Lecture 11: Inductive Program Synthesis

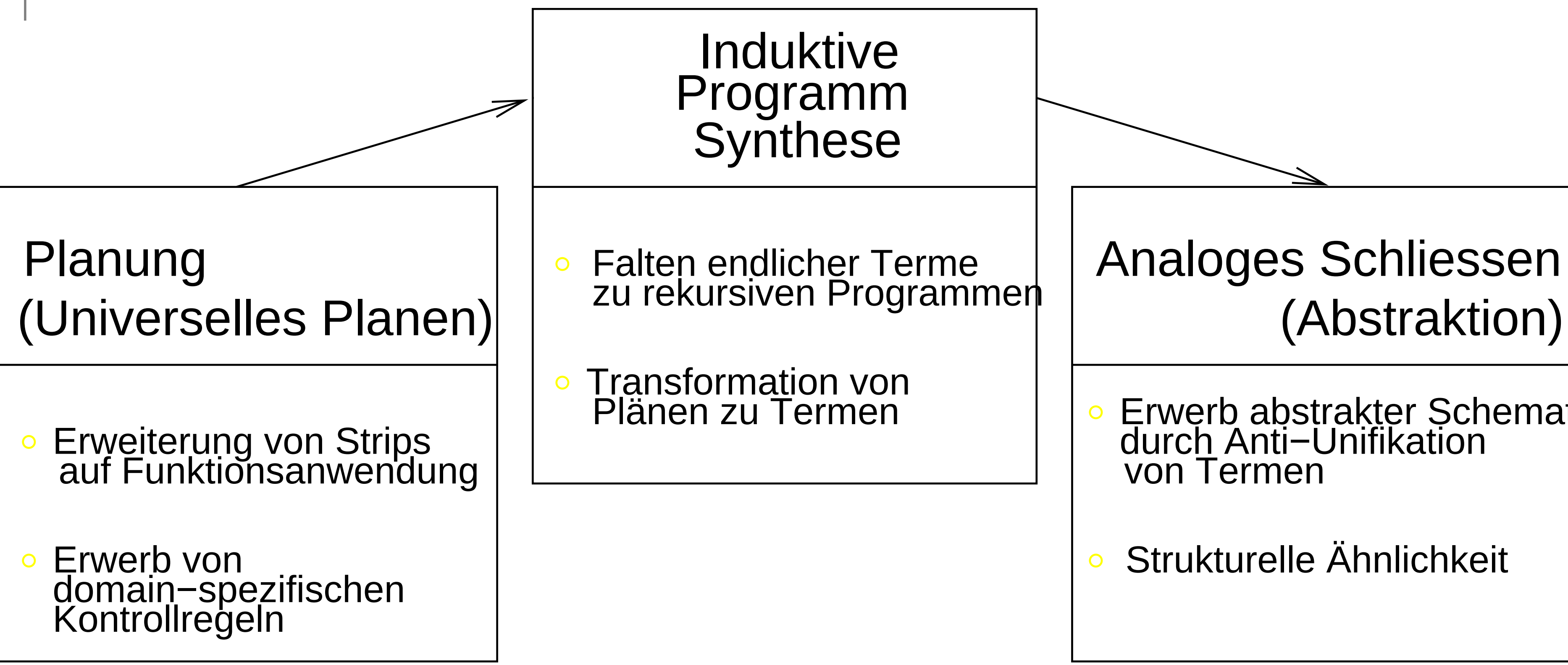
Cognitive Systems II - Machine Learning

SS 2005

Part III: Learning Programs and Strategies

Functional Programs, Program Schemes, Traces, Folding

Lernen aus Problemlöse-Erfahrung



Klassifikations- und Handlungsregeln

einfache Klassifikationsregel (Entscheidungsbaum)

```
if (x1 = 1) & (x2 = 0) then k else not k
```

rekursive Klassifikationsregel (definite Hornklauseln)

```
ancestor(P, F) :- parent(P,F).
ancestor(P, F) :- parent(P, I), ancestor(I, F).
```

einfache Handlungsregel (Entscheidungsbaum)

```
if (pressure=high) & (oxygen=low) then reduce-temp else do-nothing
```

rekursive Handlungsregel (funktionales Programm)

```
unloadall(tr, s) = if empty(tr) then s
                   else unload(getobj(tr), unloadall(restobj(tr), s))
```

$\hookrightarrow$ Lernen rekursiver Handlungsregeln mit induktiver Programmsynthese

Programmsynthese

Automatisches Programmieren:

Automatisierung/Unterstützung eines möglichst groSSen Teils des Software-Entwicklungsprozesses.

(Spezifikations-Akquisition, Umsetzung in Programmcode, Verifikation, Validierung, ...)

Programmsynthese: Automatisierung oder Unterstützung der Generierung von Programmcode aus Spezifi kationen.

