

Kapitel VI.2

Induktives logisches Programmieren (ILP)

© Institut AIFB, 2002.

Alle Rechte vorbehalten. Nachdruck oder photomechanische Wiedergabe nur mit Genehmigung des Verfassers.
Zuwendungen unterliegen den strafrechtlichen Bedingungen des Urheberrechtsgesetzes.

VI.2 Induktives Logisches Programmieren

(Inductive Logic Programming - ILP)

(Lavrac / Dzeroski 1994)

VI.2.1 Einführung

- ILP kombiniert Methoden des induktiven maschinellen Lernens (I) mit Methoden der Logikprogrammierung (LP).

$$ILP = I + LP$$

- Ziel ist das Lernen *relationaler* Beschreibungen in Form von Logikprogrammen - ggf. unter Verwendung von *Hintergrundwissen* (background knowledge).
- prädikatenlogischer Formalismus ist *ausdrucksstärker* als Attribut-Wert-Darstellung

Abduktion
Induktion
Deduktion

Kapitel VI.2: Induktives logisches Programmieren

Beispiel: "Spielkarten"

Farbe = { Herz, Pik, Kreuz, Karo }

Wert = { 7, 8, ... }

- Attribut-Wert-Darstellung:

- „Kreuz 7“: [(Farbe = Kreuz) \wedge (Wert = 7)]
- intensionale Beschreibung von „Kartenpaar“ durch Regel:

$$\begin{aligned} &\text{IF } [(\text{Wert}_1 = 7) \wedge (\text{Wert}_2 = 7)] \vee \\ &\quad [(\text{Wert}_1 = 8) \wedge (\text{Wert}_2 = 8)] \vee \\ &\quad \dots \\ &\text{THEN Kartenpaar} \end{aligned}$$

D.h., jede Attribut-Wert-Kombination muß explizit aufgeführt werden!

- Hornklausel-Darstellung:

- „Kreuz 7“ wird durch die atomare Formel
 $\text{Karte}(\text{Kreuz}, 7)$

mit dem Prädikatensymbol ‚Karte‘ und den Konstanten Kreuz und 7 dargestellt.

- „Kartenpaar“ wird intensional durch die Regel beschrieben:

$$\text{Paar}(\text{Farbe}_1, \text{Wert}_1, \text{Farbe}_2, \text{Wert}_2) \leftarrow (\text{Wert}_1 = \text{Wert}_2)$$

(Prädikatensymbol: „Paar“,
Variablen: $\text{Farbe}_1, \text{Farbe}_2, \text{Wert}_1, \text{Wert}_2$)

Die Verwendung von Variablen ermöglicht eine kompakte Beschreibung!

Kapitel VI.2: Induktives logisches Programmieren

a) verwendete Begriffe:

- *Fakt*: einzelnes Objekt, z.B. "Kreuz 7"
 - *Objektbeschreibungssprache*
z.B. "Karte(Kreuz,7)"
- *Begriff*: Beschreibung einer Objektmenge („Kartenpaar“)
unterscheide: *extensionale* Begriffsdefinition
intensionale Begriffsdefinition (z.B. "Paar"-Regel)
- *Konzeptbeschreibungssprache L*
L legt fest, welche Sprachelemente zur Beschreibung von Konzepten verwendet werden dürfen
 - *language bias* (Sprachvorgabe)
- Ein Begriff *erklärt* ein Objekt, wenn die Objektbeschreibung der Begriffsdefinition genügt (concept description covers the object description).

Kapitel VI.2: Induktives logisches Programmieren

- *Hypothese* (hypothesis): zu lernende intensionale Konzeptbeschreibung
 - E^+ : Menge der *positiven* Beispiele
 - E^- : Menge der *negativen* Beispiele
 - $E = E^+ \cup E^-$: Menge der Beispiele
- *Lernen eines Konzepts* aus Beispielen :
 - geg. Menge E von Beispielen
 - finde Hypothese H in vorgegebener Konzept - beschreibungssprache , so daß
 - H jedes positive Beispiel $e \in E^+$ erklärt,
 - H kein negatives Beispiel $e \in E^-$ erklärt.

Kapitel VI.2: Induktives logisches Programmieren

- Funktion '*covers*':

$$\text{covers}(H,e) = \begin{cases} \text{true, H erklärt e} \\ \text{false, sonst} \end{cases}$$

Verallgemeinerung auf Beispielmengen:

