeln, der die Ausagenlogik erweitert (so daß die dort erarbeiteten Resultat sich übertragen lassen), aber bessere Modellierungsmöglichkeiten zur Verfügung stellt.

Motivation: Deklarative Wissensrepräsentation

Beispiel: Blockswelt



- Objekte:
  {a,b,c,d,e, red, blue}
- Funktionen:
  color: {<a,red>, <b,red >, <c,red >, <d,blue>, <e,blue>}
- Relationen:
  on: {<a,b>, <b,c>,<d,e>}
  ontable: {c,e}
  clear: {a,d}

Konzeptualisierung:
<{a,b,c,d,e,red,blue},{color},{on,ontable,clear}>

- Die Formalisierung von Wissen in deklarativer Form beginnt mit einer **Konzeptualisierung:**
  - Universe of discourse: Objekte, die in der Welt existieren
  - Beziehungen zwischen den Objekten (Funktionen und Relationen)
- Formal ist eine Konzeptualisierung ein Tripel aus Objekten, Funktionen und Relationen
- Die klassische Sprache zur deklarativen Wissensrepräsentation ist die **Prädikatenlogik**

Prädikatenlogik 1. Stufe (PL1): Syntax

- Die **Syntax** der Prädikatenlogik legt die Datenobjekte fest und definiert damit die Sprache, in der Aussagen formuliert werden
- In der Prädikatenlogik erster Stufe sind folgende **Datenobjekte** verfügbar:
  - K: Menge der Konstantensymbole z.B. {a, b, c, ...}
  - V: Menge der Variablensymbole z.B. {x, y, z, x_1, ...}
  - F: Menge der Funktionssymbole z.B. {f, g, h, f_1, ...}
    jedem Funktionssymbol ist eine Stelligkeit zugeordnet
  - P: Menge der Prädikatensymbole z.B. {p, q, r, p_1, ...}
    jedem Prädikatensymbol ist eine Stelligkeit zugeordnet
  - O: Menge der Operatoren (Junktoren und Quantoren)
- Beispiele für Junktoren: ¬ (nicht)
  & (und)
  v (oder)
  → (impliziert)
  ↔ (äquivalent)
- Beispiele für Quantoren: ∀ (für alle)
  ∃ (existiert)

Prädikatenlogik 1. Stufe: Syntax (Forts.)

- Die Menge der **Terme** ist wie folgt definiert:
  1. alle Variablen- und Konstantensymbole sind ein Term
  2. sei f_n ein n-stelliges Funktionssymbol und seien $t_1,...,t_n$ Terme, dann ist $f_n(t_1,...,t_n)$ ein Term
  3. Alle Terme entstehen aus 1. mittels Iteration von 2.
- Die Menge der **Formeln** ist wie folgt definiert::
  1. Sei p_n ein n-stelliges Prädikatensymbol und seien $t_1,...,t_n$ Terme, dann ist $p_n(t_1,...,t_n)$ eine Formel (atomare Formel oder Atom).
  2. Seien A und B Formeln und sei die Menge der Junktoren gegeben durch $\{\neg, \&, v, \rightarrow, \leftrightarrow\}$, dann sind (A), $\neg A$, A & B, A v B, $A \rightarrow B$, $A \leftrightarrow B$ ebenfalls Formeln.
  3. Sei x eine Variable, A eine Formel und sei die Menge der Junktoren gegeben durch $\{\forall,\exists\}$, dann sind $\forall x$ A und $\exists x$ A ebenfalls Formeln.
  4. Alle Formeln entstehen aus 1. mittels Iteration von 2. und 3.

Semantik der Prädikatenlogik 1. Stufe (Modelltheorie / Tarski-Semantik)

- Eine **Interpretation** I ist ein Paar <U,f>.
  - U ist eine nichtleere Menge von Individuen, das Universum
  - Die Abbildung f ordnet jedem syntaktischen Objekt ein semantisches Objekt zu:
    - jedem Konstantensymbol wird ein Element aus U zugeordnet,
    - jedem Funktionssymbol eine Funktion über U,
    - jedem Prädikatensymbol eine Relation über U mit entsprechender Stelligkeit.
- Unter einer Interpretation läßt sich jeder variablenfreie Term zu einem Element des Universums und jede Formel zu einem **Wahrheitswert** auswerten:
  - Eine atomare Formel p(t1,...,tn) kann zu einem Wahrheitswert ausgewertet werden, indem man die dem Prädikat p zugeordnete Relation für die den Termen $t_1,...,t_n$ zugeordneten Elemente prüft.
  - Aus den Wahrheitswerten atomarer Formeln kann man den Wahrheitswert zusammengesetzter Formeln bestimmen.
- Eine Interpretation, die eine Formel F erfüllt, heißt ein **Modell** für F.
- Ein ausgezeichnetes Universum für die Untersuchung von Eigenschaften logischer Formeln ist das Herbranduniversum, das aus der Menge der Grundterme besteht.

