Hauptseminar
Machine Learning

# Inductive Logic Programming
# Relational Data Mining

Christoph Petzinger

WS 2003/2004

# Inhaltsverzeichnis

# 1 Relational Data Mining

Data Mining Systeme haben unter anderem die Aufgabe bisher unbekannte Zusammenhänge und Muster in grossen Datenmengen zu erkennen. Dabei wurden die meisten Data Mining Methoden dazu entwickelt, Daten zu analysieren, welche in Matrix Form vorliegen. Dabei stellen die Zeilen z.B. einzelne Beobachtungen und Messungen und die Spalten Variablen dar.

Diese Art der Darstellung, welche häufig in der Statistik eingesetzt wird, ist allerdings für viele Probleme ungeeignet. Dies ist dann der Fall, wenn verschiedene Objekt-Typen und Beziehungen zwischen diesen Typen existieren. Die Daten liegen also nicht in Matrix-Form vor sondern sind in einer relationalen Datenbank mit mehreren Tabellen gespeichert.

Eine Lösung wäre es nun Daten aus mehren Tabellen in einer Tabelle zu vereinen und dann klassische Data Mining Werkzeuge darauf anzuwenden. Durch diese Vereinigung kann es allerdings zum Verlust von Informationen kommen, was dieses Verfahren für viele Fälle uninteressant macht.

Einen anderen Weg gehen nun die ILP (Inductive Logic Programming) Ansätze, welche direkt auf die multi-relationalen Datenbanken angewandt werden können. ILP kombiniert dabei Methoden des induktiven maschinellen Lernens mit Methoden der Logikprogrammierung, wobei das Ziel das Erlernen von relationalen Beschreibungen in Form von Logikprogrammen ist.

Neben den Grundlagen der Induktiven Logikprogrammierung sollen im Rahmen dieser Ausarbeitung exemplarisch die beiden ILP-Systeme FOIL und TILDE vorgestellt werden.

# 2 Inductive Logic Programming

Die von ILP-Systemen erlernten relationalen Beschreibungen werden üblicherweise anhand von Logikprogrammen dargestellt, welche eine wichtige Teilmenge der Prädikatenlogik (*first-order logic*) sind. Dieser prädikatenlogische Formalismus ist viel ausdrucksstärker als die Attribut-Wert-Darstellung herkömmlicher Data Mining Methoden.

Gelernt wird eine Zielrelation $p$ mit Hilfe von Trainingsbeispielen, welche bereits dahingehend klassifiziert sind, ob sie zur Zielrelation gehören oder nicht. Dabei werden diejenigen Beispiele, die zur Zielrelation gehören, als *positiv* bezeichnet. Beispiele die nicht zur Zielrelation gehören sind *negative* Beispiele. Der Lernprozess kann ausserdem durch Hintergrundinformationen $q_i$, sowie durch eine Sprachvorgabe (*language bias*), welche diejenigen Sprachelemente festlegt, die in der Hypothese $H$ zur Beschreibung von $p$ verwendet werden können, gezielt beeinflusst werden. Die Hintergrundinformationen liegen dabei ebenfalls in Form eines Logikprogramms vor. Aufgabe des ILP-Systems ist es nun eine *konsistente* und *vollständige* Definition von $p$ zu finden.

Formell lässt sich dies folgendermassen darstellen:
Für eine Menge von Beispielen, $E = P \cup N$, wobei $P$ die Mengen von positiven und $N$ die Menge von negativen Beispielen ist, sowie eine Menge von Hintergrundinformationen $B$ wird eine Hypothese $H$ gesucht so dass gilt:

$$\forall e \in P : B \wedge H \vDash e \text{ ($H$ ist vollständig)}$$
$$\forall e \in N : B \wedge H \nvDash e \text{ ($H$ ist konsistent)}$$

Die Vollständigkeit und Konsistenz einer Hypothese wird am besten durch eine Funktion $covers()$ überprüft:

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

Die Funktion $covers(H, e)$ kann erweitert werden, um eine Menge von Beispielen zu verarbeiten. Diese Funktion gibt die Menge von Beispielen aus $E$ zurück, welche von $H$ erklärt werden:

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

Eine Hypothese $H$ ist also *Vollständig* bezüglich den Beispielen $E$, wenn sie alle *positiven* Beispiele beschreibt, also wenn gilt:

