Künstliche Intelligenz

Prof. Dr. Dirk Krechel
Hochschule RheinMain

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim Geisenheim

Inhalte

- Einführung
- **Symbolische Verfahren, Logik**
  - Aussagenlogik, **Prädikatenlogik**
  - Horn Logik, Prolog
- Suchen und Bewerten
  - Problemlösen durch Suche
    - Uninformierte Suche
    - Heuristische Suche
    - Spielbäume

Skolem Normalform

- Eine prädikatenlogische Formel ist in *Skolem Normalform*, wenn
  - sie in Pränex Normalform ist, also in der Form $Q_1x_1\ Q_2x_2 \ldots Q_nx_n\ \alpha$ und
  - die Q_i alle Allquantoren sind ($Q_i = \forall$)
- *Skolemisierung*: Wir können für jede Formel in PNF die Skolem Normalform erzeugen
  - Entferne den am weitesten links stehenden Existenzquantor $\exists x$
  - Ersetze in der Matrix die Variable x durch den Term $f(y_1, \ldots, y_n)$ wobei
    f ein neues Funktionssymbol ist und
    $y_1, \ldots, y_n$ alle Variablen der Allquantoren links des entfernten Existenzquantors sind
- Das Ergebnis ist in Skolem Normalform
  Die neuen Funktionen heißen *Skolemfunktionen*
  - Idee: Der Zeuge der Existenz des Objekts in einem Universum kann nur davon abhängen welche Objekte den Variablen in den Allquantoren links des entsprechenden Existenzquantors vorher zugewiesen wurde. Die Soklemfunktion „berechnet“ eines dieser Objekte

Skolem Normalform – Beispiele

- Formel $\forall x \, \exists u \, P(x, y) \vee P(a, z) \vee \neg Q(u, z)$
  Skolemisierung mit $u = f(x)$
  $\forall x \, P(x, y) \vee P(a, z) \vee \neg Q(f(x), z)$

- Formel $\forall x \, \forall y \, \exists z \, P(x, f(y)) \vee P(a, z)$
  Skolemisierung mit $z = g(x, y)$
  $\forall x \, \forall y \, P(x, f(y)) \vee P(a, g(x, y))$

- Formel $\forall x \, \forall y \, \exists z \, \forall u \, \exists v \, P(x, z) \wedge P(y, v)$
  Skolemisierung mit $z = f(x, y)$, $v = g(x, y, u)$
  $\forall x \, \forall y \, \forall u \, P(x, f(x,y)) \wedge P(y, g(x, y, u))$

Skolemisierung für Entscheidungsverfahren

- Satz:
  - Eine Formel ist inkonsistent gdw die Skolem Normalform ist inkonsistent
  - Eine Formel ist erfüllbar gdw ihre Skolem Normalform ist erfüllbar
- Praktische Aufgabenstellung:
  Zeige, dass

  $$\Sigma = \{\alpha_1, \ldots, \alpha_n\} \models \beta$$

- Umsetzung (wie in der Aussagenlogik):
  Zeige, dass

  $\Sigma = \{\alpha_1, \ldots, \alpha_n\} \cup \{\neg\beta\}$ inkonsistent

  was äquivalent ist
- Es gilt, dass $\{\alpha_1, \ldots, \alpha_n\} \cup \{\neg\beta\}$ inkonsistent gdw

  $\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta$ inkonsistent gdw

  Soklem Normalform von $\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta$ inkonsistent

Klauselnormalform

- Eine Klausel ist eine endliche Menge von Literalen $\{L_1, \ldots, L_n\}$, die als Disjunktion dieser Literale interpretiert wird
  - Mehrfachvorkommen desselben Literals in der Disjunktion sind ausgeschlossen
- Die leere Klausel wird immer auf false abgebildet und mit $\square$ bezeichnet
  - Die leere Klausel ist der Widerspruch
- Eine Menge von Klauseln wird interpretiert als Konjunktion dieser Klauseln
  - Mehrfachvorkommen derselben Klausel in der Konjunktion sind ausgeschlossen
- Jede Formel in Skolem Normalform mit einer Matrix in CNF kann direkt auf eine endliche Menge von Klauseln abgebildet werden
  - Wir nehmen dabei immer an, dass keine freien Variablen vorkommen, bzw. alle (freien) Variablen sind allquantifiziert
  - Die endliche Menge von Klauseln ist die *Klauselnormalform*

Klauselnormalform – Beispiele

- Formel $\forall x \, \exists u \, P(x, y) \vee P(a, z) \vee \neg Q(u, z)$
  Skolemisierung mit $u = f(x)$
  $\forall x \, P(x, y) \vee P(a, z) \vee \neg Q(f(x), z)$
  Klauselnormalform
  $\{ \{P(x, y), P(a, z), \neg Q(f(x), z)\} \}$
- Formel $\forall x \, \forall y \, \exists z \, P(x, f(y)) \vee P(a, z)$
  Skolemisierung mit $z = g(x, y)$
  $\forall x \, \forall y \, \exists z \, P(x, f(y)) \vee P(a, g(x, y))$
  Klauselnormalform
  $\{ \{P(x, f(y)), P(a, g(x, y))\} \}$
- Formel $\forall x \, \forall y \, \exists z \, \forall u \, \exists v \, P(x, z) \wedge P(y, v)$
  Skolemisierung mit $z = f(x, y)$, $v = g(x, y, u)$
  $\forall x \, \forall y \, \forall u \, P(x, f(x,y)) \wedge P(y, g(x, y, u))$
  Klauselnormalform
  $\{ \{P(x, f(x,y))\}, \{P(y, g(x, y, u))\} \}$

Entscheidung in Klauselnormalform – Beispiel

- Sei $\Sigma = \{ \forall x\, (\neg P(x) \vee \exists y\, \neg Q(y)), P(a) \}$
  Gilt $\Sigma \models \exists x\, \neg Q(x)$?
- Gilt gdw
  $\forall x\, (\neg P(x) \vee \exists y\, \neg Q(y)) \wedge P(a) \wedge \neg \exists x\, \neg Q(x)$ inkonsistent
- Transformation in PNF
  $\forall x\, (\neg P(x) \vee \exists y\, \neg Q(y)) \wedge P(a) \wedge \forall x\, Q(x)$
  $\forall x\, (\neg P(x) \vee \exists y\, \neg Q(y)) \wedge P(a) \wedge \forall z\, Q(z)$
  $\forall x \exists y\, (\neg P(x) \vee \neg Q(y)) \wedge P(a) \wedge \forall z\, Q(z)$
  $\forall x \exists y \forall z\, ((\neg P(x) \vee \neg Q(y)) \wedge P(a) \wedge Q(z))$
- Skolemisierung
  $\forall x \forall z\, ((\neg P(x) \vee \neg Q(f(x))) \wedge P(a) \wedge Q(z))$
- Klauselnormalform: $\{ \{\neg P(x), \neg Q(f(x))\}, \{P(a)\}, \{Q(z)\} \}$
- Klauselnormalform ist inkonsistent, also gilt $\Sigma \models \exists x\, \neg Q(x)$
  - Q(z) gilt immer, also darf P(x) für kein x gelten, aber P(a) gilt