Programmsynthese

- Deduktive Programmsynthese: vollständige, formale (nicht ausführbare) Spezifi kationen.

  last(l) $\Leftarrow$ *find z such that for some y, l = y* $\circ$ *[z]*
  where islist(l) and l $\neq$ *[ ]* (Manna & Waldinger)

$$\forall\, x\, \exists\, y\, [Pre(x) \Rightarrow Post(x, y)]$$

$$\forall\, x\, [Pre(x) \Rightarrow Post(x, f(x))]$$

Programmsynthese

- Induktive Programmsynthese: unvollständige Spezifikationen ("Trainingsbeispiele"), I/O-Paare oder *Traces*.

```
last([1]) = 1
last([2 7]) = 7
last([5 3 4]) = 4
```

$\hookrightarrow$ Induktive Programmsynthese ist Maschinelles Lernen

Induktive Programmsynthese

- **Genetisches Programmieren:**
  generate and test.
- **Induktive Logische Programmierung:**
  meist *bottom-up* Generalisierung aus positiven und negativen Beispielen (*relative least general generalizations*)
- **Funktionale Programmsynthese:**
  1. Umwandlung von I/O-Beispielen in Berechnungsspuren (*traces*)
  2. Rekurrenz-Entdeckung (Falten endlicher Terme)

Fuktionale Programmsynthese

- Schritt 2: Programmsynthese aus Berechnungsspuren
  - $\hookrightarrow$ Pattern-Matching auf Termen
  - $\hookrightarrow$ Theoretische Basis von Summers (1977)
    Synthese von Lisp-Programmen aus
    Berechnungsspuren
- Schritt 1 basiert dagegen auf Hintergrundwissen

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$
\begin{aligned}
F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\
& \ldots, \\
& p_k(x) \rightarrow f_k(x), \\
& T \rightarrow C(F(b(x)), x))
\end{aligned}
$$

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$
\begin{aligned}
F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\
& \ldots, \\
& p_k(x) \rightarrow f_k(x), \\
& T \rightarrow C(F(b(x)), x))
\end{aligned}
$$

$p_1, \ldots p_k$: Prädikate *atom(*b_i*(x))*

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$
\begin{aligned}
F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\
& \ldots, \\
& p_k(x) \rightarrow f_k(x), \\
& T \rightarrow C(F(b(x)), x))
\end{aligned}
$$

b, b_i: Basis-Funktionen $car(x)$, $cdr(x)$ und Kombinationen ($cadr(x)$, $cddr(x)$, ...)

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$
\begin{aligned}
F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\
& \ldots, \\
& p_k(x) \rightarrow f_k(x), \\
& T \rightarrow C(F(b(x)), x))
\end{aligned}
$$

$f_1, \ldots, f_k$: S-Expression (nil, b, $cons(w, x)$)

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$
\begin{aligned}
F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\
& \ldots, \\
& p_k(x) \rightarrow f_k(x), \\
& T \rightarrow C(F(b(x)), x))
\end{aligned}
$$

T: Wahrheitswert “true”

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$\begin{aligned} F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\ & \ldots, \\ & p_k(x) \rightarrow f_k(x), \\ & T \rightarrow C(F(b(x)), x)) \end{aligned}$$

$C(w, x)$: cons-Expression, in der w genau einmal vorkommt

Summers' RPS

Hypothesensprache: linear rekursives Programmschema (RPS)

$$
\begin{aligned}
F(x) \leftarrow \quad & (p_1(x) \rightarrow f_1(x), \\
& \ldots, \\
& p_k(x) \rightarrow f_k(x), \\
& T \rightarrow C(F(b(x)), x))
\end{aligned}
$$

$$
\begin{aligned}
u(x) \leftarrow \quad & (\text{atom(cdr(x))} \rightarrow \text{cons(x, nil)}, \\
& T \rightarrow \text{cons(cons(car(x), nil), u(cdr(x)))})
\end{aligned}
$$

Beispiel

I/O Beispiele:

nil $\longrightarrow$ nil

$(A) \longrightarrow ((A))$

$(A\ B) \longrightarrow ((A)\ (B))$

$(A\ B\ C) \longrightarrow ((A)\ (B)\ (C))$

Beispiel

nil $\rightarrow$ nil

$(A) \rightarrow ((A))$

$(A\ B) \rightarrow ((A)\ (B))$

$(A\ B\ C) \rightarrow ((A)\ (B)\ (C))$

Berechnungsspuren:

$F_L(x) \leftarrow$ (atom(x) $\rightarrow$ nil,
atom(cdr(x)) $\rightarrow$ cons(x, nil),
atom(cddr(x)) $\rightarrow$ cons(cons(car(x), nil), cons(cdr(x), nil)),
T $\rightarrow$ cons(cons(car(x), nil), cons(cons(cadr(x),nil),
cons(cddr(x),nil))))

Beispiel

$$F_L(x) \leftarrow \quad (\text{atom(x)} \rightarrow \text{nil},$$
$$\text{atom(cdr(x))} \rightarrow \text{cons(x, nil)},$$
$$\text{atom(cddr(x))} \rightarrow \text{cons(cons(car(x), nil), cons(cdr(x), nil))},$$
$$\text{T} \rightarrow \text{cons(cons(car(x), nil), cons(cons(cadr(x),nil),}$$
$$\text{cons(cddr(x),nil))))}$$

Synthetisiertes Programm:

$$\text{unpack(x)} \leftarrow \quad (\text{atom(x)} \rightarrow \text{nil},$$
$$\text{T} \rightarrow \text{u(x)})$$
$$\text{u(x)} \leftarrow \quad (\text{atom(cdr(x))} \rightarrow \text{cons(x, nil)},$$
$$\text{T} \rightarrow \text{cons(cons(car(x), nil), u(cdr(x))))}$$

