Deduktives Lernen – Explanation-Based Learning

- Algorithmus
  - grundlegende Idee anhand eines Beispiels
- Eigenschaften von EBL
  - Utility Problem
- Implementierung als PROLOG Meta-Interpreter
- Operationalisierung

Explanation-Based Learning

- Grundidee:

  Ein erfolgreicher Beweis für ein Fakt wird
  - **kompiliert:** sodaß er das nächste Mal schneller erfolgen kann
  - **generalisiert:** sodaß auch andere Beispiele effizienter bewiesen werden können

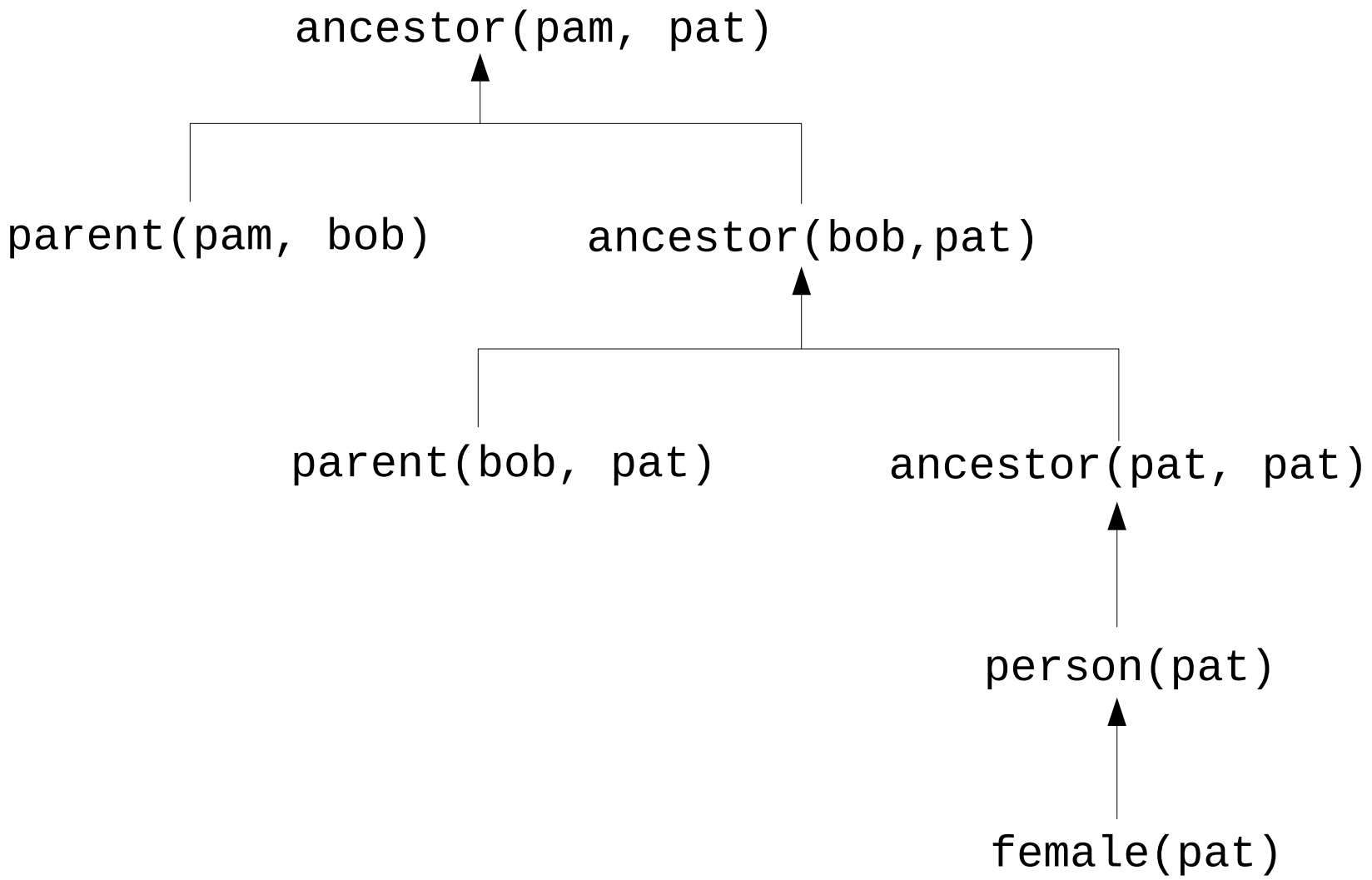
  → das System lernt: es zieht aus einem Beispiel Rückschlüsse, die es auf anderen Beispielen anwenden kann
- Charakteristika:
  - es wird das Vorhandensein einer korrekten (= vollständigen und konsistenten) Domän-Theorie vorausgesetzt
    - **Vollständigkeit:**
      für jedes positive Beispiel kann ein Beweis gefunden werden
    - **Korrektheit:**
      für kein negatives Beispiel kann ein Beweis gefunden werden
  - Es kann aus einem einzigen Beispiel gelernt werden
  - Generalisierung dieses Beispiels erfolgt deduktiv
    - das Resultat des Lernens ist beweisbar korrekt

