Induktives Schließen

- Induktion vs. Deduktion
- Maschinelles Lernen
  - Definition
  - Anwendungsbeispiele
  - Konzept-Lernen
- Induktive Logik-Programmierung / Relationales Lernen / Foil
  - Separate-and-Conquer Lernen von Regel-Mengen
  - Hill-Climbing-Suche zum Lernen einzelner Regeln
  - Lernen rekursiver Regeln
  - Determinate Literals
- Induktion in propositionaler Logik
  - Konvertierung Datalog → propositionale Logik (Linus/Dinus)
- Overfitting und Pruning
  - Grundidee von Foil's MDL-Kriterium

Literatur

- Artikel
  - J. R. Quinlan, R. Mike Cameron-Jones: Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing* 13(3&4): 287-312, 1995. http://www.rulequest.com/Personal/q+cj.ngc95.ps
  - J. Fürnkranz: Separate-And-Conquer Rule Learning, *Artificial Intelligence Review* 13(1): 3-54, 1999. http://www.oefai.at/cgi-bin/tr-online?number+96-25
- Bücher
  - T. M. Mitchell: *Machine Learning*, McGraw-Hill, 1997
    (Gutes Lehrbuch für Maschinelles Lernen, hier vor allem Abschnitt 10)
  - N. Lavrac, S. Dzeroski: *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood 1994.
    (Klassiker für ILP, etwas veraltet, dafür on-line verfügbar)
    http://www-ai.ijs.si/SasoDzeroski/ILPBook/
- Software
  - R. Quinlan: Foil (eines der ersten und bekanntesten ILP-Systeme, in vieler Hinsicht jedoch suboptimal) http://www.rulequest.com/Personal/
  - S. Muggleton: Progol (erfolgreichstes System in Anwendungen. Hauptunterschied zu Foil: vollständige statt heuristischer Suche, daher oft genauer aber weniger effizient) http://www.doc.ic.ac.uk/~shm/

Induktives und Deduktives Schließen

- Deduktives Schließen
  - ist das Ableiten von neuen Fakten aus alten Fakten und allgemeingültigen Regeln
  - kann in Prädikatenlogik erster Stufe formalisiert werden
    - hat eine korrekte (vollständige und konsistente) Beweisregel (Resolution bzw. EPP)
    - kann daher in Programmen automatisiert werden
  - und das Ergebnis wird sicher stimmen
- Induktives Schließen
  - ist das Ableiten von allgemeingültigen Regeln aus bekannten Fakten
  - kann nicht streng formalisiert werden
    - obwohl es natürlich formale Methoden gibt
  - das Ergebnis eines induktiven Schlusses ist notwendigerweise nicht beweisbar korrekt
    - man kann gefundene Regeln nur empirisch überprüfen

Deduktives Schließen

- Gegeben:
  - Eine Theorie T
    - d.h. Regeln, die das gesuchte Ergebnis beschreiben
  - Hintergrundwissen B
    - eine Menge von Fakten oder Regeln, die das Wissen beschreiben, das von T verwendet wird
- Gesucht:
  - Eine Menge von Fakten E
    - die konkrete Folgerungen von *B* und T sind

$$T \quad \cup \quad B \quad \rightarrow \quad E$$

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

$\cup$

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

$\rightarrow$

```
parent(penelope,victoria).
parent(penelope,arthur).
parent(christopher,victoria).
parent(christopher,arthur).
```

Induktives Schließen

- Gegeben:
  - Eine Menge von Beispielen E
    - die den gesuchten Sachverhalt beschreiben (üblicherweise Fakten)
  - Hintergrundwissen B
    - eine Menge von Fakten oder Regeln, die das Wissen beschreiben, das verwendet werden kann um T zu formulieren
- Gesucht:
  - Eine Theorie T
    - die die Beispiele E mit Hilfe von B erklärt (d.h. aus B und T muß deduktiv E folgern)

$$E \quad \cup \quad B \quad \rightarrow \quad T$$

```
parent(penelope,victoria).
parent(penelope,arthur).
parent(christopher,victoria).
parent(christopher,arthur).
```

$\cup$

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

$\rightarrow$

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

Der Vollständigkeit halber: Abduktives Schließen

- Gegeben:
  - Eine Menge von Beispielen E
    - die den gesuchten Sachverhalt beschreiben (üblicherweise Fakten)
  - Eine Theorie T
    - die die Beispiele E erklären soll
- Gesucht:
  - Hintergrundwissen B
    - das notwendig ist, um aus B und T deduktiv E zu folgern (Teile dieses Wissens können schon vorhanden sein)

$$E \quad \cup \quad T \quad \rightarrow \quad B$$

```
parent(penelope,victoria).
parent(penelope,arthur).
parent(christopher,victoria).
parent(christopher,arthur).
```