Inferenzkalkül

- Ziel:
  - Prädikatenlogische Formeln direkt syntaktisch manipulieren
  - Erzeugen von *korrekten* prädikatenlogische Formeln
- *Inferenzkalkül*
  - Vorschriften oder *Inferenzregeln*
  - Aus gegebenen prädikatenlogischen Formeln
    neue prädikatenlogische Formen generieren
- Definition (wie Aussagenlogik): Eine Formel β *folgt syntaktisch* aus einer Formelmenge $\Sigma=\{\alpha_1,\ldots,\alpha_n\}$ und Inferenzregeln IR gdw
  - Es gibt eine Folge von $\Sigma=\Sigma_0, \Sigma_1, \Sigma_2, \ldots$ mit β aus einem Σ_i
  - $\Sigma_{i+1} = \Sigma_i \cup \{\gamma_i\}$; und γ_i und entsteht aus Anwendung einer Regel in IR auf Σ_i
  - Wir schreiben: $\Sigma_i \vdash \gamma_i$, $\Sigma \vdash^* \beta$ oder kurz $\Sigma \vdash \beta$ und $\Sigma_i \vdash \Sigma_j$ für $i \leq j$;
    um explizit auf IR hinzuweisen schreibt man auch $\vdash_{IR}$ statt $\vdash$

Korrektheit und Vollständigkeit

- Ziel
  - Ein Kalkül soll *vollständig* sein:
    Alles was (semantisch) korrekt ist soll (syntaktisch) herleitbar sein
  - Ein Kalkül soll *korrekt* sein:
    Alles was (syntaktisch) hergleitet werden kann soll (semantisch) korrekt sein
- Korrektheit und Vollständigkeit: $\vdash \; = \; \models$
  - Korrektheit: Für alle Σ, β gilt: Falls $\Sigma \vdash \beta$ gilt, dann gilt $\Sigma \models \beta$
  - Vollständigkeit: Für alle Σ, β gilt: Falls $\Sigma \models \beta$ gilt, dann gilt $\Sigma \vdash \beta$
- Satz von Gödel: Es gibt einen korrekten und vollständigen
  Kalkül für die Prädikatenlogik erster Stufe (PL1)
- Aber: PL1 ist unentscheidbar
  - Es gibt kein Verfahren, das für ein gegebene Formel entscheidet ob die Formel eine Tautologie ist oder nicht
  - Für ein vollständiges korrektes Kalkül gibt es
    keine maximale Länge der Ableitungsfolge

Inferenzregeln

- Notation wie in der Aussagenlogik
  - Prämisse: Ein Muster, dem bestehende Formeln aus E entsprechen
  - Konklusion: Die neue hergeleitete Formel

$$\frac{\text{Prämisse}}{\text{Konklusion}}$$

- Beispiele
  - Modus Ponens

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta}$$

  - Instanziierung

$$\frac{\forall x\ \alpha}{\alpha\ [x/t]}$$

  - Existenzielle Aussage

$$\frac{\alpha\ [x/t]}{\exists x\ \alpha}$$

- Definition: Beweise (unter IR)
  - Ein *Beweis* von α aus Σ ist eine Folge von Formeln α_i (i=1, ...,n), so dass
  - α_i in Σ oder α_i ist Konklusion aus Anwendung einer Inferenzregel mit $\{ \alpha_j \mid 0 < j < i \} \cup \Sigma$ als Quelle für die Prämisse
  - $\alpha_n = \alpha$

Beweis – Beispiel

- Sei $\Sigma = \{ \exists x\, P(x) \Rightarrow \forall x\, Q(x), P(a) \}$
  Gilt $\Sigma \vdash Q(a)$?
- Beweis unter Verwendung der Inferenzregeln
  - $\alpha_1 = \exists x\, P(x) \Rightarrow \forall x\, Q(x)$ (aus Σ)
  - $\alpha_2 = P(a)$ (aus Σ)
  - $\alpha_3 = \exists x\, P(x)$ (Existenzielle Aussage mit α_1 und α_2)
  - $\alpha_4 = \forall x\, Q(x)$ (Modus Pones mit α_1 und α_3)
  - $\alpha_5 = Q(a)$ (Instantziierung mit α_4)

Resolution für die Prädikatenlogik

- Satz: Die Resolution ist korrekt und vollständig in PL1
- Resolutionsbeweis für $\Sigma \models \beta$
  Wir zeigen Inkonsistenz von $\Sigma \cup \{\neg \beta\}$

- Konvertiere $\Sigma \cup \{\neg \beta\}$ in Klauselnormalform
- Verwende die folgenden beiden Inferenzregeln solange wie möglich
  - Resolutionsregel für die Prädikatenlogik
  - Faktorisierungsregel für die Prädikatenlogik
- Wenn die leere Klausel abgeleitet werden kann, dann ist die Klauselmenge inkonsistent und folglich gilt $\Sigma \models \beta$

Resolutionsregel für PL1

- Resolutions-regel

$$\frac{\{L_1, L_2, \ldots, L_n, A\} \quad \{\neg A', K_1, K_2, \ldots, K_m\}}{\{\sigma(L_1), \sigma(L_2), \ldots, \sigma(L_n), \sigma(K_1), \sigma(K_2), \ldots, \sigma(K_m)\}}$$

- Annahme: Die Klauseln $\{L_1, \ldots, L_n, A\}$ und $\{\neg A', K_1, \ldots, K_m\}$ haben disjunkte Variablenmengen, gegebenenfalls können Variablen vorher umbenannt werden
- Die L_i und die K_j sind beliebige (positive oder negative) Literale
- A und A' sind positive Literale
- A und A' sind unifizierbar und mgu(A, A') = σ

- Die Konklusion heißt auch *Resolvente*
- Resolutionsregel nur anwendbar wenn A und A' unifzierbar
- Ähnlich zur Aussagenlogik, plus Unifikation und Substitution

Faktorisierungsregel für PL1

- Faktorisierungs-regel

$$\frac{\{L_1, L_2, \ldots, L_n\}}{\{\sigma(L_1), \sigma(L_2), \ldots, \sigma(L_n)\}}$$

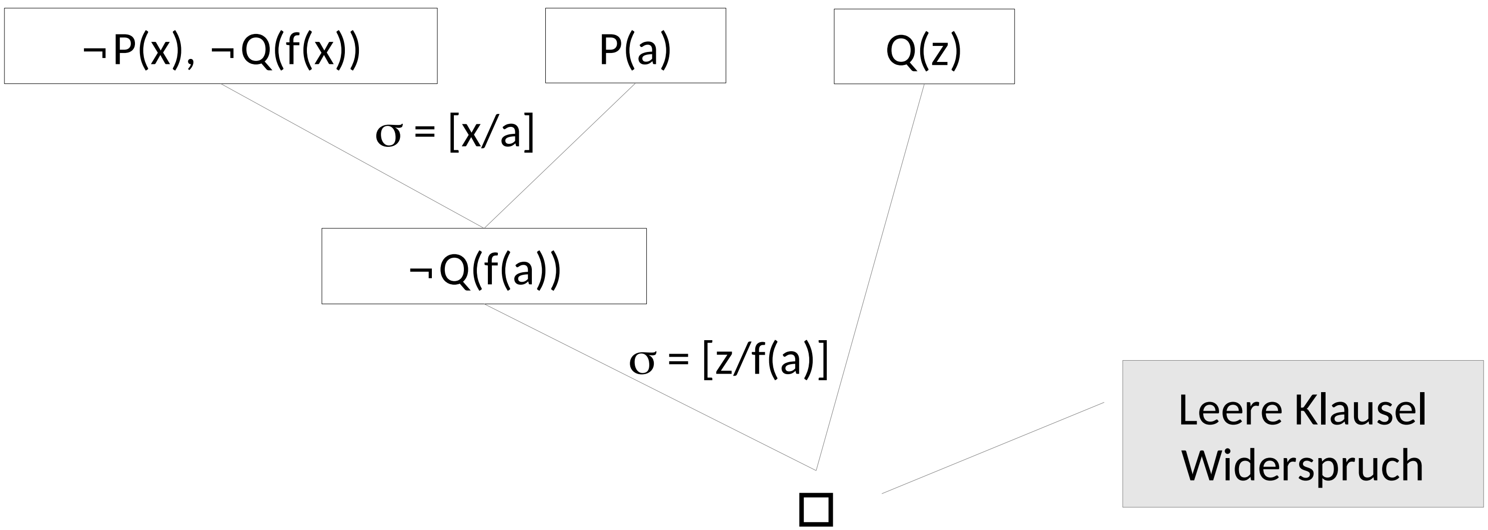
- Die L_i und sind beliebige (positive oder negative) Literale
- Es gibt zwei unifizierbare Literale L_i und L_j mit $i \neq j$
- $mgu(L_i, L_j) = \sigma$