$$\text{covers}(H,E) = \{e \in E \mid \text{covers}(H,e) = \text{true}\}$$

- Hypothese H ist *vollständig* bezüglich Beispielmenge E, wenn

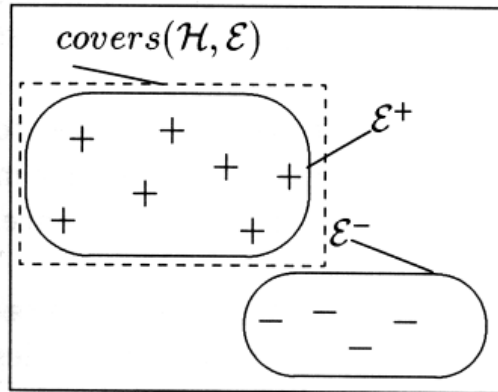
$$\text{covers}(H, E^+) = E^+$$

- Hypothese H ist *konsistent* bezüglich Beispielmenge E, wenn

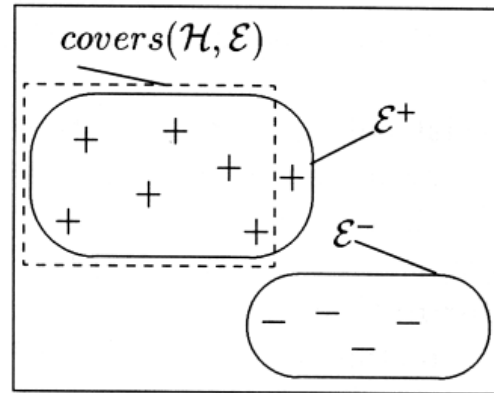
$$\text{covers}(H, E^-) = \emptyset$$

Kapitel VI.2: Induktives logisches Programmieren

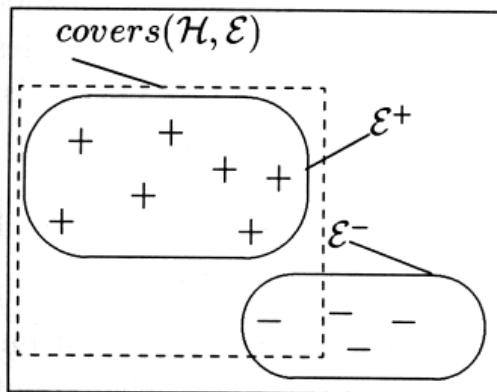
\mathcal{H} : complete, consistent



\mathcal{H} : incomplete, consistent



\mathcal{H} : complete, inconsistent



\mathcal{H} : incomplete, inconsistent

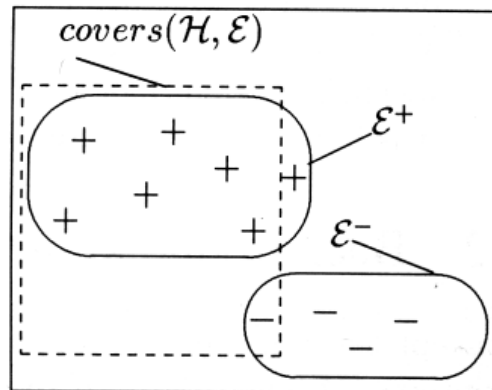


Abbildung VI.2.1:
Completeness and
consistency of a
hypothesis

(Lavrač/ Dzeroski 1994)

Kapitel VI.2: Induktives logisches Programmieren

b) Hintergrundwissen

- Zusätzlich zur Beispielmenge E kann Hintergrundwissen B (*background knowledge*) zum Lernen verwendet werden:
- Man verwendet zur Begriffsbeschreibung auch Relationen (Begriffe, Beschreibungen) aus B.

- *Lernen eines Begriffs mit Hintergrundwissen*

- geg.: Menge E von Beispielen
Hintergrundwissen B
 - gesucht: Hypothese H in vorgegebener Begriffs-
beschreibungssprache L, so dass H vollständig und
konsistent bezüglich B und E ist.
- Funktion ‚covers‘ mit Einbeziehung von B:
- $$\text{covers}(B, H, e) = \text{covers}(B \cup H, e)$$
- $$\text{covers}(B, H, E) = \text{covers}(B \cup H, E)$$

Kapitel VI.2: Induktives logisches Programmieren

- *Vollständigkeit* einer Hypothese H bei Einbeziehung von Hintergrundwissen B: $\text{covers}(B, H, E^+) = E^+$
- *Konsistenz* einer Hypothese H bei Einbeziehung von Hintergrundwissen B: $\text{covers}(B, H, E^-) = \emptyset$
- I.a. wird zur Beschreibung der Beispiele E, des Hintergrundwissens B sowie der Hypothesen H *dieselbe* Sprache L verwendet
 - Beispiele sind *Grundfakten* des Zielprädikates.
 - Hintergrundwissen ist definiert unter Verwendung entsprechender *Prädikatsymbole*.
 - Die Hypothese ist eine intensionale Beschreibung des *Zielprädikates (target predicate)*.

Kapitel VI.2: Induktives logisches Programmieren

Beispiel:

- Lerne intensionale Beschreibung des *Zielprädikates daughter* (X, Y) mit der Bedeutung „Person X ist Tochter von Person Y “.
- Das Hintergrundwissen stellt *zwei Relationen* zur Verfügung in Form
 - des 1-stelligen Prädikatensymbols ‚*female*‘
 - des 2-stelligen Prädikatensymbols ‚*parent*‘
- Die Beispielmenge E enthält 2 positive und 2 negative Beispiele

Training examples		Background knowledge	
Daughter (mary, ann).	+	Parent (ann, mary).	Female (ann).
Daughter (eve, tom).	+	Parent (ann, tom).	Female (mary).
Daughter (tom, ann).	-	Parent (tom, eve).	Female (eve).
Daughter (eve, ann).	-	Parent (tom, ian).	

Tabelle VI.2.1: A simple ILP problem: learning the daughter relation.

Buch online unter <http://www-ai.ijs.si/SasoDzeroski/ILPBook>

→ (Lavrac / Dzeroski 1994)

Kapitel VI.2: Induktives logisches Programmieren

Training examples		Background knowledge	
Daughter (mary, ann).	+	Parent (ann, mary).	Female (ann).
Daughter (eve, tom).	+	Parent (ann, tom).	Female (mary).
Daughter (tom, ann).	-	Parent (tom, eve).	Female (eve).
Daughter (eve, ann).	-	Parent (tom, ian).	