Semantik der Prädikatenlogik 1. Stufe (Forts.)

Sei I = <U,f> eine Interpretation, sei ϕ ein Ausdruck, dann ist die Wahrheit von ϕ in I (symbolisch I |= ϕ) wie folgt definiert:

1. Falls ϕ die Form $p(t_1,...t_n)$ hat, dann gilt I |= ϕ, genau dann wenn $<f(t_1),...f(t_n)> \in f(p)$
2. Falls ϕ die Form $\neg\psi$ hat, dann gilt I |= ϕ, genau dann wenn es nicht der Fall ist daß I |= ψ
3. Falls ϕ die Form χ & ψ hat, dann gilt I |= ϕ, genau dann wenn I |= χ und I |= ψ
4. Falls ϕ die Form χ v ψ hat, dann gilt I |= ϕ, genau dann wenn I |= χ oder I |= ψ oder beides
5. Falls ϕ die Form $\chi \rightarrow \psi$ hat, dann gilt I |= ϕ, genau dann wenn I |≠ χ oder I |= ψ oder beides

Sei a eine Konstante die nicht in ϕ vorkommt.

6. Falls ϕ die Form $\forall x\ \psi$ hat, dann gilt I |= ϕ, genau dann wenn gilt I‘ |= ψ [x/a] für alle Interpretationen I‘, die mit I mit Ausnahme der Abbildung für a übereinstimmen.
7. Falls ϕ die Form $\exists\ x\ \psi$ hat, dann gilt I |= ϕ, genau dann wenn es eine Interpretation I‘ gibt, die mit I mit Ausnahme der Abbildung für a übereinstimmt, und es gilt I‘ |= ψ [x/a]

Logische Konsequenz und Herleitbarkeit in PL1

- Eine Aussage ϕ ist eine logische Folgerung (**gültige Konsequenz)** aus einer Menge von Aussagen S (symbolisch S |= ϕ) genau dann wenn für alle Interpretationen I gilt: falls I |= ψ für alle $\psi \in$ S, dann gilt auch I |= ϕ.
- Eine Aussage ϕ kann aus einer Menge von Aussagen S **hergeleitet** werden (symbolisch S |— ϕ) falls es einen **Beweis** für ϕ aus S gibt.
- Eine Herleitung wird durch **Inferenzregeln** berechnet, die Steuerung zur Auswahl und Anwendung der Inferenzregeln heißt **Inferenzprozedur.**
- Korrektheit und Vollständigkeit von Inferenzprozeduren:
  - **Korrektheit:** Wenn S |— ϕ dann S |= ϕ (d.h. wenn ϕ aus S herleitbar ist, dann ist ϕ eine Konsequenz aus S)
  - **Vollständigkeit:** Wenn S |= ϕ dann S |— ϕ (d.h. wenn ϕ eine Konsequenz aus S ist, dann ist der Beweis auch herleitbar)

  Ziel: Finde eine Inferenzprozedur, die korrekt und vollständig ist

Beispiele für Inferenzregeln der Prädikatenlogik 1. Stufe

Instantiierung: $\dfrac{\forall x \; P}{P\{x/a\}}$

Modus Ponens: $\dfrac{\phi, \; \phi \rightarrow \psi}{\psi}$

Instantiierung und Modus Ponens: $\dfrac{P(a), \forall x \; P(x) \rightarrow Q(x)}{Q(a)}$

Modus Tollens: $\dfrac{\neg \psi, \; \phi \rightarrow \psi}{\neg \phi}$

Die modelltheoretische Semantik der Prädikatenlogik ist unabhängig von operationaler Semantik, so daß unterschiedliche Inferenzregeln anwendbar sind

Problem: Finde eine Inferenzprozedur, die einfach automatisierbar ist.