$\cup$

```
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

$\rightarrow$

```
mother(penelope,victoria).
mother(penelope,arthur).
father(christopher,victoria).
father(christopher,arthur).
```

Maschinelles Lernen

- Maschinelles Lernen ist die Wissenschaft, die sich mit dem automatischen induktiven Schließen beschäftigt
  - Lernen ist schwer zu definieren (→ folgende Folien)
- Heutzutage beschäftigt sich maschinelles Lernen vorwiegend mit der Aufgabe, in großen Datenmengen Muster und Regelmäßigkeiten zu finden
  - Stichwort: Data Mining
  - Die Ähnlichkeiten zur Statistik werden immer fließender

Definitionen für Lernen
(aus der Psychologie)

„Lernen ist der Sammelname für Vorgänge, Prozesse oder nicht unmittelbar beobachtbare Veränderungen im Organismus, die durch Erfahrungen entstehen und zu Veränderungen des Verhaltens führen."

[Bergius,1971]

„Lernen bedeutet Veränderungen in der Wahrscheinlichkeit, mit der Verhaltensweisen in bestimmten Situationen auftreten."

[Hilgard,1973]

„Lernen ist eine Verhaltensänderung, die *nicht* durch Reifungsvorgänge, Verletzungen oder Erkrankungen, Ermüdungsprozesse oder durch Anlagen erklärt werden kann."

[Joerger,1976]

Definitionen für Lernen
(aus der Künstlichen Intelligenz)

„Learning denotes changes in the system that ... enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

[Simon,1983]

„Learning is making useful changes in our minds."

[Minsky,1985]

„Learning is constructing or modifying representations of what is being experienced."

[Michalski,1986]

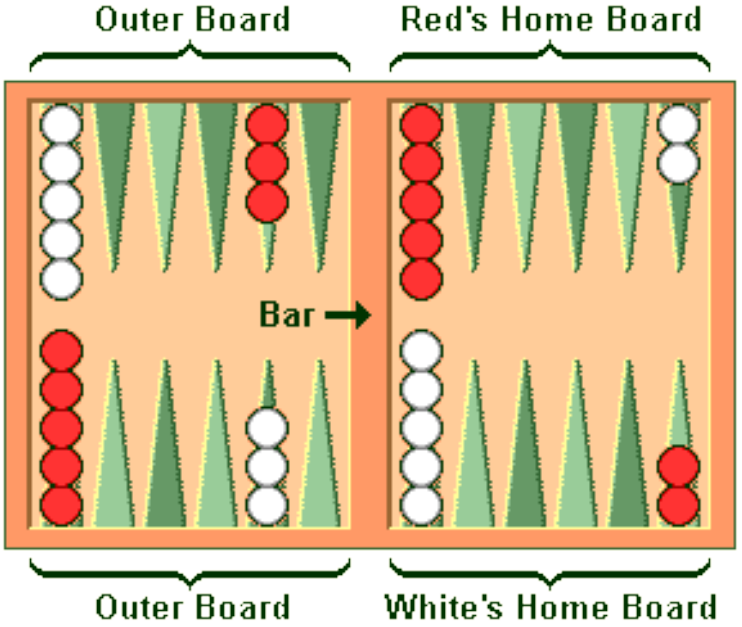
Definition Maschinelles Lernen

- Definition (Mitchell 1997)

  „A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.“
- Given:
  - a task T
  - a performance measure P
  - some experience E with the task
- Goal:
  - generalize the experience in a way that allows to improve your performance on the task

Learning to Play Backgammon

- Task:
  - play backgammon
- Performance Measure:
  - percentage of games won
- Experience:
  - previous games played



TD-Gammon:

- learned a neural network for evaluating backgammon boards
- from playing millions of games against itself
- successively improved to world-champion strength
- `http://www.research.ibm.com/massive/tdl.html`
  GNU Backgammon: `http://www.gnu.org/software/gnubg/`

Recognizing Spam-Mail

- Task:
  - sort E-mails into categories (e.g., Regular / Spam)
- Performance Measure:
  - Weighted Sum of Mistakes (letting spam through is not so bad as misclassifying regular E-mail as spam)
- Experience:
  - Handsorted E-mail messages in your folder

In Practice:

- Many Spam-Filters (e.g., Mozilla) use Bayesian Learning for recognizing spam mails

Market Basket Analysis

- Task:
  - discover items that are frequently bought together
- Performance Measure:
  - ? (revenue by making use of the discovered patterns)
- Experience:
  - Supermarket check-out data

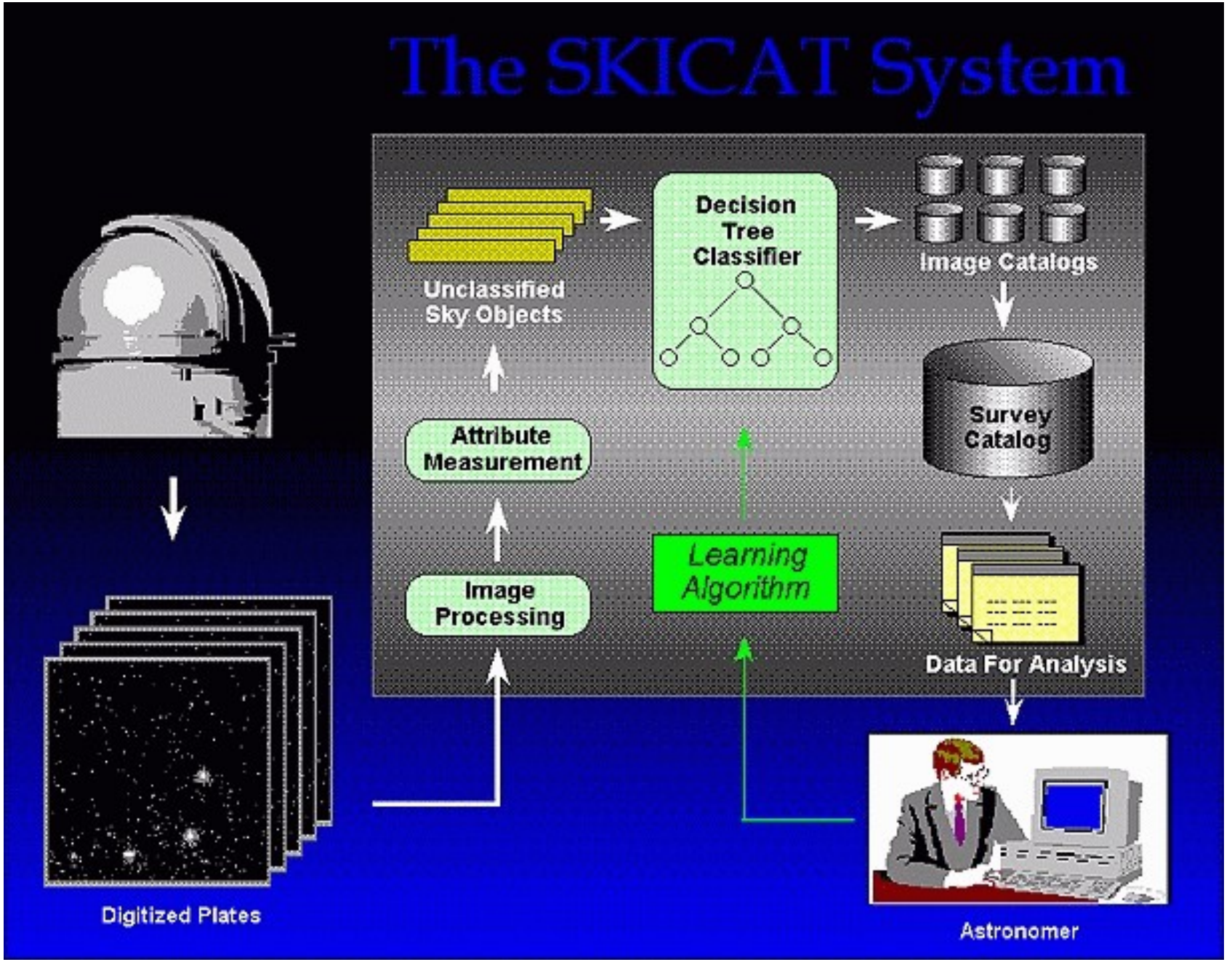
Myth:

- The most frequently cited result is:

$$\text{diapers} \rightarrow \text{beer}$$

Learning to Classify Stars

- Task:
  - Classification of celestial bodies
- Data:
  - 3000 images (23,040 x 23,040 pixels) of the Palomar Sky Observatory, 3 Terabytes of data,
  - classified into $\approx 10^7$ galaxies, $\approx 10^8$ stars, $\approx 10^5$ quasars
  - representation with 40 attributes (image-processing)
- Method:
  - learning of multiple decision trees
  - combining the best rules of each tree
- SKICAT Performance:
  - 94% accuracy, better than astronomers
  - discovery of 16 new quasars (12/95)


The SKICAT System
Decision Tree Classifier
Image Catalogs
Unclassified Sky Objects
Survey Catalog
Attribute Measurement
Learning Algorithm
Image Processing
Data For Analysis
Digitized Plates
Astronomer

Konzept-Lernen

- Eine der wichtigsten Aufgaben im maschinellen Lernen
  - Aufgabe ist, die Definition eines Begriffs z.B. in der Form einer Relation, aus Beispielen und Hintergrundwissen zu lernen.
- Gegeben:
  - Positive Beispiele E^+
    - Beispiele für das zu lernende Konzept
  - Negative Beispiele E^-
    - Gegenbeispiele für das zu lernende Konzept
  - Hintergrundwissen B
- Gesucht:
  - Eine korrekte Theorie T, d.h. mit folgenden Eigenschaften
    - **Vollständigkeit:** aus dem Hintergrundwissen und der Theorie kann man die positiven Beispiele herleiten
    - **Konsistenz:** aus dem Hintergrundwissen und der Theorie kann man kein negatives Beispiel herleiten

Logische Formalisierung des Konzept-Induktionsproblems

- A priori Constraints
  - **Prior Necessity**: $\exists e \in E^+: B \not\models e$
    - mindestens ein positives Beispiel darf nicht von vornherein aus dem Hintergrundwissen folgen
    - sonst bräuchten wir ja nicht zu lernen
  - **Prior Satisfiability**: $\forall e \in E^-: B \not\models e$
    - die negativen Beispiele dürfen nicht aus dem Hintergrundwissen folgen
    - das könnte eine Ergänzung zu B nicht mehr reparieren
- A posteriori Desiderata
  - **Posterior Sufficiency (Vollständigkeit)**: $\forall e \in E^+: B \wedge T \models e$
    - aus der gefundenen Theorie sollen alle positiven Beispiele logisch folgern
  - **Posterior Satisfiability (Konsistenz)**: $\forall e \in E^-: B \wedge T \not\models e$
    - aus der gefundenen Theorie soll kein negatives Beispiel logisch folgern

Deduktives vs. Induktives Lernen

- Deduktives Lernen (EBL)
  - Die Beispiele sind bereits aus dem Hintergrundwissen ableitbar
  - Bereits ein Beispiel kann zu einer sinnvollen Generalisierung führen
  - Die Generalisierungen sind beweisbar (sicher richtig)
  - Die Hintergrundtheorie dient zur Fokussierung auf die relevanten Merkmale
- Induktives Lernen
  - Die Beispiele sind nicht aus dem Hintergrundwissen ableitbar (prior necessity)
  - Man braucht viele Beispiele um verläßliche Generalisierungen zu treffen
  - Die Generalisierungen können nur statistisch überprüft werden
  - Die Anzahl der verschiedenen Trainingsbeispiel führt zur Fokussierung

Lernen als Suche

- Lernen ist ein Suchproblem
  - Im Raum aller möglichen Theorien wird nach einer Theorie gesucht, die vorgegebene Kriterien (z.B. Korrektheit) erfüllt
- Gegeben:
  - Hypothesenraum H:
    - Ein Raum von möglichen Theorien
  - Eine Bool'sche Funktion `covers(`h,e`)`
    - die feststellt, ob die Hypothese h das Beispiel e als Teil des zu lernenden Konzepts ansieht oder nicht
    - man sagt dann, daß die Hypothese h das Beispiel e abdeckt
- Gesucht:
  - eine Hypothese $h \in H$, die
    - vollständig ist: $\forall e \in E^+$: `covers(`h,e`)`
    - konsistent ist: $\nexists e \in E^-$: `covers(`h,e`)`

Konzept-Lernen in Datalog: Inductive Logic Programming (ILP)

- Ziel-Relation
  - Eine Relation, für die eine Beschreibung gelernt werden soll
- Trainings-Beispiele
  - Eine Menge von Tupeln der Relation als positive Beispiele
  - Eine Menge von Tupeln der Relation als negative Beispiele
- Hintergrundwissen:
  - Eine Menge von Datalog-Relationen, die verwendet werden können, um eine Definition der Ziel-Relation zu formulieren
  - Oft auch Selektion einzelner Werte oder Wertebereiche aus den Attributen der Ziel-Relation
    - wenn nur Selektion gestattet ist → propositionale Logik
- Hypothesenraum:
  - alle möglichen Datalog-Programme, die sich aus den Relationen im Hintergrundwissen bilden lassen
  - möglicherweise Einschränkungen durch Type Constraints, etc.

Beispiel

- Ziel-Relation:

