

Automatisierte Logik und Programmierung

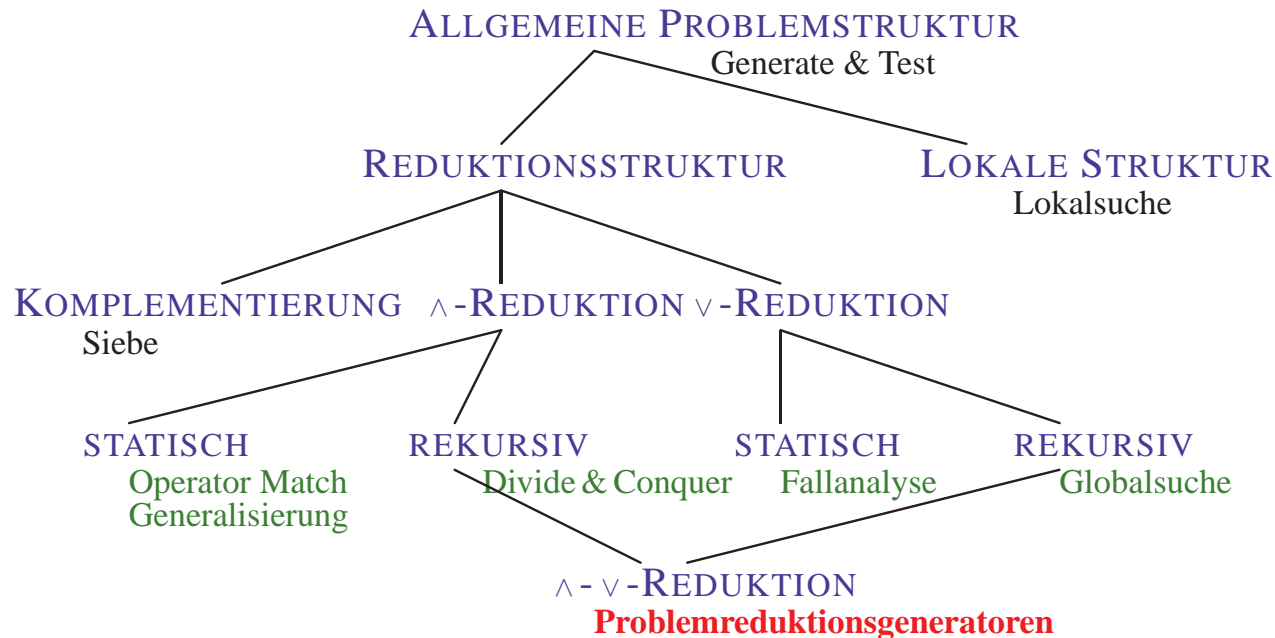
Einheit 21

Problemreduktionsgeneratoren



1. Algorithmenschema
2. Korrektheit
3. Synthesestrategie

PROBLEMREDUKTIONSGENERATOREN



- **Rekursive \vee - \wedge -Reduktion von Problemen**

- Gesamtlösung ist “Summe” unabhängiger Einzellösungen (\vee -Reduktion)
- Einzellösungen aus Teillösungen zusammengesetzt (\wedge -Reduktion)
- Verallgemeinert Dynamisches Programmieren, Spielbaumsuche, ...

- **Häufig Kern der Lösung von “Optimierungs”problemen**

- Betrachte Problem als größtes einer Menge von Teilproblemen
- Löse Teilprobleme durch \vee - \wedge -Reduktion auf kleinere Teilprobleme

- **Synthese ähnlich zu Divide & Conquer Techniken**

● Bestimme äquivalente Zustände in Automaten

- Definition: $p \cong q \equiv \forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$
- Bestimme Relationen $\not\cong(p, q, k) \equiv \exists w \in \Sigma^{\leq k}. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \notin F$
Dann ist $p \cong q \Leftrightarrow \not\cong(p, q, |Q|)$, wobei Q Menge aller Zustände
- Lösungsverfahren verwendet Startwerte für $k=0$ und \vee - \wedge -Reduktion
 - $\not\cong(p, q, 0) \equiv p \in F \Leftrightarrow q \notin F$
 - $\not\cong(p, q, k+1) \equiv \not\cong(p, q, k) \vee \exists a \in \Sigma. \not\cong(\delta(p, a), \delta(q, a), k)$
- \wedge -Reduktion: Berechnung von $\delta(p, a) / \delta(q, a)$ und Test $\not\cong(p', q', k)$
- \vee -Reduktion: Disjunktion aller Möglichkeiten für Unterschiedlichkeit

● Syntaxanalyse in Chomsky-Normalform Grammatiken

- Definition: $w \in L(G) \equiv S \xrightarrow{*} w$
- Bestimme $V_{i,j} \equiv \{A \in V \mid A \xrightarrow{*} w_i \dots w_j\}$, dann $w \in L(G) \Leftrightarrow S \in V_{1,|w|}$
- Lösungsverfahren verwendet Startwerte für $i=j$ und \vee - \wedge -Reduktion
 - $V_{i,i} \equiv \{A \in V \mid A \rightarrow w_i \in P\}$
 - $V_{i,j} \equiv \{A \in V \mid \exists i \leq k < j. \exists A \rightarrow BC \in P. B \in V_{i,k} \wedge C \in V_{k+1,j}\}$
- \wedge -Reduktion für jedes k : Eintrag von A benötigt $B \in V_{i,k}$ und $C \in V_{k+1,j}$
- \vee -Reduktion: Vereinigung der Resultate aller k zwischen i und $j-1$