- Mögliche Definition des Zielprädikates ‚daughter‘:

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

- Modellierung des Hintergrundwissens mit den Prädikaten

female, mother, father

ergibt eine andere Definition des Zielprädikates

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

$$\text{daughter}(X, Y) \leftarrow \text{female}(X), \text{father}(Y, X)$$

VI.2.2 Grundlagen des ILP

a) Begriffe der Logikprogrammierung

- Eine *Klausel* ist eine prädikatenlogische Formel der Form

$$\forall X_1 \forall X_2 \dots \forall X_s (L_1 \vee L_2 \vee \dots \vee L_m)$$

mit Variablen X_i ($i=1, \dots, s$)
und Literalen L_j ($j=1, \dots, m$)

- Schreibweisen:

(i) $L_1 \vee L_2 \vee \dots \vee L_{i-1} \vee \bar{L}_i \vee \bar{L}_{i+1} \vee \dots \vee \bar{L}_m$

(ii) $\{L_1, L_2, \dots, L_{i-1}, \bar{L}_i, \bar{L}_{i+1}, \dots, \bar{L}_m\}$ Mengenschreibweise

(iii) $L_1 \vee L_2 \vee \dots \vee L_{i-1} \leftarrow L_i \wedge L_{i+1} \wedge \dots \wedge L_m$

(iv) $L_1, L_2, \dots, L_{i-1} \leftarrow L_i, L_{i+1}, \dots, L_m$

Kapitel VI.2: Induktives logisches Programmieren

- Eine *Hornklausel* ist eine Klausel mit höchstens einem positiven Literal
- Eine *definite Hornklausel* ist eine Klausel mit genau einem positiven Literal

$$\begin{array}{l} \text{Kopf-Literal} \rightarrow T \leftarrow L_1, \dots, L_m \\ \text{mit } T, L_i \text{ atomare Formeln} \end{array}$$

- Eine *Programmklausel* ist eine Klausel der Form

$$\begin{array}{l} T \leftarrow L_1, \dots, L_m \\ \text{mit } T \text{ atomare Formel} \\ L_i \text{ Literal} \end{array}$$

Kapitel VI.2: Induktives logisches Programmieren

b) grundlegende Definitionen

- Erklärung [coverage]:

- geg. Hintergrundwissen B, Hypothese H, Menge E von Beispielen
- H erklärt [covers] Beispiel $e \in E$ unter Berücksichtigung von B, wenn

$$B \cup H \models e,$$

$$\text{d.h. covers}(B, H, e) = \text{true, wenn } B \cup H \models e$$

ILP-Lernaufgabe:

gegeben:

- eine Menge E von Beispielen in Form von Grundfakten eines Prädikatsymbols p (= Zielprädikat).
- eine Beschreibungssprache L, sowie Hintergrundwissen B mit Prädikatsymbolen.

gesucht:

- Hypothese H für p in Beschreibungssprache L, so dass H vollständig und konsistent ist bezüglich E und B

c) Strukturierung des Hypothesenraumes

- Lernen einer Konzeptbeschreibung kann als *Suche* im Hypothesenraum interpretiert werden.
- Die Suche ist beeinflusst durch die *Struktur* des Suchraumes, die *Suchstrategie* und *Suchheuristik*.
- Viele ILP-Verfahren benutzen die θ -Subsumtion als syntaktisches Kriterium zur Strukturierung des Suchraumes (θ -Subsumtions-Verband)
- Definition θ -Subsumtion (θ -subsumption):

Seien c, c' zwei Programmklauseln in Mengennotation; c θ -subsumiert c' , wenn es eine Substitution θ gibt, so daß $c\theta \subseteq c'$.

Wenn c θ -subsumiert c' , dann $c \models c'$

Eigentlich will man \models als Suchkriterium verwenden, ist aber zu teuer zu berechnen.

Kapitel VI.2: Induktives logisches Programmieren

Beispiel:

Sei $c = \text{daughter}(X, Y) \leftarrow \text{parent}(Y, X)$

bzw. in Mengenschreibweise

$$c = \{ \text{daughter}(X, Y), \neg \text{parent}(Y, X) \}$$

und $c' = \text{daughter}(\text{mary}, \text{ann}) \leftarrow (\text{female}(\text{mary}), \text{parent}(\text{ann}, \text{mary}))$

bzw. in Mengenschreibweise

$$c' = \{ \text{daughter}(\text{mary}, \text{ann}), \neg \text{female}(\text{mary}), \neg \text{parent}(\text{ann}, \text{mary}) \}.$$

Mit der Substitution $\theta = \{ X/\text{mary}, Y/\text{ann} \}$ gilt:

$$(i) \quad c\theta = \{ \text{daughter}(\text{mary}, \text{ann}), \neg \text{parent}(\text{ann}, \text{mary}) \}$$

$$(ii) \quad c \theta\text{-subsumiert } c', \text{ da } c\theta \subseteq c'.$$

Kapitel VI.2: Induktives logisches Programmieren

- Zwei ProgrammklauseIn c , c' heißen θ -subsumtionsäquivalent, wenn $c\theta \subseteq c'$ und $c'\theta \subseteq c$.
- Eine Programmklausel heißt *reduziert* (*reduced*), wenn sie zu keiner echten Teilmenge von sich selbst θ -subsumtionsäquivalent ist.
- Definition: Syntaktische *Generalisierung* / *Spezialisierung*:
Seien c , c' ProgrammklauseIn;
 - (i) c ist mindestens so generell wie c' ($c \leq c'$), wenn c θ -subsumiert c' .
 - (ii) c ist genereller als c' ($c < c'$), wenn ($c \leq c'$) und $\neg (c' \leq c)$.

c heißt Generalisierung von c' , c' heißt Spezialisierung von c .

Kapitel VI.2: Induktives logisches Programmieren

- *Eigenschaften der θ -Subsumtion:*
 - (i) Wenn c θ -subsumiert c' , dann $c \models c'$
(Umkehrung gilt nicht)
 - (ii) Die \leq -Relation induziert einen *Verband* auf der Menge der reduzierten Klauseln, d.h. zwei Klauseln c, c' haben jeweils eine kleinste obere Schranke und eine größte untere Schranke.