Eine logik-basierte Inferenzprozedur: Resolution [Robinson, 1965]

- Resolution basiert auf dem Widerlegungsprizip (refutation)
  - Um einen Satz ϕ aus einer Menge von Aussagen S zu beweisen, nehme an, daß $\neg\phi$ gilt und versuche einen Widerspruch herzuleiten.
- Sei S eine Menge von Klauseln (die Theorie) und ϕ eine Anfrage, die man beweisen will, dann gilt:

  S |= ϕ gdw S $\cup$ {$\neg \phi$} widersprüchlich
- Das Resolutionsverfahren besteht aus drei Prozessen:
  - Umwandlung der Aussagen in Klauselform (durch Eliminierung von Implikation und Quantoren)
  - Unifikation: Vereinheitlichung von Formeln durch Substitution von Variablen
  - Resolution: Anwendung der Inferenzregeln

Klauselform für PL1

- Eine Menge S von Formeln der Prädikatenlogik kann transformiert werden in eine Menge von Klauseln
  (Klauseln sind implizit UND-verknüpft)
- Klauseln sind Mengen von Literalen
  (Literale in Klauseln sind implizit ODER-verknüpft)
- Literale sind atomare Formeln oder negierte atomare Formeln
- Alle Variablen in Klauseln sind all-quantifiziert

Beispiel:

Sei S gegeben durch die beiden Formeln

$\neg(\exists x)\ [p(X)]$

$\forall z\ [\exists y)[qQ(z,y) \vee r(z)] \rightarrow p(Z)]$

S ist äquivalent zu:

$(\neg p(x)) \wedge (\neg q(z,y) \vee p(z)) \wedge (\neg r(z1) \vee p(z1))$

geschrieben in Klauselform ergibt:

$\{\neg p(x)\}$

$\{\neg q(z,y), p(z)\}$

$\{\neg r(z1), p(z1)\}$

Umwandlung in Klauselform: Erläuterung

1. Eliminiere $\rightarrow$: $\phi \rightarrow \psi$ wird ersetzt durch $\neg\phi \vee \psi$
   $\phi \leftrightarrow \psi$ wird ersetzt durch $(\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$
2. Schränke den Bereich von $\neg$ ein:
   $\neg(\neg\phi)$ wird ersetzt durch ϕ
   $\neg(\phi \wedge \psi)$ wird ersetzt durch $\neg\phi \vee \neg\psi$ (de Morgan Regel)
   $\neg(\phi \vee \psi)$ wird ersetzt durch $\neg\phi \wedge \neg\psi$ (de Morgan Regel)
   $\neg\forall x\ \phi$ wird ersetzt durch $\exists x\ \neg\phi$
   $\neg\exists x\ \phi$ wird ersetzt durch $\forall x\ \neg\phi$
3. Benenne Variablen um, so daß keine zwei Quantoren die gleiche Variable binden
   Beispiel: $(\forall x\ (p(x,x)) \wedge (\exists x\ q(x))$ wird zu $(\forall x\ (p(x,x)) \wedge (\exists y\ q(y))$
4. Ziehe alle Quantoren nach links unter Beibehaltung der Reihenfolge. Eliminiere Existenzquantoren: Skolemisierung (siehe Extrafolie)
5. Lasse die Allquantoren weg (alle Variablen sind nun implizit allquantifiziert)
6. Wandle die Formel um in eine Konjunktion von Disjunkten (konjunktive Normalform) durch Anwendung der Regel $(\phi \wedge \psi) \vee \chi = (\phi \vee \chi) \wedge (\psi \vee \chi)$
7. Jedes Konjunkt heißt Klausel. Eine Klausel ist eine Disjunktion von Literalen, die man als Menge schreiben kann, Literale sind atomare Formeln oder negierte atomare Formeln
   Beispiel: $P \wedge (Q \vee \neg R)$ wird zu $\{P\},\ \{Q,\ \neg R\}$
8. Benenne die Variablen um, so daß keine zwei Klauseln die gleichen Variablennamen haben

Skolemisierung: Eliminierung von Existenzquantoren

- Ein Existenzquantor besagt, daß es für die durch ihn quantifizierte Variable ein Individuum gibt, der für ihn substitutiert werden kann, so daß die Aussage wahr wird
- Man kann den Existenzquantor eliminieren, indem man das Individuum benennt:
  - Falls der Existenzquantor nicht im Bereich eines Allquantors vorkommt, ersetze die existentiell quantifizierte Variable durch eine neue Konstante (Skolemkonstante)

