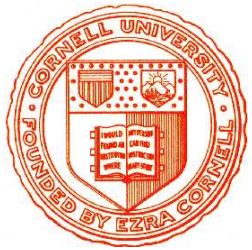
Automatisierte Logik und Programmierung

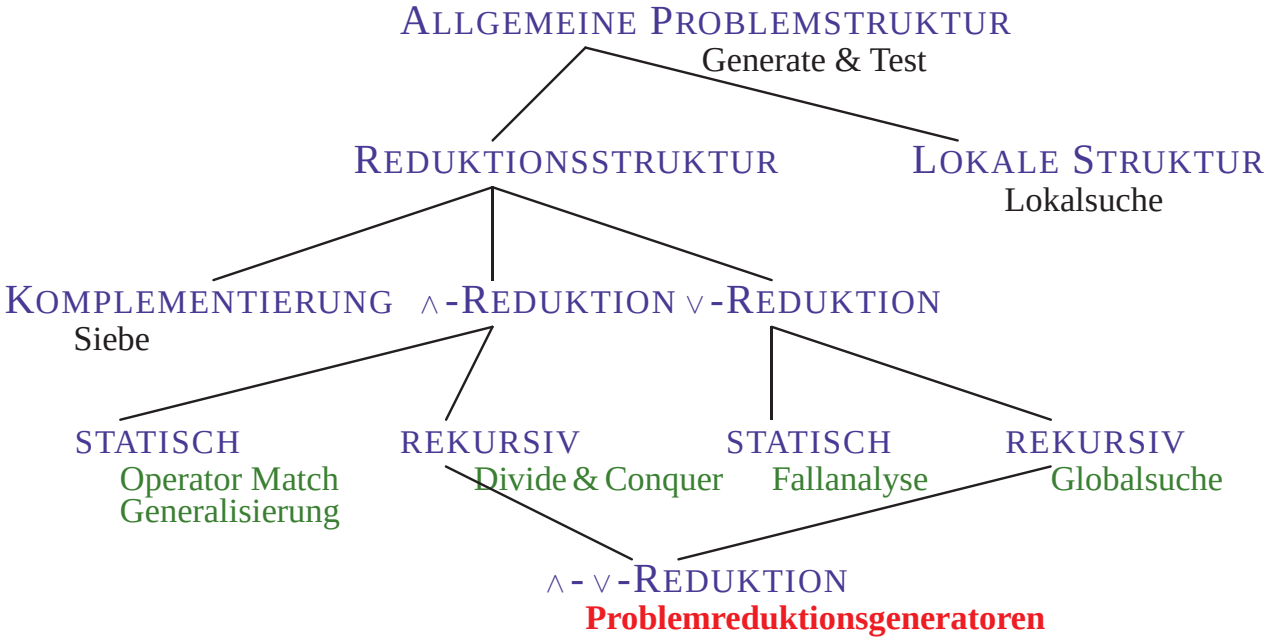
Einheit 21



Problemreduktionsgeneratoren

1. Algorithmenschema
2. Korrektheit
3. Synthesestrategie
4. Optimierungsvariante

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**

Anwendungen von Problemreduktionsgeneratoren

- **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 $p \cong q \Leftrightarrow \neg(\not\cong(p,q,|Q|))$, 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 $V_{i,j} \equiv \{A \in V \mid A \xrightarrow{*} w_k \ldots w_j\}$, dann $w \in L(G) \Leftrightarrow S \in V_{1,|w|}$
  - Lösungsverfahren verwendet Startwerte für $i{=}j$ und $\vee$-$\wedge$-Reduktion
    - $V_{i,i} \equiv \{A \in V \mid A \to w_k \in P\}$
    - $V_{i,j} \equiv \{A \in V \mid \exists i \leq k < j.\ \exists A \to BC \in P.\ B \in V_{i,k} \wedge C \in V_{k+1,j}\}$

  $\wedge$-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 $j{-}1$

Anwendungen von Problemreduktionsgeneratoren (2)