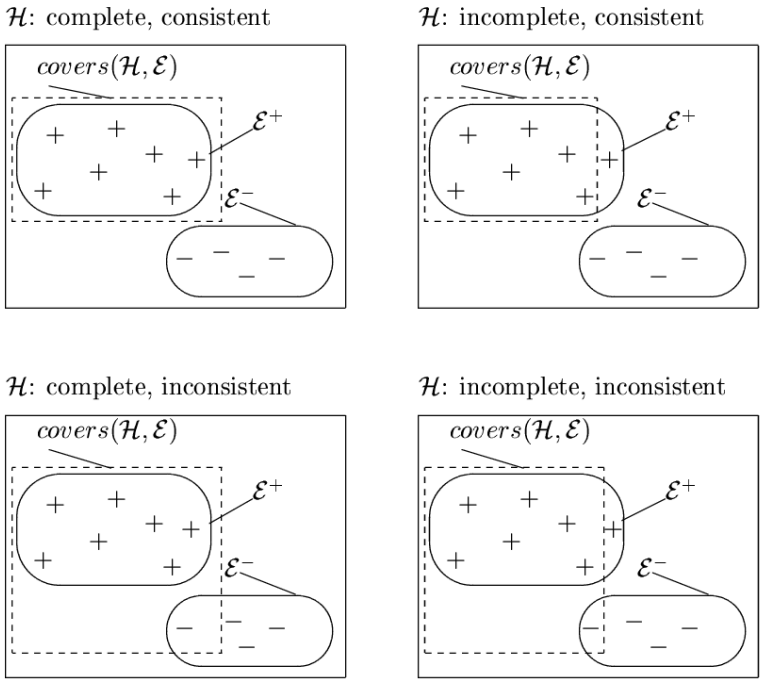
$$covers(H, E^+) = E^+$$

Sie ist *Konsistent* bezüglich den Beispielen $E$, wenn sie keine *negativen* Beispiele beschreibt, also wenn gilt:

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

Folgende Darstellung zeigt mögliche Situationen, welche die Vollständigkeit und Konsistenz der Hypothese $H$ betreffen: Soll zudem auch noch die Menge Hintergrundinforma-

Abbildung 2.1: Vollständigkeit und Konsistenz




Completeness and consistency of a hypothesis.

tionen $B$ in den Lernprozess eingebunden werden, so erhält man eine weitere Erweiterung der $covers()$ Funktion:

$$covers(B, H, e) = covers(B \cup H, e)$$

$$covers(B, H, E) = covers(B \cup H, E)$$

Tabelle 2.1: *daughter* Beispiel

| *Training examples* | | *Background knowledge* | |
|---|---|---|---|
| $daughter(mary, ann).$ | $\oplus$ | $parent(ann, mary).$ | $female(ann).$ |
| $daughter(eve, tom).$ | $\oplus$ | $parent(ann, tom).$ | $female(mary).$ |
| $daughter(tom, ann).$ | $\ominus$ | $parent(tom, eve).$ | $female(eve).$ |
| $daughter(eve, ann).$ | $\ominus$ | $parent(tom, ian).$ | |

A simple ILP problem: learning the *daughter* relation.

Als kurzes Beispiel sei die Aufgabe gestellt, die Zielrelation $daughter(X, Y)$, also „X ist die Tochter von Y“, zu erlernen. Es sind jeweils 2 positive und negative Beispiele, sowie Hintergrundinformationen über die Beziehungen $female$ und $parent$ gegeben. Als vollständige und konsistente Lösung wäre dann die Zielrelation

$$daughter\,(X, Y) \leftarrow female\,(X)\,, parent\,(Y, X)$$

möglich.

## 2.1 Prädikatenlogik

Im vorherigen Abschnitt wurde die Zielrelation bereits als Klausel der Pradikatenlogik dargestellt. Eine Klausel besteht aus einem *Kopf* und einem *Körper*, wobei der Kopf die Folgerung aus dem Bedingungsteil, dem Körper der Klausel, ist. Kopf und Körper einer Klausel bestehen jeweils aus einer *Menge von Atomen*, und ein Atom ist ein Prädikat gefolgt von einem Tupel von Termen. Terme sind Variablen oder Funktionen gefolgt von einem Tupel von Termen.

$$\underbrace{\overbrace{father}^{Prädikat}\,(X, Y) \vee \overbrace{mother\,(X, Y)}^{Atom}}_{Kopf \widehat{=} Folgerung} \leftarrow \underbrace{parent\,(X, Y)}_{Körper \widehat{=} Bedingung}$$

Beispiel für eine *Klausel* - $X, Y$ sind Variablen
„*if* X is a parent of Y *then* X is the father of Y *or* X is the mother of Y“