Kapitel VI.2: Induktives logisches Programmieren

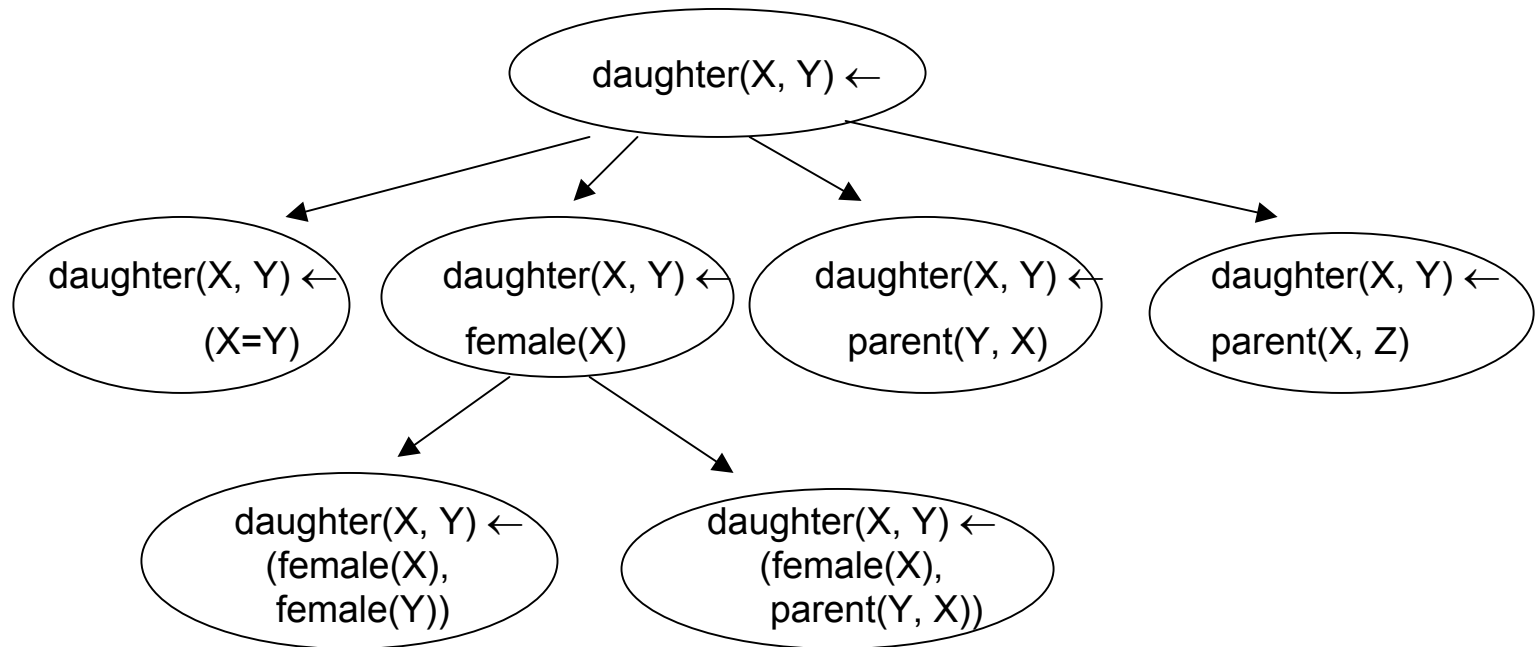


Abbildung VI.2.2: Part of the *refinement graph* for the family relations problem

(Lavrac / Dzeroski 1994)

Kapitel VI.2: Induktives logisches Programmieren

- Suchraumreduktion:

(i) wenn c zu c' generalisiert wird, dann werden alle Beispiele, die von c erklärt werden, auch von c' erklärt;

Es folgt: Wenn c ein *negatives* Beispiel erklärt,
dann erklärt auch c' dieses neg. Beispiel;

d.h.: *Generalisierungen* von c müssen *nicht*
mehr betrachtet werden.

(ii) wenn c zu c' spezialisiert wird, dann wird ein Beispiel e , das nicht von c erklärt wird, auch nicht von c' erklärt;

Es folgt: wenn c ein *positives* Beispiel e nicht
erklärt, dann erklärt auch c' dieses e nicht;

d.h.: *Spezialisierungen* von c müssen *nicht*
mehr betrachtet werden.

VI.2.3 Spezialisierende ILP-Verfahren

- Spezialisierende ILP-Verfahren durchsuchen den Hypothesenraum *top-down*, d.h. von generelleren zu spezielleren Hypothesen - unter Verwendung der *θ -Subsumtion*.

- Definition: *Verfeinerungsoperator (refinement operator)*.

Sei L eine Beschreibungssprache. Ein Verfeinerungsoperator ist eine Funktion $\varphi: L \rightarrow \wp(L)$, die jede Klausel $c \in L$ abbildet auf eine endliche Menge $\varphi(c)$ von Klauseln, die alle Spezialisierungen von c sind:

$$\varphi(c) \subseteq \{c' \mid c' \in L, c < c'\}.$$

- typische Verfeinerungsoperatoren:

(i) Anwendung einer *Substitution* auf eine Klausel

(ii) *Hinzufügen* eines *Literals* zum Körper einer Klausel

Kapitel VI.2: Induktives logisches Programmieren

MIS: Model Inference System (Shapiro 1983)

(Erstes spezialisierendes ILP-Verfahren)

- (1) Initialize hypothesis H to a (possibly empty) set of clauses in L.
 - (2) **repeat**
 - (3) Read the next (positive or negative) example.
 - (4) **repeat**
 - (5) **if** there exists a *covered negative example* e then
 - (6) Delete incorrect clauses from H.
 - (7) **if** there exists a *positive example e not covered* by H then
 - (8) With breadth-first search of the refinement graph
 - (9) develop a clause c which covers e and add it to H.
 - (10) **until** H is complete and consistent.
 - (11) **Output:** Hypothesis H.
- forever**

(A simplified MIS; see Lavrac / Dzeroski 1994, p. 54ff)

Kapitel VI.2: Induktives logisches Programmieren

- Verfeinerungsoperatoren erzeugen einen sogenannten *Verfeinerungsgraphen*: Programmklauseln sind die Knoten, Verfeinerungsbeziehungen sind die Kanten.