    $\exists x$ `president(x)` wird zu `president(a)`
  - Steht der Existenzquantor im Bereich eines Allquantors, ersetze die existentiell quantifizierte Variable durch eine Funktion, deren Argumente die Variablen des umgebenden Allquantors sind

    $\forall x \exists y$ `mother(x,y)` wird zu $\forall x$ `mother(x,f(x))`

Beispiel: Motivation für Unifikation

Beispiel:

- Aussage A: ***Martin ist ein Informatiker.***
- Aussage B: ***Jeder Informatiker kann programmieren.***
- Aussage C: ***Martin kann programmieren.***

Kann ich C aus A und B folgern?

In PL1:

```
Informatiker(martin)
∀x (Informatiker(x) → Kann_programmieren(x))
```

In Klauselform:

```
{  { Informatiker(martin)},
   { ¬Informatiker(x), Kann_programmieren(x) } }
```

Unifikation

- Ein Unifikationsalgorithmus bestimmt, ob zwei Literale gleichgemacht werden können.
- Der Unifikationsalgorithmus berechnet eine Substitution. Eine Substitution σ ist eine Liste von Bindungen, d.h. Paare von Variablen und ihren Werten (Terme).
- Wird in einer Subsitution σ eine Variable an eine Konstante oder eine variablenfreien Funktionsausdruck gebunden, dann darf es keine Bindung an einen anderen Wert geben.
- Ist σ eine Subsitution und ϕ eine Formel, dann ist $\phi\sigma$ die Anwendung der Substitution σ auf ϕ. Dabei werden die Variablen in ϕ durch ihren in σ angegebenen Wert ersetzt.
- *Unifikation:* Gegeben zwei Literale oder Terme t_1 und t_2, finde eine Substitution σ, so daß $t_1\sigma = t_2\sigma$
- Eine Substitution σ ist ein Unifikator zweier Formeln ϕ und ψ genau dann wenn $\pi\sigma = \psi\sigma$
- Ein Unifikator σ heißt allgemeinster Unifikator, wenn sich alle anderen Unifikatoren durch Instantiierung aus σ ergeben, d.h. wenn es einen weiteren Unifikator τ gibt, dann gibt es eine Substitution ρ, so daß $\tau = \sigma\rho$

Beispiele für Unifikatoren

ϕ	ψ	Unifikator σ
`p(x,x)`	`p(a,a)`	`{x/a}`
`p(x,x)`	`p(a,b)`	fail (keine Unifikation möglich)
`p(x,y)`	`p(a,b)`	`{x/a, y/b}`
`p(x,y)`	`p(a,a)`	`{x/a, y/a}`
`p(f(x),b)`	`p(f(c),z)`	`{x/c, z/b}`
`p(x,f(x))`	`p(y,z)`	`{x/y, z/f(y)}`
`p(x,f(x))`	`p(y,y)`	fail

Andreas Abecker (Folie nach Hinkelmann)
Mosbach, Sommersemester 2000
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Hier gehört ein Unifikationsalgorithmus her!

Inferenzregel der Resolution in der PL1

Sei S eine Menge von Klauseln;

Seien $R = \{\phi_1, ..., \phi_i, ..., \phi_n\}$ und $Q = \{\psi_1, ..., \neg\psi_j, ..., \psi_m\}$ Klauseln in S;

Seien ϕ_i und ψ_j unifizierbar mit Unifikator σ;

$resolve(R,Q) = \{\phi_1\sigma, ..., \phi_{i-1}\sigma, \phi_{i+1}\sigma, ..., \phi_n\sigma, \psi_1\sigma, ... \psi_{j-1}\sigma, \psi_{j+1}\sigma, ..., \psi_m\sigma\}$
ist eine neue Klausel
(resolve(R,Q) heißt eine Resolvente von R und Q)

Füge resolve(R,Q) zu S hinzu

Beachte:

- Es wird mit einem positiven und einem negativen Literal resolviert
- Die Resolvente enthät die Literale der beiden Ausgangsklauseln ohne die Literale, mit denen resolviert wurde
- Der Unifikator wird auf alle Literale der Resolvente angewandt

Beweis durch Resolution

- Sei S eine Menge von Klauseln (die Theorie) und ϕ eine Anfrage, die man beweisen will, dann gilt:

  $S \vdash \phi$ gdw $S \cup \{\neg \phi\}$ widersprüchlich gdw $S \cup \{\neg \phi\} \vdash Ł$