Klauseln lassen sich auf verschiedene Arten darstellen, wobei letztere Darstellung aufgrund ihrer Einfachheit bevorzugt wird:

$$\forall X_1 \forall X_2 \ldots \forall X_s \left(A_1 \vee A_2 \vee \ldots \vee A_h \vee \overline{B_1} \vee \overline{B_2} \vee \ldots \vee \overline{B_b}\right)$$

$$\left(A_1 \vee A_2 \vee \ldots \vee A_h \vee \overline{B_1} \vee \overline{B_2} \vee \ldots \vee \overline{B_b}\right)$$

$$\left\{A_1, A_2, \ldots, A_h, \overline{B_1}, \overline{B_2}, \ldots, \overline{B_b}\right\}$$

$$A_1 \vee A_2 \vee \ldots \vee A_h \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_b$$

$$A_1, A_2, \ldots, A_h \leftarrow B_1, B_2, \ldots, B_b$$

Für die Logikprogrammierung gibt es noch eine Reihe wichtiger Sonderformen von Klauseln:

**Horn Klausel** - Sie hat *höchstens 1* Atom im Kopf

**definite Klausel** - Sie hat *genau 1* Atom im Kopf

$$ancestor\,(X, Y) \leftarrow parent\,(Z, Y)\,, ancestor\,(X, Z)$$

**Programm Klausel** - Sie hat *genau 1* Atom im Kopf und es sind *negierte* Atome im Körper erlaubt

$$mother\,(X, Y) \leftarrow parent\,(X, Y)\,, not\ male\,(X)$$

**fact** - Tatsachen haben *genau 1* Atom im Kopf und einen *leeren* Körper

$$parent\,(mother\,(X)\,, X)$$

**ground fact** - wie Tatsachen; Es kommen keinerlei Variablen vor

$$daughter\,(mary, ann)$$

Klauseln können zusammengefasst werden, und so bildet ein Menge von Programm Klauseln ein Logikprogramm. Die zu lernende Zielrelation hat die Form einer Prädikat Definition, welche aus einer Menge von Programm Klauseln besteht, die das selbe Prädikat im Kopf haben. Die Hintergrundinformationen bestehen aus den beiden Tatsachen (*facts*) Arten, und sind stets gültig.

## 2.2 covering approach

Für ILP-Systeme ist der *covering approach* zur Induktion von Klauseln sehr weit verbreitet. Der covering Algorithmus besteht aus zwei verschachtelten Schleifen.

In einer *Hauptschleife* konstruiert der Algorithmus eine Hypothese. Beginnend mit einer leeren Menge fügt er Klauseln zur Hypthese hinzu, welche einige der positiven aber keine negativen Beispiele beschreiben. Beispiele die von einer Klausel beschrieben werden, werden aus der Beispielmenge entfernt. Der Algorithmus stoppt und gibt eine vollständige Hypothese zurück, wenn keine positiven Beispiele mehr in der Beispielmenge vorhanden sind.

In der *inneren Schleife* wird eine Klausel aus dem Suchraum aller möglichen Klauseln ausgewählt. Dieser Suchraum wird durch Generalisierungs- und Spezialisierungs-Operatoren auf eine Ordnung abgebildet. Die Grösse des Raumes wird dabei vor allem durch die Sprachvorgaben und Hintergrundinformationen bestimmt, und es wird versucht, einen grossen Suchraum durch geeignete Massnahmen zu verkleinern, um effizienter zu einem Ergebnis zu gelangen. So können zum Beispiel Schranken eingeführt werden oder auch Rekursionen und die Einführung neuer Variablen verhindert werden. Wie genau die Klauseln im Suchraum bewertet werden und wie die beste Klausel für die Hypothese ausgesucht wird, ist von dem jeweiligen ILP-System abhängig.

## 2.3 $\theta$-subsumption

Bei der Strukturierung des Suchraumes für die Auswahl einer geeigneten Klausel spielt die *$\theta$-subsumption* eine entscheidende Rolle. Die $\theta$-subsumption wendet dabei eine Substitution auf Klauseln an.