● Rucksackproblem

- $(g_1..g_n, a_1..a_n, G, A) \in KP \equiv \exists J \subseteq \{1..n\}. \sum_{i \in J} g_i \leq G \wedge \sum_{i \in J} a_i \geq A \}$
- Bestimme Nutzen $N(k, g) \equiv \max \{ \sum_{i \in J} a_i \mid J \subseteq \{1..k\} \wedge \sum_{i \in J} g_i \leq g \}$
 Dann ist $(g_1..g_n, a_1..a_n, G, A) \in KP \Leftrightarrow N(n, G) \geq A$
- Lösungsverfahren verwendet Startwerte $k=0$ und $g=0$ und Reduktion
 - $N(k, 0) = 0, N(0, g) = 0,$
 - $N(k, g) = \max\{N(k-1, g-g_k) + a_k, N(k-1, g)\}$ \wedge -Reduktion: Berechnung von $g-g_k / k-1, N(k, g),$ Addition von a_k
 \vee -Reduktion: Auswahl aus zwei Möglichkeiten

● (Alle) kürzesten Verbindungen in gewichteten Graphen

- $d(i, j) \equiv \min\{w(i_1, \dots, i_n) \mid i_1, \dots, i_n \text{ Pfad von } i \text{ nach } j\}$
- Bestimme $d^k(i, j) \equiv \min\{w(i_1, \dots, i_n) \mid i=i_1, \dots, i_n=j \text{ mit } i_2, \dots, i_{n-1} \leq k\}$
 Dann ist $d(i, j) := d^{|V|}(i, j)$
- Lösungsverfahren verwendet Startwerte für $k=0$ und \vee - \wedge -Reduktion
 - $d^0(i, j) = w(i, j)$ Floyd Warshall Algorithmus
 - $d^{k+1}(i, j) = \min\{d^{k-1}(i, j), d^{k-1}(i, k-1) + d^{k-1}(k-1, j)\}$

SCHEMATISIERUNG DER DARSTELLUNG

● Aufruf einer Hilfsfunktion für Teilprobleme

FUNCTION `Parse`($G=(V, T, P, S)$, $w:\text{Grammars} \times T^*$)
RETURNS $\{A:V \mid A \xrightarrow{*} w\}$
 $\equiv \text{aux}(G, w, 1, |w|)$

● Hilfsfunktion reduziert auf mehrere Teilprobleme

FUNCTION `aux`($G, w, i, j:\text{Grammars} \times T^* \times \mathbb{N} \times \mathbb{N}$)
WHERE $i, j \in \{1..|w|\} \wedge i \leq j$ RETURNS $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$
 \equiv if $i=j$ then $\{A \mid A \rightarrow w_i \in P\}$
else $\{A \mid \exists k \in \{i..j-1\}. \exists B, C:V. A \rightarrow BC \in P \wedge$
 $B \in \text{aux}(G, w, i, k) \wedge C \in \text{aux}(G, w, k+1, j)\}$

● Komponenten der Hilfsfunktion

- Dekomposition der Suche $i..j$ an Stelle k in $\{i..k, k+1..j\}$
- Rekursive Bestimmung der Lösung für $i..j$ wenn $i < j$
Direkte Bestimmung der Lösung wenn $i=j$
- Komposition der Lösungen für k (Rückwärtsanwendung der Regeln)
- Gesamtergebnis ist Vereinigung aller so entstandenen Lösungen

FUNCTION `aux`($G, w, i, j:\text{Grammars} \times T^* \times \mathbb{N} \times \mathbb{N}$) ...
 $\equiv \bigcup_k (\text{Compose}_k \circ (\text{aux}_{k_1} \times \text{aux}_{k_2}) \circ \text{Decompose}_k)(i, j)$

- **Verallgemeinertes Divide & Conquer Schema**

- Unabhängige Divide & Conquer Algorithmen für jede Einzellösung
 - Dekompositionen und Kompositionen verschieden
 - Teilprobleme können einander überlappen
 - Basisfall für primitive Eingaben wird “triviales” Divide & Conquer
- Lösung durch **Vereinigung** aller Einzellösungen berechnen

- **Allgemeines Algorithmenschema (der Hilfsfunktion)**

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x,y]\}$
 $\equiv \bigcup_i (\text{Compose}_i \circ (f_{i_1} \times \dots \times f_{i_k}) \circ \text{Decompose}_i)(x)$

- **4 zentrale Komponenten der Algorithmentheorie**

- $\text{Decompose}_i: D \rightarrow D_{i_1} \times \dots \times D_{i_k}$ Aufspalten der Eingabe in Teilprobleme
 - Hilfsfunktionen $f_{i_j}: D_{i_j} \rightarrow R_{i_j}$ evtl. rekursiver Aufruf von f
 - $\text{Compose}_i: R_{i_1} \times \dots \times R_{i_k} \rightarrow R$ Zusammensetzen der Teillösungen
 - Wohlfundierte Ordnung \succ für Terminierungsgarantie
- ...sowie **Erweitertes Strong Problem Reduction Principle**, Domains, ...

Korrektheit folgt aus wenigen Voraussetzungen

↪ Folie 8

● Schematischer Algorithmus der Hilfsfunktion

FUNCTION *aux* ... $\equiv \bigcup_k (Compose_k \circ (aux_{k_1} \times aux_{k_2}) \circ Decompose_k) (i, j)$

● Formale Komponenten des Schemas

– Grammatik G und Wort w sind Konstante für die Hilfsfunktion *aux*

– $Decompose_k(i, j) \equiv ((i, k), (k+1, j))$ $(k \in \{i..j-1\})$

$Decompose_j(i, j) \equiv i$ $(i=j)$

– $aux_{k_1/2}(x, y) \equiv aux(G, w, x, y)$ $(k \in \{i..j-1\})$

$aux_j(x) \equiv x$

– $Compose_k(V_1, V_2) \equiv \{A \in V \mid \exists B \in V_1. \exists C \in V_2. A \rightarrow BC \in P\}$ $(k \in \{i..j-1\})$

$Compose_j(y) \equiv \{A \in V \mid A \rightarrow w_y \in P\}$

● Instantiierter Algorithmus

FUNCTION *aux*(G, w, i, j : Grammars \times $T^* \times \mathbb{N} \times \mathbb{N}$) WHERE $i, j \in \{1..|w|\} \wedge i \leq j$