Beispiel: Anwendung von MIS auf die Familienbeziehungen

- (1) (i) Die Wurzel des Verfeinerungsgraphen ist das allgemeinste Zielprädikat:

$c = \text{daughter}(X, Y) \leftarrow$

- (ii) Hypothese: $H = \{c\}$

- 2x (2) (iii) H erklärt e1, e2

- (5) (iv) H erklärt negatives Beispiel e3
d.h., wir entfernen c aus H.

Training examples		Background knowledge	
Daughter (mary, ann).	+	Parent (ann, mary).	Female (ann).
Daughter (eve, tom).	+	Parent (ann, tom).	Female (mary).
Daughter (tom, ann).	-	Parent (tom, eve).	Female (eve).
Daughter (eve, ann).	-	Parent (tom, ian).	

daughter(X, Y) ←
steht für
daughter(X, Y) ← true

Kapitel VI.2: Induktives logisches Programmieren

(7)

(v) verfeinere $\text{daughter}(X, Y)$ mit

$\varphi(c) = \{\text{daughter}(X, Y) \leftarrow L \mid L \in \{ X = Y, \text{female}(X), \text{female}(Y),$

$\text{parent}(X, X), \text{parent}(X, Y),$

$\text{parent}(Y, X), \text{parent}(Y, Y),$

$/* \text{ (Literale nur mit Variablen aus Klauselkopf) } */$

$\text{parent}(X, Z), \text{parent}(Z, X),$

$\text{parent}(Y, Z), \text{parent}(Z, Y) \} \}$

$/* \text{ (Literale mit neuer Variable } Z \text{ (} Z \neq X, Z \neq Y \text{);$

Literale müssen wenigstens eine Variable aus
Klauselkopf enthalten) $*/$

$\text{daughter}(X, Y) \leftarrow (X = Y)$ erklärt nicht e1, wird also übergangen.

Kapitel VI.2: Induktives logisches Programmieren

$c' = \text{daughter}(X, Y) \leftarrow \text{female}(X)$

- Ende von (7)
-
- (4) (vi) • erklärt e1,
- (4) (vii) • erklärt e2,
- (4) (viii) • ist konsistent mit e3
- (2+5) (viii) • ist inkonsistent mit e4, d.h. ein neg. Beispiel wird erklärt
- (ix) ...
- (ω) Die Verfeinerung von c' :
- $\text{daughter}(X, Y) \leftarrow (\text{female}(X), \text{parent}(Y, X))$
- erklärt e1, e2 und
 - ist konsistent mit e3, e4.

Ende.

Bemerkung: Sofern möglich wird die Einführung *neuer* Variablen *vermieden* !

D.h. man verwendet bevorzugt Literale aus der ersten

Menge aus Literalen.

Kapitel VI.2: Induktives logisches Programmieren

Generisches Top-down Verfahren

- basiert auf Top-Down-Suche in Refinement-Graphen (wie MIS); umfasst FOIL und weitere.

- *grundlegende Terminologie:*

(i) Eine Hypothese ist eine Menge von Programmklauseln , d.h. $H \in \mathcal{P}_{\text{fin}}(L)$

(ii) Menge E von Beispielen: $E = E^+ \cup E^-$

$$|E| = n, |E^+| = n^+, |E^-| = n^-$$

(iii) aktuelle Klausel hat Form $T \leftarrow Q$

mit T atomare Formel $p(X_1, \dots, X_n)$, p Zielprädikat

und Q Konjunktion von Literalen L_1, \dots, L_m

(iv) aktuelle Menge E_{cur} von Trainingsbeispielen

zur Konstruktion der aktuellen Klausel c

$$|E_{\text{cur}}| = n_{\text{cur}}, E_{\text{cur}} = E_{\text{cur}}^+ \cup E_{\text{cur}}^-$$

Kapitel VI.2: Induktives logisches Programmieren

Generell besteht das Verfahren aus *drei Schritten*:

- (1) Vorverarbeitung der Trainingsbeispiele
- (2) Konstruktion einer Hypothese
- (3) Nachverarbeitung der Hypothese

(1) Vorverarbeitung (vgl. Kap. IV)

- Handhabung *fehlender* Werte (missing values)
- sofern negative Beispiele fehlen, *Generierung negativer* Beispiele, z.B. unter Verwendung der *Closed-World-Assumption*:
Alle Beispiele, die nicht als positive Beispiele gegeben sind, werden generiert und als negative Beispiele gekennzeichnet.

(2) Konstruktion einer Hypothese H

- *covering*-Schleife:

- Konstruiere neue Klausel c
- füge c in H ein
- entferne aus der Menge der aktuellen Trainingsbeispiele alle Beispiele, die von c erklärt werden

- *specialization*-Schleife:

- aktuelle Klausel $c = T \leftarrow Q$ wird verfeinert zu

$$c' = T \leftarrow Q, L \quad (L \text{ Literal}) \text{ mit } c' \in \varphi(c)$$

- lokal beste Verfeinerung wird jeweils ausgewählt (*Hill-climbing* Verfahren)
- verschiedene *Heuristiken* anwendbar für Bewertung der Klauseln (vgl. C4.5)