Bei Anwendung einer Substitution $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$ auf eine Klausel $F$ erhält man die Klausel $F\theta$, in dem alle Vorkommnisse der Variablen $V_i$ durch den Term $t_i$ ersetzt werden, wie in folgendem Beispiel verdeutlicht wird.

$$c = daughter\,(X, Y) \leftarrow parent\,(Y, X)$$

$$\theta = \{X/mary, Y/ann\}$$

$$c\theta = daughter\,(mary, ann) \leftarrow parent\,(ann, mary)$$

Der geordnete Hypothesen-Raum ist ein gerichteter acyclischer Graph, wobei die Knoten Klauseln und die Kanten Operationen zur Verallgemeinerung bzw. Spezialisierung sind. Durch diese Ordnung ist es dann möglich den Graph systematisch zu durchsuchen.

Wenn $c$ und $c^{'}$ Klauseln sind, dann *$\theta$-subsumiert* $c\ c^{'}$, wenn es eine Substitution $\theta$ gibt, so dass gilt: $c\theta \subseteq c^{'}$ Das bedeutet für die Klausel:

- $c$ ist mindesten genau so allgemein wie $c^{'}$ $(c \leq c^{'})$
- $c$ ist ist eine Verallgemeinerung von $c^{'}$
- $c^{'}$ ist ist eine Spezialisierung von $c$

## 2.4 top-down Suche

Bei der Suche nach neuen Klauseln für die Hypothese gibt es zwei grundlegende Ansätze, welche beide auf der $\theta$-subsumption beruhen.

Bei der *top-down* Suche erfolgt die Suche von einer *allgemeinen* Klausel aus. Durch einen Spezialisierungs-Operator werden die jeweils nächsten spezielleren Klauseln gesucht, solange bis diese keine negativen mehr und mindestens ein positives Beispiel beschreibt.

Bei der *bottom-up* Suche erfolgt die Suche von der *speziellsten* Klausel aus, die noch ein positives Beispiel beschreibt. Durch einen Generalisierunga-Operator wird die jeweils nächste allgemeinere Klausel gesucht, solange diese keine negativen Beispiele beschreibt.

Im weiteren wird nur noch auf den top-down Ansatz eingegangen. Der Spezialisierungs-Operator hat dabei die Form:

$$p\left(c\right) = \left\{c^{'} | c^{'} \in L, c < c^{'}\right\}$$

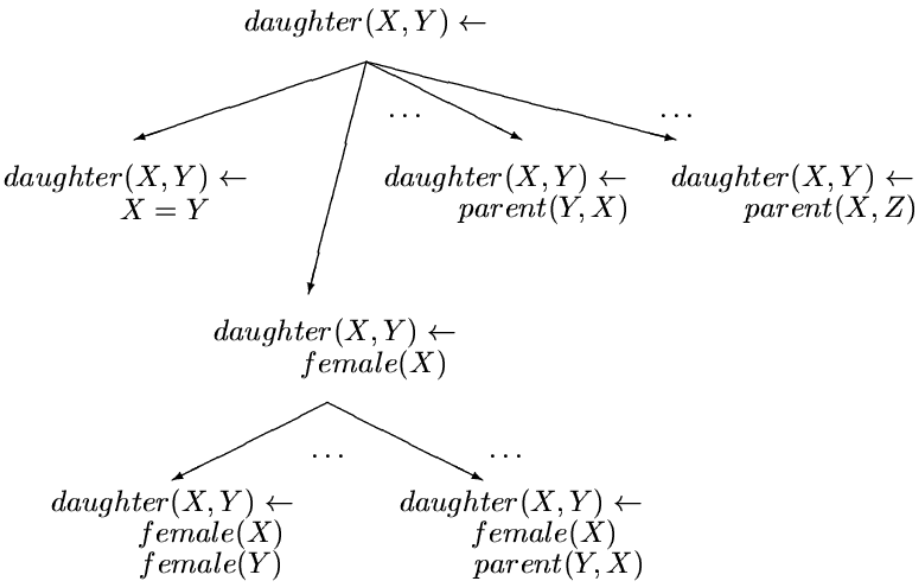
Abbildung 2.2: generischer top-down Algorithmus

**Algorithm 3.2 (Generic top-down ILP algorithm)**

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

**Output:** Hypothesis $\mathcal{H}$.

Abbildung 2.3: Teil eines Spezialisierungs-Graphen für das *daughter* Beispiel



# 3 Das ILP-System FOIL

## 3.1 Der Algorithmus

Das FOIL System baut auf dem generischen top-down Algorithmus auf. Der covering-Algorithmus bricht ab, sobald es keine positiven Beispiele mehr in den Trainingsdaten gibt, also alle bereits von Klauseln beschrieben werden, oder ein Fehler auftritt.