RETURNS $\{A \in V \mid A \xrightarrow{*} w_i \dots w_j\}$

$\equiv \bigcup \{ \{A \in V \mid A \rightarrow w_i \in P\} \mid i=j \}$

$\cup \bigcup \{ \{A \in V \mid \exists B \in aux(G, w, i, k). \exists C \in aux(G, w, k+1, j). A \rightarrow BC \in P\} \mid k \in \{i..j-1\} \}$

SYNTAXANALYSE ALS PROBLEMRDUKTIONSGENERATOR

```
FUNCTION Parse( $G=(V,T,P,S)$ ,  $w:\text{Grammars}\times T^*$ ) RETURNS  $\{A:V \mid A \xrightarrow{*} w\}$   
 $\equiv aux(G,w,1,|w|)$   
FUNCTION aux( $G,w,i,j:\text{Grammars}\times T^*\times\mathbb{N}\times\mathbb{N}$ ) WHERE  $i,j\in\{1..|w|\}\wedge i\leq j$   
  RETURNS  $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$   
 $\equiv \bigcup\{\{A\in V \mid A\rightarrow w_i\in P\} \mid i=j\}$   
   $\cup \bigcup\{\{A\in V \mid \exists B\in aux(G,w,i,k). \exists C\in aux(G,w,k+1,j). A\rightarrow BC\in P\}$   
     $\mid k\in\{i..j-1\}\}$ 
```

● Schematischer Algorithmus nach Simplifikationen

```
FUNCTION Parse( $G=(V,T,P,S)$ ,  $w:\text{Grammars}\times T^*$ )  
  RETURNS  $\{A:V \mid A \xrightarrow{*} w\}$   
 $\equiv aux(G,w,1,|w|)$   
FUNCTION aux( $G,w,i,j:\text{Grammars}\times T^*\times\mathbb{N}\times\mathbb{N}$ )  
  WHERE  $i,j\in\{1..|w|\}\wedge i\leq j$   
  RETURNS  $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$   
 $\equiv$  if  $i=j$  then  $\{A\in V \mid A\rightarrow w_j\in P\}$   
  else  $\{A\in V \mid \exists k\in\{i..j-1\}. \exists B\in aux(G,w,i,k).$   
     $\exists C\in aux(G,w,k+1,j). A\rightarrow BC\in P\}$ 
```


KORREKTHEIT VON PROBLEMREDUKTIONSGENERATOREN

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x,y]\}$
 $\equiv \bigcup_i (\text{Compose}_i \circ (f_{i_1} \times \dots \times f_{i_k}) \circ \text{Decompose}_i)(x)$

ist korrekt, wenn 5 Axiome erfüllt sind

1. O ist rekursiv zerlegbar in O_{D_i} , $O_{i_1} \times \dots \times O_{i_k}$ und O_{C_i} (SPRP)

$$O[x, z] \Leftrightarrow \exists i:\mathbb{N}, \bar{y}_i = (y_{i_1}, \dots, y_{i_k}): D_{i_1} \times \dots \times D_{i_k}, \bar{w}_i = (w_{i_1}, \dots, w_{i_k}): R_{i_1} \times \dots \times R_{i_k} \\ O_{D_i}[x, \bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \wedge O_{C_i}[\bar{w}_i, z]$$

2. Dekompositionen erfüllen O_{D_i} und ‘verkleinern’ Problem

FUNCTION $f_{d_i}(x:D)$ WHERE $I[x]$
RETURNS $\{\bar{y}_i: D_{i_1} \times \dots \times D_{i_k} \mid O_{D_i}[x, \bar{y}_i] \wedge x \succ \bar{y}_i \wedge I_{i_1, \dots, i_k}[\bar{y}_i]\}$

wobei $x \succ \bar{y}_i \equiv x \succ y_{i_j}$ für alle j mit $D_{i_j} = D$

3. Hilfsfunktionen f_{i_j} erfüllen O_{i_j}

FUNCTION $f_{i_j}(y_{i_j}: D_{i_j})$ WHERE $I_{i_j}[y_{i_j}]$ RETURNS $\{w_{i_j}: R_{i_j} \mid O_{i_j}[y_{i_j}, w_{i_j}]\}$

4. Kompositionen erfüllen O_{C_i}

FUNCTION $f_{c_i}(\bar{w}_i: R_{i_1} \times \dots \times R_{i_k})$ WHERE true RETURNS $\{z_i: R \mid O_{C_i}[\bar{w}_i, z_i]\}$

5. Verkleinerungsrelation \succ ist wohlfundierte Ordnung auf D

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x,y]\}$
 $\equiv \bigcup_i (\text{Compose}_i \circ (f_{i_1} \times \dots \times f_{i_k}) \circ \text{Decompose}_i)(x)$

- **Partielle Korrektheit: strukturelle Induktion über (D, \succ)**

- $z \in f(x) \Leftrightarrow \exists i. z \in (\text{Compose}_i \circ (f_{i_1} \times \dots \times f_{i_k}) \circ \text{Decompose}_i)(x)$
- $\text{Decompose}_i[x]$ liefert alle $\bar{y}_i = (y_{i_1}, \dots, y_{i_k})$ mit $O_{D_i}[x, \bar{y}_i]$ und $x \succ \bar{y}_i$ Axiom 2
- Jedes $f_{i_j}(y_{i_j})$ liefert alle w_{i_j} mit $O_{i_j}[y_{i_j}, w_{i_j}]$ Axiom 3
- $\text{Compose}_i[w_{i_1}, \dots, w_{i_k}]$ liefert alle z_i mit $O_{C_i}[w_{i_1}, \dots, w_{i_k}, z_i]$ Axiom 4
- z ist eines dieser z_i und es gilt $O[x, z]$ Axiom 1

$$O[x, z] \Leftrightarrow \exists i:\mathbb{N}, \bar{y}_i = (y_{i_1}, \dots, y_{i_k}):D_{i_1} \times \dots \times D_{i_k}, \bar{w}_i = (w_{i_1}, \dots, w_{i_k}):R_{i_1} \times \dots \times R_{i_k} \cdot \\ O_{D_i}[x, \bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \wedge O_{C_i}[\bar{w}_i, z]$$