```
father(A,B)
```

- Trainings-Beispiele:

⊕: `father(christopher,arthur).`
`father(christopher,victoria).`

⊖: `father(penelope,arthur).`
`father(christopher,penelope).`

- Hintergrundwissen:

```
parent(christopher,arthur).
parent(christopher,victoria).
parent(penelope,arthur).
parent(penelope,victoria).
```

```
male(christopher).
male(arthur).

female(victoria).
female(penelope).
```

Beispiel (Ctd.)

- Suchraum:
  - alle möglichen Konjunktionen von Termen, die man aus den Relationen `parent/2`, `male/1`, `female/1` bilden kann.
- Mögliche Lösungen:

```
father(A,B) :- parent(A,B), male(A).

father(A,B) :- male(A), parent(A,B).
```

  - falls Negationen erlaubt sind:

```
father(A,B) :- parent(A,B), \+ female(A).
```

Beispiel in propositionaler Logik

- Ziel-Relation:

```
object(Shape,Color,Size)
```

- Trainings-Beispiele:

No.	*Attributes*			*Class*
	Shape	Color	Size	
1	triangle	black	big	⊕
2	square	white	big	⊖
3	triangle	black	small	⊕
4	circle	white	small	⊖

- Hintergrundwissen:
  - Gleichheit von Attributwerten

Beispiel in propositionaler Logik (Ctd.)

- Suchraum:
  - alle Kombinationen von möglichen Selektionen auf Werte, die in den Daten vorkommen
  - d.h. `Shape = triangle, Shape = circle, Shape = square, Color = black, Color = white, Size = big, Size = small`

- Mögliche Lösungen:

```
object(Shape,Color,Size) :- Shape = triangle.

object(Shape,Color,Size) :- Color = black.

object(Shape,Color,Size) :- Shape = triangle,
                            Color = black.
```

Foil: First Order Inductive Learner
(Quinlan, 1990)

- eines der ersten und bekanntesten ILP-Systeme
- Such-Raum:
  - Beispiele und Hintergrundwissen werden durch Tupel beschrieben
    - keine Regeln im Hintergrundwissen zulässig
    - wäre aber prinzipiell möglich
  - Hypothesenraum sind rekursive Datalog-Programme
- Such-Strategie:
  - Separate-and-Conquer um Regeln zu Theorie zu kombinieren
  - Top-Down Hill-Climbing Suche nach bestem Literal, um eine Regel fortzusetzen
  - Weighted Information Gain zur Bewertung der Literale
  - Determinate Literals
  - MDL-Kriterion für Noise Handling
- Etliche Erweiterungen:
  - backtracking, Lernen ohne negative Beispiele, ...

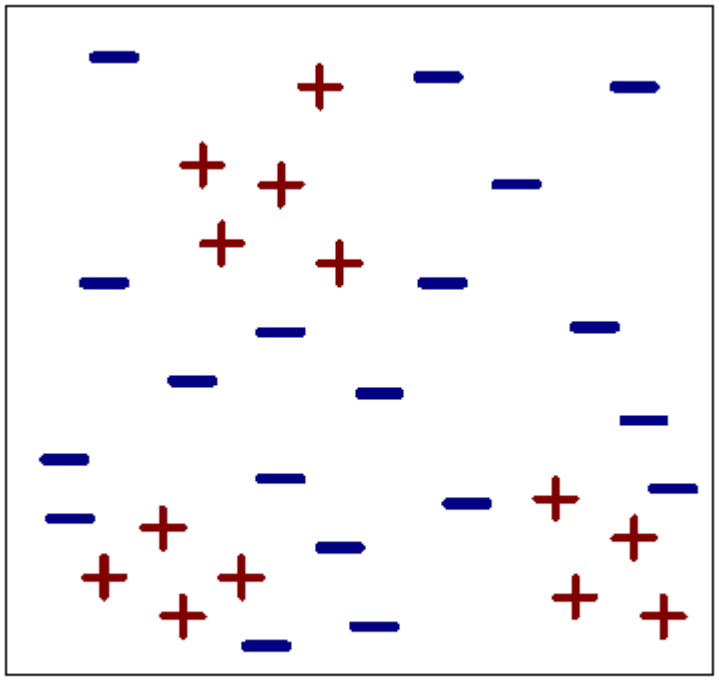
Separate-and-Conquer Rule Learning

- findet eine konsistente und vollständige Theorie, indem
  - konsistente Regeln gefunden werden
  - die verschiedene Teile des Beispielraums abdecken
  - solange bis alle Trainingsbeispiele von zumindest einer Regel abgedeckt werden
- Um eine Regelmenge zu finden:

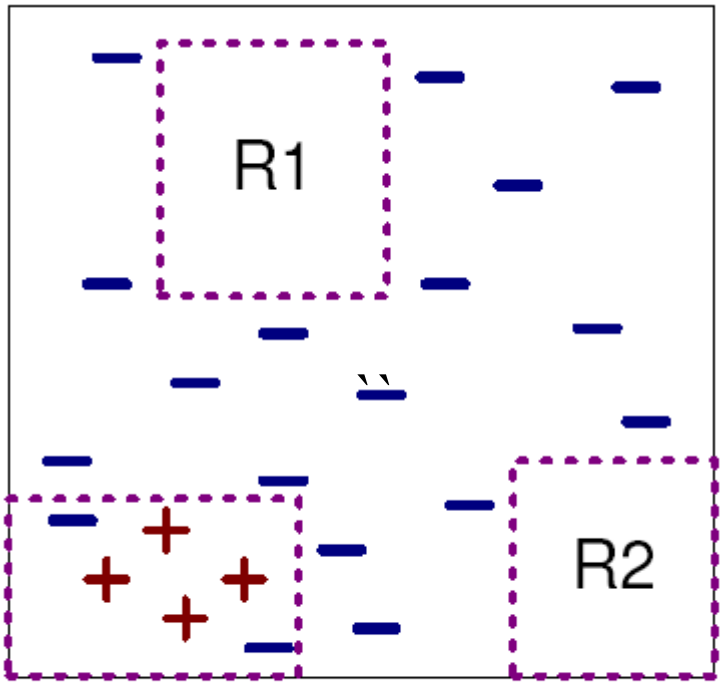
1. Generalisierung: erweitere die momentane Theorie um eine Regel, die
   - einige positive
   - aber keine negativen

   Beispiele abdeckt
2. Separate: entferne alle Beispiele, die durch diese Regel abgedeckt werden
3. Conquer: wenn es noch positive Beispiele gibt, goto 1.

Separate-and-Conquer Rule Learning



(i) Original Data



(iv) Step 3

Quelle für Grafiken: http://www.cl.uni-heidelberg.de/kurs/ws03/einfki/KI-2004-01-13.pdf

Separate-and-Conquer Rule Learning

- Eine große Anzahl von Regel-Lernalgorithmen verfolgt diese Strategie
  - gemeinsam ist allen Algorithmen, daß sie die positiven Beispiele sukzessive durch neue Regeln abdecken
  - diese Algorithmen sind daher auch als “Covering”-Algorithmen bekannt
- Die einzelnen Algorithmen unterscheiden sich in der Art und Weise, wie sie einzelne Regeln finden
  - z.B. EBL zum Finden einer operationalen *Theorie*

1. Wähle ein positives Beispiel aus
2. Suche nach einem Beweis für dieses Beispiel
3. Generalisiere den Beweis
4. Entferne alle Beispiele, die von diesem Beweis erklärt werden
5. Sind noch positive Beispiele über, goto 1.

Hill-Climbing-Suche nach einer Regel

- die verbreiteste Art ist, eine Hill-Climbing-Suche nach einer guten Regel durchzuführen
- findet eine konsistente Regel, indem
  - einzelne Bedingungen zu einer Regel hinzugefügt werden
  - solange bis die Regel nur mehr positive Beispiele abdeckt
- Algorithmus:

1. Versuche alle möglichen Bedingungen an die momentane Regel anzuhängen
2. Wähle eine Bedingung aus, die
   - möglichst viele positive Beispiele
   - möglichst wenige negative Beispiele

   abdeckt
3. Ist die Regel konsistent, retourniere sie als Ergebnis, ansonsten goto 1.

Separate-and-Conquer Rule Learning

- Pseudo-Code für den gesamten Algorithmus mit Hill-Climbing-Suche nach einer Regel

```
procedure SeparateAndConquer(Examples)