- Durch Anwendung der Faktorisierungsregel wird die resultierende Klausel kürzer als die Ausgangsklausel

- Keine Entsprechung in der Aussagenlogik

Resolution – Beispiel

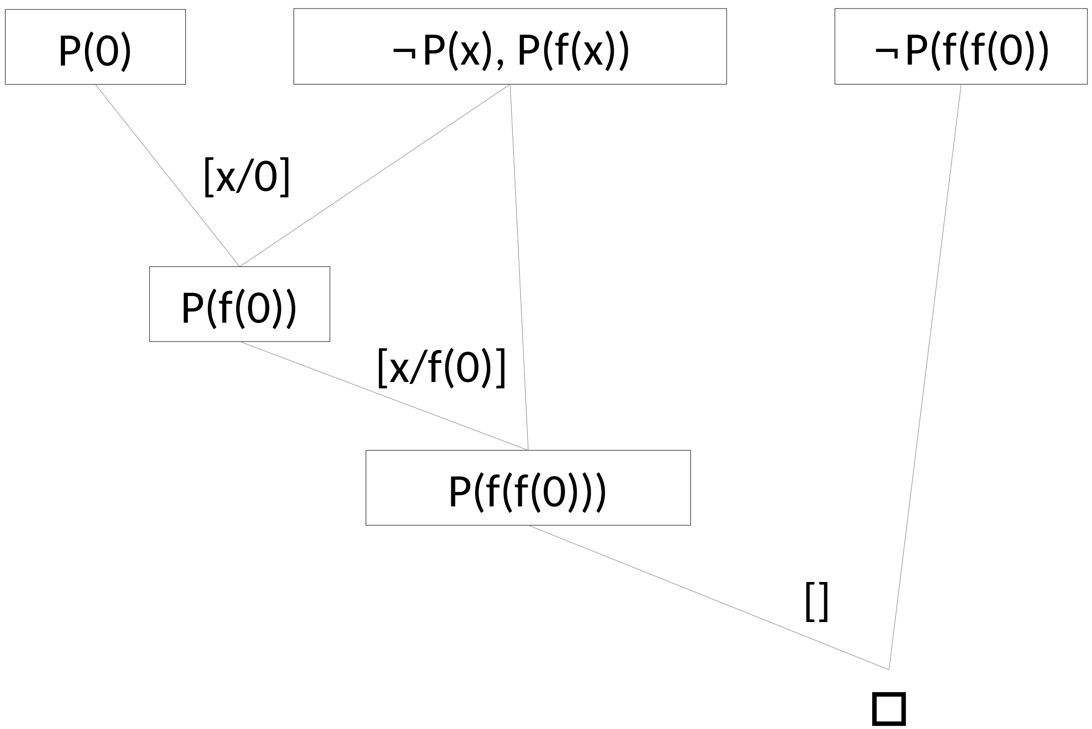
- Sei Σ = { ∀x (¬P(x) v ∃y ¬Q(y)), P(a) }
  Gilt Σ ├ ∃x ¬ Q(x) ?
- Klauselnormalform wie vorher
  { {¬P(x), ¬Q(f(x))}, {P(a)}, {Q(z)} }
- Resolutionsbeweis



- [Im Beispiel nur Resolutionsregel ausreichend]

Resolution – Beispiel

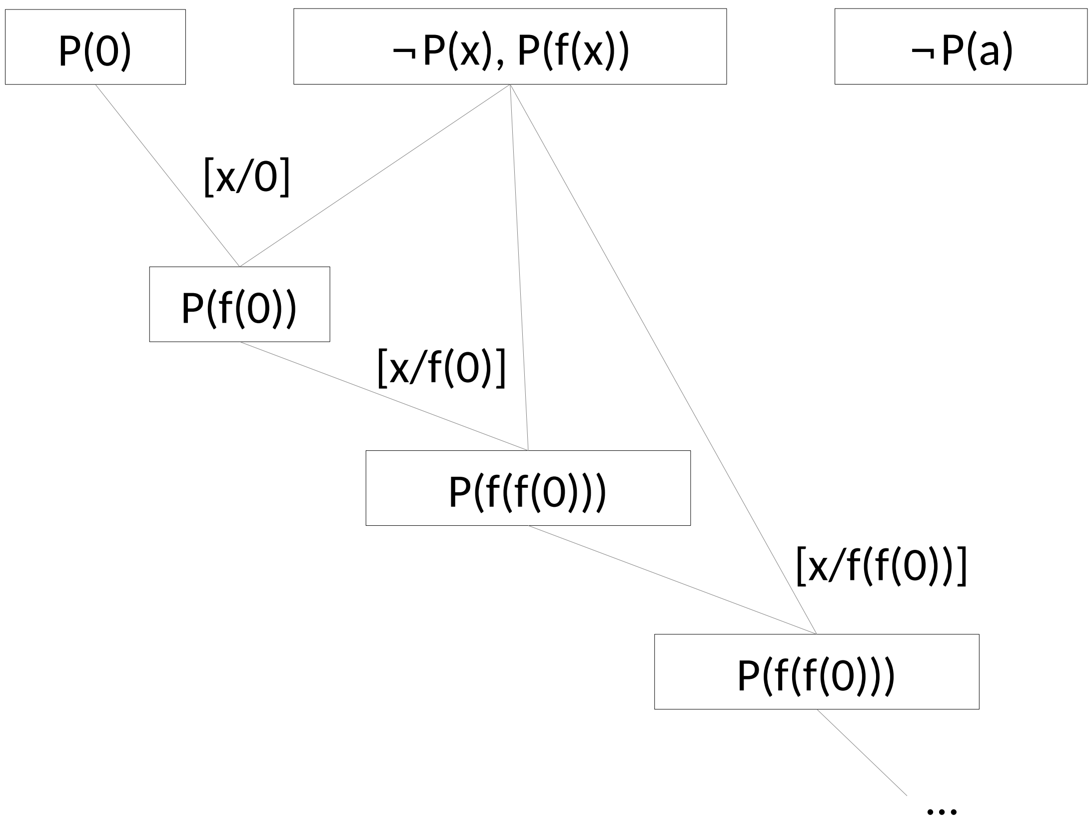
- $\{ P(0), \forall x\, (P(x) \Rightarrow P(f(x))) \} \vdash P(f(f(0)))$