  Ł ist die leere Klausel, die als falsch (false, Widerspruch) interpretiert wird

- Um zu beweisen, daß eine Formel ϕ aus einer Menge S von Formeln folgt, gehe wie folgt vor:
  - Füge die Negation von ϕ zu der Menge S hinzu: S' := $S \cup \{\neg\phi\}$
  - Wandle S' um in Klauselform: SK := convert(S')
  - Wende die Inferenzregel der Resolution so lange auf SK an, bis die leere Klausel hergeleitet wird
- Resolution ist korrekt und vollständig

Problem der Steuerung bei der Resolution

- 4 Entscheidungspunkte beim Resolutionsbeweis:
  - Auswahl der beiden Klauseln für die Resolution
  - Auswahl der Literale der ausgewählten Klauseln für die Unifikation
- Jede noch so gute Steuerung kann Entscheidungen treffen, die nicht zum Ziel führen.
- Ein Weg, das Problem der Steuerung zu behandeln, ist
  - die Wahl einer weniger ausdrucksmächtigen Sprache
  - die Festlegung einer Strategie, die Entscheidungsfreiheiten reduziert

Einschränkung der Syntax: Hornklauseln

- **Hornklauseln** sind Klauseln mit höchstens einem positiven Literal
- Es gibt drei Arten von Hornklauseln:
  - Regel: $P \leftarrow P_1 \& P_2 \& ... \& P_n$ ist äquivalent zu $\{P, \neg P_1, \neg P_2, ..., \neg P_n\}$
  - Fakt: P ist äquivalent zu $\{P\}$
  - Anfrage: $\leftarrow P_1 \& P_2 \& ... \& P_n$ ist äquivalent zu $\{\neg P_1, \neg P_2, ..., \neg P_n\}$

  (Das positive Literal einer Hornklausel bezeichnet man auch als *Konklusion* ; die negativen Literale heißen auch *Prämissen,* Hornklauseln werden so notiert, daß das positive Literal vorne steht)
- Eine Hornklausel-Wissensbasis besteht aus einer Menge von Regeln und Fakten
- Für Hornklausel-Wissensbasen gilt die *Closed-World Assumption* (CWA), d.h. es wird nur das als wahr angenommen, was in der Wissensbasis steht

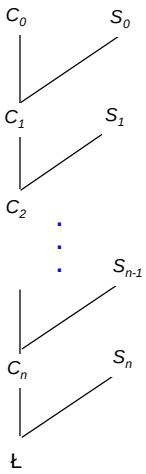
Beispiel für eine Hornklausel-Wissensbasis

```
man(X) ← human(X) & male(X)

woman(X) ← human(X) & female(X)

parent(X,Y) ← mother(X,Y)
parent(X,Y) ← father(X,Y)

ancestor(X,Y) ← parent(X,Y)
ancestor(X,Y) ← parent(X,Z) &
   ancestor(Z,Y)

human(john)
human(paul)
human(mary)

male(john)
male(paul)

female(mary)

father(john,mary)

mother(mary,paul)
```

Beispiele für Anfragen:

```
← human(john)
← human(X)
← man(john)
← man(X)
← parent(mary,X)
← ancestor(X,Y) & male(X)
```

Entscheidungspunkte bei OLD-Resolution und Hornklauseln



C_i: Center Clauses (Zentrumsklauseln)
S_i: Side Clauses (Seitenklauseln)

- 4 Entscheidungspunkte bei Resolution:
  - Welche Klauseln wählt man für die Resolution?
  - Welche beiden Literale innerhalb der Klauseln werden unifiziert?
- OLD-Resolution
  - starte mit der Anfrage als einer Klausel
  - Lineare Resolution: nimm die Resolvente als eine Klausel im nächsten Resolutionsschritt
  - Ordered Linear Resolution: Literale einer Klausel sind geordnet, resolviere mit dem ersten Literal
- Es bleibt nur noch eine Entscheidungsfreiheit: die Wahl der Seitenklausel
  - C_i: C_0 = Anfrage / C_i = Resolvente α
  - Literal in C_i: erstes Literal α
  - S_i: Klausel aus Wissensbasis ?
  - Literal in S_i: erstes Literal (Kopf) α