Theory = ∅
while Positive(Examples) ≠ ∅
        Clause = ∅
        Cover = Examples
        while Negative(Cover) ≠ ∅
                Clause = Clause ∪ SelectCondition(Clause, Cover)
                Cover = Cover(Clause, Cover)
        Examples = Examples − Cover
        Theory = Theory ∪ Clause
return(Theory)
```

Auswahl einer Bedingung

- Für jedes Prädikat im Hintergrundwissen:
  - generiere alle möglichen Literale
- Für jedes Literal
  - berechne einen heuristischen Wert
- Wähle das beste Literal aus

```
procedure SelectCondition(Clause,Examples)

Vmax = 0
Literals = AllPossibleLiterals(Clause)
for Literal in Literals
    V = HeuristicValue(Clause,Literal,Examples)
    if V > Vmax
        Vmax = V
        BestLiteral = Literal
return(BestLiteral)
```

Alle möglichen Bedingungen

- Im Prinzip alle möglichen Literale bestehend aus
  - einem Prädikat im Hintergrundwissen
  - Variablen, die im Head bzw. den bisherigen Bedingungen vorkommen
  - eventuell neuen Variablen
  - eventuell Konstanten-Symbolen, die in den Beispielen vorkommen
- Das können ziemlich viele werden
  - da jede Möglichkeit an den Daten getestet werden muß, kann das zu sehr hohen Laufzeiten führen
- Mögliche Einschränkungen:
  - **Language Bias:** Spezifikationssprache für gültige Literale
  - **Typen:** Variablen und Argumente von Prädikaten haben Typen, nur gültige Zuweisungen werden versucht
  - **Templates oder Schemata:** bestimmte Kombinationen werden forciert oder ausgeschlossen

Beispiel

- Head der Regel: `father(A,B) :-`
- Mögliche Bedingungen (wenn keine zusätzlichen Einschränkungen vorliegen)

```
male(A)      \+male(A)      parent(A,A)      parent(B,A)
male(B)      \+male(B)      \+parent(A,A)    \+parent(B,A)
female(A)    \+female(A)    parent(A,B)      parent(B,B)
female(B)    \+female(B)    \+parent(A,B)    \+parent(B,A)
```

- Klarerweise suboptimal, z.B.
  - `male(A)` ist gleichbedeutend mit `\+ female(A)`
  - `parent(A,A)` wird immer falsch sein

Beispiel für Ablauf

```
%%%
%%% EXAMPLE FOR SEPARATE-AND-CONQUER
%%% LEARNING ALGORITHM
%%%

| ?- foil(father(A,B),Clauses).

2 positive and 2 negative instances left.

male(A):        covers 2 + and 1 -
\+male(A):      covers 0 + and 1 -
male(B):        covers 1 + and 1 -
\+male(B):      covers 1 + and 1 -
female(A):      covers 0 + and 1 -
\+female(A):    covers 2 + and 1 -
female(B):      covers 1 + and 1 -
\+female(B):    covers 1 + and 1 -
parent(A,A):    covers 0 + and 0 -
\+parent(A,A):  covers 2 + and 2 -
parent(A,B):    covers 2 + and 1 -
\+parent(A,B):  covers 0 + and 1 -
parent(B,A):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 2 -
parent(B,B):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 2 -

Chose literal male(A) (covers 2 + and 1 -)
```

Suche nach erster Bedingung

```
male(A):        covers 2 + and 1 -
\+male(A):      covers 0 + and 0 -
male(B):        covers 1 + and 0 -
\+male(B):      covers 1 + and 1 -
female(A):      covers 0 + and 0 -
\+female(A):    covers 2 + and 1 -
female(B):      covers 1 + and 1 -
\+female(B):    covers 1 + and 0 -
parent(A,A):    covers 0 + and 0 -
\+parent(A,A):  covers 2 + and 1 -
parent(A,B):    covers 2 + and 0 -
\+parent(A,B):  covers 0 + and 1 -
parent(B,A):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 1 -
parent(B,B):    covers 0 + and 0 -
\+parent(B,A):  covers 2 + and 1 -

Chose literal parent(A,B) (covers 2 + and 0 -)

Found clause: father(A,B):-male(A),parent(A,B)


Clauses = [(father(A,B):-male(A),parent(A,B))] ?

yes
| ?-
```

Suche nach zweiter Bedingung

Auswahl von Bedingungen

- Welche Bedingungen sollen ausgewählt werden?
  - Ziel ist, daß die Regelmenge korrekt, d.h. komplett und konsistent wird
- Daher sollte die Regel
  - möglichst viele positive Beispiele abdecken
  - möglichst wenige negative Beispiele abdecken
- Daher sollten Maße ausgewählt werden, deren Wert
  - geringer wird
    - wenn weniger positive Bespiele abgedeckt werden
    - wenn mehr negative Beispiele abgedeckt werden
  - größer wird
    - wenn mehr positive Beispiele abgedeckt werden
    - wenn weniger negative Beispiele abgedeckt werden
- Es gibt viele Möglichkeiten solche Maße zu definieren
  - eine klare Aussage läßt sich nur schwer treffen

Precision

- Erste Idee:
  - Verwende die Genauigkeit (Precision) einer Regel als Suchheuristik
  - Die Genauigkeit ist der Prozentsatz der positiven Beispiele p in allen $p + n$ Beispielen, die von einer Regel abgedeckt werden.

$$precision = \frac{p}{p+n}$$

- Nachteile:
  - Eine Regel, die nur 1 positives Beispiel abdeckt wird als gleichwertig angesehen mit einer Regel, die 1000 positive Beispiele (und kein negatives) abdeckt
    - intuitiv ist aber letztere klar stärker
  - Eine Precision von 0.5 kann
    - ein guter Fortschritt sein, wenn die vohergehende Regel nur 0.25 Precision hatte
    - ein großer Rückschritt sein, wenn sie 0.75 Precision hatte

Zusatzforderungen

- Neben Konsistenz und Vollständigkeit wird daher auch gewünscht
  - eine Regel soll möglichst viele Beispiele abdecken
    - um eine besser verständliche Lösung zu produzieren
    - um besser zu generalisieren (→ Overfitting)
  - eine Regel soll möglichst kurz sein
    - aus denselben Gründen wie oben
    - kürzere Regeln decken tendenziell mehr Beispiele ab (Warum?)
  - die Änderung im heuristischen Wert gegenüber der vorigen Regel sollte möglichst groß (und positiv) sein.

Weighted Information Gain

- Foil verwendet daher (in etwa) folgendes Maß:

Die Logarithmen ändern nichts an der Sortierung der Regeln nach Precision, aber Unterschiede nahe bei 0 werden wichtiger

$$WIG = p\left(\log_2 \frac{p}{p+n} - \log_2 \frac{p'}{p'+n'}\right)$$

Der Informationgewinn pro Beispiel wird mit der Anzahl der von R abgedeckten positiven Beispiele gewichtet.

Qualität der Regel R, die sich aus dem Hinzufügen einer Bedingung L zu R' ergibt (R = R' ∪ L)

Qualität der momentanen Regel R'

Wie viel hat das Hinzufügen von L gebracht?
(**Beachte:** beide Logarithmen sind < 0!)

Motivation in der Informationstheorie

- Frage:
  - Wie viel Information steckt in den von einer Regel abgedeckten Beispielen?
- Idealfall:
  - Keine Information, da Information bereits in der Regel steckt
  - das heißt, idealerweise sind alle abgedeckten Beispiele positiv
- Eine Nachricht X, die mit Wahrscheinlichkeit $p(X)$ auftritt, hat einen Informationsgehalt $-\log_2 p(X)$
  - das heißt, ich brauche mindestens $-\log_2 p(X)$ Bits, um eine Nachricht, die mit Wahrscheinlichkeit $p(X)$ auftritt, zu kodieren
  - je seltener eine Nachricht ist, desto mehr Information trägt sie
- Wir messen, um wie viel sich der Informationsgehalt eines Beispiels durch den Übergang von R zu R' verringert
- und multiplizieren das mit der Anzahl der verbleibenden positiven Beispiele p

Weighted Information Gain

- Informationstheoretische Interpretation

$$WIG = p\left((-\log_2 \frac{p'}{p'+n'}) - (-\log_2 \frac{p}{p+n})\right)$$

Der Informationsgewinn pro Beispiel wird mit der Anzahl der von R abgedeckten positiven Beispiele gewichtet.

Informationsgehalt der Regel R'

Informationsgehalt der Regel R = R' ∪ L

Informationsgewinn beim Übergang von R' zu R (R wird i.a. weniger Information enthalten als R')

Beispiel

```
| ?- foil(father(A,B),Clauses).