Kapitel VI.2: Induktives logisches Programmieren

Algorithm VI.2.2 (generic top-down ILP algorithm) (Lavrac/Dzeroski 1994)

```
Initialize  $\varepsilon_{cur} := \varepsilon$ .
Initialize  $H := \emptyset$ .
repeat {covering}
  Initialize  $c := T \leftarrow$ .
  repeat {specialization}
    Find the best refinement  $c_{best} \in \varphi(c)$ .
    Assign  $c := c_{best}$ .
  until Necessity stopping criterion is satisfied.
  Add  $c$  to  $H$  to get new hypothesis  $H' := H \cup \{c\}$ .
  Remove positive examples covered by  $c$  from  $\varepsilon_{cur}$  to get new
  training set  $\varepsilon'_{cur} := \varepsilon_{cur} - covers(B, H', \varepsilon_{cur}^+)$ .
  Assign  $\varepsilon_{cur} := \varepsilon'_{cur}$ ,  $H := H'$ .
until Sufficiency stopping criterion is satisfied.

Output: Hypothesis  $H$ .
```

Kapitel VI.2: Induktives logisches Programmieren

- *necessity* criterion:

Konsistenz, d.h. c erklärt kein negatives Beispiel

- *sufficiency* criterion:

Vollständigkeit, d.h. H erklärt E⁺

(3) Nachverarbeitung

- Nachverarbeitung beinhaltet die Entfernung *irrelevanter Literale* aus Klauseln, sowie die Entfernung *irrelevanter Klauseln* aus der Hypothese (vgl. die erzeugten Regeln bei C4.5).

Kapitel VI.2: Induktives logisches Programmieren

- Definition (*Irrelevanz eines Literals*): Sei $c \in H$, und die Klausel d entstehe aus c durch Entfernen eines Literals L aus dem Körper von c . L ist irrelevant in c , wenn

$$\text{covers}(B, H \cup \{d\} \setminus \{c\}, E^-) \subseteq \text{covers}(B, H, E^-)$$

d.h. die Anzahl der abgedeckten negativen Beispiele nimmt nicht zu.

- Definition (*Irrelevanz einer Klausel*): Eine Klausel $c \in H$ ist irrelevant, wenn:

$$\text{covers}(B, H, E^+) \subseteq \text{covers}(B, H \setminus \{c\}, E^+)$$

Kapitel VI.2: Induktives logisches Programmieren

VI.2.4 Der ILP Algorithmus Foil

(Lavrač/Dzeroski 1994) (siehe auch Quinlan 1990)

VI.2.4.1 Überblick

- Foil ist eine konkrete Realisierung des in VI.2.3 beschriebenen generischen Top-down Verfahrens (spezialisierendes ILP-Verfahren)
- Foil gehört zur Klasse der empirischen ILP Systeme (empirical ILP system):
 - alle Trainingsbeispiele müssen im vorhinein gegeben sein (batch learner)
 - kein Orakel verfügbar, das Generalisierungen bzw. Spezialisierungen validiert (non-interactive)
 - einzelne Konzeptbeschreibung wird gelernt (single predicate learner)
 - keine Hypothesen dürfen vorgegeben sein (learning from scratch)

- Merkmale von Foil

- Konzeptbeschreibungssprache L ist beschränkt auf funktionsfreie ProgrammklauseIn
 - Literale im Rumpf einer Klausel haben entweder ein Prädikatensymbol q_i aus dem Hintergrundwissen B oder das Prädikatensymbol des Zielprädikats
 - ⇒ rekursive Klauseln können gelernt werden
 - wenigstens eine Argumentvariable eines Rumpf-Literals muß im Kopf der Klausel oder in einem weiter links stehenden Literal auftreten
 - Literale, die eine Variable an eine Konstante binden, sind nicht erlaubt

Kapitel VI.2: Induktives logisches Programmieren

d.h. variablenfrei

- Trainingsbeispiele sind als funktionsfreie Grundfakten (function-free ground facts) gegeben
- Hintergrundwissen B besteht aus extensionalen Prädikatendefinitionen, d.h. aus einer endlichen Menge von funktionsfreien Grundfakten
 - Hintergrundwissen führt Prädikatensymbole q_i ($i = 1, \dots, n$) ein
 - Extension der Prädikate sind Tupel von Konstanten
- zentraler Bestandteil von Foil ist die Konstruktion einer Hypothese
 - ⇒ setzt geeignete Vorverarbeitung der Trainingsbeispiele voraus

Vorverarbeitung der Trainingsbeispiele

- sofern negative Beispiele fehlen:

negative Beispiele werden unter Verwendung der Closed-World-Assumption (CWA) generiert:

alle Beispiele, die nicht als positive Trainingsbeispiele gegeben sind, werden generiert und als negative Trainingsbeispiele gekennzeichnet

- Bemerkung:

Trainingsbeispiele müssen in einem bestimmten Datenformat vorliegen, damit sie von Foil verarbeitet werden können

VI.2.4.2 Konstruktion einer Hypothese

(i) Covering-Teil („outerloop“)

- Hypothese besteht aus einer endlichen Menge von Klauseln (die durch Disjunktion verbunden sind).
- Klauseln werden der Reihe nach gelernt und zu der bereits gebildeten Hypothese hinzugefügt.
- positive Trainingsbeispiele, die von der zuletzt gelernten Klausel abgedeckt werden, werden aus der Menge der Trainingsbeispiele ε_{cur} entfernt, ehe die nächste Klausel gelernt wird.
- Klauselbildung bricht ab, sobald alle positiven Trainingsbeispiele von einer Klausel erklärt werden (sofern die Daten nicht verrauscht sind).