Rekurrenz-Detektion

$$
\begin{aligned}
F_L(x) \leftarrow \quad (&\text{atom(x)} \rightarrow \text{nil}, \\
&\text{atom(cdr(x))} \rightarrow \text{cons(x, nil)}, \\
&\text{atom(cddr(x))} \rightarrow \text{cons(cons(car(x), nil), cons(cdr(x), nil))}, \\
&\text{T} \rightarrow \text{cons(cons(car(x), nil), cons(cons(cadr(x),nil),} \\
&\qquad\qquad\qquad \text{cons(cddr(x),nil))))}
\end{aligned}
$$

Rekurrenz-Detektion

$F_L(x) \leftarrow$ (atom(x) $\rightarrow$ nil,

atom(cdr(x)) $\rightarrow$ cons(x, nil),

atom(cddr(x)) $\rightarrow$ cons(cons(car(x), nil), cons(cdr(x), nil)),

T $\rightarrow$ cons(cons(car(x), nil), cons(cons(cadr(x),nil),

cons(cddr(x),nil))))

Differenzen:

$p_2(x) = p_1($cdr(x)$)$

$p_3(x) = p_2($cdr(x)$)$

$p_4(x) = p_3($cdr(x)$)$

Rekurrenz-Relation:

$p_1(x) =$ atom(x)

$p_{k+1}(x) = p_k($cdr(x)$)$ for $k = 1, 2, 3$

Rekurrenz-Detektion

$F_L(x) \leftarrow$ (atom(x) $\rightarrow$ nil,
atom(cdr(x)) $\rightarrow$ cons(x, nil),
atom(cddr(x)) $\rightarrow$ cons(cons(car(x), nil), cons(cdr(x), nil)),
T $\rightarrow$ cons(cons(car(x), nil), cons(cons(cadr(x),nil),
cons(cddr(x),nil))))

Differenz:

$f_2(x) = \textit{cons}(x, f_1(x))$

Rekurrenz-Relation:

$f_1(x) = \textit{nil}$

$f_2(x) = \text{cons}(x, f_1(x))$

Rekurrenz-Detektion

$F_L(x) \leftarrow$ (atom(x) $\rightarrow$ nil,
atom(cdr(x)) $\rightarrow$ cons(x, nil),
atom(cddr(x)) $\rightarrow$ cons(cons(car(x), nil), cons(cdr(x), nil)),
T $\rightarrow$ cons(cons(car(x), nil), cons(cons(cadr(x),nil),
cons(cddr(x),nil))))

Differenz:

$f_3(x) =$ *cons(cons(car(x),nil)*, $f_2(cdr(x)))$

$f_4(x) =$ *cons(cons(car(x),nil)*, $f_3(cdr(x)))$

Rekurrenz-Relation:

$f_2(x) =$ *cons*$(x, f_1(x))$

$f_{k+1}(x) =$ *cons(cons(car(x),nil)*, $f_k($cdr(x)$))$ for $k = 2, 3$

Falten endlicher Terme

- Wird eine Rekurrenz-Relation in einem endlichen Term erkannt,

  $p_1(x) =$ *atom(x)*

  $p_{k+1}(x) = p_k($*cdr(x)*$)$ for $k = 1, 2, 3$

  $f_1(x) =$ *nil*

  $f_2(x) =$ *cons*$(x, f_1(x))$

  $f_{k+1}(x) =$ *cons(cons(car(x),nil)*, $f_k(cdr(x)))$ for $k = 2, 3$

- so kann der Term in eine rekursive Funktion gefaltet werden.

  u(x) $\leftarrow$ $($atom(cdr(x)) $\rightarrow$ cons(x, nil),

  T $\rightarrow$ cons(cons(car(x), nil), u(cdr(x)))$)$

Summers' Synthese-Theorem (1)

- Entfalten einer rekursiven Funktion: Syntaktisches Einsetzen des Funktionskörpers für den Funktionsaufruf mit entsprechender Substitution der Parameter
- Ausnutzen der Beziehung zwischen einem RPS und seinen Entfaltungen für die Induktion (Faltung) eines RPS aus einem endlichen Term.
- Term wird als k-te Approximation der gesuchten Funktion $\mathbf{F}(x)$ angenommen.

Folge von Unfoldings für *unpack(x)*

für keine Eingabe defi niert $\qquad U^0 \leftarrow \Omega$

Folge von Unfoldings für *unpack(x)*

für keine Eingabe defi niert $U^0 \leftarrow \Omega$

für leere Liste defi niert $U^1 \leftarrow$ (atom(x) $\rightarrow$ nil,

$T \rightarrow \Omega)$

Folge von Unfoldings für *unpack(x)*

für keine Eingabe definiert $\quad U^0 \leftarrow \Omega$

für leere Liste definiert

$$U^1 \leftarrow (\text{atom(x)} \rightarrow \text{nil}, \\ T \rightarrow \Omega)$$

für leere und einelementige Liste definiert

$$U^2 \leftarrow (\text{atom(x)} \rightarrow \text{nil}, \\ \text{atom(cdr(x))} \rightarrow \text{cons(x, nil)}, \\ T \rightarrow \Omega)$$

Folge von Unfoldings für *unpack(x)*

für keine Eingabe definiert $U^0 \leftarrow \Omega$

für leere Liste definiert
$$U^1 \leftarrow \begin{array}{l}(\text{atom(x)} \rightarrow \text{nil},\\ T \rightarrow \Omega)\end{array}$$

für leere und einelementige Liste definiert

$$U^2 \leftarrow \begin{array}{l}(\text{atom(x)} \rightarrow \text{nil},\\ \text{atom(cdr(x))} \rightarrow \text{cons(x, nil)},\\ T \rightarrow \Omega)\end{array}$$

...
für Listen bis n Elemente definiert

Summers' Synthese-Theorem (2)

- $\hookrightarrow$ Partielle Ordnung über der Menge der Funktionsapproximationen (aufsteigende Kette).
- $\hookrightarrow$ Supremum der Kette von Funktionsapproximationen ist die gesuchte Funktion (Fixpunkt-Semantik).