- [In diesem Beispiel auch nur Resolutionsregel, aber eine Klausel wurde mehrfach angewendet]

Resolution – Beispiel

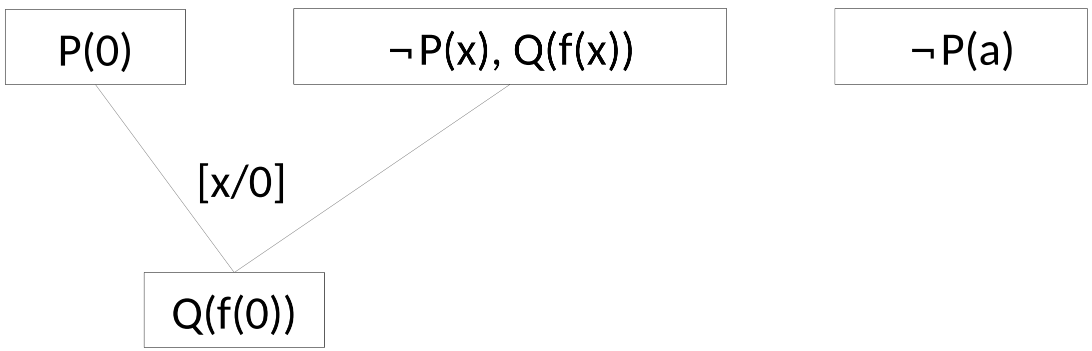
- $\{ P(0), \forall x\, (P(x) \Rightarrow P(f(x))) \} \vdash P(a)$



- [Unendliche Ableitung. P(a) folgt nicht, aber Resolution findet das nicht heraus]

Resolution – Beispiel

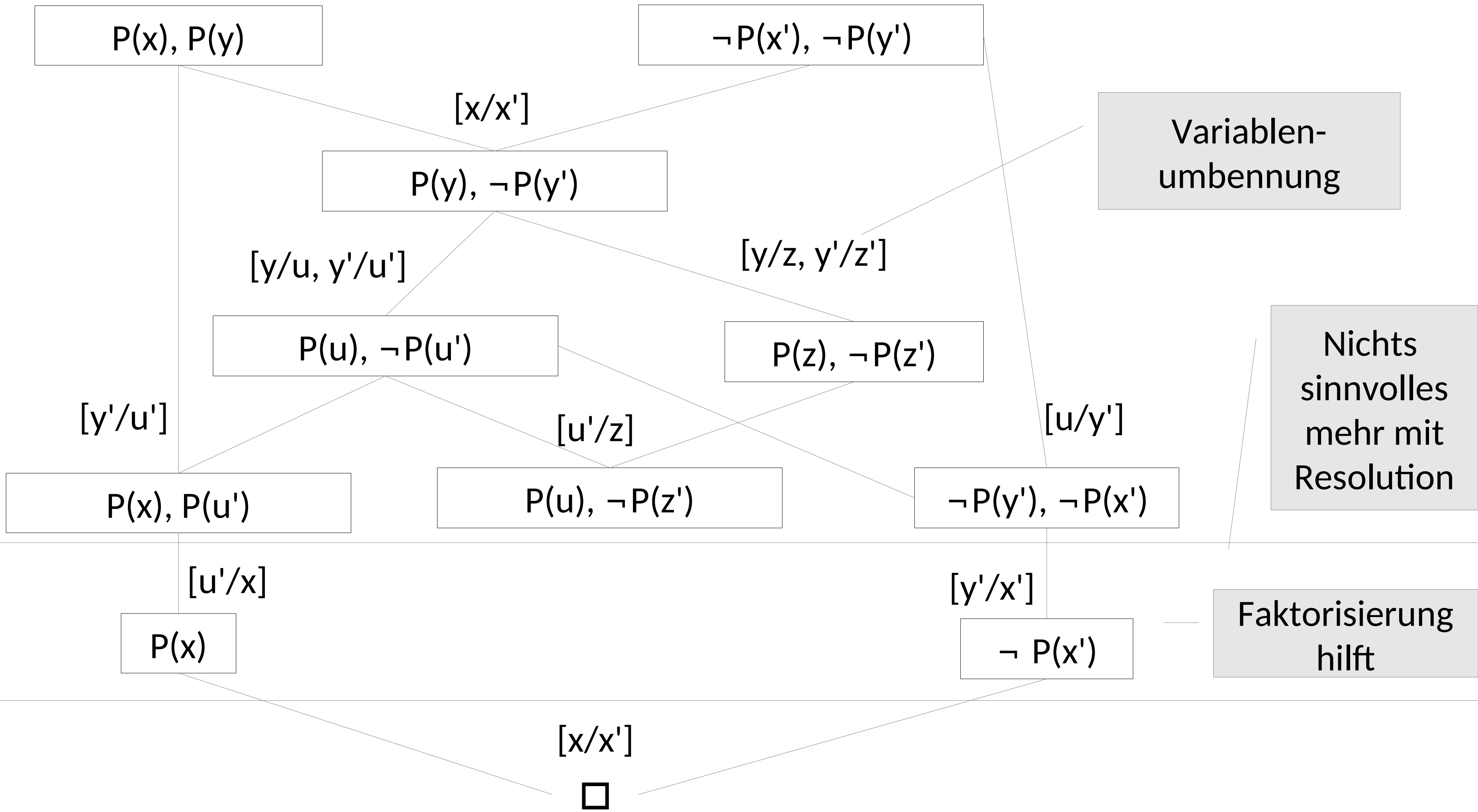
- { P(0), ∀x (P(x) ⇒ Q(f(x))) } ├ P(a)



- [Keine weitere Ableitung möglich (keine Resolution, keine Faktorisierung). Die leere Klausel kann nicht abgeleitet werden. P(a) folgt nicht, und die Resolution findet das heraus]

Resolution – Beispiel

- $\{ \forall x\, \forall y\; P(x) \vee P(y) \} \vdash \exists x'\, \exists y'\; P(x') \wedge P(y')$?



Resolution für Prädikatenlogik

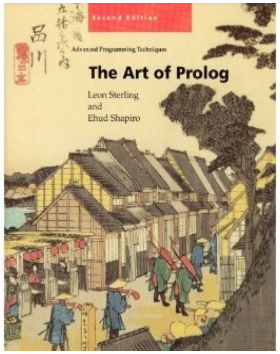
- Prädikatenlogik ist unentscheidbar
  - Es gibt kein berechenbares Entscheidungsverfahren das entscheidet,
    ob eine Formel eine logische Folgerung aus einer Formelmenge ist oder nicht
- Trotzdem ist die Resolution ein vollständiger und korrekter Kalkül
- Drei Situationen können beim Beweisen von $\Sigma \vdash \beta$ auftreten
  - Die leere Klausel wird abgeleitet: Es gilt $\Sigma \vdash \beta$
  - Wir haben alle Klauseln konstruiert, die man mit Resolution und Faktorisierung ableiten kann und die leere Klausel konnte dabei nicht erzeugt werden: Es gilt **nicht** $\Sigma \vdash \beta$
  - Der Ableitungsvorgang dauert unendlich, beziehungsweise wir können noch keine Aussage zur Terminierung machen.
    Wir können keine Aussage zu $\Sigma \vdash \beta$ machen