Kapitel VI.2: Induktives logisches Programmieren

Algorithm 4.1 (FOIL – the covering algorithm)

Initialize $\mathcal{E}_{cur} := \mathcal{E}$.
Initialize $\mathcal{H} := \emptyset$.
repeat {covering}
 Initialize clause $c := T \leftarrow$.
 Call the *SpecializationAlgorithm*(c, \mathcal{E}_{cur})
 to find a clause c_{best} .
 Assign $c := c_{best}$.
 Post-process c by removing irrelevant literals to get c' .
 Add c' to \mathcal{H} to get a new hypothesis $\mathcal{H}' := \mathcal{H} \cup \{c'\}$.
 Remove positive examples covered by c' from \mathcal{E}_{cur} to get
 a new training set $\mathcal{E}'_{cur} := \mathcal{E}_{cur} - covers_{ext}(\mathcal{B}, \{c'\}, \mathcal{E}_{cur}^+)$.
 Assign $\mathcal{E}_{cur} := \mathcal{E}'_{cur}, \mathcal{H} := \mathcal{H}'$.
until $\mathcal{E}_{cur}^+ = \emptyset$ or encoding constraints violated.
Output: Hypothesis \mathcal{H} .

Algorithmus VI.2.3: FOIL - the covering algorithm (Lavrač / Dzeroski 1994)

(ii) Specialization-Teil (specialization algorithm - „inner loop“)

- Es werden Klauseln der Form

$$p(X_1, \dots, X_n) \leftarrow (L_1, \dots, L_m)$$

gebildet.

- Die Suche nach Klauseln startet mit der allgemeinsten Klausel

$$p(X_1, \dots, X_n) \leftarrow \quad (p(X_1, \dots, X_n) \text{ ist Zielprädikat}),$$

die schrittweise verfeinert wird.

- Spezialisierungsschritt fügt der bisher gebildeten Klausel

$$c_i = p(X_1, \dots, X_n) \leftarrow (L_1, L_2, \dots, L_{i-1})$$

ein neues Rumpf-Literal L_i hinzu; L_i ist atomare Formel A oder deren Negation $\neg A$

Kapitel VI.2: Induktives logisches Programmieren

A ist atomare Formel der Form

(1) $X_j = X_s$ mit X_j, X_s Variablen in c_i ,

(2) $q_k (Y_1, \dots, Y_{n_k})$ mit

- q_k Prädikatsymbol aus B

- Y_j ist Variable in c_i oder neue existentiell quantifizierte Variable ($j = 1, \dots, n_k$); wenigstens eine Variable Y_j muß in c_i sein;

(3) $p(Z_1, \dots, Z_n)$ mit gewissen Einschränkungen an die Variablen Z_j zur Vermeidung unendlicher Rekursionen

- Spezialisierungsschritt verwendet lokale Beispielmenge ε_i (local training set),

- die aus m-Tupeln von Konstanten besteht, wenn c_i m verschiedene Variablen enthält.

- ε_{i+1} ist der natürliche Verbund (natural join) von ε_i mit der zu L_i gehörenden Relation.

Kapitel VI.2: Induktives logisches Programmieren

Algorithm 4.2 (FOIL - the specialization algorithm)

Initialize $i := 1$.

Initialize local training set $\mathcal{E}_i := \mathcal{E}_{cur}$.

Initialize current clause $c_i := c$.

while $\mathcal{E}_i^- \neq \emptyset$ or encoding constraints not violated do

 Find the best literal L_i to add to the body of $c_i = T \leftarrow Q$

 and construct $c_{i+1} := T \leftarrow Q, L_i$.

 Form a new local training set \mathcal{E}_{i+1} as a set of extensions
 of the tuples in \mathcal{E}_i that satisfy L_i .

 Assign $c := c_{i+1}$.

 Increment i .

endwhile.

Output : Clause c .

Algorithmus VI.2.4:FOIL - the specialization algorithm

(Lavrač / Dzeroski 1994)

Kapitel VI.2: Induktives logisches Programmieren

Beispiel: Lernen der Beschreibung des Zielprädikates daughter (X, Y)
(vgl. Abschnitt VI.2.1)

- Spezialisierungsschritt startet mit allgemeiner Klausel
 $c_1 = \text{daughter}(X, Y) \leftarrow$
 $\varepsilon_1 = \varepsilon_c$ (4 Trainingsbeispiele)
- Spezialisierungsschritt verfeinert c_1 zu
 $c_2 = \text{daughter}(X, Y) \leftarrow \text{female}(X)$
 $\varepsilon_2 =$ Menge aller Trainingsbeispiele, die durch c_2
erklärt werden
- Spezialisierungsschritt verfeinert c_2 zu
 $c_3 = \text{daughter}(X, Y) \leftarrow \text{female}(X), \text{parent}(X, Y)$
 ε_3 enthält nur noch positive Trainingsbeispiele,
damit: konsistente neue Klausel c_3 gefunden

Bemerkung: Spezialisierungsschritte repräsentieren „inner loop“ aus
FOIL-Algorithmus

Kapitel VI.2: Induktives logisches Programmieren

Training Examples		Background Knowledge	
Daughter (mary, ann)	+	Parent (ann,mary)	Female (ann)
Daughter (eve,tom)	+	Parent(ann, tom)	Female (mary)
Daughter (tom,ann)	-	Parent (tom,eve)	Female (eve)
Daughter (eve,ann)	-	Parent (tom,ian)	

Tabelle VI.2.1: Ein einfaches ILP-Problem (Lavrac, Dzeroski 1994)

<i>Current clause c_1 : daughter(X, Y) ←</i>			
\mathcal{E}_1	(mary, ann) ⊕		$n_1^{\oplus} = 2$ $I(c_1) = 1.00$
	(eve, tom) ⊕		$n_1^{\ominus} = 2$
	(tom, ann) ⊖	$L_1 = female(X)$	
	(eve, ann) ⊖	$Gain(L_1) = 0.84$	$n_1^{\oplus\oplus} = 2$
<i>Current clause c_2 : daughter(X, Y) ← female(X)</i>			
\mathcal{E}_2	(mary, ann) ⊕		$n_2^{\oplus} = 2$ $I(c_2) = 0.58$
	(eve, tom) ⊕		$n_2^{\ominus} = 1$
	(eve, ann) ⊖	$L_2 = parent(Y, X)$	
		$Gain(L_2) = 1.16$	$n_2^{\oplus\oplus} = 2$
<i>Current clause c_3 : daughter(X, Y) ← female(X), parent(Y, X)</i>			
\mathcal{E}_3	(mary, ann) ⊕		$n_3^{\oplus} = 2$ $I(c_3) = 0.00$
	(eve, tom) ⊕		$n_3^{\ominus} = 0$

