Inductive Logic Programming & Relational Data Mining

Christoph Petzinger

petzinge@in.tum.de

Überblick

1. Einführung
2. wichtige Begriffe der *first-order logic*
3. Anwendung von ILP
4. *covering approach*
5. Suchraum der clauses
6. *top-down* Suche
7. ILP-System FOIL
8. ILP-System TILDE

1. Einführung - relational data minin

- data mining Systeme: Auffinden von bisher unbekannten Zusammenhängen in grossen Datenmengen.
- klassischer Ansatz: Daten liegen in Matrix Form vor.
  Zeilen $\widehat{=}$ Messungen, Beobachtungen, Datensätze
  Spalten $\widehat{=}$ Attribute, Variablen
  attribute-value learning oder *propositional learning*
- relational data mining: Daten sind in einer Datenbank gespeichert.
  Dabei gibt es verschiedene Objekt-Typen (Tabellen) und Beziehungen zwischen diesen Typen.
  first-order learning oder *relational learning*

1. Einführung - ILP

- *preprocessing* ermöglicht Auswertung mit klassischen data mining Mitteln.
  $\Rightarrow$ Informationsverlust
- ILP Ansätze können direkt auf multi-relationale Daten angesetzt werden um Gesetzmässigkeiten zu finden
- ILP Systeme können bekannte Hintergrundinformationen benutzen.
  Diese liegen in Form eines *logic programs* vor.
- Erkannte Muster werden ebenfalls in Form von *logic programs* dargestellt.
  (Eine Teilmenge der *first-order / predicate logic*)

2. clausal theory

- *clausal theory* $\mapsto$ Menge von *clauses*
- *clause* $\mapsto$ *head* und *body*
- *head* und *body* $\mapsto$ Menge von *atoms*
- *atom* $\widehat{=}$ *predicate* mit Variablen und Funktionsaufrufen

$$\overbrace{\overbrace{father}^{predicate}\,(X,Y) \vee mother\,(X,Y)}^{head \widehat{=} conclusion} \leftarrow \overbrace{parent\,(X,Y)}^{body \widehat{=} condition}$$

Beispiel für eine *full clause* - 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"

2. alternative Schreibweisen

- *clauses* lassen sich auch in anderer Form darstellen:

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

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

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

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

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

2. logic program

- *Horn clause* = höchstens 1 *atom* im *head*
- *definite clause* = genau 1 *atom* im *head*

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

- *program clause* = genau 1 *atom* im *head* und *negated atoms* im *body* möglich

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

- *logic program* $\mapsto$ Menge von *program clauses*
- *predicate definition* $\mapsto$ Menge von *program clauses* mit dem selben *predicate* im *head*

2. fact

- *fact* = genau 1 *atom* im *head* und leerer *body*

$$parent\left(mother\left(X\right),X\right) \leftarrow \text{ oder } parent\left(mother\left(X\right),X\right)$$

- *ground fact* = es kommen keinerlei Variablen in der *clause* vor

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

3. Anwendung von ILP

- Gegeben ist eine Menge von bereits klassifizierten Beispielen:
  - positive Beispiele gehören zur *target relation* p
  - negative Beispiele gehören nicht zur *target relation* p
- Hintergrundinformationen q_i werden bei der Erlernung von p benutzt.
- Der *language bias* legt fest, welche *predicates* in der Hypothese H zur Beschreibung von p verwendet werden können.
- Aufgabe ist es nun eine *konsistente* und *vollständige* Definition von p zu finden.

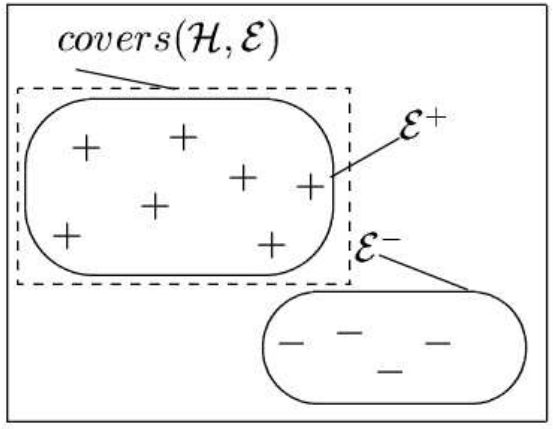
3. Anwendung der ILP - formell

- Menge an Beispielen: $E = P \cup N$
  P : positive Beispiele; N : negative Beispiele;
- Hintergrundinformation B
- gesucht: H so dass,