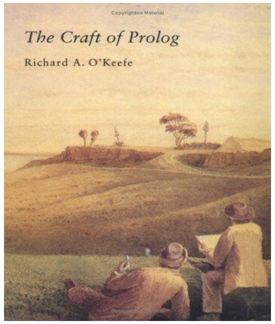
Inhalte

- Einführung
- **Symbolische Verfahren, Logik**
  - Aussagenlogik, Prädikatenlogik
  - **Horn Logik, Prolog**
- Suchen und Bewerten
  - Problemlösen durch Suche
    - Uninformierte Suche
    - Heuristische Suche
    - Spielbäume
  - Information Retrieval
- Lernen
  - Entscheidungstheorie
  - Naive Bayes
  - Entscheidungsbäume
  - Neuronale Netze
  - unüberwachtes Lernen

Hornlogik und Prolog

- Hornlogik
  - Prädikatenlogik erster Stufe mit Einschränkungen
    - Klauselnormalform, nur ein positives Literal, weniger ausdrucksmächtig
  - Resolution vollständig, keine Faktorisierung
- Prolog
  - Logisches/Deklaratives Programmieren
  - Basiert auf Horn-Logik und SLD-Resolution
  - Programmiersprache Prolog
- Literatur und Online-Quellen
  - SWI-Prolog, http://www.swi-prolog.org
    - Umfassende Features, ausgereift, plattformunabhängig
  - Prolog-Programmieren – Kunst und Handwerk
    - The Art of Prolog
      Sterling Shapiro, 1994
    - The Craft of Prolog
      O'Keefe, 1990„*Elegance is not optional*“





Hornlogik

- Hornlogik
  - Echte Untermenge der Prädikatenlogik
  - Formeln in Klauselnormalform
  - Aber nur ein positives Literal!
- Modellierung
  - Implikationen einfach zu realisieren
  - Typisch, bei Abbildung von Wissen
  - Vorwärtsregeln: Wenn das und das und das und .... dann das
  - Aber *echte* Untermenge, manche Aussagen nicht ausdrückbar
- Beispiel
  - {P(x), Q(y)} ist *keine* Hornklausel
  - {¬P(x), Q(y)} ist eine Hornklausel
  - {¬P(x), ¬Q(y), R(y)} ist eine Hornklausel
  - {P(x)} ist eine Hornklausel

$\alpha, \alpha_i, \beta, \beta_i$ sind Atomformeln

$\neg\alpha_1 \vee \ldots \vee \neg\alpha_n \vee \beta$

$\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta$

$\cancel{\alpha_1 \wedge \alpha_2 \Rightarrow \beta_1 \vee \beta_2}$

$\cancel{\neg\alpha_1 \vee \neg\alpha_2 \vee \beta_1 \vee \beta_2}$

Hornlogik – Fakten, Regeln, Anfragen

- Wir nennen einen Hornklausel
- *Fakt*
  - wenn sie nur aus einer positiven atomaren Formel besteht
  - Beispiel: { P(x) } $\Rightarrow P(x)$
- *Regel*
  - wenn sie aus einer positiven und mindestens einer negativen atomaren Formel besteht $P(x) \land Q(y) \Rightarrow R(x)$
  - Beispiel: {¬P(x), ¬Q(y), R(x)}
- *Anfrage*
  - wenn sie ausschließlich aus negativen atomaren Formeln besteht
  - Beispiel: { ¬P(x), ¬Q(x) } $P(x) \land Q(x) \Rightarrow$
    Gilt P(x) und Q(x)?
    Kann man P(x) und Q(x) aus Wissensbasis Σ folgern? … gdw … $\Sigma \models P(x) \land Q(x)$
    Ist ¬P(x), ¬Q(x) zusammen mit Σ inkonsistent? $\Sigma \cup \{\neg P(x) \lor \neg Q(x)\}$
  - Inferenzverfahren um leere Klausel herzuleiten ist Resolution

Hornlogik und Prolog

- Konvention Groß-/Kleinschreibung
  - Prädikatssymbole und Funktionssymbole aus Kleinbuchstaben
  - Variablensymbole aus Großbuchstaben

equals, add
mutter, f, g, p, q

X, Y, Z, X0, Y1
A, B, P, Q

- Fakten
  - als atomare Formel gefolgt von Punkt(.)

```
mag(hans, brot).
equals(X, X).
```

- Regeln als Implikation
  - Konklusion (*Head*) zuerst
  - Visualisierung der Implikation ⇐ durch :-
  - Prämissen (*Body*) danach kommasepariert

```
mag(hans, X) :- scharf(X).
equals(X, Y) :-
  equals(Y, X).
equals(X, Z) :-
  equals(X,Y), equals(Y,Z).
```

$$\text{Equals}(x,y) \wedge \text{Equals}(y,z) \Rightarrow \text{Equals}(x,z)$$

- Anfrage
  - Beginnt mit ?-
    Das Fragezeichen um An*frage* zu visualisiern
  - Atomare Formeln danach kommasepariert

```
?- mag(hans, chili).
?- equals(a, X), equals(X, b).
```

Hornlogik und Prolog – Beispiele

Indian(Curry).
...

$\forall x\ Indian(x) \wedge Mild(x) \Rightarrow Likes(Sam, x)$

Gilt $\Sigma \models Likes(Sam, Dahl)$?

Fakten

```
indian(curry).
indian(dahl).
indian(tandoori).
indian(kurma).
mild(dahl).
mild(tandoori).
mild(kurma).
chinese(chow_mein).
chinese(chop_suey).
chinese(sweet_and_sour).
italian(pizza).
italian(spaghetti).
```

Regeln

```
likes(sam, Food) :-
        indian(Food), mild(Food).
likes(sam, Food) :- chinese(Food).
likes(sam, Food) :- italian(Food).
likes(sam, chips).
```

Σ

Anfragen

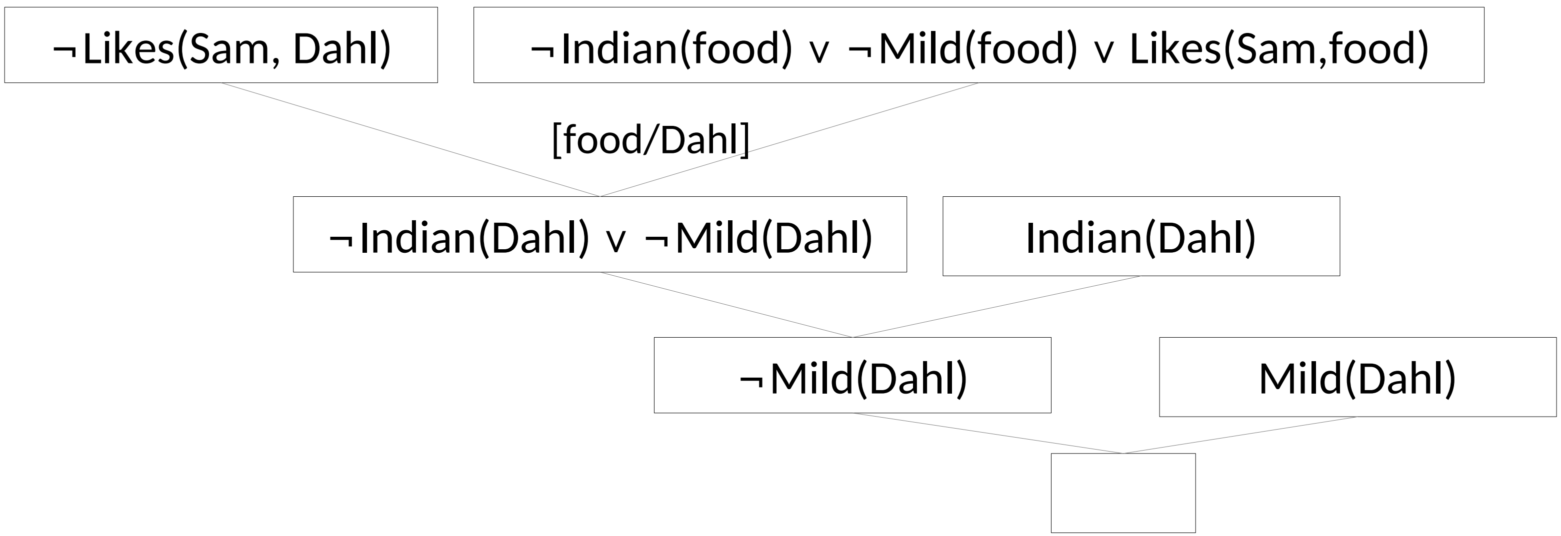
```
?- likes(sam, dahl).
?- likes(sam, chop_suey).
?- likes(sam, pizza).
?- likes(sam, chips).
?- likes(sam, curry), indian(tandoori).
```