Tabelle VI.2.2: FOIL trace for the family relation problem (Lavrac / Dzeroski 1994)

Kapitel VI.2: Induktives logisches Programmieren

- Sofern Spezialisierung ein Literal mit neuen Variablen einführt, erhöht sich die Stelligkeit der Tupel in der lokalen Trainingsmenge; ferner kann sich die Anzahl der Tupel erhöhen

Bsp.: daughter (X, Y)

- wähle im Spezialisierungsschritt

$L_1 = \text{parent}(Y, Z)$

- damit $c_2 = \text{daughter}(X, Y) \leftarrow \text{parent}(Y, Z)$
- damit Tupel in ε_2 dreistellig; ferner verdoppelt sich Anzahl der Tupel, da jedes Tupel aus ε_1 mit 2 parent-Tupeln aus B gejoined werden kann.

Kapitel VI.2: Induktives logisches Programmieren

<i>Current clause $c_1 : daughter(X, Y) \leftarrow$</i>			
\mathcal{E}_1	$(mary, ann)$	\oplus	$n_1^{\oplus} = 2$
	(eve, tom)	\oplus	$n_1^{\ominus} = 2$
	(tom, ann)	\ominus	$L_1 = parent(Y, Z)$
	(eve, ann)	\ominus	$n_1^{\oplus\oplus} = 2$
<i>Current clause $c_2 : daughter(X, Y) \leftarrow parent(Y, Z)$</i>			
\mathcal{E}_2	$(mary, ann, mary)$	\oplus	$n_2^{\oplus} = 4$
	$(mary, ann, tom)$	\oplus	
	(eve, tom, eve)	\oplus	
	(eve, tom, ian)	\oplus	
	$(tom, ann, mary)$	\ominus	$n_2^{\ominus} = 4$
	(tom, ann, tom)	\ominus	
	$(eve, ann, mary)$	\ominus	
	(eve, ann, tom)	\ominus	

Tabelle VI.2.3: The effect of new variables on the current training set of tuples in FOIL (Lavrac / Dzeroski 1994)

Kapitel VI.2: Induktives logisches Programmieren

- Auswahl des Literals L_i im Spezialisierungsschritt für Klausel c_i :
 - Grundlage ist Maß für gewichteten Informationsgewinn (weighted information gain) (vgl. Kapitel über C4.5)
 - Bezeichnung:

n_i : Anzahl Tupel in lokaler Trainingsmenge ε_i

n_i^+ : Anzahl positiver Tupel

n_i^- : Anzahl negativer Tupel

- Informationsgehalt einer Klausel: Informationsgehalt, um mitzuteilen, daß ein Tupel aus ε_i bzw. ε_{i+1} positiv ist:

$$I(c_i) = -\log_2\left(\frac{n_i^+}{n_i}\right) = -\log_2\left(\frac{n_i^+}{n_i^+ + n_i^-}\right)$$

$$I(c_{i+1}) = -\log_2\left(\frac{n_{i+1}^+}{n_{i+1}}\right)$$

Kapitel VI.2: Induktives logisches Programmieren

- Sei n_i^{++} Anzahl positiver Tupel in Trainingsmenge ε_i , die auch in Trainingsmenge ε_{i+1} enthalten sind (gegebenenfalls mit höherer Stelligkeit); damit ist gewichteter Informationsgewinn definiert als

$$\text{Gain}(L_i) = n_i^{++} \cdot (I(c_i) - I(c_{i+1}))$$

Beispiel: Klauseln $c_1 = \text{daughter}(X, Y)$

$c_2 = \text{daughter}(X, Y) \leftarrow \text{female}(X)$

$$\left. \begin{array}{l} I(c_1) = -\log_2\left(\frac{2}{2+2}\right) = 1.0 \\ I(c_2) = -\log_2\left(\frac{2}{2+1}\right) = 0.58 \end{array} \right\} \text{ sowie } n_1^{++} = 2$$

liefert

$$\text{Gain}(\text{female}(X)) = 2 \cdot (I(c_1) - I(c_2)) = 2 \cdot 0.42 = 0.84$$

VI.2.4.3 Weitere Aspekte von FOIL

a) Verrauschte Daten

- Bisher beschriebener Algorithmus setzt voraus, daß Trainingsbeispiele nicht verrauscht sind (noise-free)
- bei verrauschten Daten wird ein Maß für die maximale Länge einer Klausel verwendet, um die Spezialisierung von Klauseln abubrechen, sog. necessity stopping criterion:

encoding length restriction: „Beschreibungslänge einer Klausel muß kleiner sein als die Beschreibungslänge der Daten, die die Klausel erklärt“

- in gleicher Weise werden keine neuen Klauseln mehr generiert (sufficiency stopping criterion), wenn nur noch Klauseln generiert werden können, die die encoding length restriction verletzen
- Nachteil: Heuristik erlaubt die Bildung komplexer Klauseln, die nur wenige positive Beispiele erklären

b) Nachverarbeitung der Hypothese

- FOIL verwendet eine zu C4.5 ähnliche Strategie, um Klauseln zu vereinfachen:
 - jede Regel wird einzeln solange vereinfacht, bis eine weitere Verfeinerung die geschätzte Fehlerrate erhöhen würde
 - vereinfachte Regeln werden entsprechend ihrer Fehlerrate sortiert und in der Reihenfolge zur Klassifizierung neuer Beispiele verwendet