Summers' Synthese-Theorem (2)

- $\hookrightarrow$ Partielle Ordnung über der Menge der Funktionsapproximationen (aufsteigende Kette).
- $\hookrightarrow$ Supremum der Kette von Funktionsapproximationen ist die gesuchte Funktion (Fixpunkt-Semantik).

Wird ein aus Beispielen konstruiertes endliches Programm (Berechnungsspuren) als k-tes Unfolding eines unbekannten rekursiven Programms aufgefasst

und können Rekurrenz-Beziehungen im endlichen Programm entdeckt werden,

dann kann ein rekursives Programm, das dieses endliche Programm erzeugen kann, induziert werden!

Verallgemeinerung

- $+$ Sprachunabhängig: Terme einer beliebigen Termalgebra $\mathcal{T}_\Sigma(X)$.
- $+$ *Mengen* rekursiver Funktionen mit beliebiger (aber nicht wechselseitiger) *calls*-Relation.
- $+$ Beliebige Anzahl von formalen Parametern, mit wechselseitigen Abhängigkeiten.
- $+$ Unvollständige Entfaltungen.

$\hookrightarrow$ Mächtigere Hypothesensprache

$\hookrightarrow$ Verallgemeinerte Synthesetheoreme

Hypothesensprache

RPS $\mathcal{S} = (\mathcal{G}, t_0)$ mit $t_0 \in \mathcal{T}_{\Sigma \cup \Phi}(X)$ und $\mathcal{G}$ als System von n Gleichungen:

$$\begin{aligned} \mathcal{G} \quad = \langle \quad & G_1(x_1, \ldots, x_{m_1}) = t_1, \\ & \vdots \\ & G_n(x_1, \ldots, x_{m_n}) = t_n \rangle \end{aligned}$$

mit $t_i \in \mathcal{T}_{\Sigma \cup \Phi}(X), i = 1 \ldots n$.

Hypothesensprache

RPS $\mathcal{S} = (\mathcal{G}, t_0)$ mit $t_0 \in \mathcal{T}_{\Sigma \cup \Phi}(X)$ und $\mathcal{G}$ als System von n Gleichungen:

$$\mathcal{G} = \langle \quad G_1(x_1, \ldots, x_{m_1}) = t_1, \\ \vdots \\ G_n(x_1, \ldots, x_{m_n}) = t_n \rangle$$

mit $t_i \in \mathcal{T}_{\Sigma \cup \Phi}(X), i = 1 \ldots n$.

$\hookrightarrow$ Summers: *eine* Gleichung G_i

Hypothesensprache

RPS $\mathcal{S} = (\mathcal{G}, t_0)$ mit $t_0 \in \mathcal{T}_{\Sigma \cup \Phi}(X)$ und $\mathcal{G}$ als System von n Gleichungen:

$$\begin{aligned}\mathcal{G} \quad = \langle \quad & G_1(x_1, \ldots, x_{m_1}) = t_1, \\ & \vdots \\ & G_n(x_1, \ldots, x_{m_n}) = t_n \rangle\end{aligned}$$

mit $t_i \in \mathcal{T}_{\Sigma \cup \Phi}(X), i = 1 \ldots n$.

- Signatur Σ
- Funktionsvariablen Φ mit $\Sigma \cap \Phi = \emptyset$ und Stelligkeit $\alpha(G_i) = m_i > 0$

Hypothesensprache

RPS $\mathcal{S} = (\mathcal{G}, t_0)$ mit $t_0 \in \mathcal{T}_{\Sigma \cup \Phi}(X)$ und $\mathcal{G}$ als System von n Gleichungen:

$$\begin{aligned} \mathcal{G} \quad = \langle \quad & G_1(x_1, \ldots, x_{m_1}) = t_1, \\ & \vdots \\ & G_n(x_1, \ldots, x_{m_n}) = t_n \rangle \end{aligned}$$

mit $t_i \in \mathcal{T}_{\Sigma \cup \Phi}(X), i = 1 \ldots n$.

- Abstraktes funktionales Programm mit t_0 als initialer Aufruf (“main”)
  und $\mathcal{G}$ als Menge rekursiver Funktionen.

Hypothesensprache

RPS $\mathcal{S} = (\mathcal{G}, t_0)$ mit $t_0 \in \mathcal{T}_{\Sigma \cup \Phi}(X)$ und $\mathcal{G}$ als System von n Gleichungen:

$$\begin{aligned} \mathcal{G} \quad = \langle \quad & G_1(x_1, \ldots, x_{m_1}) = t_1, \\ & \vdots \\ & G_n(x_1, \ldots, x_{m_n}) = t_n \rangle \end{aligned}$$

mit $t_i \in \mathcal{T}_{\Sigma \cup \Phi}(X), i = 1 \ldots n$.