2 positive and 2 negative instances left.

male(A):        Gain: 0.83 (2+/1-)
\+male(A):      Gain: 0.00 (0+/1-)
male(B):        Gain: 0.00 (1+/1-)
\+male(B):      Gain: 0.00 (1+/1-)
female(A):      Gain: 0.00 (0+/1-)
\+female(A):    Gain: 0.83 (2+/1-)
female(B):      Gain: 0.00 (1+/1-)
\+female(B):    Gain: 0.00 (1+/1-)
parent(A,A):    Gain: 0.00 (0+/0-)
\+parent(A,A):  Gain: 0.00 (2+/2-)
parent(A,B):    Gain: 0.83 (2+/1-)
\+parent(A,B):  Gain: 0.00 (0+/1-)
parent(B,A):    Gain: 0.00 (0+/0-)
\+parent(B,A):  Gain: 0.00 (2+/2-)
parent(B,B):    Gain: 0.00 (0+/0-)
\+parent(B,B):  Gain: 0.00 (2+/2-)

Chose literal male(A). Gain: 0.83 (2+/1-).
```

Beispiele für Literale, die keinen Informationsgewinn bringen.

```
male(A):        Gain: 0.00 (2+/1-)
\+male(A):      Gain: 0.00 (0+/0-)
male(B):        Gain: 0.58 (1+/0-)
\+male(B):      Gain:-0.42 (1+/1-)
female(A):      Gain: 0.00 (0+/0-)
\+female(A):    Gain: 0.00 (2+/1-)
female(B):      Gain:-0.42 (1+/1-)
\+female(B):    Gain: 0.58 (1+/0-)
parent(A,A):    Gain: 0.00 (0+/0-)
\+parent(A,A):  Gain: 0.00 (2+/1-)
parent(A,B):    Gain: 1.17 (2+/0-)
\+parent(A,B):  Gain: 0.00 (0+/1-)
parent(B,A):    Gain: 0.00 (0+/0-)
\+parent(B,A):  Gain: 0.00 (2+/1-)
parent(B,B):    Gain: 0.00 (0+/0-)
\+parent(B,B):  Gain: 0.00 (2+/1-)
Chose literal parent(A,B). Gain: 1.17 (2+/0-).

Found clause: father(A,B):-male(A),parent(A,B)

Clauses = [(father(A,B):-male(A),parent(A,B))]

yes
```

Drei Literale mit maximalem Gain (zufälliges Tie Break)

Nach dem zweiten Literal werden keine negativen Beispiele mehr abgedeckt.

Einführen Neuer Variablen im Body

- Manche Regeln können formuliert werden, wenn im Body der Regel neue Variablen eingeführt werden
  - Beispiel:

```
grandfather(A,B) :- father(A,C), parent(C,B).
```

- Vorgehensweise:
  - Literale mit neuen Variablen werden genauso ausprobiert
  - Die neuen Variablen können an alle Werte gebunden werden, die sich aus dem Hintergrundwissen ergeben
    - im Prinzip alle Tupel <A,B,C>, die sich als Antwort auf die Query, die dem Body der Regel entspricht, ergeben.

Beispiele und Beweise

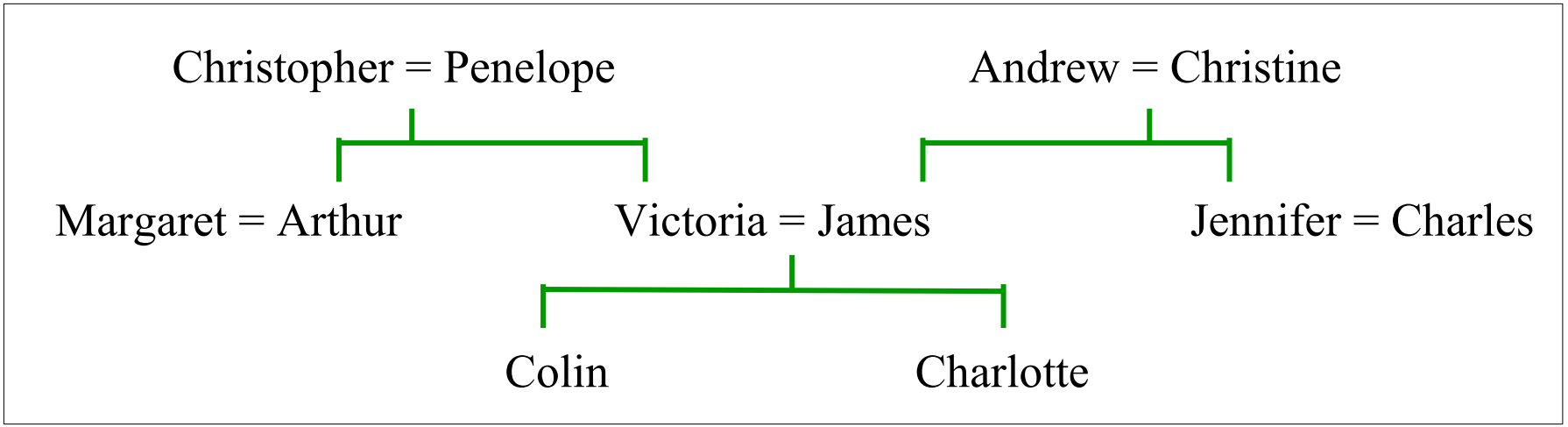
- Durch das Einführen neuer Variablen kann eine gelernte Regel für ein positives Beispiel durch mehrere Variablen-belegungen erfüllt werden
  - es gibt also mehrere Beweise für ein Beispiel
  - in anderen Worten:
    - die Anzahl der Tupel, die als Antwort auf eine Datenbank-Query mit dem Body der Regel zurückkommt, ist größer als die Anzahl der Beispiele, die im Head bewiesen werden sollen.
  - Beispiel:

```
sibling(A,B) :- parent(X,A), parent(X,B), A \= B.
```

    - Anzahl der Tupel `<A,B,X>`, die sich als Lösung für die Query im Body ergeben, kann größer sein als die Anzahl der Tupel `<A,B>`, die als positive Beispiele für `sibling/2` gegeben sind.
- Zwei Möglichkeiten zu zählen:
  - **Beweise zählen:** Alle wahren Tupel im Body werden gezählt
  - **Beispiele zählen:** All beweisbaren Heads werden gezählt

Beispiel

- Hintergrundwissen: `parent(X,Y), father(X,Y)`



- Zielrelation: `grandfather(X,Y)`

- Trainingsbeispiele:

⊕:
```
grandfather(christopher,colin).
grandfather(christopher,charlotte).
grandfather(andrew,colin).
grandfather(andrew,charlotte).
```

⊖:
```
grandfather(penelope,colin).
grandfather(christine,charlotte).
grandfather(victoria,christopher).
grandfather(james,charlotte).
grandfather(christopher,james).
grandfather(andrew,james).
```

Zählen von Beweisen

- Der Algorithmus führt die momentane Menge der positiven und negativen Tupel mit
  - am Anfang entspricht das der Menge der positiven und negativen Trainings-Beispiele
- kommt eine neue Variable hinzu, wird jedes Tupel mit allen gültigen Belegungen dieser Variable erweitert
  - falls eine Variable mehrere Belegungen hat, kann sich die Anzal der Tupel vergrößeren
  - natürlich kann sie sich (wie bisher) auch verkleinern, falls ein Tupel keine Belegung mehr hat (also durch Hinzufügen einer Bedingung nicht mehr beweisbar ist)

Beispiel

- Das Hinzunehmen des Literals father(A,C) führt zu einer Verdoppelung der Tupel.
  - Trainings-Tupel:

<⊕,christopher,colin>
<⊕,andrew,colin>
<⊖,penelope,colin>
<⊖,victoria,christopher>
<⊖,christopher,james>
<⊕,christopher,charlotte>
<⊕,andrew,charlotte>
<⊖,christine,charlotte>
<⊖,james,charlotte>
<⊖,andrew,james>

  - Trainings-Tupel für `grandfather(A,B) :- father(A,C).`

<⊕,christopher,colin,victoria>
<⊕,christopher,colin,arthur>
<⊕,christopher,charlotte,victoria>
<⊕,christopher,charlotte,arthur>
<⊖,james,charlotte,colin>
<⊖,christopher,james,victoria>
<⊖,andrew,james,james>
<⊕,andrew,colin,james>
<⊕,andrew,colin,jennifer>
<⊕,andrew,charlotte,james>
<⊕,andrew,charlotte,jennifer>
<⊖,james,charlotte,charlotte>
<⊖,christopher,james,arthur>
<⊖,andrew,james,jennifer>

- In diesem Fall ändert sich am Ergebnis nichts
  - kann aber i.a. schon passieren

Zählen von Beispielen vs. Zählen von Beweisen (Tupeln)

- Leere Regel:
  - abgedeckte Beispiele/Tupel: $4\oplus/6\ominus$
  - Informationsgehalt: $-\log_2\frac{4}{4+6}=1.31\,bits$
- Beispiele nach Auswahl des Literals `father(A,C)`:
  - abgedeckte Beispiele: $4\oplus/3\ominus$
  - Informationsgehalt: $-\log_2\frac{4}{4+3}=0.81\,bits$
  - Informationsgewinn: $4(1.31-0.81)=2.05\,bits$
- Tupel nach Auswahl des Literals `father(A,C)`:
  - abgedeckte Tupel: $8\oplus/6\ominus$
  - Informationsgehalt: $-\log_2\frac{8}{8+6}=0.81\,bits$
  - Informationsgewinn: $4(1.31-0.81)=2.05\,bits$

In diesem Fall ändert sich am Ergebnis nichts.
Kann aber i.a. schon passieren!

Hier steht die Anzahl der Tupel in der Vorgänger-Regel

Unterschiede

- Zählen von abgedeckten Beispielen ist semantisch klarer
  - letztendlich sollten ja alle positiven Beispiele beweisbar sein, und es ist egal wie oft
  - andererseits kann es eine Rolle spielen, ob es nur mehr einen oder noch zwanzig Beweise für ein negatives Beispiel gibt
  - wird üblicherweise durch Testen der gesamten Regel implementiert
- Zählen von Beweisen kann u.U. effizienter sein
  - wird üblicherweise durch das Mitführen der Tupel implementiert
  - muß nicht darauf achten, welche Tupel zu den gleichen Beispielen im Head führen
  - bzw. man muß nicht jedes Mal einen neuen Beweis finden
- empirisch wurde noch kein Unterschied gefunden

→ Welche Methode besser ist, ist noch unklar.

  - Üblicher ist Zählen von Beispielen
  - aber Foil verwendet Zählen von Beweisen

Nützliche Literale ohne Gain

- Literale, die neue Variablen einführen, haben möglicherweise keine discriminative Power
  - d.h. sie helfen nicht, zwischen positiven und negativen Beispielen zu unterscheiden
  - haben daher keinen Information Gain
    - optimieren auch kein anderes Maß, das darauf setzt, Verbesserungen in Konsistenz und Vollständigkeit zu optimieren
- Aber solche Literale sind unbedingt notwendig
  - da sie die Variablen einführen, die dann in anderen Literalen getestet werden
- Beispiel:

```
grandchild(A,B) :- child(A,C), child(C,B).
```

  - Das Literal `child(A,C)` hilft nicht, zwischen Enkelkindern, und nicht Enkelkindern zu unterscheiden, da jeder ein Kind von jemandem ist → kein Information Gain
  - **Anm:** Wie wird in diesem Beispiel die Regel trotzdem gefunden?

Determinate Literals

- Ein Literal ist *determinate*, wenn es
  - eine oder mehrere neue Variablen einführt
  - für jede Belegung von alten Variablen mit positivem Head gibt es genau eine Belegung für diese neuen Variablen
  - für jede Belegung von alten Variablen mit negativem Head gibt es höchstens eine Belegung für diese neuen Variablen
- Beispiele:
  - `succ(N,N1)` ist determinate
    - jede Zahl hat genau einen Nachfolger
  - `components(List,Head,Tail)` ist determinate
    - sowohl wenn `List` gegeben ist, und `Head` und `Tail` frei
    - als auch wenn `Head` und `Tail` gegeben sind und `List` frei ist
  - `father(X,A)` ist determinate, wenn `A` gegeben ist
    - es ist nicht determinate, wenn `X` gegeben ist!
  - `parent(X,A)` ist nicht determinate
    - es gibt für jedes `A` zwei Eltern!

Determinate Literals in Foil

- wenn kein Literal mit ausreichendem Gain gefunden werden kann
  - ausreichend sind 80% des maximalen Gains
  - der maximale Gain ergibt sich aus der Annahme, daß man mit dem nächsten Literal alle positiven Beispiele und keins der negativen Beispiele abdeckt
- wird versucht, dem Lerner aus der Patsche zu helfen, indem der momentanen Regel *alle* Determinate Literals dazugefügt werden
  - mit Determinate literals werden aber nicht alle möglichen Problemfälle behandelt
  - sondern nur die, die effizient zu behandeln sind.
- nach dem Lernen der Regel werden alle unnötigen Literale aus der Regel gelöscht
  - d.h. alle Literale, die gelöscht werden können, ohne die Genauigkeit zu beeinträchtigen

Lernen rekursiver Konzepte

- Eine rekursive Regel entsteht, wenn das Prädikat, das im Head vorkommt, auch im Body der Regel vorkommt
  - Beispiel:

```
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

- Der Wahrheitsgehalt der entsprechenden Literale kann anhand der extensional gegebenen Trainingsbesipiele evaluiert werden
  - d.h. man gibt die vorhandenen positiven Trainingsbeispiele als Fakten zum Hintergrundwissen dazu
  - dann können sie genauso verwendet werden wie andere Prädikate im Hintergrundwissen
- Dabei sind allerdings einige Besonderheiten zu beachten
  - z.B. würde ohne entsprechende Vorsichtsmaßnahmen einfach die triviale Rekursion `ancestor(X,Y) :- ancestor(X,Y).` gelernt werden

Probleme bei rekursiven Literalen

- Vermeidung unendlicher Rekursionsketten:
  - nicht nur die triviale Rekursion ist ein Problem (→ nächste Folie)
- Ordnung der gelernten Regeln:
  - Der Covering-Algorithmus kann die Regeln in beliebiger Reihenfolge lernen
  - daher müssen die Regeln nach dem Lernen sortiert werden
    - **Anm:** Das gilt für dir prozedurale Semantik von Prolog, bei der Fix-Punkt-Semantik von Datalog ist das nicht notwendig!
- Vollständigkeit der Trainings-Beispiele:
  - alle Beispiele im Beweisbaum einer rekursiven Ableitung müssen im Trainings-Set sein.
  - Beispiel:
    - um eine rekursive Regel für `ancestor(christopher,colin)` zu lernen, muß (neben `parent(christopher,victoria)`) auch `ancestor(victoria,colin)` bekannt sein.

Sicherstellen korrekter Rekursionen

- Das rekursive Literal muß eine andere Variablenbelegung als der Head haben
  - Problem: `married(X,Y) :- married(Y,X).`
- Das rekursive Literal muß eine neue Variable einführen
  - Problem: `parent(X,Y) :- married(X,Z), parent(Z,Y).`
- Die Konstanten der Argumente des rekursiven Aufrufs müssen geordnet werden, sodaß die Aufrufe einem Minimum oder einem Maximum zustreben
  - ein rekursiver Aufruf wird nur zugelassen, wenn für zumindest ein Argument ein Vorwärtsschritt in der Ordnung erfolgt
  - Problem mit mehreren rekursiven Aufrüfen

    ```
    r(A,B,C) :- r(A+1,B-1,X), r(A-1,B+1,Y), C is X+Y.
    ```
- Alles in allem ein nicht triviales Problem
  - Lösung in Foil: (Details in (Quinlan & Cameron-Jones, 1995))
    - Finden einer Ordnungsrelation für alle Argumente des rekursiven Aufrufs gleichzeitig

Größe des Suchraums

- Die Grundversion des Algorithmus durchsucht
  - alle Prädikate
  - mit allen möglichen Variabilisierungen
  - inklusive möglicher neuer Variablen
- Viele dieser Möglichkeiten sind klar unnötig
  - Explosion des Suchraums
  - neu eingeführte Variablen sorgen für zusätzliche Komplexität
    - da sie in allen zukünftigen Prädikaten eingesetzt werden können
- Problem:
  - Wie kann der Suchraum reduziert werden, ohne das Lernergebnis zu gefährden?

Einschränkungen bei neuen Variablen

- Anzahl neuer Variablen
  - Es wird nur ein bestimmte Anzahl von neuen Variablen zugelassen
  - in Foil: Option `-V`
- Tiefenbeschränkung der Variablen
  - Die Tiefe einer Variable, die im Head vorkommt, ist 0
  - Die Tiefe jeder anderen Variable ist 1 plus das Maximum aller alten Variablen im Literal, in dem die Variable eingeführt wird
  - Beispiel:

```
ackermann(A,B,C) :- succ(D,B), ackermann(A,D,F),
                    succ(E,A), ackermann(E,F,C).
```

    - Tiefe: `A/0, B/0, C/0, D/1, E/1, F/2`
  - Beschränkung der Tiefe hilft lange Ketten von Literalen, die neue Variablen einführen, zu verhindern.
  - in Foil: Option `-d`

Typen, Modes, Symmetrien

- Typ-Spezifikationen:
  - nicht alle Variablen können in allen Argumenten verwendet werden
  - z.B. `components(list,atom,list)`
- Mode-Spezifikationen:
  - Nicht alle Stellen eignen sich zur Einführung neuer Variablen
    - `+`: nur alte Variablen können verwendet werden
    - `-`: alte oder neue Varialben können verwendet werden
    - `#`: Konstanten sollen verwendet werden (in Foil: alte Variablen)
    - Beispiel: `components(List,Head,Tail)  #-- / -##`
- Symmetrie-Spezifikationen:
  - eigene Literale sind symmetrisch in ihren Argumenten, sodaß nur eine Richtung überprüft werden muß
  - z.B. `adjacent(A,B)` und `adjacent(B,A)`