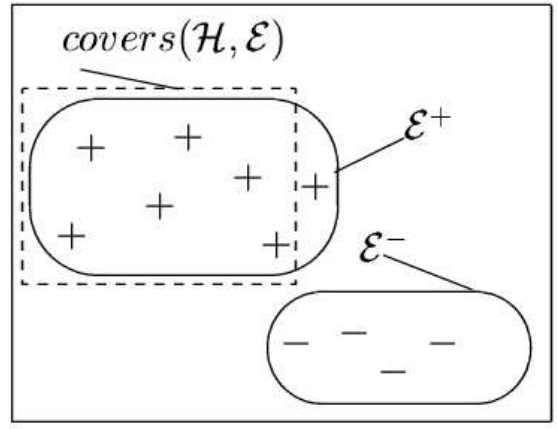
  $\forall e \in P : B \wedge H \models e$ (H ist vollständig)
  $\forall e \in N : B \wedge H \not\models e$ (H ist konsistent)
- Veranschaulichung durch eine Funktion
  $covers\,(H, B, E) = covers\,(H \cup B, E)$

3. Vollständigkeit / Konsistenz

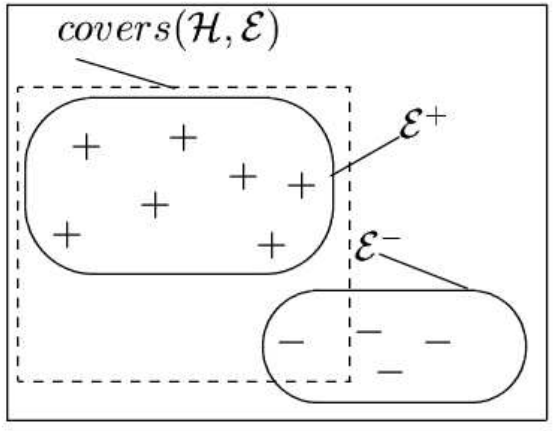
$\mathcal{H}$: complete, consistent



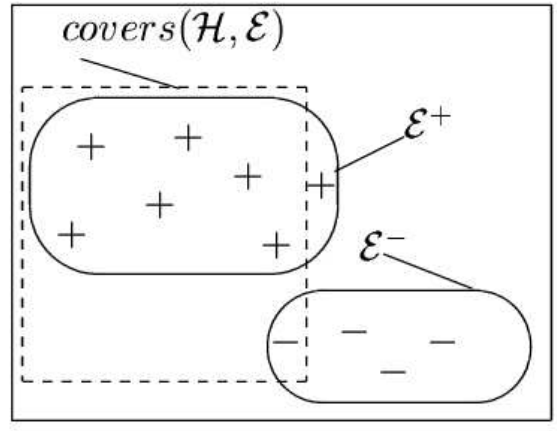
$\mathcal{H}$: incomplete, consistent



$\mathcal{H}$: complete, inconsistent



$\mathcal{H}$: incomplete, inconsistent



Completeness and consistency of a hypothesis.

3. Anwendung von ILP - 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.

- Es soll die Relation $daughter(X, Y)$ definiert werden.
- Hintergrundinformation: $female$ und $parent$
- Es gibt jeweil 2 positive und negative Beispiele für die *target relation* $daughter$
- Ergebnis:

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

4. covering approach

- Die meisten ILP Systeme benutzen ein solchen Algorithmus.
  (z.B. MIS, FOIL)
- In einer **Hauptschleife** konstruiert ein *covering* Algorithmus eine Menge von *clauses*. Beginnend mit einer leeren Hypothese fügt er *clauses* hinzu, die einige positive Beispiele abdecken. Diese Beispiele werden entfernt.

4. covering approach

- In einer **inneren Schleife** werden die einzelnen *clauses* aus dem Suchraum aller möglichen *clauses* ausgewählt. Dieser Raum ist durch Generalisierungs- oder Spezialisierungsoperatoren strukturiert.
- Die Art, wie die einzelnen clauses ausgewählt werden, ist von System zu Sytem verschieden.
- Die Hypothese ist vollständig, wenn alle positiven Beispiele abgedeckt sind.