- **Terminierung: Wohlfundiertheit von \succ**

Axiom 5

- **Rekursion und Basisfälle implizit in Hilfsfunktionen f_{i_j} enthalten**

Es sind viele neue Komponenten zu bestimmen

- **Zusätzliche Datentypen**

- Ein-/Ausgabetypen der Hilfsfunktionen $f_{i_j} (D_{i_j}, R_{i_j})$

- **Zusätzliche Ein-/Ausgabebedingungen**

- Für Dekompositionen (O_{D_i}), Kompositionen (O_{C_i}), Hilfsfunktionen (I_{i_j}, O_{i_j})

- **Komponenten des Algorithmus**

- *Decompose* _{i} : $D \rightarrow \text{Set}(D_{i_1} \times \dots \times D_{i_k})$

Axiom 2

- $f_{i_j}: D_{i_j} \rightarrow R_{i_j}$

Axiom 3

- *Compose* _{i} : $R_{i_1} \times \dots \times R_{i_k} \rightarrow \text{Set}(R)$

Axiom 4

- **Terminierungsordnung für Rekursion**

- Wohlfundierte Ordnung \succ auf D

Axiom 5

Lösungen müssen Beweis der Axiome 1–5 liefern

SYNTHESE EINES \wedge - \vee -REDUKTIONSSALGORITHMUS

SYNTAXANALYSE

Spezifikation: FUNCTION `cyk`(G, w, i, j : Grammars $\times T^* \times \mathbb{N} \times \mathbb{N}$)
WHERE $i, j \in \{1..|w|\} \wedge i \leq j$
RETURNS $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$

Ziel: Bestimme Komponenten des \wedge - \vee -Reduktionsschemas für `cyk`

SYNTHESE EINES \wedge - \vee -REDUKTIONSSALGORITHMUS

SYNTAXANALYSE

Spezifikation: FUNCTION $\text{cyk}(G, w, i, j : \text{Grammars} \times T^* \times \mathbb{N} \times \mathbb{N})$
 WHERE $i, j \in \{1..|w|\} \wedge i \leq j$
 RETURNS $\{A : V \mid A \xrightarrow{*} w_i..w_j\}$

Ziel: Bestimme Komponenten des \wedge - \vee -Reduktionsschemas für cyk

1. Es gibt nur wenige sinnvolle Arten, die Eingabe zu zerlegen

Ein Intervall $[i..j]$ mit $i < j$ kann an jeder Stelle $k \in \{i..j-1\}$ in zwei Teilintervalle zerlegt werden. $[i..i]$ kann nur in die Zahl i “zerlegt” werden.

Dies liefert folgende Komponenten

- $\text{Decompose}_k(i, j) \equiv ((i, k), (k+1, j))$, falls $k \in \{i..j-1\}$, i , falls $k=i=j$
- Extra-Domänen: $D_{k_l} \equiv \mathbb{N} \times \mathbb{N}$, falls $k \in \{i..j-1\}$, \mathbb{N} sonst
- Ausgabebedingungen $O_{D_k}[(i, j), (x_1, y_1), (x_2, y_2)] \equiv i=x_1, k=y_1, k+1=x_2, j=y_2$
 $O_{D_j}[(i, j), x] \equiv x=i=j \quad \mapsto \text{Axiom 2}$
- Wohlfundierte Ordnung: $(i, j) \succ (i', j') \equiv |j-i| > |j'-i'| \quad \mapsto \text{Axiom 5}$

Zerlegungen dieser Art sollten in Wissensbank gespeichert sein

SYNTHESE EINES \wedge - \vee -REDUKTIONENSALGORITHMUS (2)

SYNTAXANALYSE

2. Die Hilfsfunktionen f_{k_l} müssen konstruiert werden

– Auf Intervalle kann rekursiv die Funktion f angewandt werden

Nicht-Intervalle sollten unverändert bleiben werden

– Dies liefert weitere Komponenten und die Gültigkeit von \mapsto **Axiom 3**

```
FUNCTION  $f_{k_l}(x, y: \mathbb{N} \times \mathbb{N})$  WHERE  $x, y \in \{1 \dots |w|\} \wedge x \leq y$   
  RETURNS  $\{A: V \mid A \xrightarrow{*} w_x \dots w_y\}$   
 $\equiv \text{cyk}(G, w, x, y)$  ist korrekt
```

für $k \in \{i..j-1\}$, $l \in \{1, 2\}$ und für $k=j$

```
FUNCTION  $f_j(x: \mathbb{N}): \mathbb{N}$  WHERE true  
  RETURNS  $y$  SUCH THAT  $x=y$   
 $\equiv x$  ist korrekt
```

SYNTHESE EINES \wedge - \vee -REDUKTIONSSALGORITHMUS (3)

SYNTAXANALYSE

3. Konstruiere Ausgabebedingungen von *Compose_k* mit \mapsto **Axiom 1**

$$\begin{aligned}
 A \xrightarrow{*} w_i..w_j &\Leftrightarrow \exists \dots \quad i=x_1 \wedge k=y_1 \wedge k+1=x_1 \wedge j=y_2 \\
 &\wedge V_1 = \{B:V \mid B \xrightarrow{*} w_{x_1}..w_{y_1}\} \wedge V_2 = \{C:V \mid C \xrightarrow{*} w_{x_2}..w_{y_2}\} \\
 &\wedge O_{C_k}[V_1, V_2, A]
 \end{aligned}$$

Umschreiben $V_1 = \{B:V \mid B \xrightarrow{*} w_i..w_k\}$, $V_2 = \{C:V \mid C \xrightarrow{*} w_{k+1}..w_j\}$

liefert $O_{C_k}[V_1, V_2, A] \equiv \exists B \in V_1. \exists C \in V_2. A \xrightarrow{*} BC$ und wegen