Rückwärtsverkettung: Beantwortung von Fragen

- Die OLD-Resolution nennt man auch Rückwärtsverkettung, da die Regeln rückwärts angewendet werden:
  Eine Regel ist anwendbar, wenn die *Konklusion* der Regel mit der Anfrage unifiziert werden kann:
  - Variablen werden mit Konstanten belegt
  - eine Variable muß bei jedem Vorkommen den gleichen Wert haben
- Rückwärtsverkettung beantwortet Anfragen an eine Regelbasis: Enthält die Anfrage Variablen, so werden gültige Werte der Variablen berechnet
- Durch Anwendung der Inferenzregel werden neue Anfragen generiert (die Bedingungen der Regel werden zur Anfrage hinzugefügt)
- Ein Literal der Anfrage wird beantwortet (bewiesen), wenn ein Fakt in der Datenbasis existiert, der mit der Anfrage unifiziert werden kann
- Falls für ein Literal der Anfrage keine anwendbare Regel oder Fakt gefunden wird, ist das Literal nicht beweisbar (vgl. Closed-World Assumption)
  ➠ Backtracking (siehe unten)
- Die Rückwärtsverkettung ist abgeschlossen, wenn die Liste der Anfragen leer ist. Das Ergebnis ist die Menge der Variablen mit ihren Werten (die Substitution).

Beispiele für Anfragen an eine Hornklausel-Wissensbasis

```
man(X) ← human(X) & male(X)

woman(X) ← human(X) & female(X)

parent(X,Y) ← mother(X,Y)
parent(X,Y) ← father(X,Y)

ancestor(X,Y) ← parent(X,Y)
ancestor(X,Y) ← parent(X,Z) &
   ancestor(Z,Y)

human(john)
human(paul)
human(mary)

male(john)
male(paul)

female(mary)

father(john,mary)

mother(mary,paul)
```

Beispiele für Anfragen:

← `human(john)`
Resultat: `yes`

← `female(john)`
Resultat: `no`

← `female(peter)`
Resultat: `no`

← `human(X)`
Resultat: `X = john` oder
`X = paul` oder
`X = mary`

← `man(john)`
Resultat: `yes`

← `man(X)`
Resultat: `X = john` oder
`X = paul`

← `parent(mary,X)`
Resultat: `X = paul`

← `ancestor(X,Y) & male(X)`
Resultat: `X = john, Y = mary` oder
`X = john, Y = paul`

Algorithmus für die Rückwärtsverkettung: Prinzip

- Suchstrategie: Tiefensuche
- Die Regeln werden in der Reihenfolge ihres Auftretens in der Wissensbasis auf Anwendbarkeit getestet
- Falls eine Entscheidung nicht zum Erfolg führt, muß man eine alternative Klausel wählen:
  - ➠ Backtracking
- Für die Wahl der Alternative muß man zu dem Zustand des letzten Entscheidungspunkts zurücksetzen
- Die Entscheidungspunkte sind definiert durch:
  - die aktuelle Anfrage L, die mit der Konklusion der Regel unifiziert hat,
  - die Liste der restlichen Literale der Anfrageliste,
  - die aktuelle Substitution (Variablenbelegungen),
  - die Klauseln, die für die aktuelle Anfrage noch nicht angewendet wurden
- Der Algorithmus für Rückwärtsverkettung benötigt zwei Stacks:
  - GOALLIST: Liste der aktuellen Literale der Anfrage
  - CHOICEPOINTS: Entscheidungspunkte für Backtracking

Algorithmus für die Rückwärtsverkettung

Sei P eine Wissensbasis bestehend als Regeln und Fakten, sei G eine Anfrage,
dann berechnet der folgende Algorithmus eine Variablenbelegung σ, falls G aus P herleitbar ist

1. Initialisierung: *GOALLIST* := G; *CHOICEPOINTS* := Ø, σ := Ø;
2. falls *GOALLIST* = Ø
   dann stop, *SUCCESS* = (true, σ);
3. L := top(*GOALLIST*), *GOALLIST* := pop(*GOALLIST*), *CLAUSES* := P;
4. falls *CLAUSES* = Ø
   dann falls *CHOICEPOINTS* := Ø
   dann *SUCCESS* = (false, Ø)
   sonst (L, *GOALLIST* , σ, *CLAUSES*) := top(*CHOICEPOINTS*);
5. *S*:= top(*CLAUSES*), *CLAUSES* := pop(*CLAUSES*), H := Konklusion(*S*), L' := Lσ, H' := Hσ;
6. falls L'τ = H'τ für eine Substitution τ
   dann σ := στ,
   GOALLIST := Prämissen(S) ∪ *GOALLIST* ,
   STATE := (L, *GOALLIST* , σ, *CLAUSES*),
   CHOICEPOINTS := push(*STATE*,*CHOICEPOINTS*),
   gehe zu 2
   sonst gehe zu 4;