- Linke Seite einer Gleichung: Programmkopf (Parameter x_i paarweise verschieden).
- Rechte Seite einer Gleichung: Programmkörper (t_i enthält mindestens einen Aufruf von G_i und evtl. Aufrufe von $G_j \in \mathcal{G}$).

Beispiel

$\mathcal{G} = \quad \langle$

ModList(l, n) = if(empty(l),
nil,
cons(if(eq0(Mod(n, head(l))), T, nil),
ModList(tail(l), n))),

Mod(k, n) = if($<$(k, n), k, Mod($-$(k, n), n))

$\rangle$

$t_0 = ModList(l, n)$

Beispiel

$\mathcal{G} = \quad \langle$

ModList(l, n) = if(empty(l),
nil,
cons(if(eq0(Mod(n, head(l))), T, nil),
ModList(tail(l), n))),

Mod(k, n) = if($<$(k, n), k, Mod($-$(k, n), n))

$\rangle$

$t_0 = ModList(l, n)$

- Zwei rekursive Gleichungen.
- *ModList* ruft *Mod* auf.

Beispiel

$\mathcal{G} = \quad \langle$

ModList(l, n) = if(empty(l),
nil,
cons(if(eq0(Mod(n, head(l))), T, nil),
ModList(tail(l), n))),

Mod(k, n) = if($<$(k, n), k, Mod($-$(k, n), n))

$\rangle$

$t_0 = ModList(l, n)$

- t_0: uninstantiierter Aufruf von *ModList*
- Initialisierung *ModList([9, 4, 7], 8)*
- Einbettung *head(Modlist([9, 4, 7], 8))*

Beispiel

$\mathcal{G} = \quad \langle$

ModList(l, n) = if(empty(l),

nil,

$\uparrow$ cons(if(eq0(Mod(n, head(l))), T, nil),

ModList(tail(l), n))),

Mod(k, n) = if($<$(k, n), k, Mod($-$(k, n), n))

$\rangle$

$t_0 = ModList(l, n)$

- Unvollständige Entfaltungen

  (Planer-/Nutzergenerierte Berechnungsspuren)

Allgemeiner Ansatz

- Betrachtung eines RPS als Termersetzungssystem
- Charakterisierung der Sprache eines RPS
- Induktion: Ist gegebener Term Element der Sprache?

Folding-Algorithmus

1. Hypothetische Segmentierung eines Terms in potentielle Rekursionspunkte: Rekurrenzrelation entlang Pfaden zu Ω's (Backtracking-Punkt).
2. Bildung eines maximalen Funktions-Körpers durch Anti-Unifi kation der Segmente.
3. Ermittlung der Substitutionsvorschrift.

Beispielterm

```
 (EMPTY L) NIL
ONS (IF (EQ0 (IF (< (HEAD L) N) (HEAD L) OMEGA)) T NIL)
IF (EMPTY (TAIL L)) NIL
(CONS (IF (EQ0 (IF (< (HEAD (TAIL L)) N) (HEAD (TAIL L))
    (IF (< (- (HEAD (TAIL L)) N) N) (- (HEAD (TAIL L)) N) OMEGA)))
  T NIL)
 (IF (EMPTY (TAIL (TAIL L))) NIL
  (CONS (IF (EQ0
     (IF (< (HEAD (TAIL (TAIL L))) N) (HEAD (TAIL (TAIL L)))
      (IF (< (- (HEAD (TAIL (TAIL L))) N) N) (- (HEAD (TAIL (TAIL L))) N)
       (IF (< (- (- (HEAD (TAIL (TAIL L))) N) N) N)
        (- (- (HEAD (TAIL (TAIL L))) N) N) OMEGA))))
    T NIL)
   (IF (EMPTY (TAIL (TAIL (TAIL L)))) NIL
    (CONS (IF (EQ0
       (IF (< (HEAD (TAIL (TAIL (TAIL L)))) N) (HEAD (TAIL (TAIL (TAIL L))))
        (IF (< (- (HEAD (TAIL (TAIL (TAIL L)))) N) N)
         (- (HEAD (TAIL (TAIL (TAIL L)))) N)
         (IF (< (- (- (HEAD (TAIL (TAIL (TAIL L)))) N) N) N)
          (- (- (HEAD (TAIL (TAIL (TAIL L)))) N) N) OMEGA))))
      T NIL)   OMEGA)))))))
```

Generierung von Berechnungsspuren

- Folding: rein syntaktisch, allgemein formulierbar, mit Pattern-Matching lösbar
- Generierung von Berechnungsspuren: domain-abhängig

Generierung von Berechnungsspuren

- Folding: rein syntaktisch, allgemein formulierbar, mit Pattern-Matching lösbar
- Generierung von Berechnungsspuren: domain-abhängig

$\hookrightarrow$ Summers: strukturelle Listenprobleme
$\hookrightarrow$ I/O Beispiele können *eindeutig* in Berechnungsspuren transformiert werden.