EBL Algorithmus

1. Beweis:
   - Das Trainingsbeispiel wird mit Hilfe der vorhandenen Domän-Theorie bewiesen
   - Ein Beweis wird auch als Erklärung bezeichnet
     → daher der Name “Explanation-Based Learning”
2. Generalisierung:
   - Der Beweis für das Beispiel wird verallgemeinert, indem Konstanten durch Variablen ersetzt werden
   - Beachte, daß Constraints auf den Variablen erhalten bleiben!
     - z.B. `ancestor(Y,Y)` im Beweisbaum des folgenden Beispiels legt fest, daß die beiden Argumente an dieser Stelle gleich sein müssen
3. Formulierung der Regel:
   - Die Konjunktion der Generalisierungen aller Fakten, die im Beweis verwendet wurden, bildet eine neue Regel
   - d.h. die Konjunktion aller Blätter im Beweis-Baum

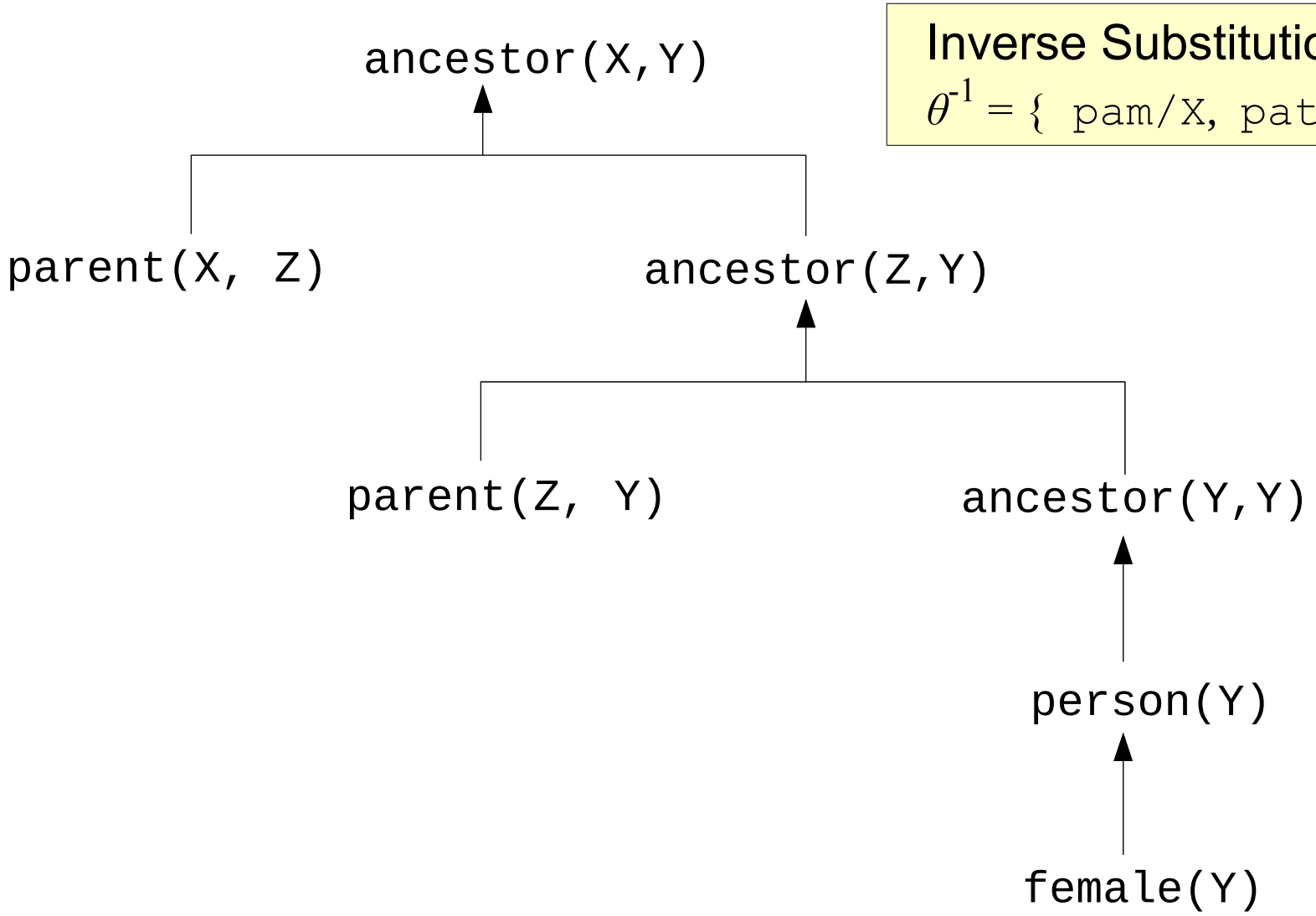
EBL: Beispiel

1. Schritt: Beweis des Beispiels



EBL: Beispiel

2. Schritt: Generalisierung des Beweises

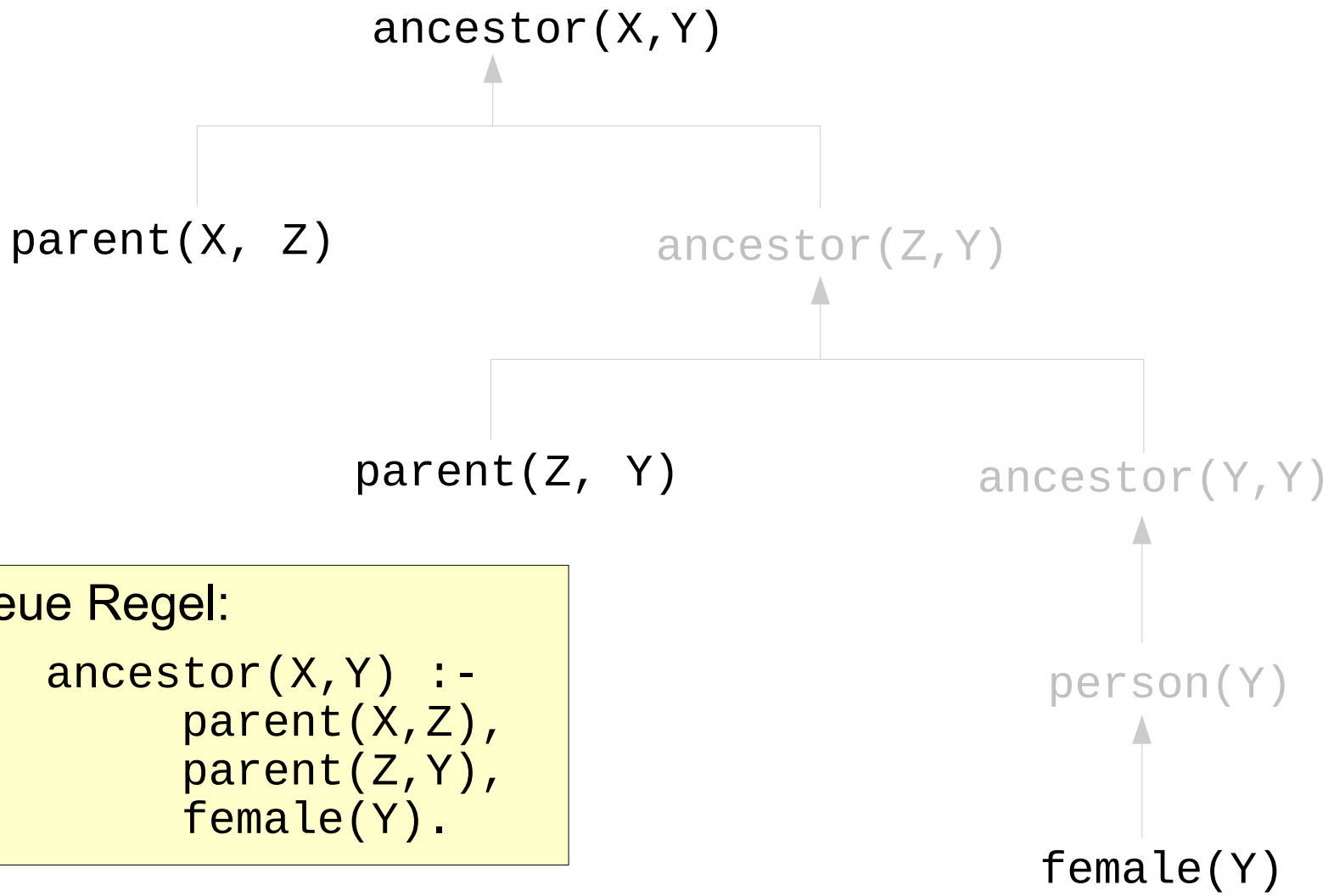


Inverse Substitution

$\theta^{-1} = \{ \texttt{pam/X}, \texttt{pat/Y}, \texttt{bob/Z} \}$