Problem: Eingeschränkte Aussagemächtigkeit von Hornklauseln

- Mit Hornklauseln kann keine Disjunktion (ODER) und Negation (NICHT) repräsentiert werden
  - Die Hauptstadt von Australien ist Sydney oder Canberra
  - Die Hauptstadt von Australien ist nicht Sydney
- Hornklauseln können negierte Prämissen haben. Sie heißen dann *normale Klauseln* und haben die Form:

  $p_0(X_0) \leftarrow p_1(X_1), p_2(X_2), ..., p_n(X_n), not(p_{n+1}(X_{n+1})), ..., not(p_{n+m}(X_{n+m}))$
- Für Hornklauseln mit Negation gibt es verschiedene Interpretationen, die bekannteste ist die *Negation-as-Failure* Regel (NAF), die die CWA voraussetzt. Umgangssprachlich bedeutet die NAF: "not(G) folgt aus einer Wissensbasis WB, wenn der Beweis von G in *endlicher Zeit* scheitert."
  (not steht für „nicht beweisbar")
- Beispiel:

  Wissensbasis:
  ```
  loves(john,X) ← girl(X), not(likes(X,wine))
  girl(mary)
  girl(susan)
  likes(susan,wine)
  ```
  Anfrage:
  ```
  ← loves(john,Y)
  ```

Von der Hornlogik zu PROLOG

- Auf der Hornklausellogik basieren die logische Programmierung (insbesondere die Programmiersprache Prolog) und deduktive Datenbanken
- Prolog ...
  - ... verwendet eine andere Syntax, z.B.
    - Variable beginnen mit Großbuchstaben
    - & wird zu ,
    - ← wird zu :-
    - jede Klausel wird mit einem . beendet
  - ... stellt Funktionen und Prozeduren zur Verfügung, z.B.
    - Arithmetik
    - Prozeduren zum Drucken, Lesen
  - ... bietet die Möglichkeit, die Bearbeitung zu beeinflussen, um prozedurale Konstrukte wie Schleifen zu implementieren

Von Vögeln und Philosophen (Lösungssuche mit Backtracking)

Fakten:

```
bird(tweety).
cat(garfield).
greek(sokrates).
```

Regeln:

```
human(Being) :- greek(Being).
animal(Beast) :- bird(Beast).
mortal(Creature) :- human(Creature).
mortal(Creature) :- animal(Creature).
```

Fragen:

```
? :- mortal(garfield).
? :- human(sokrates).
? :- animal(sokrates).
? :- mortal(tweety).
```

Von Katzen und Mäusen (inkrementelle Erweiterung der Wissensbasis)

Fakten:

```
bird(tweety).
cat(garfield).
greek(sokrates).
mouse(jerry).
```

Regeln:

```
human(Being) :- greek(Being).
animal(Beast) :- bird(Beast).
mortal(Creature) :- human(Creature).
mortal(Creature) :- animal(Creature).
```

Frage:

```
? :- mortal(jerry).
```

Was muß ich in die Wissensbasis einfügen, damit die Frage positiv beantwortet wird?

Welche **Begriffshierarchie** versteckt sich hinter den angegebenen Regeln?

ABER: PROLOG ist gerade **nicht** primär / ausschließlich zur Darstellung von taxonomischem Wissen gedacht!

Darstellung relationalen Wissens mit PROLOG

Fakten:

```
maennlich(bernd).       weiblich(moni).          ehepaar(bernd,moni).
maennlich(willi).       weiblich(anita).         ehepaar(willi,anita).
maennlich(richard).     weiblich(gertrud).       mutter(anita,bernd).
maennlich(werner).      weiblich(ida).           mutter(gertrud,anita).
```

Regeln:

```
vater(Vater,Kind) :-     ehepaar(Vater,Mutter),
                         mutter(Mutter,Kind).

geschwister(K1,K2) :-    mutter(Mutter,K1),
                         mutter(Mutter,K2).

bruder(B1,B2) :-   geschwister(B1,B2),
                   maennlich(B1).

onkel(Onkel,Person)  :-  bruder(Onkel,Mutter),
                         mutter(Mutter,Person).
```