Chomsky-NF $O_{C_k}[V_1, V_2, A] \equiv \exists B \in V_1. \exists C \in V_2. A \rightarrow BC \in P$

für $k \in \{i..j-1\}$, $l \in \{1, 2\}$ und für $k=j$

$$A \xrightarrow{*} w_i..w_j \Leftrightarrow \exists x, y:\mathbb{N}. i=j \wedge x=i \wedge y=x \wedge O_{C_j}[y, A]$$

liefert nach Umschreiben $O_C[y, A] \equiv A \xrightarrow{*} w_y$

und wegen Chomsky-NF $O_{C_j}[y, A] \equiv A \rightarrow w_y \in P$

SYNTHESE EINES \wedge - \vee -REDUKTIONSSALGORITHMUS (4)

4. Erneute Synthesen liefern Algorithmen für $Compose_k$ \mapsto Axiom 4

```
FUNCTION  $f_c(V_1, V_2: Set(V) \times Set(V))$  WHERE true  
  RETURNS  $\{A \in V \mid \exists B \in V_1. \exists C \in V_2. A \rightarrow BC \in P\}$   
 $\equiv Compose_k(V_1, V_2)$  ist korrekt
```

für $k \in \{i..j-1\}$, $l \in \{1, 2\}$ und für $k=j$

```
FUNCTION  $f_c(y: \mathbb{N})$  WHERE true  
  RETURNS  $\{A \in V \mid A \rightarrow w_y \in P\}$   
 $\equiv Compose_k(y)$  ist korrekt
```

In beiden Fällen liefert die Ausgabebedingung bereits den Algorithmus

SYNTHESE EINES \wedge - \vee -REDUKTIONERALGORITHMUS (4)

4. Erneute Synthesen liefern Algorithmen für $Compose_k$ \mapsto Axiom 4

```
FUNCTION  $f_c(V_1, V_2: Set(V) \times Set(V))$  WHERE true  
  RETURNS  $\{A \in V \mid \exists B \in V_1. \exists C \in V_2. A \rightarrow BC \in P\}$   
 $\equiv Compose_k(V_1, V_2)$ 
```

ist korrekt

für $k \in \{i..j-1\}, l \in \{1, 2\}$ und für $k=j$

```
FUNCTION  $f_c(y: \mathbb{N})$  WHERE true  
  RETURNS  $\{A \in V \mid A \rightarrow w_y \in P\}$   
 $\equiv Compose_k(y)$ 
```

ist korrekt

In beiden Fällen liefert die Ausgabebedingung bereits den Algorithmus

5. Instantiierter Algorithmus

```
FUNCTION  $cyk(G, w, i, j: Grammars \times T^* \times \mathbb{N} \times \mathbb{N})$   
  WHERE  $i, j \in \{1..|w|\} \wedge i \leq j$  RETURNS  $\{A: V \mid A \xrightarrow{*} w_i..w_j\}$   
 $\equiv \bigcup \{ \{A \in V \mid A \rightarrow w_i \in P\} \mid i=j \}$   
   $\cup \bigcup \{ \{A \in V \mid \exists B \in cyk(G, w, i, k). \exists C \in cyk(G, w, k+1, j). A \rightarrow BC \in P\}$   
     $\mid k \in \{i..j-1\} \}$ 
```

SYNTHESESTRATEGIE IST ÄHNLICH ZU DIVIDE&CONQUER

• Grundstrategie

1. Wähle Dekompositionsstruktur $Decompose_i$ aus Wissensbank
2. Konstruiere Hilfsfunktionen f_{i_j} (Identität oder rekursiver Aufruf von f)
3. Konstruiere Kompositionen $Compose_i$ mit Korrektheitsaxiom 2

$$I[x] \wedge I_{i_1, \dots, i_k}[\bar{y}_i] \wedge O_{D_i}[x, \bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \Rightarrow (O_{C_i}[\bar{w}_i, z] \Leftrightarrow O[x, z])$$

Vereinfache $O[x, z]$ im Kontext zu einer Formel über \bar{w}_i und z

4. Wähle \succ aus der Wissensbank und verifiziere $Decompose_i$
 5. Verifiziere Vollständigkeitsaxiom
- $$I[x] \wedge O[x, z] \wedge O_{C_i}[\bar{w}_i, z] \Rightarrow \exists \bar{y}_i: D_{i_1} \times \dots \times D_{i_k}. (I_{i_1, \dots, i_k}[\bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}])$$
6. Instantiiere Algorithmenschema und vereinfache das Resultat

• Umgekehrte Strategie analog

- Wähle $Compose_i$ aus Wissensbank, bestimme f_{i_j} , $Decompose_i$ und \succ

Durchführung im Detail erheblich aufwendiger

Parametrisierung von Divide & Conquer Wissen

- **Standard-Zerlegungen von Datentypen** $\mapsto \text{Decompose}_i$
 - Endliche Listen, Intervalle: Spaltung in Teillisten/-intervalle an Stelle i
 - Endliche Mengen: Spaltung nach Größe der Elemente
 - Bäume: Spaltung in Wurzel und alle Teilbäume
 - ⋮
- **Standard-Wohlordnungen auf Datentypen** $\mapsto \gamma$
 - Längen und Größenordnungen, Lexikographische Ordnung, etc
- **Standard-Kompositionen für Typen** $\mapsto \text{Compose}_i$
 - Endliche Folgen/Intervalle: Verkettung der Teile
 - Endliche Mengen: Vereinigung von Teilmengen(familien)
 - Bäume: Zusammensetzung von Teilbäumen mit neuer Wurzel
 - ⋮