Hornlogik – Inferenzverfahren

- Inferenzverfahren für $\Sigma \models \beta_1 \land \ldots \land \beta_n$
  - Ziel ist immer das Zeigen der Inkonsistenz von $\Sigma \cup \{\neg\beta_1 \lor \ldots \lor \neg\beta_n\}$
  - Dazu ist in der Hornlogik Resolution ausreichend (Faktorisierungsregel ist nicht notwendig)
  - Spezielle Strategie zur Regelauswahl
- SLD-Resolution
  - Selective Linear Resolution with Definite clauses
  - Suchbaum für Resolution von einer Anfrage
    - Auswahl der möglichen Regeln, immer aus Anfrage, Reihenfolge unbestimmt
  - Resolventen sind wieder Hornklauseln
- Umsetzung in Prolog
  - Literale der Anfrage von links nach rechts
  - Regelalternativen von oben nach unten, Tiefensuche
- Anfrage implizit existenzquantifiziert
  - Man sucht ein „Beispiel“ für die Inkonsistenz
  - Beispiel ist Antwortsubstitution, die Kumulation der Unifikatoren

Hornlogik/Prolog – Beispiel

- Gilt $\Sigma \models$ Likes(Sam, Dahl)?




- ```
  ?- likes(sam, dahl).

  ?- indial(dahl), mild(dahl).
  ?- mild(dahl).
  ?-

  yes
  ```

```
likes(sam, Food) :-
  indian(Food), mild(Food).
indian(dahl).
mild(dahl).
```
```

Beispiel mit Antwortsubstitution

- Gilt $\Sigma \models$ Likes(Sam, x)?

¬Likes(Sam, x)

¬Indian(food) ∨ ¬Mild(food) ∨ Likes(Sam,food)

[food/x]

¬Indian(x) ∨ ¬Mild(x)

Indian(Dahl)

[x/Dahl]

¬Mild(Dahl)

Mild(Dahl)

```
likes(sam, Food) :-
  indian(Food), mild(Food).
indian(dahl).
mild(dahl).
```

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

  ?- indial(dahl), mild(dahl).
  ?- mild(dahl).
  ?-

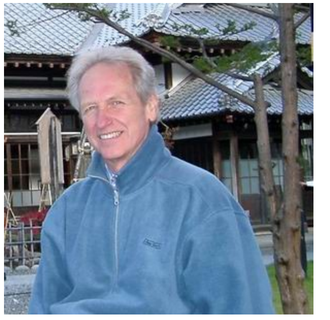
  X=dahl ?
  ```

weitere mögliche Antwortsubstitutionen auf Rückfrage
  ```

Logische Programmierung

- Deklarative Programmierung
  - Programmierparadigma (wie imperativ, funktional, objektorientiert)
  - Problembeschreibung statt Lösungsweg
  - Die Umgebung findet die Lösung
- Logische Programmierung
  - ist deklarative Programmierung
  - Prädikatenlogische Formeln für Beziehungen zwischen Objekten
  - Formeln sind Problembeschreibung
  - Inferenzmaschine berechnet Lösung
- Logische Programmierung mit Prolog
  - Einschränkungen (Horn-Logik, links/rechts Tiefensuche SLD-Resolution) und damit Inferenzmaschine effizient realisierbar
  - Ungetypte Terme als Datenstrukturen
  - + weitere Kompromisse und/oder Constraint-Systeme

Prolog

- Historie
  - Alain Colmerauer, Robert Kowalski, 1972
    Universität von Marseille, ursprüngliches Ziel:
    Verständnis natürlicher Sprache, Französisch
  - Prädikatenlogik erster Stufe, Regelsysteme
    Einschränkung und vollständiges Suchverfahren (SLD-Resolution)
    Ausführung Backtracking
  - 80er Jahre: Freie und kommerzielle Prolog-Implementierungen,
    Neben LISP die Sprache für KI
  - 90er Jahre: Japan 5th Generation Computer Projekt, Compiler/WAM
    Constraint Logic Programming (CLP)
  - Heute: Spezialanwendungen in KI, Logik, Deduktive Datenbanken, Symbolische Berechnungen, Operations Research, …
    Ausgereifte Implementierungen (SICStus, SWI, …)
- SWI-Prolog, http://www.swi-prolog.org/
  - Seit 1986, WAM-basiert, schnell und stabil, Plattformunabhängig
  - Graphische Oberfläche, CLP (Q, FD, …)

Programmieren

- Programmieren
  - Fakten abgeben
  - Regeln angeben
- Datenbasis laden
  - Dateiname mit .pl am Ende
  - Einlesen mit ?- [dateiname]. ohne .pl am Ende
  - Alternative mit consult/comile
- Anfragen stellen
  - Aussage hinter ?-

verwandschaft.pl

```
frau(carol).     % Carol ist eine Frau
frau(eva).
frau(susi).
frau(lilith).
mann(abel).      % Abel ist ein Mann
mann(adam).

elter(adam, kain).  % Adam ist ein Elternteil von Kain
elter(eva, kain).
elter(kain, susi).
elter(carol, susi).
elter(lilith, carol).

mutter(X, Y) :-     % X ist eine Mutter von Y
    elter(X, Y),    % wenn X Elternteil von Y ist und
    frau(X).        % X eine Frau ist

oma(X, Z) :-        % X ist eine Oma von Z
    mutter(X, Y),   % wenn X eine Mutter von Y ist und
    elter(Y, Z).    % Y ein Elternteil von Z ist
```