EBL: Beispiel

3. Schritt: Formulierung einer neuen Regel



Neue Regel:

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

EBL als Prolog Meta-Interpreter

- Funktioniert fast wie der Meta-Interpreter für Beweisbäume
  - nur die Blätter des Beweisbaumes werden gesammelt
  - Literale werden generalisiert

```
ebl_prove( L, GenL, GenL) :-
        clause(L,true).
ebl_prove( (C1,C2), (GenC1,GenC2),
           (GenFacts1,GenFacts2))  :-
        !,
        ebl_prove(C1, GenC1, GenFacts1),
        ebl_prove(C2, GenC2, GenFacts2).
ebl_prove( Head,  GenHead,  GenFacts )  :-
        clause( GenHead , GenBody ),
        GenBody \= true,
        copy_term( (GenHead :- GenBody),
                   (Head :- Body) ),
        ebl_prove(Body, GenBody, GenFacts).
```

Aufruf des EBL Meta-Interpreters

- `ebl_prove(ancestor(pam,pat), ancestor(X,Y), GenFacts)`
  - `GenFacts` enthält dann die auf die angegebenen Variablen generalisierten Facts

- Man kann noch einen Aufruf programmieren, der den generalisierten Head und die generalisierten Facts zusammensetzt:

```
ebl( L, GenL, (GenL :- GenFacts)) :-
        ebl_prove(L, GenL, GenFacts).
```

- Aufruf:

```
?- ebl(ancestor(pam,pat),ancestor(X,Y),P).

X = _G160
Y = _G161
P = ancestor(_G160, _G161) :-  parent(_G160, _G267),
                               parent(_G267, _G161),
                               female(_G161)
```

Die neue Regel ist ...

- *beweisbar* immer richtig
  - d.h., wenn die Bedingungen des Bodies erfüllt sind, ist der Head sicher gültig
  - da sich das aus dem zugrundeliegenden Beweisbaum ergibt
- für mehr als ein Beispiel gültig
  - sie wurde zwar aus einem Beispiel erzeugt (“gelernt”)
  - aber sie gilt für alle Beispiele, die denselben Beweisweg nehmen
    - z.B. auch für `ancestor(tom,ann).`
- nicht für alle Beispiele gültig
  - es gibt natürlich Beispiele, die sich nicht mit dieser Regel herleiten lassen
    - z.B. alle, die über mehr oder weniger als 2 Generationen gehen