Abbildung 3.1: FOIL covering Algorithmus

---

**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 $\mathcal{H}$.

---

Der specialization Algorithmus beginnt mit einer leeren Klausel für die Zielrelation in der Form

$$P\,(X_1, X_2, \ldots, X_k) \leftarrow$$

Danach wird eine lokale Trainingsmenge $T_1$ mit der aktuellen Trainingsmenge initialisiert und der Zähler $i$ auf 1 gesetzt. Solange es negative Beispiele in $T_i$ gibt, wird nun in jedem Durchlauf mit Hilfe der Heuristik das beste Literal $L_i$ gefunden, welches am Körper der Klausel angehängt wird. Danach wird eine neue Trainingsmenge $T_{i+1}$ erstellt, welche aus denjenigen Beispielen aus $T_i$ besteht, die $L_i$ erfüllen. $i$ wird inkrementiert und ein neuer Durchlauf wird gestartet.

Abbildung 3.2: FOIL specialization Algorithmus

---

**Algorithm 4.2 (FOIL – the specialization algorithm)**

Initialize local training set $\mathcal{E}_i := \mathcal{E}_{cur}$.
Initialize current clause $c_i := c$.
Initialize $i := 1$.
**while** $\mathcal{E}_i^- \neq \emptyset$ or *encoding constraints 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 $c$.

---

## 3.2 Heuristik

Bei der Bewertung der Literale wird der Informationsgewinn $Gain(L_i)$ durch das Literal $L_i$ quantitativ errechnet. Dabei ist vor allem die Anzahl der positiven Beispiele in der jeweiligen Trainingsmenge vor und nach dem hinzufügen von $L_i$ zur Klausel ausschlaggebend. Folgende Berechnungen werden durchgeführt:

Informationsgehalt der Trainingsmenge $T_i$:

$$I(T_i) = -\log_2\left(T_i^+ / \left(T_i^+ + T_i^-\right)\right)$$

Informationsgehalt der Trainingsmenge $T_{i+1}$, falls $L_i$ gewählt wird:

$$I(T_{i+1}) = -\log_2\left(T_{i+1}^+ / \left(T_{i+1}^+ + T_{i+1}^-\right)\right)$$

Daraus errechnet sich der Informationsgewinn durch das Literal $L_i$:

$$Gain(L_i) = T_i^{++} \times (I(T_i) - I(T_{i+1}))$$

wobei $T_i^{++}$ die Anzahl der positiven Beispiele ist, die sowohl in $T_i$ als auch in $T_{i+1}$ sind.

## 3.3 Beispiel

Hier ist der komplette Durchlauf des FOIL Algorithmus für das *daughter* Beispiel gezeigt. Gut zu erkennen ist dabei, wie die einzige Klausel der Hypothese in 3 Durchläufen des specialization Algorithmus aufgebaut wird und der Informationsgehalt der Trainingsmenge abnimmt.

Tabelle 3.1: FOIL Beispiel

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

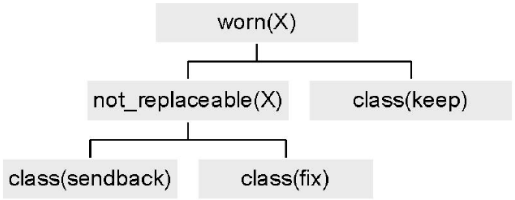
FOIL trace for the family relation problem.

# 4 Das ILP-System TILDE

## 4.1 Der Algorithmus

Das TILDE System beschreitet einen etwas anderen Weg als das FOIL System. Zwar benutzt es auch einen top-down Ansatz, allerdings werden keine Klauseln für eine Hypothese gesucht, sondern es wird ein Entscheidungsbaum aufgebaut. Dadurch ist eine Klassifizierung der Daten in mehr als zwei Klassen möglich, und nicht nur in positive und negative. Um ein Beispiel zu klassifizieren müssen alle Tests entsprechend dem Entscheidungsbaum abgearbeitet werden. Gut zu erkennen ist dies im direkten Vergleich des Baumes mit einem equivalenten Prolog Programm, das zeilenweise abgearbeitet wird bis ein Test erfüllt ist:
$class\,(sendback) : -worn\,(\_X)\,, not\_replaceable\,(\_X)\,, !.$
$class\,(fix) : -worn\,(\_X)\,, !.$
$class\,(keep)\,.$

Abbildung 4.1: TILDE Entscheidungsbaum




TILDE startet mit einem leeren Baum. für jede Ebene des Baumes werden alle mögliche Tests erzeugt und danach bewertet, wie gut sie die Klassen voneinander trennen. Der beste Test wird in den jeweiligen Knoten aufgenommen. Sobald ein Knoten nur Beispiele einer einzigen Klasse enthält, ist die Suche in diesem Pfad beendet.