```
?- [verwandschaft].   % Laden
?- oma(X, susi).      % Welche X sind Oma von Susi?
X = eva ;             % Eva ist Oma von Susi
X = lilith ;          % Lilith ist Oma von Susi
No                    % sonst niemand
?-
```

Anfrage und Berechnung

- Anfrage
  - Für welche X gilt X ist die oma von susi?

oma(X, susi)

- Berechnung
  - Probiere alle Regeln
  - Suchbaum

oma(X, susi)

mutter(X, Y), elter(Y, susi)

elter(X,Y), frau(X), elter(Y,susi)

elter(adam, kain), frau(adam), elter(kain, susi)

elter(eva, kain), frau(eva), elter(kain,susi)

Ja, wenn X=eva ist

....

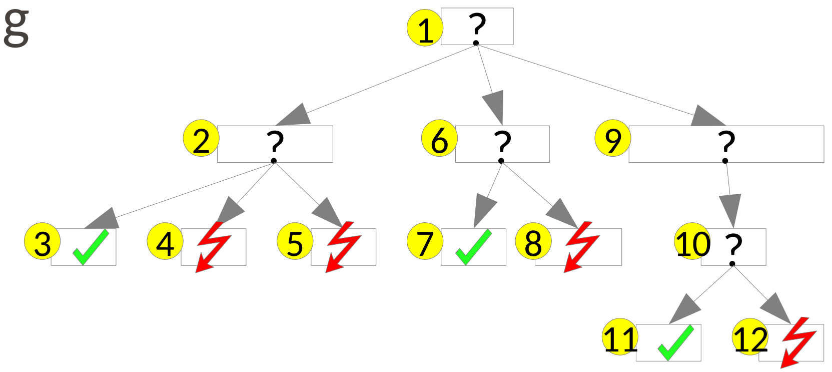
....

elter(lilith, carol), frau(lilith), elter(carol, susi)

Ja, wenn X=lilith ist

Berechnung in Prolog

- Suche nach einer Variablenbelegung
  - Von links nach rechts
  - Tiefensuche, Backtracking
    - Kann schiefgehen, wenn ein Pfad unendlich lang wird



- Erfolg/Mißerfolg
  - Bei Erfolg „Ja“ und Ausgabe der aktuellen Variablenbelegung
    - Bei ; weiter probieren
    - Bei Return beenden
  - Bei Mißerfolg „Nein“

```
?- oma(X, susi).      % Welche X sind Oma von Susi?
X = eva ;             % Eva ist Oma von Susi
X = lilith ;          % Lilith ist Oma von Susi
No                    % sonst niemand
?-
```

- Wenn nichts gefunden wurde „Nein“
  - Annahme, dass alles Wissen vorhanden ist
  - Closed World Assumption

```
?- elter(adam, carol).
No
```

Könnte sein, aber ist nicht in der Menge der Fakten.

Mehrere Regeln – Vorfahren

- X ist Vorfahre von Z
  - Wenn X ein Elternteil von Z ist
  - Oder wenn X ein Elternteil von Y ist und Y ein Vorfahre von Z

```
vorfahr(X, Z) :-
        elter(X, Z).
vorfahr(X, Z) :-
        elter(X, Y),
        vorfahr(Y, Z).
```

```
?- vorfahr(X, susi).
X = kain ;
X = carol ;
X = adam ;
X = eva ;
X = lilith ;
No
?- vorfahr(X, carol).
X = lilith ;
No
?- vorfahr(eva, X).
X = kain ;
X = abel ;
X = susi ;
No
?-
```

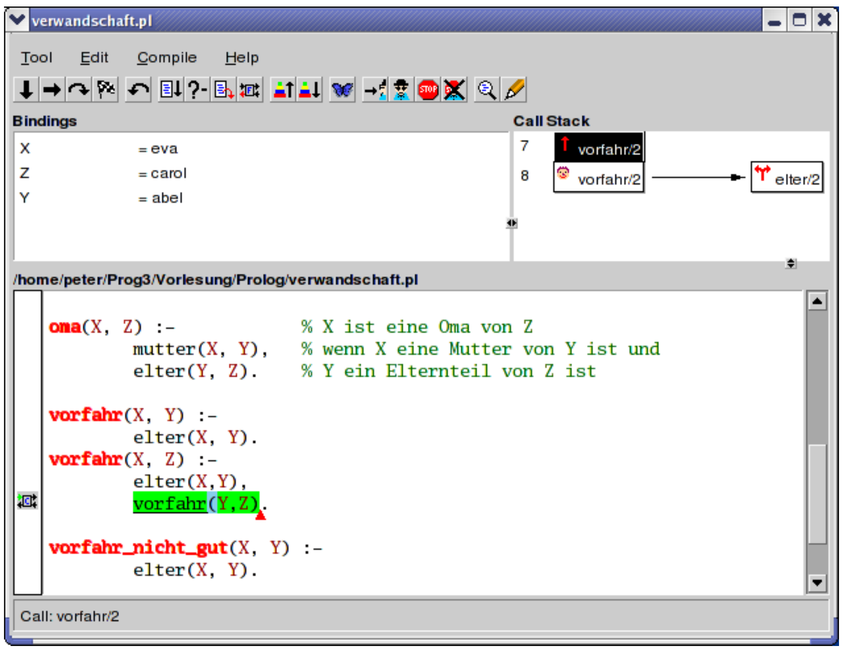
- Beispiel:
  - kain, carol, … lilith sind Vorfahren von susi
  - lilith ist der einzige Vorfahr von carol
  - eva ist Vorfahr von kain, abel und susi
- Achtung – Reihenfolge!
  - von links nach rechts mit Tiefensuche kann schief gehen

```
vorfahr_nicht_gut(X, Y) :-
    elter(X, Y).
vorfahr_nicht_gut(X, Z) :-
    vorfahr_nicht_gut(Y, Z),
    elter(X, Y).
```

```
?- vorfahr_nicht_gut(susi, X).
ERROR: Out of local stack
?- vorfahr_nicht_gut(X, abel).
X = adam ;
X = eva ;
ERROR: Out of local stack
?-
```

Prolog bei der Ausführung beobachten

- Aufrufe und Variablen-belegung bei der Tiefensuche
  - Text-basiert mit trace.
  - Graphisch mit guitracer.

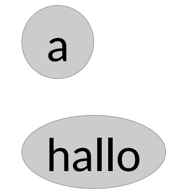



```
?- trace.
[trace]  ?- vorfahr(X, carol).
   Call: (8) vorfahr(_G315, carol) ? creep
   Call: (9) elter(_G315, carol) ? creep
   Exit: (9) elter(lilith, carol) ? creep
   Exit: (8) vorfahr(lilith, carol) ? creep
X = lilith ;
   Redo: (8) vorfahr(_G315, carol) ? creep
   Call: (9) elter(_G315, _L192) ? creep
   Exit: (9) elter(adam, kain) ? creep
   Call: (9) vorfahr(kain, carol) ? creep
   Call: (10) elter(kain, carol) ? creep
   Fail: (10) elter(kain, carol) ? creep
   Redo: (9) vorfahr(kain, carol) ? creep
   …
   Redo: (10) vorfahr(susi, carol) ? creep
   Call: (11) elter(susi, _L214) ? creep
   Fail: (11) elter(susi, _L214) ? creep
No
?- guitracer.
% The graphical front-end will be used for subsequent tracing
Yes
?- trace.
Yes
[trace]  ?- vorfahr(X, carol).
```

Datenstrukturen sind Terme

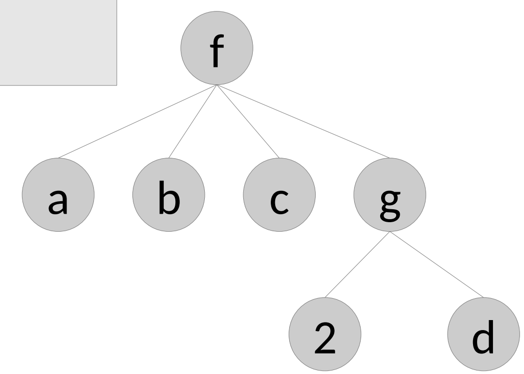
- Atome
  - Kleingeschriebene Wörter
  - Funktoren ohne Parametern