→ *EBL liefert eine hinreichende, aber nicht notwendige Regel.*

Beachte

- Die neue Regel beinhaltet kein neues Wissen!
  - Alles was, mit der neuen Regel bewiesen werden kann, könnte auch ohne ihr bewiesen werden!
- Aber der Beweis wird abgekürzt!
  - Die neue Regel greift nur direkt auf die Fakten zu.
    - dadurch werden Irrwege im Finden dieses Ableitungsbaums vermieden (SLD Search Tree)
    - das heißt beim nächsten Mal wird dieses Beispiel effizienter bewiesen.
  - Die neue Regel deckt auch neue Fälle ab, die dann ebenfalls effizienter bewiesen werden könne

- Kurz gefaßt:
  - Man weiß nach dem Lernen nichts neues, aber man weiß es schneller!
  - Wird auch “Speed-Up Learning” genannt

Utility Problem

- Wurde zuerst von Minton (1988) im Prodigy Lern- und Plan-System beobachtet
  - Hier wurde EBL zur Effizienzsteigerung in der Suche nach Plänen eingesetzt
  - gefundene Pläne wurden mit EBL generalisiert und konnten in der Folge wiederverwertet werden
- Gelernte Regeln
  - können die Problemlösungszeit reduzieren, da das Ausführen einer Regel effizienter ist, als eine Suche durchzuführen
  - erhöhen aber andererseits die Problemlösungszeit, da alle Regeln in Betracht gezogen werden müssen (was wiederum Zeit kostet)
- Daher ist der Nutzen der Regeln nicht immer klar, z.B.
  - Regeln, die selten verwendet werden
  - Regeln, deren Bedingungen teuer zu matchen sind

Operationalisierung von Wissen

- Eine Sichtweise auf EBL ist, daß es versucht, das vorhandene Wissen zu “operationalisieren”.
  - Operationales Wissen ist Wissen, das unmittelbar (d.h. ohne lange Beweisketten überprüft werden kann.
- Einfachste Realisierung:
  - Operationale Prädikate sind die Fakten, die in der Datenbank gespeichert sind (EDB)
  - Nicht-operationale Prädikate sind die Regeln der IDB
- Es sind aber auch andere Varianten denkbar

Realisierung von Operationalisierung

- Ein Prädikat definieren, das angibt, ob ein Literal operational ist:
  - alle Fakten sind operational

```
operational(L) :-
    clause(L,true).
```

  - zusätzliche operationale Prädikate können (z.B.) einfach aufgelistet werden

```
operational( person(_) ).
```

- In der ersten Regel des Programms wird dann
  - nicht mehr überprüft, ob ein Fakt vorliegt
  - sondern überprüft, ob ein operationales Prädikat vorliegt

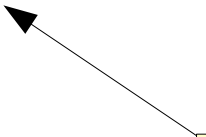
```
ebl_prove( L, GenL, GenL) :-
    operational(L),
    call(L).
```

`call(L)` ruft Prolog's eingebauten Beweiser auf, um festzustellen, ob `L` gilt oder nicht.

Ergebnis

```
?- ebl(ancestor(pam,pat),ancestor(X,Y),P).

X = _G160
Y = _G161
P = ancestor(_G160, _G161) :-
        parent(_G160, _G267),
        parent(_G267, _G161),
        person(_G161)
```



Die Generalisierung ist nun auch für männliche Nachkommen gültig.

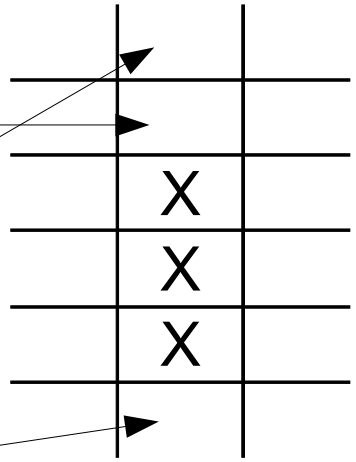
Anwendungen von EBL

- Search Control
  - Während einer Suche muß man ständig entscheiden, welcher Zustand als nächstes ausgewählt werden soll
- Query Optimization
  - der IDB Teil teurer Datalog Queries kann durch EBL in EDB-Abfragen “kompiliert” werden
- Spiele
  - Spiele wie Schach haben eine klar formalisierbare Domäntheorie (die Regeln des Spieles)
  - Für jede Stellung des Spiels läßt sich (theoretisch) beweisen, wer gewonnen hat
    - Der Beweis ist aber sehr aufwendig
  - Daher die Hoffnung:
    - Generalisierung gefunder Beweise mittels EBL erlaubt, auch in neuen Situationen effizienter zu spielen
  - hat sich aber in der Praxis als undurchführbar herausgestellt