4. covering approach

- Bei der Such nach neuen *clauses* für die Hypothese gibt es 2 grundlegende Ansätze, die beide auf der θ-*subsumtion* beruhen:
  - *top-down* Suche (FOIL, CLAUDIEN)

    Suche erfolgt von einer **allgemeinen** *clause* aus. Durch einen sog. *specialization / refinement operator* werden die nächsten spezielleren *clauses* gesucht, bis keine negativen Beispiele mehr beschrieben werden.
  - *bottom-up* Suche (GOLEM, CIGOL)

    Suche erfolgt von der **speziellsten** *clause* aus, die ein pos. Beispiel beschreibt. Durch einen sog. *generalization operator* werden allgemeinere *clauses* gesucht, solange keine neg. Beispiele beschrieben werden.

5. substitution

- *substitution* $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$
- Die Anwendung der *substituion* θ auf einen *term*, ein *atom* oder eine *clause* F, ergibt den *term*, das *atom* oder die *clause* $F\theta$.
- Alle Vorkommnisse der Variablen V_i werden dabei durch die Terme t_i ersetzt.
- Beispiel:

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

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

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

5. theta-subsumption

- Der Hypothesen-Raum der *clauses* ist ein gerichteter acyclischer Graph.
- Dabei sind Knoten *clauses*, und Kanten *refinement operations*.
- Um diesen Graph von *clauses* systematisch durchsuchen zu können ist es praktisch, wenn eine Ordnung vorliegt. Eine solche Ordnung basiert auf der θ-*subsumption*.
- c und c' sind clauses.
  c θ-*subsumes* c', wenn es eine Substitution θ gibt, so dass gilt:
  $c\theta \subseteq c'$

5. theta-subsumption

- Beispiel:
  $c = daughter\,(X, Y) \leftarrow parent\,(Y, X) =$
  $\left\{daughter\,(X, Y)\,, \overline{parent\,(Y, X)}\right\}$
  $\theta = \emptyset \;\Rightarrow\; c\theta = c$
  $c^{'} = daughter\,(X, Y) \leftarrow female\,(X)\,, parent\,(Y, X) =$
  $\left\{daughter\,(X, Y)\,, \overline{female\,(X)}, \overline{parent\,(Y, X)}\right\}$
- $c\theta \subseteq c^{'} \;\Rightarrow\; c$ θ-subsumes $c^{'}$
- c ist mindestens genau so allgemein wie $c^{'}$ $\left(c \leq c^{'}\right)$
- c ist eine Verallgemeinerung von $c^{'}$
- $c^{'}$ ist eine Spezialisierung / *refinement* von c

6. top-down Suche

- Die Suche erfolgt hierbei in Richtung von einer allgemeinen zu einer speziellen *clause*.
- Der *specialization/refinement operator* basiert auf der θ*-subsumption*.
- *refinement operator* $p(c) = \left\{ c' | c' \in L, c < c' \right\}$
- Die Spezialisierung einer *clause* wird solange wiederholt, bis sie keine negativen Beispiele mehr abdeckt, mindestens jedoch ein positives.