```
?- X = a, Y = hallo.
X = a
Y = hallo
```



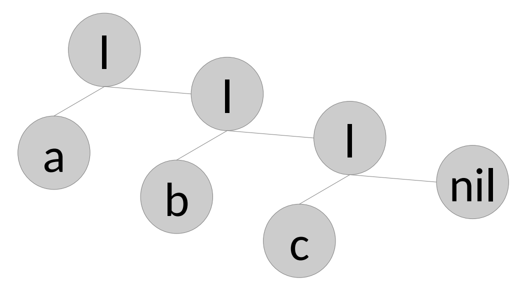
- Terme
  - Baumstruktur
  - Funktoren mit mehreren Argumenten
  - Argumente sind Terme
  - Atome sind Terme

```
?- X = f(a,b,c, g(2, d)).
X = f(a, b, c, g(2, d))
```



- Listen
  - Darstellbar als Baum
  - Spezielle Notation in Prolog möglich

```
l(a, l(b, l(c, nil)))
```

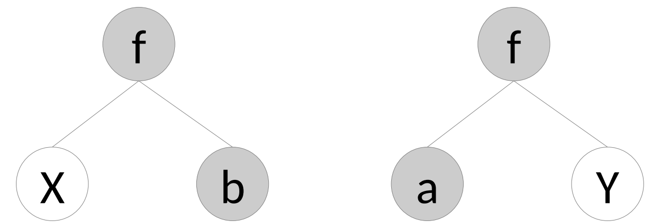


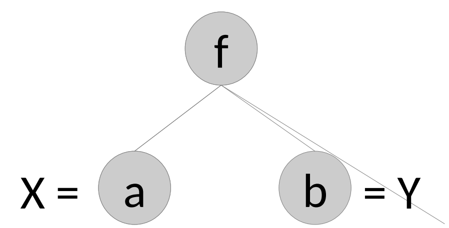

```
?- X = [a,b,c], Y = [a | [b | [c | []]]].
X = [a, b, c]
Y = [a, b, c]
Yes
```

Unifikation

- =
  - Nicht Zuweisung sondern Unifikation
  - Versucht eine Variablenbelegung zu finden zwei Terme gleich zu machen
  - Variablen an beliebigen Stellen im Term
- Unifikationsmethode
  - Gleichungsmenge
  - Wenn Funktoren nicht gleich, FAIL
  - Ersetze Funktionsgleichung durch
  - Entferne triviale Gleichungen zwischen Atomen
  - Wenn eine Seite eine Variable ist, dann ersetze jedes Vorkommen der Variable durch andere Seite
  - Ergebnis wie Robinson-Unifkation
- Zyklische Terme vermeiden
  - Je nach Prolog erlaubt, mit Occurscheck verboten

```
?- f(X, a) = f(b, Y).
```






```
?- f(X) = X.
X = f(**)
Yes
?- unify_with_occurs_check(f(X),X).
No
```

Unifikation – Beispiele

- Reihenfolge beliebig, Menge von Gleichungen, Vereinfachung
  - { X = f(Y), Y= a }
    { X = f(a), Y=a }
- Gleichsetzen der Unterterme, Vereinfachung
  - { f(X,a) = f(b,Y) }
    { X = b, a = Y }
  - { f(X, a, g(Y, X)) = f(c, a, Z) }
    { X = c, a = a, g(Y,X) = Z }
    { X = c, g(Y, X) = Z}
    { X = c, g(Y, c) = Z}
- Nicht immer eine Lösung
  - { f(X,b) = f(a,X) }
    { X =a, b = X}   % Widerspruch

```
?- X = f(Y), Y = a.
X = f(a)
Y = a
Yes
?- f(Y) = X, Y = a.
Y = a
X = f(a)
Yes

?- f(X, a) = f(b, Y).
X = b
Y = a
Yes

?- f(X,a,g(Y, X)) = f(c,a,Z).
X = c
Z = g(Y, c)
Yes

?- f(X, b) = f(a, X).
No
```

Terminierung

- Terminierung nicht garantiert
  - Aufgrund der Strategie endlose Inferenzketten möglich
  - Im Beispiel liefert Anfrage
    `?- p(a).`
    kein Ergebnis
  - Es wird immer wieder (nach Umbenennung der Variablen X) mit der ersten Regel resolviert, aber kein Fortschritt erzielt
- Wie beim Programmieren denken
  - Reihenfolge beachten
  - Problemreduktion,wie Rekursion
    - Problem muss kleiner werden Ordnung
    - Es darf nur endlich viele Schritte bis zum trivialen Fall geben, diskrete Ordnung

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

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

```
?- p(a).
ERROR: Out of local stack
```

wäre ok

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

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

```
?- p(b).
ERROR:
```

```
p(a).
```

```
?- p(b).
false
```

nicht herleitbar

Negation und Closed World Assumption

- Negation
  - Nicht unterstützt in Hornlogik
  - Nicht unterstützt in Prolog
- Negation as Failure
  - Wenn mit den vorhandenen Regeln eine Anfrage nicht bewiesen werden kann, dann wird angenommen die Anfrage gilt nicht
  - Statt Negation besser „nicht herleitbar“ $\nvdash P(x)$
  - Eingebautes Prädikat \+ sieht ähnlich aus wie $\nvdash$
    (hieß früher not())
    - Metaprädikat (hat Prädikat als Argument)
    - Vermeiden!
  - ACHTUNG: Eine solche Aussage kann sich ändern,
    wenn neue Fakten hinzu kommen, also wird **nicht bewiesen** $\vdash \neg P(x)$
    keine Negation in Hornlogik
- Closed World Assumption
  - Alles was nicht als wahr gezeigt werden kann ist falsch

```
p(a).
```

```
?- p(b).
false.
?- \+(p(b)).
true.
```

Arithmetik

- Arithmetik
  - Repräsentation von natürlichen Zahlen durch Terme
    - 0 durch `0`
    - 1 durch `succ(0)`
    - 2 durch `succ(succ(0))`
    - …
  - Addition
    - Neutrales Element 0
    - x+y = (x-1) + (y+1)
    - (oft) terminierend, da kleiner werdend und gegen 0
    - Idealerweise „Sorte“ festlegen
- Anfragen
  - Rechnen
  - Aber auch umstellen!
  - Und Lösungen aufzählen!

```
zahl(0).
zahl(succ(X)) :-
   zahl(X).

add(0, X, X).
add(succ(X), Y, succ(Z)) :-
   add(X, Y, Z).
```

```
add(0, X, X) :-
      zahl(X).
```

```
?- zahl(X).
X = 0 ;
X = succ(0) ;
X = succ(succ(0)) ;
X = succ(succ(succ(0))) .

?- add(succ(succ(0)), succ(0), X).
X = succ(succ(succ(0))).

?- add(succ(succ(0)), X, succ(succ(succ(succ(0))))).
X = succ(succ(0)).

?- add(X, Y, succ(succ(0))).
X = 0,   Y = succ(succ(0)) ;
X = succ(0),   Y = succ(0) ;
X = succ(succ(0)),   Y = 0 ;
false.
```