Generierung von Berechnungsspuren

- Folding: rein syntaktisch, allgemein formulierbar, mit Pattern-Matching lösbar
- Generierung von Berechnungsspuren: domain-abhängig

$\hookrightarrow$ Programming by Demonstration: Erfassung über Benutzerinteraktion

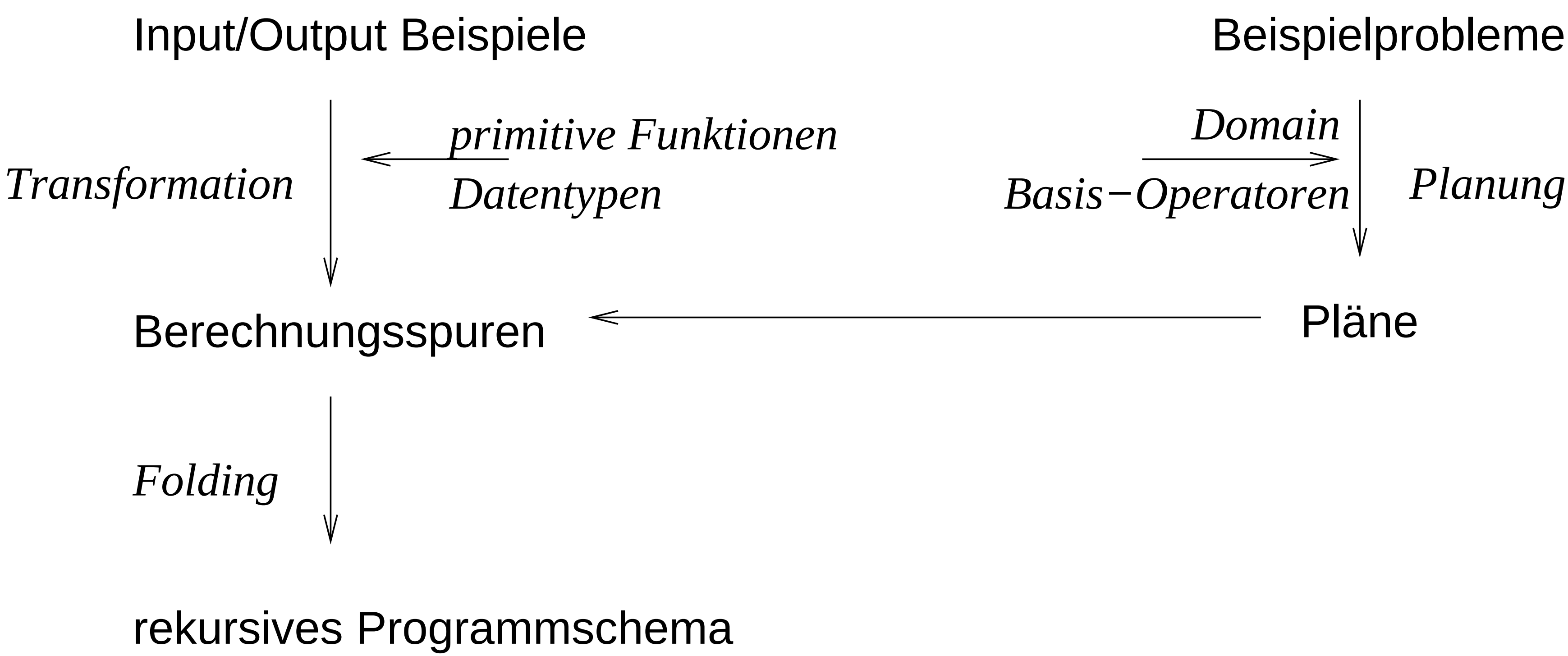
Generierung von Berechnungsspuren

- Folding: rein syntaktisch, allgemein formulierbar, mit Pattern-Matching lösbar
- Generierung von Berechnungsspuren: domain-abhängig

$\hookrightarrow$ KI-Planung: Pläne als Folge von Operator-Anwendungen zur Transformation eines Anfangszustand (Input) in einen Zielzustand (Output)
$\hookrightarrow$ Pläne sind Berechnungsspuren