6. top-down Suche

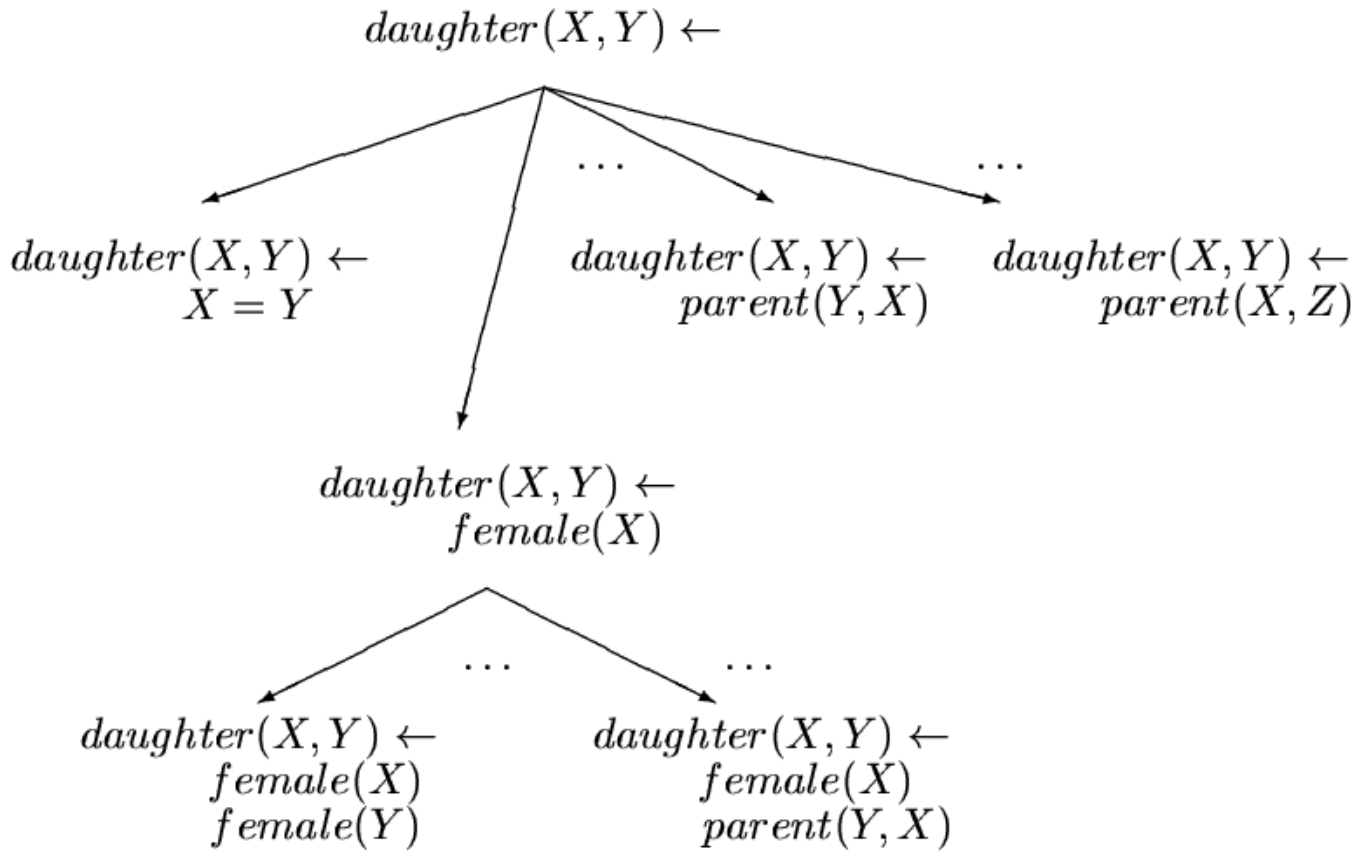
Algorithm 3.2 (Generic top-down ILP algorithm)

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 $\mathcal{H}$.

6. top-down Suche

Teil eines *refinement graph* für das $daughter$ Beispiel



7. FOIL

- Das FOIL Sytem baut auf dem generischen *top-down* auf.
- Der *covering-loop* des Algorithmus bricht ab, sobald es keine $\oplus$ Trainings Beispiele mehr gibt.
- Der *specialization-loop* beginnt mit der leeren *clause* für die *target relation*

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

7. FOIL - 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 $\mathcal{H}$.

7. FOIL - 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.

7. FOIL - specialization

- Ein lokales Trainingsset T_1 wird mit dem aktuellen Trainingsset initialisiert. $i = 1$
- solange $\ominus$ Beispiele in T_i sind:
  - Suche ein *literal* L_i, um es am *body* der *clause* anzuhängen.
  - Erstelle ein neues Trainingsset T_{i+1}, basierend auf den Beispielen aus T_i, welche L_i erfüllen.
  - i inkrementieren und fortfahren
- Die *literals* L_i werden mit Hilfe der Heuristik gefunden.

7. FOIL - Heuristik

- Informationsgehalt des Trainingsset T_i:
  $I\left(T_i\right) = -\log_2\left(T_i^+ / \left(T_i^+ + T_i^-\right)\right)$
- Informationsgehalt des Trainingsset T_{i+1}, falls L_i gewählt wird:
  $I\left(T_{i+1}\right) = -\log_2\left(T_{i+1}^+ / \left(T_{i+1}^+ + T_{i+1}^-\right)\right)$

$\Rightarrow$ Informationsgewinn durch das *literal* L_i:

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

- T_i^{++} ist die Anzahl der $\oplus$Beispiele sowohl in T_i als auch in T_{i+1}

7. FOIL - 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.

7. 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.