Propositionales Lernen

- Die meisten Lern-Algorithmen arbeiten nur in propositionaler Aussagenlogik
  - d.h. im Prinzip, daß sie nur einstellige Relationen verwenden können
  - üblicherweise Selektionen auf bestimmte Werte eines Attributs
- Für Hintergrundwissen, das keine neue Variablen zuläßt, oder nur determinate Literals zuläßt ist eine einfache Konvertierung des Lernproblems möglich
  - Konvertierung Prädikatenlogik 1. Stufe → Aussagenlogik
  - es muß allerdings eine bestimmte fixe Maximaltiefe der Variablen bzw. eine maximale Beweislänge vorgegeben werden
- Für nicht determinate Literale mit neuen Variablen ist eine Übersetzung nicht so einfach möglich

Linus (Lavrač & Džeroski, 1991)

- Lernen einer First-Order Theorie mit einem propositionalem Regel-Lern Algorithmus

- Übersetzung von Prädikaten
  - Für jede gültige Variabilisierung eines Prädikats wird ein binäres Attribut A definiert
    - A = true für ein Beispiel, wenn das Literal durch die Instantiierung des Heads mit den Werten des Beispiels beweisbar wird
    - A = false wenn nicht

Beispiel

- Das ursprüngliche Lern-Problem
  - Target: `daughter(X,Y)`

Training examples		*Background knowledge*		
$daughter(sue, eve).$	$\oplus$	$parent(eve, sue).$	$female(ann).$	$male(pat).$
$daughter(ann, pat).$	$\oplus$	$parent(ann, tom).$	$female(sue).$	$male(tom).$
$daughter(tom, ann).$	$\ominus$	$parent(pat, ann).$	$female(eve).$	
$daughter(eve, ann).$	$\ominus$	$parent(tom, sue).$		

- Das entsprechende propositionale Lern-Problem

	Variables		*Propositional features*							
C	X	Y	$f(X)$	$f(Y)$	$m(X)$	$m(Y)$	$p(X,X)$	$p(X,Y)$	$p(Y,X)$	$p(Y,Y)$
$\oplus$	sue	eve	$true$	$true$	$false$	$false$	$false$	$false$	$true$	$false$
$\oplus$	ann	pat	$true$	$false$	$false$	$true$	$false$	$false$	$true$	$false$
$\ominus$	tom	ann	$false$	$true$	$true$	$false$	$false$	$false$	$true$	$false$
$\ominus$	eve	ann	$true$	$true$	$false$	$false$	$false$	$false$	$false$	$false$

Beispiel aus (Lavrac & Dzeroski, 1994)

Rückübersetzung gelernter Regeln

- Aus dem propositionalen Datenset wurde folgende Regel gelernt:

$$Class = \oplus \quad \textbf{if} \quad [female(X) = true] \quad \wedge \quad [parent(Y, X) = true]$$

  - **Beachte:** die Regel verwendet nur Select-Statements auf Attribute der Tabelle auf der vorigen Folie!

- Attribute werden dann entsprechend zurückübersetzt
  - Die resultierende Regel ist:

```
daughter(X,Y) :- female(X), parent(Y,X).
```

Dinus

- Dinus = Linus mit Determinate Literals
- Übersetzung von Determinate Literals
  - Für jede Output-Variable wird ein neues Attribut angelegt
  - Der Wert des Attributs ist der Wert, der sich aus den Input-Variablen ergibt (kann maximal einer sein, da determinate)
  - Die Variablen-Tiefe muß natürlich beschränkt werden, da sich sonst unendlich viele Literale finden lassen würden.

Beispiel

- Das ursprüngliche Lern-Problem
  - Target: `grandmother(X,Y)`

Training examples	
$grandmother(ann, bob).$	$\oplus$
$grandmother(ann, sue).$	$\oplus$
$grandmother(bob, sue).$	$\ominus$
$grandmother(tom, bob).$	$\ominus$

Background knowledge		
$father(zak, tom).$	$father(pat, ann).$	$father(zak, jim).$
$mother(ann, tom).$	$mother(liz, ann).$	$mother(ann, jim).$
$father(tom, sue).$	$father(tom, bob).$	$father(jim, dave).$
$mother(eve, sue).$	$mother(eve, bob).$	$mother(jean, dave).$

- Das entsprechende propositionale Lern-Problem

	Variables		*New variables*					*Propositional features*		
$g(X,Y)$	X	Y	$f(U,X)$	$f(V,Y)$	$m(W,X)$	$m(Z,Y)$	...	$m(X,V)$	$m(X,Z)$	...
Class	X	Y	U	V	W	Z		A_1	A_2	
$\oplus$	ann	bob	pat	tom	liz	eve		$true$	$false$	
$\oplus$	ann	sue	pat	tom	liz	eve		$true$	$false$	
$\ominus$	bob	sue	tom	tom	eve	eve		$false$	$false$	
$\ominus$	tom	bob	zak	tom	ann	eve		$false$	$false$	

Beispiel aus (Lavrac & Dzeroski, 1994)

Rückübersetzung gelernter Regeln

- Nehmen wir an, aus dem propositionalen Datenset wurden folgende Regeln gelernt (in unserem Beispiel würde nur die erste gelernt werden)

$$Class = \oplus \quad \textbf{if} \quad [A_1 = true]$$
$$Class = \oplus \quad \textbf{if} \quad [A_2 = true]$$

- Attribute werden dann entsprechend zurückübersetzt

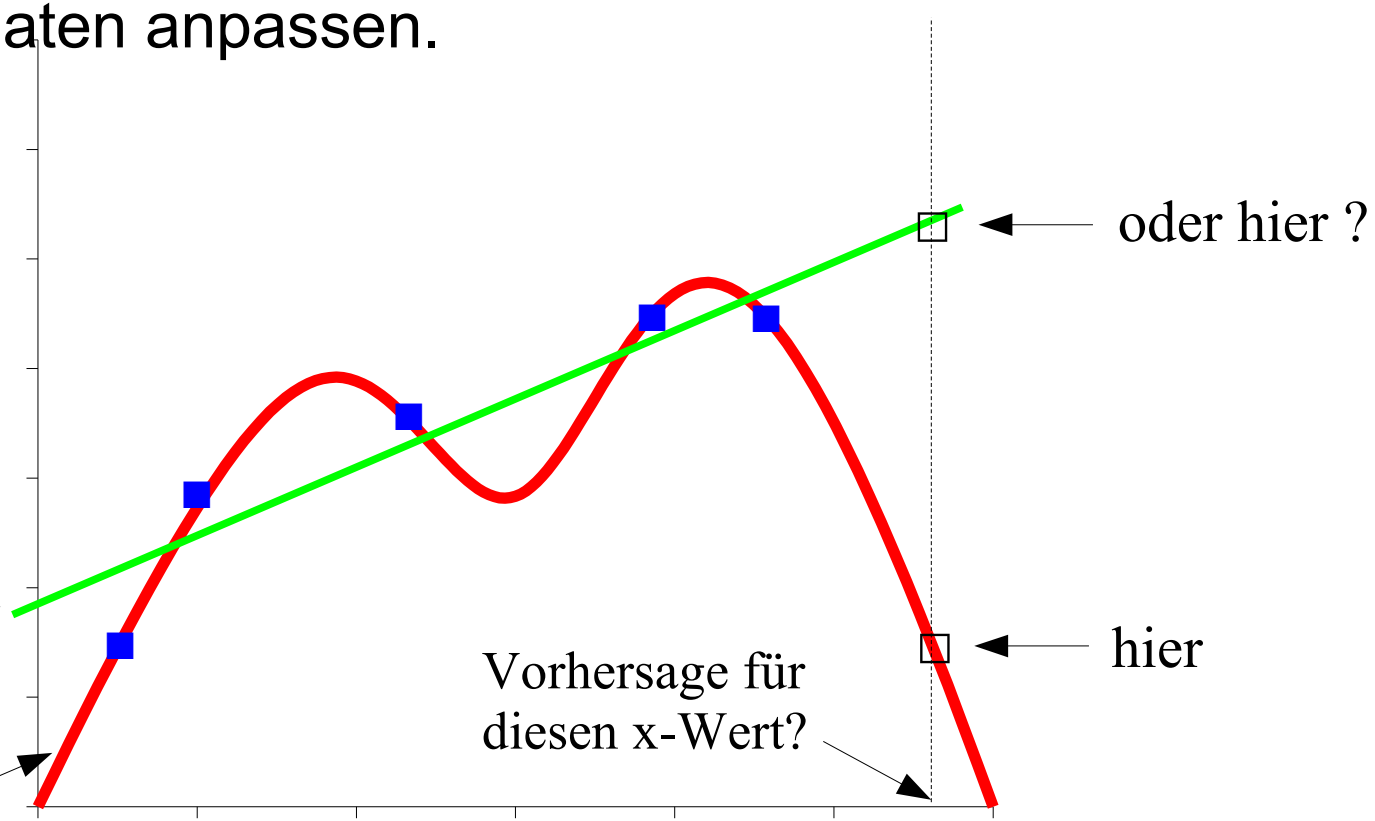
```
grandmother(X,Y) :- mother(X,V).
grandmother(X,Y) :- mother(X,Z).
```