UNTERSTÜTZENDE TECHNIKEN

- **Heuristische Fixierung der Hilfsfunktionen**

↪ f_{i_j}

- Wähle $f_{i_j} := f$, wo möglich, sonst $f_{i_j} := \text{id}$

- **Inferenzmechanismen zur Steuerung von Vorwärtsinferenz**

- Abgeleitete Vorbedingungen, Fallanalyse, Operator Match

- **Aufspaltung des Reduktionsprinzips in 2 Axiome**

- **Starke Korrektheit** bzgl. Komposition und Dekomposition

$$\forall i:\mathbb{N}, x:D, \bar{y}_i:D_{i_1} \times \dots \times D_{i_k}, \bar{w}_i:R_{i_1} \times \dots \times R_{i_k}, z:R. \quad (1)$$

$$I[x] \wedge I_{i_1, \dots, i_k}[\bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \wedge O_{C_i}[\bar{w}_i, z] \Rightarrow (O_{D_i}[x, \bar{y}_i] \Leftrightarrow O[x, z])$$

$$\forall i:\mathbb{N}, x:D, \bar{y}_i:D_{i_1} \times \dots \times D_{i_k}, \bar{w}_i:R_{i_1} \times \dots \times R_{i_k}, z:R. \quad (2)$$

$$I[x] \wedge I_{i_1, \dots, i_k}[\bar{y}_i] \wedge O_{D_i}[x, \bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \Rightarrow (O_{C_i}[\bar{w}_i, z] \Leftrightarrow O[x, z])$$

Hilfsmittel zur Konstruktion von *Decompose_i* bzw. *Compose_i*

- **Vollständigkeit** bzgl. Komposition

$$\forall i:\mathbb{N}, x:D, \bar{w}_i:R_{i_1} \times \dots \times R_{i_k}, z:R.$$

$$I[x] \wedge O[x, z] \wedge O_{C_i}[\bar{w}_i, z] \Rightarrow \exists \bar{y}_i:D_{i_1} \times \dots \times D_{i_k}. (I_{i_1, \dots, i_k}[\bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}])$$

SYNTHESE MIT UMGEKEHRTER STRATEGIE

BINÄRE SUCHBÄUME FÜR ENDLICHE MENGEN

● Datenstruktur mit Unterstützung für schnelle Suche

- Binärbaum über geordnetem Datentyp α (`nil` oder `fork(t1, a, t2)`)
Tiefenbasierte Suche im Baum muß Ordnung widerspiegeln
- **All-BST**: Bestimme alle legalen binären Suchbäume für Menge M

Spezifikation:

```
FUNCTION All-BST(M:Set( $\alpha$ )) WHERE true  
  RETURNS {t:BinTree( $\alpha$ ) | members(t)=M  $\wedge$  is-bst(t)}
```

Ziel: Bestimme Komponenten des \wedge - \vee -Reduktionsschemas für All-BST

● Variante: Such-optimale binäre Suchbäume

→ Folie 24

- Elemente von α können Gewichte (e.g. Häufigkeiten) haben
- Knoten haben eine Suchtiefe `level(x, t)` im Baum für Menge S
- Kosten `cost(t)` sind gewichtete Summe der Knotenlevel in t

```
FUNCTION O-BST(M:Set( $\alpha$ )) WHERE true  
  RETURNS optima(cost, All-BST(M))
```

wobei `optima(c, S) \equiv {x \in S | \forall y \in S. c(x) \leq c(y)}`

SYNTHESE MIT UMGEKEHRTER STRATEGIE (2)

1. Wähle zwei *Compose* Operationen

- $Compose_1 \equiv \text{fork}$ und $Compose_2 \equiv \text{nil}$
- Liefert $O_{C_1}(\cdot, t) \equiv t = \text{nil}$ und $O_{C_2}(t_1, a, t_2, t) \equiv t = \text{fork}(t_1, a, t_2)$
und $R_1 \equiv \text{Unit}$, $R_{2_{1/3}} \equiv \text{BinTree}(\alpha)$, $R_{2_2} \equiv \alpha$

2. Konstruiere Hilfsfunktionen f_{ij} heuristisch

- $f_1 \equiv \text{id}$, $f_{2_{1/3}} \equiv \text{All-BST}$, $f_{2_2} \equiv \text{id}$
- Liefert $D_1 \equiv \text{Unit}$, $D_{2_{1/3}} \equiv \text{Set}(\alpha)$, $D_{2_2} \equiv \alpha$
und $I_{ij}(y) \equiv \text{true}$ für alle i, j

3. Konstruiere Ausgabebedingungen O_{D_i} mit Axiom 1

- $\text{members}(t) = M \wedge \text{is-bst}(t) \Leftrightarrow \exists y : \text{Unit}. t = \text{nil} \wedge y = () \wedge O_{D_1}[M, Y]$

Umschreiben liefert: $O_{D_1}[M, Y] \equiv M = \emptyset \wedge Y = ()$

- $\text{members}(t) = M \wedge \text{is-bst}(t)$

$$\Leftrightarrow \exists y : \text{Unit}. t = \text{fork}(t_1, a, t_2) \wedge \text{members}(t_1) = M_1 \wedge \text{is-bst}(t_1) \\ \wedge \text{members}(t_2) = M_2 \wedge \text{is-bst}(t_2) \wedge a = y \wedge O_{D_1}[M, (M_1, Y, M_2)]$$

Umschreiben: $O_{D_1}[M, (M_1, Y, M_2)] \equiv M_1 \cup \{Y\} \cup M_2 = M \wedge M_1 < Y \wedge M_2 > Y,$

wobei $M < z \equiv \forall x \in M. x < z$

SYNTHESE MIT UMGEKEHRTER STRATEGIE (3)

4. Konstruiere *Decompose* und \succ mit Axiom 3

- (Mengenwertige) Spezifikation O_{D_1} wird erfüllt durch $\{ () \mid M=\emptyset \}$
- Spezifikation O_{D_1} wird erfüllt durch
$$\{ (M_1, y, M_2) \mid M_1 \cup \{y\} \cup M_2 = M \wedge M_1 < y < M_2 \}$$
Operation ist ineffizient ausführbar, kann aber optimiert werden
- Wegen $|M| > |M_1|$ und $|M| > |M_2|$ wähle wohlfundierte Ordnung
$$M \succ M' \equiv |M| > |M'|$$