Beispiel: Go-Moku (Minton, 1984)

- Das Spiel Go-Moku (5-Gewinnt ohne Schwerkraft) wurde durch eine Domän-Theorie kodiert
- Immer wenn das Programm verloren hat, versuchte es zu beweisen, warum es verloren hat
- Und generalisierte dann den Beweis zu einer Regel
- Beispiel einer gelernten Regel:

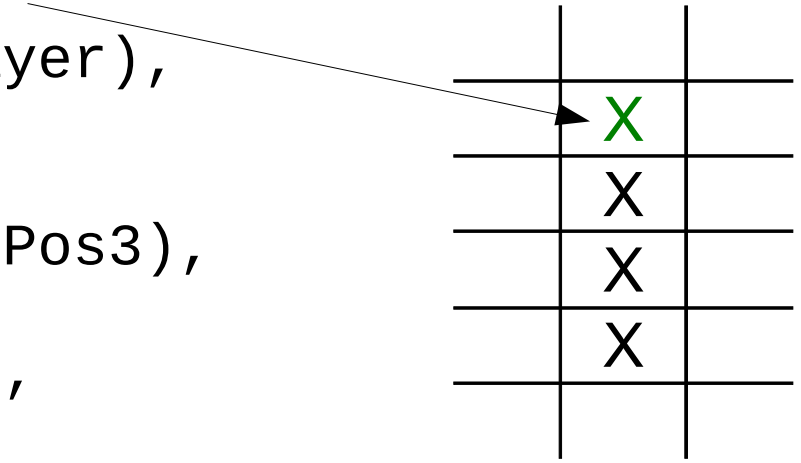
```
move(Square,Player) :-
   three-in-row(Pos3,Player),
   is_empty(Square),
   extends(Pos3,Square),
   composes(Pos4,Square,Pos3),
   is_empty(Square1),
   extends(Pos4,Square1),
   is_empty(Square2),
   extends(Pos4,Square2),
   Square1 \= Square2.
```



Beispiel: Go-Moku (Minton, 1984)

- Das Spiel (5-Gewinnt ohne Schwerkraft) wurde durch eine Domän-Theorie kodiert
- Immer wenn das Programm verloren hat, versuchte es zu beweisen, warum es verloren hat
- Und generalisierte dann den Beweis zu einer Regel
- Beispiel einer gelernten Regel:

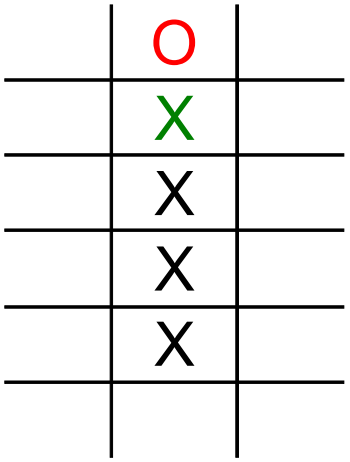
```
move(Square,Player) :-
   three-in-row(Pos3,Player),
   is_empty(Square),
   extends(Pos3,Square),
   composes(Pos4,Square,Pos3),
   is_empty(Square1),
   extends(Pos4,Square1),
   is_empty(Square2),
   extends(Pos4,Square2),
   Square1 \= Square2.
```



Beispiel: Go-Moku (Minton, 1984)

- Das Spiel (5-Gewinnt ohne Schwerkraft) wurde durch eine Domän-Theorie kodiert
- Immer wenn das Programm verloren hat, versuchte es zu beweisen, warum es verloren hat
- Und generalisierte dann den Beweis zu einer Regel
- Beispiel einer gelernten Regel:

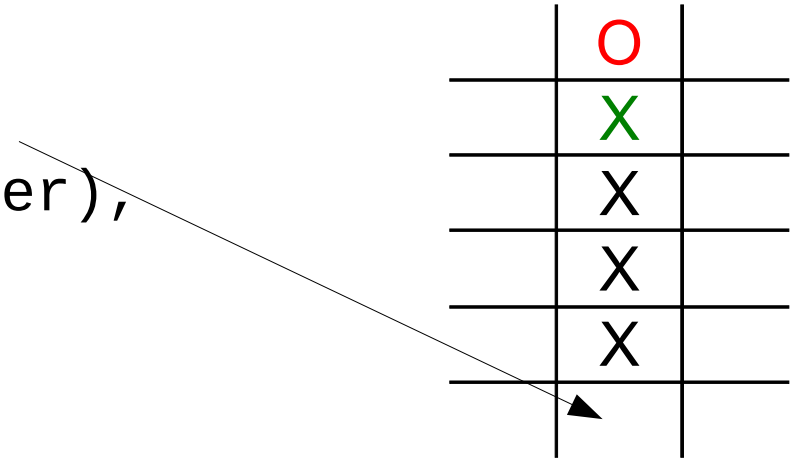
```
move(Square,Player) :-
   three-in-row(Pos3,Player),
   is_empty(Square),
   extends(Pos3,Square),
   composes(Pos4,Square,Pos3),
   is_empty(Square1),
   extends(Pos4,Square1),
   is_empty(Square2),
   extends(Pos4,Square2),
   Square1 \= Square2.
```



Beispiel: Go-Moku (Minton, 1984)

- Das Spiel (5-Gewinnt ohne Schwerkraft) wurde durch eine Domän-Theorie kodiert
- Immer wenn das Programm verloren hat, versuchte es zu beweisen, warum es verloren hat
- Und generalisierte dann den Beweis zu einer Regel
- Beispiel einer gelernten Regel:

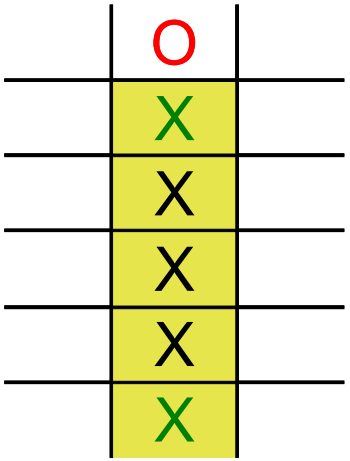
```
move(Square,Player) :-
   four-in-row(Pos4,Player),
    is_empty(Square),
    extends(Pos4,Square).
```



Beispiel: Go-Moku (Minton, 1984)

- Das Spiel (5-Gewinnt ohne Schwerkraft) wurde durch eine Domän-Theorie kodiert
- Immer wenn das Programm verloren hat, versuchte es zu beweisen, warum es verloren hat
- Und generalisierte dann den Beweis zu einer Regel
- Beispiel einer gelernten Regel:

```
move(Square,Player) :-
   four-in-row(Pos4,Player),
    is_empty(Square),
    extends(Pos4,Square).
```



Literatur

- Artikel
  - T. M. Mitchell, R. M. Keller, S. T. Kedar-Cabelli. Explanation-Based Generalization: A Unifying View. *Machine Learning* 1:47-80, 1986.
    (Der klassische Artikel zu EBL)
  - G. DeJong, R. Mooney. Explanation Based Learning: An Alternative View. *Machine Learning* 1:145-176, 1986
  - S. T. Kedar-Cabelli, L. T. McCarty. Explanation-Based Generalization as Resolution Theorem Proving, *Proceedings of the International Workshop on Machine Learning (ICML-87)*, pp. 383-389, 1987.
    (beschreibt die Implementierung als Prolog-Meta-Interpreter)
  - S. Minton. Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples. *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, pp. 251-254, 1984.
    (beschreibt die Anwendung auf das Spiel Go-Moku)
- Bücher
  - T. M. Mitchell: *Machine Learning*, McGraw-Hill, 1997
    (Klassisches Lehrbuch, hier vor allem Abschnitt 11)