8. TILDE

- TILDE benutzt ebenfalls einen *top-down* Ansatz.
- Es werden keine Hypothesen gesucht, sondern *decision trees*.
- Um ein Beispiel zu Klassifizieren, müssen die Tests im *decision tree* abgearbeitet werden.
- Einteilung in mehrere Klassen möglich.

8. TILDE - Algorithmus

- TILDE startet mit einem leeren Baum.
- Alle möglichen test werden erzeugt und bewertet.
- Der beste Test wird im gegewärtigen Knoten aufgenommen.
- Sobald ein Knoten nur Beispiele einer Klasse enthält, ist die Suche beendet.

8. TILDE - Beispiel

begin(model(2)).
class(**sendback**).
worn(engine).
worn(chain).
end(model(2)).

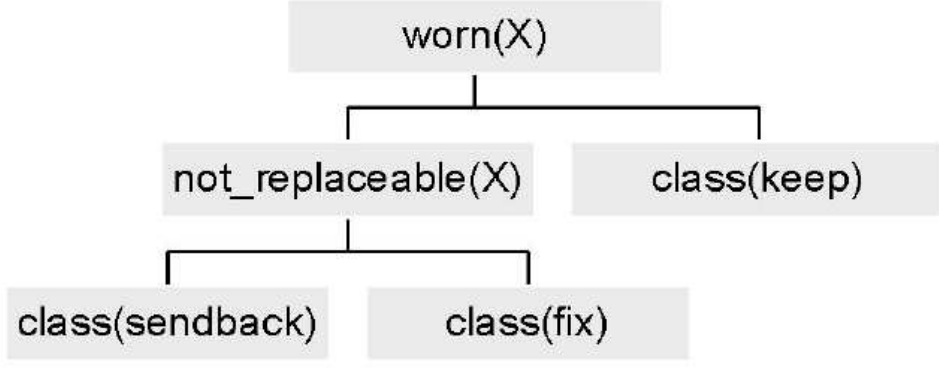
begin(model(3)).
class(**sendback**).
worn(control_unit).
end(model(3)).

begin(model(4)).
class(**fix**).
worn(chain).
end(model(4)).

begin(model(1)).
class(**fix**).
worn(gear).
worn(chain).
end(model(1)).

begin(background).
replaceable(chain)
replaceable(gear)
not_replaceable(engine)
not_replaceable(control_unit)
end(background).

begin(model(5)).
class(**keep**).
end(model(5)).

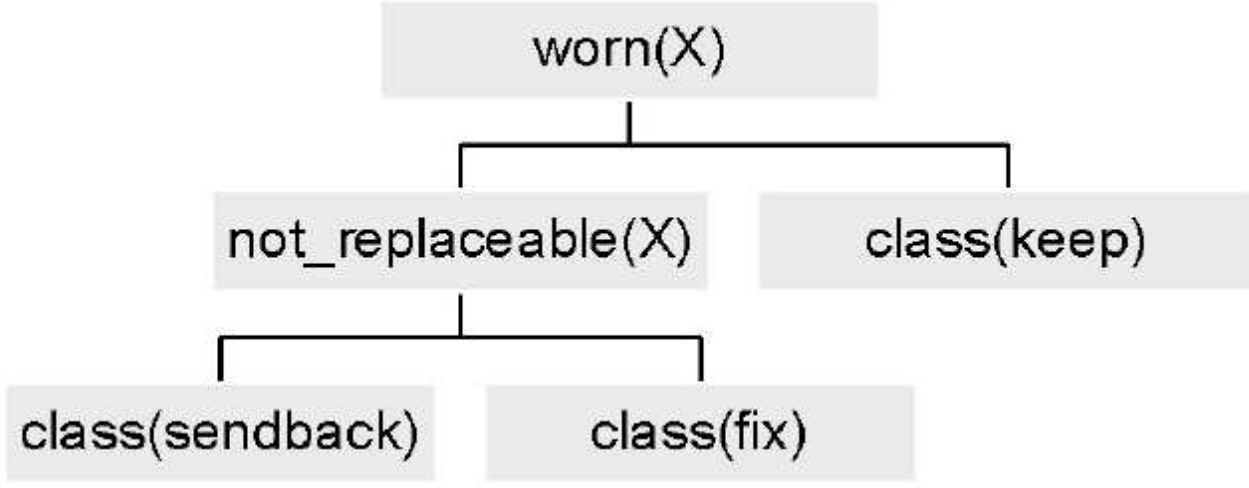


The WORN example with three classes and a corresponding tree.