5. Instantiiere das Algorithmenschema

```
FUNCTION All-BST(M:Set( $\alpha$ )) WHERE true
  RETURNS {t:BinTree( $\alpha$ ) | members(t)=M  $\wedge$  is-bst(t)}
 $\equiv$  {nil | M= $\emptyset$ }
   $\cup \bigcup$  {fork(All-BST(M1), y, All-BST(M2)) | M1 $\cup$ {y} $\cup$ M2=M  $\wedge$  M1<y<M2}
```

Nach Vereinfachungen

```
FUNCTION All-BST(M:Set( $\alpha$ )) WHERE true
  RETURNS {t:BinTree( $\alpha$ ) | members(t)=M  $\wedge$  is-bst(t)}
 $\equiv$  if M= $\emptyset$  then {nil}
  else {fork(All-BST({x $\in$ M | x<y}), y, All-BST({x $\in$ M | x>y})) | y $\in$ M}
```

Es ist nicht immer eine Vereinigung von Teillösungen

- **\wedge - \vee -Reduktion strebt oft spezifische Werte an**
 - Nichtäquivalente Zustände: Disjunktion von Teillösungen
 - Rucksackproblem: Maximum von Teillösungswerten
 - Kürzeste Verbindungen in Graphen: Minimum von Teillösungswerten
 - O-BST: Kombination kostengünstigster Teillösungswerte
- **Zielstellung ist formuliert im Aufruf der Hilfsfunktion**
 - $\not\equiv(p, q)$: Disjunktion der Teillösungen für $\hat{\delta}(p, w)$, $\hat{\delta}(q, w)$ und alle w
 - Rucksackproblem: Maximaler Nutzen aller Auswahlmöglichkeiten
 - Floyd-Warshall: Minimum aller möglichen Pfade
- **Nachträgliche Bearbeitung aller Teillösungen aufwendig**
 - Oft würden zunächst exponentiell viele Möglichkeiten generiert
 - Zielstellung muß in Reduktionsstruktur integriert werden
 - Disjunktionen, Minima, Maxima sind in jedem Schritt zu bestimmen

\wedge - \vee -Algorithmenschema muß verfeinert werden

VEREINHEITLICHUNG DER DARSTELLUNG

- **Zielstellung ist als Optimierungsproblem formulierbar**

- Rucksackproblem: Maximierung des Nutzens
- Floyd-Warshall / O-BST: Minimierung des Aufwands
- $\not\equiv(p, q)$: “Maximierung” des booleschen Werts (durch Disjunktion)

- **Allgemeine Struktur des Algorithmus**

Schema kann beibehalten werden, wenn man Menge der Optima berechnet

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\text{optima}(c, \{y:R \mid O[x, y]\})$
 $\equiv \text{optima}(c, \text{aux}(x))$

FUNCTION $\text{aux}(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x, y]\}$
 $\equiv \bigcup_i (\text{Compose}_i \circ (f_{i_1} \times \dots \times f_{i_k}) \circ \text{Decompose}_i)(x)$

- **Integration von Aufruf/Optimierung in die Hilfsfunktion**

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\text{optima}(c, \{y:R \mid O[x, y]\})$
 $\equiv \text{optima}(c, \bigcup_i (\text{Compose}'_i \circ (f'_{i_1} \times \dots \times f'_{i_k}) \circ \text{Decompose}_i)(x))$

wobei $f'_{i_j}(y) \hat{=} \text{opt}(c, f_{i_j}(y))$ und $\text{Compose}'_i(\dots, \text{opt}(c, z_i), \dots) \hat{=} \text{Compose}_i(z_1, \dots, z_k)$

Voraussetzungen für Korrektheit sind ähnlich zum einfachen Schema

KORREKTHEIT DES ALLGEMEINEREN REDUKTIONSSCHEMAS

FUNCTION $f(x:D)$ WHERE $I[x]$
RETURNS $\text{optima}(c, \{y:R \mid O[x, y]\})$
 $\equiv \text{optima}(c, \bigcup_i (\text{Compose}_i^\circ (f_{i_1} \times \dots \times f_{i_k}) \circ \text{Decompose}_i) (x))$
ist korrekt, wenn 5 Axiome erfüllt sind

1. O ist optimal rekursiv zerlegbar in O_{D_i} , $O_{i_1} \times \dots \times O_{i_k}$ und O_{C_i} (SPRP)

$O[x, z] \Leftrightarrow \exists i:\mathbb{N}, \bar{y}_i = (y_{i_1}, \dots, y_{i_k}):D_{i_1} \times \dots \times D_{i_k}, \bar{w}_i = (w_{i_1}, \dots, w_{i_k}):R_{i_1} \times \dots \times R_{i_k}.$

$O_{D_i}[x, \bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \wedge O_{C_i}[\bar{w}_i, z]$

$\forall i, \bar{y}_i, \bar{w}_i, \bar{w}'_i, z, z'. O_{D_i}[x, \bar{y}_i] \wedge O_{i_1}[y_{i_1}, w_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w_{i_k}] \wedge O_{C_i}[\bar{w}_i, z]$
 $\wedge O_{i_1}[y_{i_1}, w'_{i_1}] \wedge \dots \wedge O_{i_k}[y_{i_k}, w'_{i_k}] \wedge O_{C_i}[\bar{w}'_i, z'] \wedge c(\bar{w}_i) < c(\bar{w}'_i)$
 $\Rightarrow c(z) < c(z')$

2. Dekompositionen erfüllen O_{D_i} und ‘verkleinern’ Problem

FUNCTION $f_{d_i}(x:D)$ WHERE $I[x]$

RETURNS $\{\bar{y}_i:D_{i_1} \times \dots \times D_{i_k} \mid O_{D_i}[x, \bar{y}_i] \wedge x \succ \bar{y}_i \wedge I_{i_1, \dots, i_k}[\bar{y}_i]\}$

3. Hilfsfunktionen f_{i_j} erfüllen O_{i_j}

FUNCTION $f_{i_j}(y_{i_j}:D_{i_j})$ WHERE $I_{i_j}[y_{i_j}]$ RETURNS $\{w_{i_j}:R_{i_j} \mid O_{i_j}[y_{i_j}, w_{i_j}]\}$