Fortsetzung Familienbeispiel (relationales Wissen in Regeln)

- **Aufgabe: definiere die restlichen drei Onkel-Fälle!**
- Hier werden neue Prädikate (in der Welt: Klassen, Objektmengen, Begriffe) aus den Beziehungen anderer Objekte zueinander definiert.
- Eine Praxisanwendung: Erweiterung der Ausdrucksmöglichkeiten relationaler Datenbanken (Extensionale Relationen in Tabellen plus intensionale Relationen in Regeln).
  - Die vorab beschriebenen Relationen hätte man auch als Ausdrücke der relationalen Algebra mit SQL beschreiben können.
  - Interessanter wird es bei Benutzung rekursiver Regeln!
  - Die Logik bietet eine einheitliche Bescheibung von relationalen Datenbanken und Inferenzregeln.
- **Aufgabe: Stückliste**

Fortsetzung Familienbeispiel (Ungerichtetheit von Regeln)

- PROLOG-Prädikate sind **ungerichtete Relationen**.
- Sie beschreiben zuerst einmal nicht, wie man etwas ausrechnet, sondern **wie etwas definiert ist**. Wie bei einer mathematischen Gleichung kann man verschiedene fehlende Stücke berechnen, je nachdem, was gegeben ist.
- Im konkreten Anwendungsfall ergibt sich die Benutzung eines Prädikats aus dem Verlauf der Resolution und der aktuellen Bindungssituation dabei.
- Im Beispiel:
  - **? :- onkel(charly,willy)** **testet, ob Charly ein Onkel von Willy ist.**
  - **? :- onkel(charly,X)** **liefert als Belegungen von X alle Neffen von Charly.**
- Was bedeutet:
  - **? :- onkel(X,willy)**
  - **? :- onkel(X,Y)**

Zur Implementierungsunvollständigkeit von PROLOG

Beispiel:

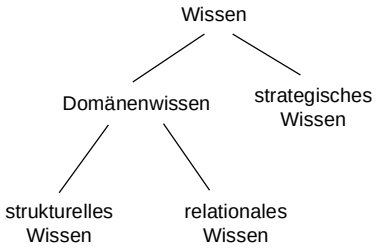
```
p(X) :- q(X).
q(X) :- p(X).
q(a).
```

Frage:

```
?:- p(Z).
```

- In der Hornlogik gilt: wenn eine Frage logisch aus der Datenbasis beantwortbar ist, dann liefert die Resolution eine Antwort.
- In PROLOG gilt dies nicht (Implementierungs-unvollständigkeit durch die Tiefensuche)!
- Abhilfen z.B. Zykelerkennung oder Tabulierung

Wissensarten: Unterscheidung zwischen Domänenwissen und strategischem Wissen



Domänenwissen: Wissen über das Anwendungsgebiet
- **strukturelles Wissen**: Entitäten des Anwendungsgebiets und strukturelle Beziehungen zwischen ihnen. Die wichtigsten strukturellen Beziehungen sind:
  - *Klassifikation* (instance-of): Objekt A ist vom Typ bzw. der Klasse K_A
  - *Subsumtion* (isa): Klasse K_A ist allgemeiner als Klasse K_B
  - *Aggregation* (part-of): Objekt A besteht aus den Teilen A_1, A_2, ...
- **relationales Wissen**: nicht-strukturelle Beziehungen und Eigenschaften

Strategisches Wissen: Wissen darüber, wie man Anwendungswissen einsetzt, um ein Problem zu lösen

Für die verschiedene Typen von Wissen gibt es unterschiedlich geeignete Repräsentationsformalismen

Wissensrepräsentation durch Logik: Diskussion

- Die Logik ist eine sehr allgemeine Sprache zur Repräsentation von Wissen
- Auf der epistemologischen Ebene stehen nur Prädikate, Konstanten und Funktionen für die Wissensrepräsentaiton zur Verfügung
- Klassen, strukturelle Beziehungen (is-a, instance-of, part-of) und Eigenschaften müssen durch Prädikate repräsentiert werden (kognitiv adäquat?)
- Die Ausdrucksmächtigkeit der Sprache und die Allgemeinheit der Inferenzen erfordert Einschränkungen der Inferenzstrategien und der Ausdrucksmächtigkeit, um effiziente Verarbeitung zu ermöglichen