8. TILDE

- Der *decision tree* als equivalentes Prolog Programm:

$$class\,(sendback) : -worn\,(\_X)\,, not\_replaceable\,(\_X)\,, !.$$
$$class\,(fix) : -worn\,(\_X)\,, !.$$
$$class\,(keep)\,.$$



8. TILDE - decision tree

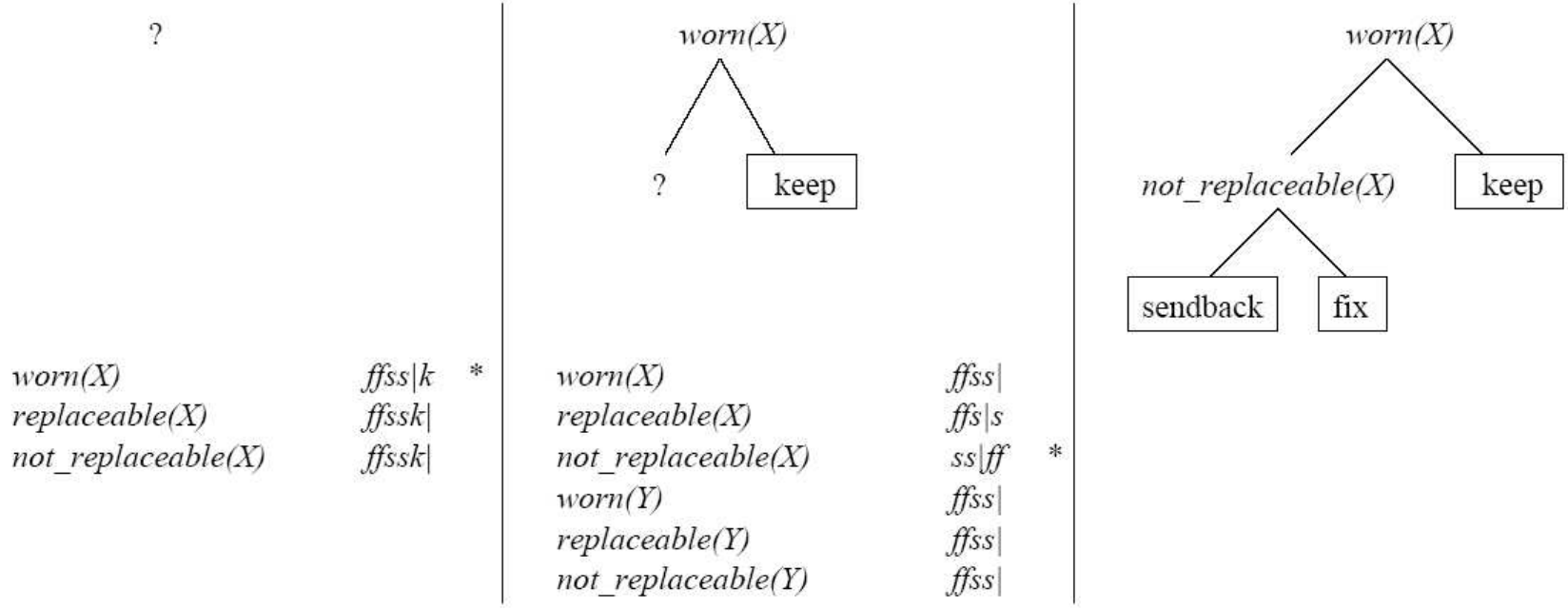


Fig. 11. TILDE illustrated on the WORN example. Each step shows the partial tree that has been built, the literals that are considered for addition to the tree, and how each literal would split the set of examples. E.g. *fss|k* means that of four examples, one with class **fix** and two with class **sendback**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

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?