Generierung von Berechnungsspuren

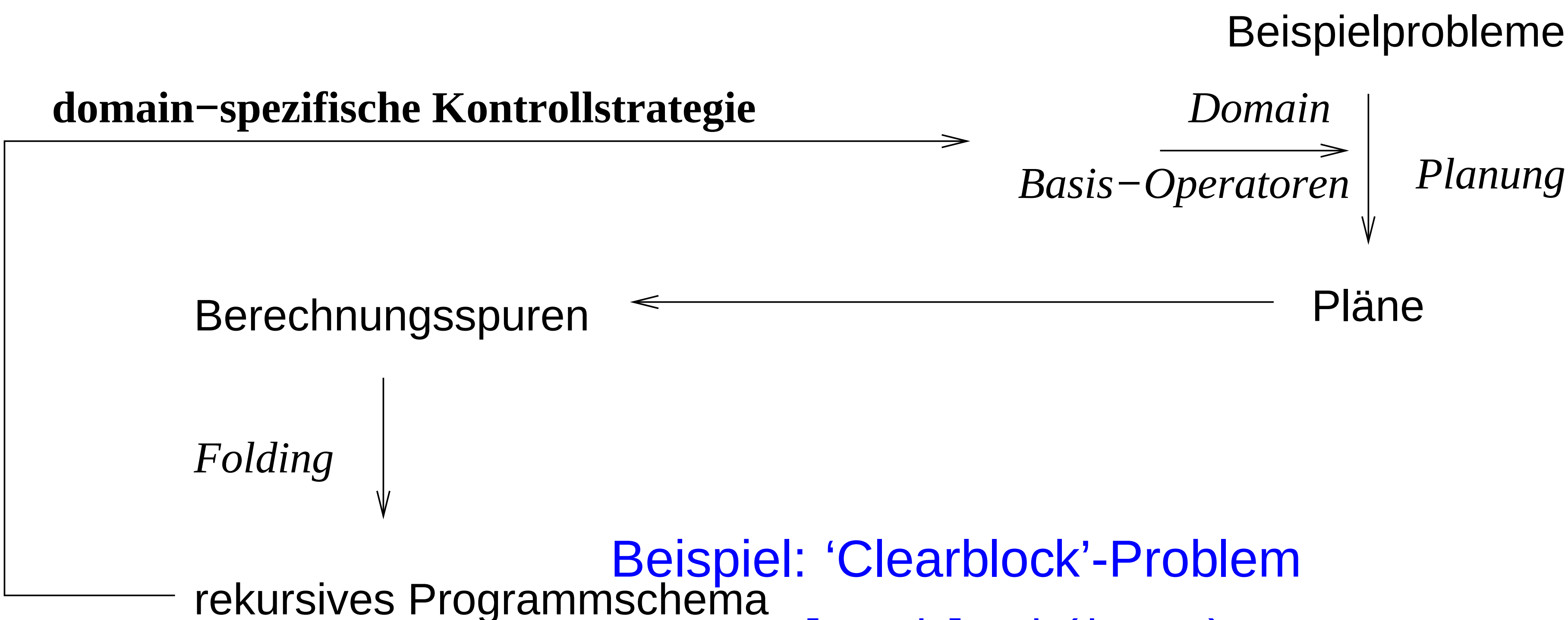
- Folding: rein syntaktisch, allgemein formulierbar, mit Pattern-Matching lösbar
- Generierung von Berechnungsspuren: domain-abhängig



Planung und Programmsynthese

- Zustandsbasiertes Planen
  (Strips, Graphplan)
  Suche im Zustandsraum
  $\hookrightarrow$ i.A. exponentieller Aufwand
- Lösung 1: domainspezifi sche Kontrollregeln zur Steuerung der Suche
  Problem: *knowledge engineering*
- Lösung 2: Lernen von Kontrollregeln aus Beispielplänen

Planung und Programmsynthese



Beispiel: ‘Clearblock’-Problem

```
clearblock(b, s) =
IF clear(b, s) THEN s
ELSE puttable(topof(b, s),
  clearblock(topof(b, s), s))
```

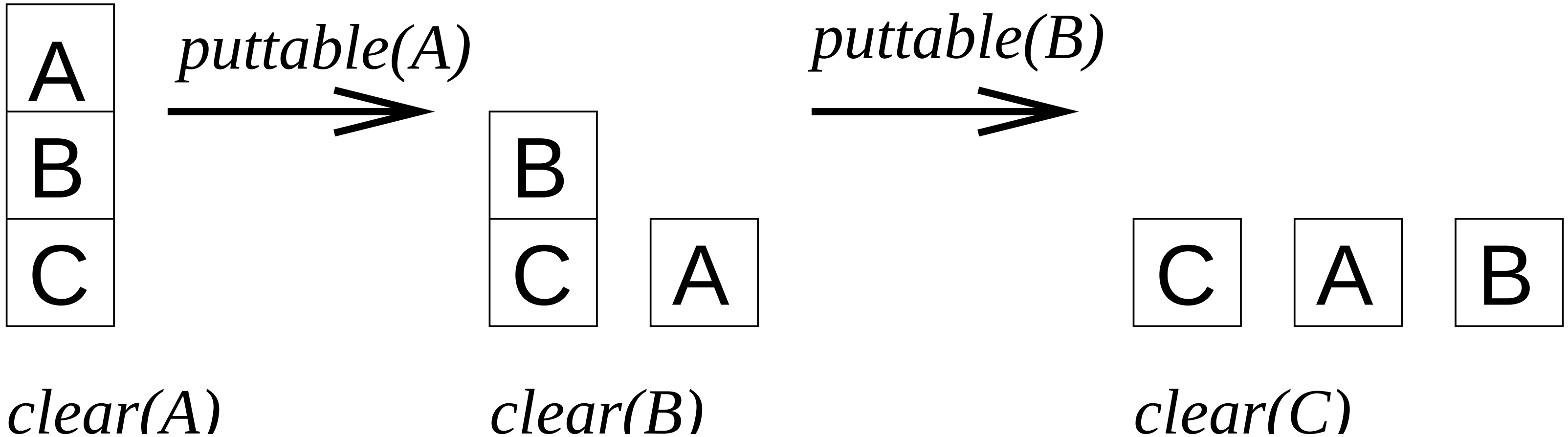
Problemlösen und Programmsynthese

- Lernen aus Problemlöseerfahrung $\approx$ Induktives Lernen aus Beispielen
- Lösen von Programmierproblemen als Spezialfall
  - "Handsimulation" der ersten n Beispiele
    Sortieren einer Liste mit bis zu drei Elementen
  - Generalisierung zur rekursiven Lösung
    insertion-sort

$\hookrightarrow$ Modellierung von Programmsynthese als Wissenserwerb aus Problemlöseerfahrung

$\hookrightarrow$ Induktive Programmsynthese als methodischer Zugang zum Lernen aus Erfahrung