  - Da die Variablen V und Z nicht im Head der Regel zu finden sind, müssen sie von einem der determinate Literals generiert worden sein.
  - Diese generierenden Literale müssen nun ebenfalls (vorne) dazugefügt werden

```
grandmother(X,Y) :- father(V,Y), mother(X,V).
grandmother(X,Y) :- mother(Z,Y), mother(X,Z).
```

Overfitting

- Problem:
  - Wenn eine Funktionenklasse beliebig viele Parameter (Freiheitsgrade) hat, kann Sie sich beliebig genau an vorgegebene Daten anpassen.
- Beispiel:
  - gegeben: 5 Punkte
  - gesucht: passende Funktion



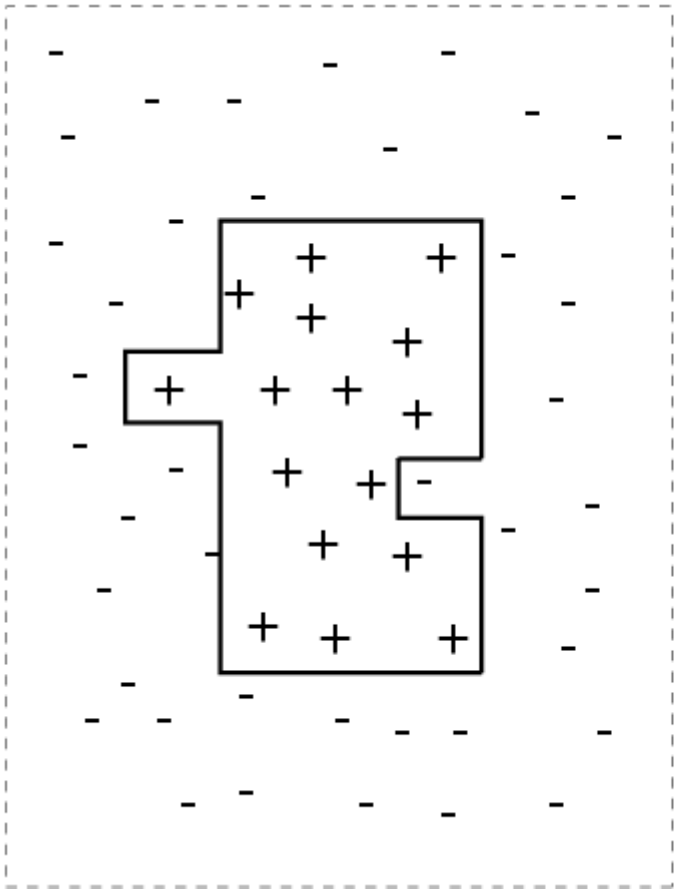
Polynom 1. Grades (Gerade)

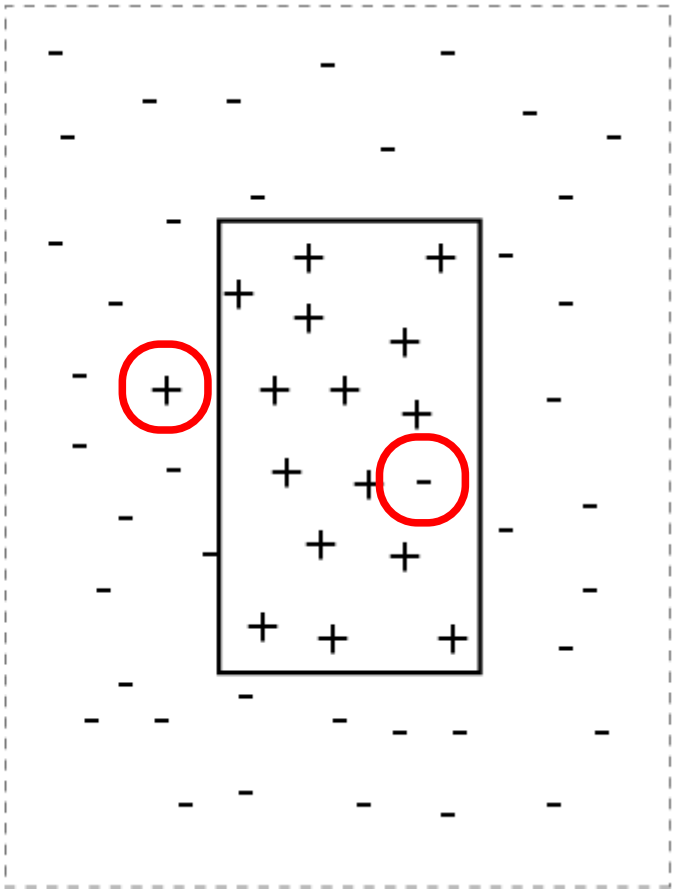
Polynom 4. Grades (n-1 Grade können n Punkte perfekt anpassen)

Overfitting

- Eine perfekte Anpassung an die gegebenen Daten ist nicht immer sinnvoll
  - Daten könnten fehlerhaft sein
    - z.B. zufälliges Rauschen (Noise)
  - Die Klasse der gewählten Funktionen könnte nicht geeignet sein
    - eine perfekte Anpassung an die Trainingsdaten ist oft gar nicht möglich
- Daher ist es oft günstig, die Daten nur ungefähr anzupassen
  - bei Kurven:
    - nicht alle Punkte müssen auf der Kurve liegen
  - beim Konzept-Lernen:
    - nicht alle positiven Beispiele müssen von der Theorie abgedeckt werden
    - einige negativen Beispiele dürfen von der Theorie abgedeckt werden

Overfitting





- beim Konzept-Lernen:
  - nicht alle positiven Beispiele müssen von der Theorie abgedeckt werden
  - einige negativen Beispiele dürfen von der Theorie abgedeckt werden

Komplexität von Konzepten

- Je weniger komplex ein Konzept ist, desto geringer ist die Gefahr, daß es sich zu sehr den Daten anpaßt
  - Für ein Polynom n-ten Grades kann man $n+1$ Parameter wählen, um die Funktion an alle Punkte anzupassen
  - Bei einer Gerade nur 2 Parameter
- Daher wird beim Lernen darauf geachtet, die Größe der Konzepte klein zu halten
  - eine kurze Regel, die viele positive Beispiele erklärt (aber eventuell auch einige negative) ist oft besser als eine lange Regel, die nur einige wenige positive Beispiele erklärt.
- Pruning: komplexe Regeln werden zurechtgestutzt
  - Pre-Pruning:
    - während des Lernens
  - Post-Pruning:
    - nach dem Lernen

MDL-Pruning in Foil

- Basiert auf dem Minimum Description Length-Prinzip (MDL)
  - ist es effektiver die Regel oder die Beispiele zu übertragen?
    - der Informationsgehalt einer Regel wird berechnet (in Bits)
    - der Informationsgehalt aller Beispiele wird berechnet (in Bits)
    - wenn die Regel mehr Bits braucht als die Beispiele dann wird die Regel nicht weiter verfeinert
  - Details → (Quinlan, 1990)
- Funkioniert nicht perfekt
  - bei nicht perfekten Regeln müßte man noch die Kosten für die Ausnahmen kodieren
    - die müssen zusätzlich zur Regel übertragen werden
  - eine informations-theoretisch perfekte Kodierung einer Regel ist praktisch nicht möglich
- Overfitting wird dadurch reduziert, aber nicht vollständig aufgehoben
  - Alternative: Post-Pruning der Regeln (behandeln wir nicht)

Ein generischer Lern-Algorithmus

```
procedure ILP(Examples)

INITIALIZE(Theories, Examples)
repeat
    T = SELECT(Theories, Examples)
    {T_i}_{i=1}^n = REFINE(T, Examples)
    Theories = REDUCE(Theories ∪ ⋃_{i=1}^n T_i, Examples)
until STOPPINGCRITERION(Theories, Examples)
return(Theories)
```

- geeignete Auswahl der Routinen kann verschiedenste Suchverfahren implementieren (e.g., depth/breadt/best-first search, genetische Suche, etc.)

Ein generischer ILP-Algorithmus

- Initialize
  - initialisiere die Menge der Theorien (e.g., Theories = {true} oder Thories = Examples)
- Select
  - Auswahl der vielversprechendsten Theorie
- Refine
  - Generieren neuer Theorien (z.B. durch Spezialisierung, Generalisierung, zufällige Störungen, etc.)
- Reduce
  - Eliminierung schlechter Theorien
- StoppingCriterion
  - bestimmt, ob die momentane Theorie hinreichend gut ist (z.B. wenn sie komplett und konsistent ist)