Abbildung 4.2: TILDE Algorithmus

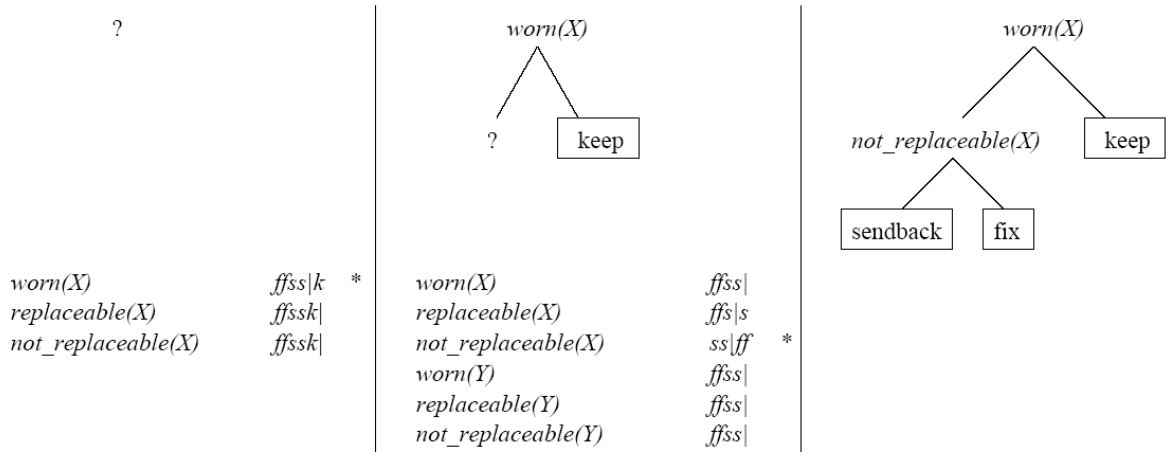
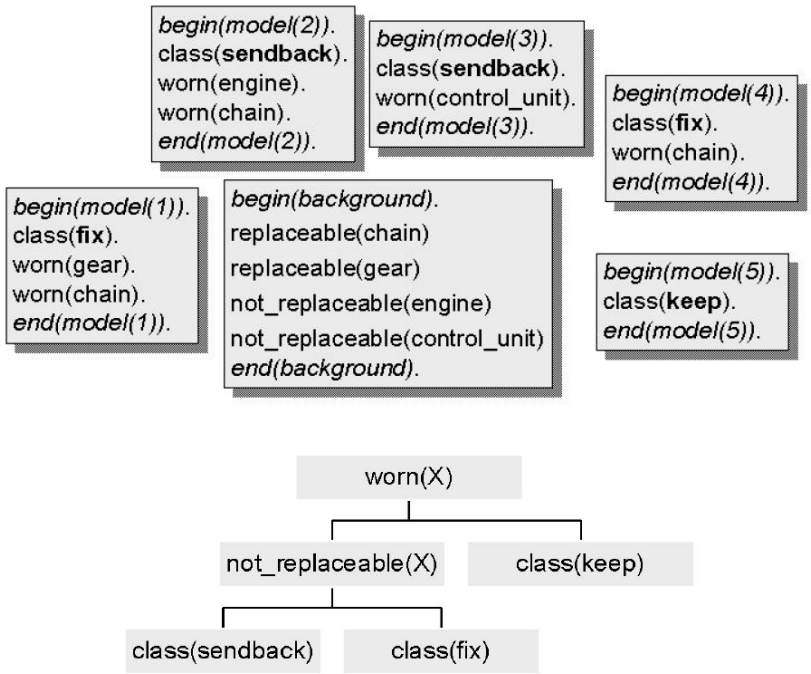



**Fig. 11.** TILDE illustrated on the WORN example. Each step shows the partial tree that has been built, the literals that are considered for addition to the tree, and how each literal would split the set of examples. E.g. *fss|k* means that of four examples, one with class `fix` and two with class `sendback` are in the left branch, and one example with class `keep` is in the right branch. The best literal is indicated with an asterisk.

## 4.2 Beispiel

Schliesslich nochmal der selbe Baum mit den zugehörigen Trainingsdaten:

Abbildung 4.3: TILDE Beispiel




The WORN example with three classes and a corresponding tree.