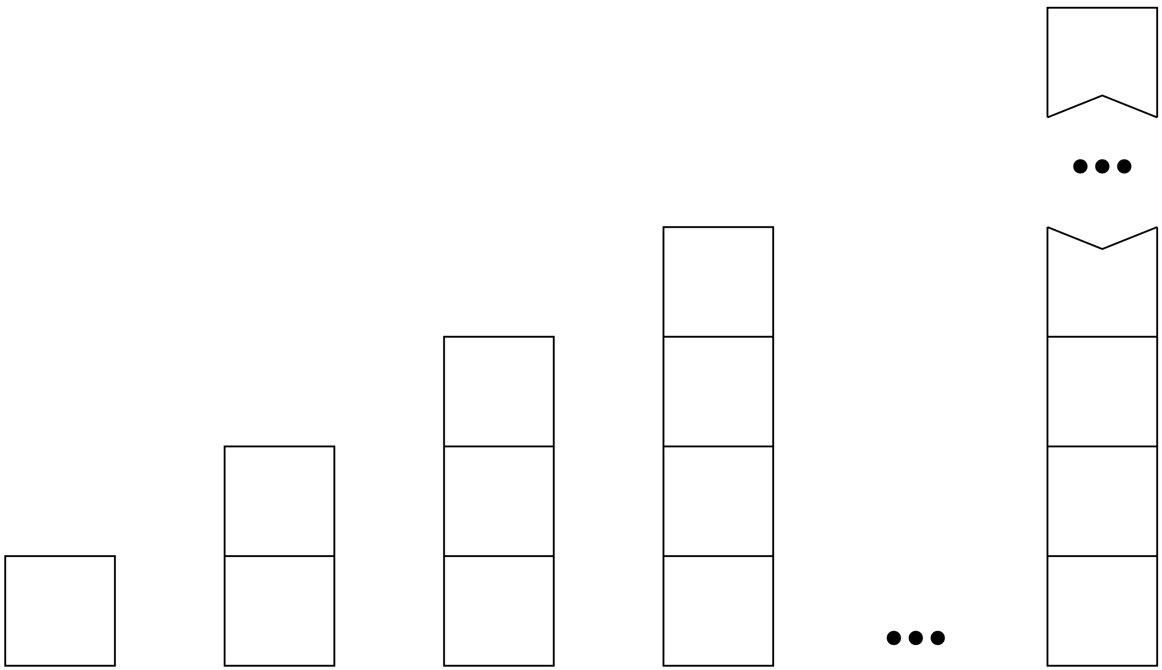
Exploration eines Problembereichs



Gegeben Problemlöse-Operatoren und Ziel

Generalisierung

über Lösungsstrukturen:



“*Wende solange* `puttable(x)` *an,*
bis `clear(`*Zielblock*`)`”

Generalisierung

über Lösungsstrukturen:

Repräsentiert als rekursives Programmschema:

```
clearblock(b, s) =
IF clear(b, s) THEN s
ELSE puttable(topof(b, s),
     clearblock(topof(b, s), s))
```

Schema-Erwerb (e.g. Norman & Rumelhart)

$\hookrightarrow$ Teilziel-Struktur

$\hookrightarrow$ Domain-spezifi sche Strategie

Generalisierung

über Lösungsstrukturen:

Repräsentiert als rekursives Programmschema:

```
clearblock(b, s) =
IF clear(b, s) THEN s
ELSE puttable(topof(b, s),
     clearblock(topof(b, s), s))
```

Teilzielbildung

Generalisierung

über Lösungsstrukturen:

Repräsentiert als rekursives Programmschema:

```
clearblock(b, s) =
IF clear(b, s) THEN s
ELSE puttable(topof(b, s),
          clearblock(topof(b, s), s))
```

Aktionsfolge

Generalisierung

über Lösungsstrukturen:

Repräsentiert als rekursives Programmschema:

```
clearblock(b, s) =
IF clear(b, s) THEN s
ELSE puttable(topof(b, s),
     clearblock(topof(b, s), s))
```

Relevante Eigenschaften

Generalisierung

über Lösungsstrukturen:

Repräsentiert als rekursives Programmschema:

```
clearblock(b, s) =
IF clear(b, s) THEN s
ELSE puttable(topof(b, s),
     clearblock(topof(b, s), s))
```

Objekt-Auswahl

Bewertung

- Induktive Programmsynthese als Ansatz zum Maschinellen Lernen von Kontrollregeln
- “Wissensentdeckung”: Identifikation von und Generalisierung über Regularitäten (Chomsky, LAD)
- Bezug zum menschlichen Lernen: *learning by doing* nicht begrenzt auf “chunking” von Regeln (Makro-Lernen, ACT), Lernen von Problemlöse-Strategien als Generalisierung über Problemlöse-Erfahrung
- Kognitive Modellierung: statt Simulation konkreter empirischer Daten Formulierung grundlegender Mechanismen (Identifikation relevanter Merkmale, Schema-Erwerb)

Anwendungen

- Lernen von Kontrollregeln für Planungssysteme
aktuell: verteiltes Planen für *ubiquituos computing* in der Transportlogistik
- Lernen von XSL Transformationen mit *recursive template application*
(Umwandlung von XML-Dokumenten)
- Analoges und fallbasiertes Schliessen als Spezialfall von Induktion
(Anti-Unifi kations-Algorithmen, Bezug zu psychologischen Theorien)