4. Kompositionen erfüllen O_{C_i}

FUNCTION $f_{c_i}(\bar{w}_i:R_{i_1} \times \dots \times R_{i_k})$ WHERE true RETURNS $\{z_i:R \mid O_{C_i}[\bar{w}_i, z_i]\}$

5. Verkleinerungsrelation \succ ist wohlfundierte Ordnung auf D

SYNTHESE OPTIMALER BINÄRER SUCHBÄUME

Spezifikation: FUNCTION O-BST(M:Set(α)) WHERE true
RETURNS optima(cost, All-BST(M))

Ziel: Bestimme Komponenten des allgemeinen Reduktionsschemas

SYNTHESE OPTIMALER BINÄRER SUCHBÄUME

Spezifikation: FUNCTION $O\text{-BST}(M:\text{Set}(\alpha))$ WHERE true
RETURNS $\text{optima}(\text{cost}, \text{All-BST}(M))$

Ziel: Bestimme Komponenten des allgemeinen Reduktionsschemas

1. Bestimme *Compose*, Hilfsfunktionen, *Decompose*, \succ wie zuvor

- $\text{Compose}_1 \equiv \text{fork}$ und $\text{Compose}_2 \equiv \text{nil}$
- $f_1 \equiv \text{id}$, $f_{2_1/3} \equiv \text{All-BST}$, $f_{2_2} \equiv \text{id}$
- $\text{Decompose}_1(M) \equiv \{() \mid M = \emptyset\}$ und $\text{Decompose}_2(M) \equiv \{M_{<y}, y, M_{>y} \mid y \in M\}$
wobei $M_{<y} \equiv \{x \in M \mid x < y\}$
- $M \succ M' \equiv |M| > |M'|$

SYNTHESE OPTIMALER BINÄRER SUCHBÄUME

Spezifikation: FUNCTION $O\text{-BST}(M:\text{Set}(\alpha))$ WHERE true
RETURNS $\text{optima}(\text{cost}, \text{All-BST}(M))$

Ziel: Bestimme Komponenten des allgemeinen Reduktionsschemas

1. Bestimme *Compose*, Hilfsfunktionen, *Decompose*, \succ wie zuvor

- $\text{Compose}_1 \equiv \text{fork}$ und $\text{Compose}_2 \equiv \text{nil}$
- $f_1 \equiv \text{id}$, $f_{2_{1/3}} \equiv \text{All-BST}$, $f_{2_2} \equiv \text{id}$
- $\text{Decompose}_1(M) \equiv \{()\mid M=\emptyset\}$ und $\text{Decompose}_2(M) \equiv \{M_{<y}, y, M_{>y} \mid y \in M\}$
wobei $M_{<y} \equiv \{x \in M \mid x < y\}$
- $M \succ M' \equiv |M| > |M'|$

2. Integriere Optimierung in Komponenten

- $f'_{2_{1/3}} \equiv O\text{-BST}$, alle anderen Komponenten können unverändert bleiben
- Das stärkere Axiom 1 kann nachgewiesen werden

SYNTHESE OPTIMALER BINÄRER SUCHBÄUME

Spezifikation: FUNCTION O-BST($M:\text{Set}(\alpha)$) WHERE true
RETURNS optima(cost, All-BST(M))

Ziel: Bestimme Komponenten des allgemeinen Reduktionsschemas

1. Bestimme *Compose*, Hilfsfunktionen, *Decompose*, \succ wie zuvor

- $Compose_1 \equiv \text{fork}$ und $Compose_2 \equiv \text{nil}$
- $f_1 \equiv \text{id}$, $f_{2_{1/3}} \equiv \text{All-BST}$, $f_{2_2} \equiv \text{id}$
- $Decompose_1(M) \equiv \{()\mid M=\emptyset\}$ und $Decompose_2(M) \equiv \{M_{<y}, y, M_{>y} \mid y \in M\}$
wobei $M_{<y} \equiv \{x \in M \mid x < y\}$
- $M \succ M' \equiv |M| > |M'|$

2. Integriere Optimierung in Komponenten

- $f'_{2_{1/3}} \equiv \text{O-BST}$, alle anderen Komponenten können unverändert bleiben
- Das stärkere Axiom 1 kann nachgewiesen werden

3. Instantiiere und vereinfache das Algorithmenschema

```
FUNCTION O-BST( $M:\text{Set}(\alpha)$ ) WHERE true
RETURNS optima(cost, { $t:\text{BinTree}(\alpha) \mid \text{members}(t)=M \wedge \text{is-bst}(t)$ })
≡ if  $M=\emptyset$  then {nil}
   else optima(cost, {fork(O-BST( $M_{<y}$ ),  $y$ , O-BST( $M_{>y}$ ))  $\mid y \in M$ })
```

- **Wissensbasierte Synthese effizienter Algorithmen**
 - Parametrisierte “Variante” von Divide & Conquer Algorithmen
Sinnvoll, wenn Problem zerlegbar in Disjunktion von Konjunktionen
 - Algorithmenschema verwendet 3 parametrisierte Basiskomponenten
 - Komponenten lassen sich mit vorgeschichtetem formalisiertem Wissen über verwendete Datenstrukturen heuristisch bestimmen
 - Synthesestrategie und Mechanismen ähnlich zu Divide & Conquer
- **Es gibt viele Anwendungsprobleme**
 - Rucksackproblem, Kürzeste Verbindungen in Graphen
 - Nichtäquivalente Zustände in endlichen Automaten
 - Größte Summe (nichtzusammenhängender) Teilsegmente einer Liste
 - Geringster Wortabstand (für Spellchecker mit Korrekturfunktion)
- **Erfolgreich in der Praxis**
 - Schematische Algorithmen lassen sich in wenigen Schritten generieren und nachträglich formal optimieren
 - Theoretische Vorarbeiten sichern Korrektheit der erzeugten Algorithmen