- **Rucksackproblem**
  - $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \equiv \exists J \subseteq \{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$
  - Bestimme Nutzen $N(k,g) \equiv \max\{\Sigma_{i\in J} a_i \mid J \subseteq \{1..k\} \wedge \Sigma_{i\in J} g_i \leq g\}$
    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**
  - $d(i,j) \equiv \min\{w(i_1,..,i_n) \mid i_1,..,i_n$ *Pfad von* i *nach* $j\}$
  - Bestimme $d^k(i,j) \equiv \min\{w(i_1,..,i_n) \mid i{=}i_1,..,i_n{=}j$ ***mit*** $i_2,..,i_{n-1} \leq k\}$
    Dann ist $d(i,j) := d^{|V|}(i,j)$
  - Lösungsverfahren verwendet Startwerte für $k{=}0$ und $\vee$-$\wedge$-Reduktion
    - $d^0(i,j) = w(i,j)$
    - $d^k(i,j) = \min\{d^{k-1}(i,j),\ d^{k-1}(i,k){+}d^{k-1}(k,j)\}$

    Floyd Warshall Algorithmus

Schematisierung der Darstellung

- **Aufruf einer Hilfsfunktion für Teilprobleme**

  FUNCTION Parse(G=(V,T,P,S),w:Grammars×T*)
  RETURNS {A:V | $A \xrightarrow{*} w$}
  ≡ aux(G,w,1,|w|)

- **Hilfsfunktion reduziert auf mehrere Teilprobleme**

  FUNCTION aux(G,w,i,j:Grammars×T*×ℕ×ℕ)
  WHERE i,j∈{1..|w|} ∧ i≤j RETURNS {A:V | $A \xrightarrow{*} w_i..w_j$}
  ≡ if i=j then {A | A→w_i∈P}
  else {A | ∃k∈{i..j-1}. ∃B,C:V. A→BC∈P ∧
  B∈aux(G,w,i,k) ∧ C∈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)
  - Gesamtresultat ist Vereinigung aller so entstandenen Lösungen

    FUNCTION aux(G,w,i,j:Grammars×T*×ℕ×ℕ) ...
    ≡ $\bigcup_k$(*Compose*$_k$ ∘ (aux_{k_1}×aux_{k_2}) ∘ *Decompose*$_k$)(i,j)

Problemreduktionsgeneratoren: Basisversion

- **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)**

  $\texttt{FUNCTION}\ f(x{:}D)\ \texttt{WHERE}\ I[x]\ \texttt{RETURNS}\ \{y{:}R \mid O[x,y]\}$

  $\equiv \bigcup_k (\mathit{Compose}_k \circ (f_{k_1} \times .. \times f_{k_i}) \circ \mathit{Decompose}_k)(x)$
- **4 zentrale Komponenten der Algorithmentheorie**
  - $\mathit{Decompose}_k$: $D \to D_{k_1} \times .. \times D_{k_i}$ Aufspalten der Eingabe in Teilprobleme
  - Hilfsfunktionen f_{k_j}: $D_{k_j} \to R_{k_j}$ evtl. rekursiver Aufruf von f
  - $\mathit{Compose}_k$: $R_{k_1} \times .. \times R_{k_i} \to R$ Zusammensetzen der Teillösungen
  - Wohlfundierte Ordnung $\succ$ für Terminierungsgarantie

  ...sowie **Erweitertes Strong Problem Reduction Principle**, Domains, ...

Korrektheit folgt aus wenigen Voraussetzungen ↦ Folie 8

Syntaxanalyse als Problemreduktionsgenerator

- **Schematischer Algorithmus der Hilfsfunktion**

  FUNCTION $aux\ldots \equiv \bigcup_k (\mathit{Compose}_k \circ (aux_{k_1} \times aux_{k_2}) \circ \mathit{Decompose}_k)\,(i,j)$

- **Formale Komponenten des Schemas**
  - Grammatik G und Wort w sind Konstante für die Hilfsfunktion aux
  - $\mathit{Decompose}_k(i,j) \equiv ((i,k),\ (k{+}1,j))$ $\quad (k \in \{i..j{-}1\})$

    $\mathit{Decompose}_j(i,j) \equiv i$ $\quad (i{=}j)$
  - $aux_{k_{1/2}}(x,y) \equiv aux(G,w,x,y)$ $\quad (k \in \{i..j{-}1\})$

    $aux_j(x) \equiv x$
  - $\mathit{Compose}_k(V_1,V_2) \equiv \{A \in V \mid \exists B \in V_1.\ \exists C \in V_2.\ A \to BC \in P\}$ $\quad (k \in \{i..j{-}1\})$

    $\mathit{Compose}_j(y) \equiv \{A \in V \mid A \to w_y \in P\}$

- **Instantiierter Algorithmus**

```
FUNCTION aux(G,w,i,j:Grammars×T*×ℕ×ℕ) WHERE i,j∈{1..|w|}∧i≤j
  RETURNS {A:V | A ⟶* w_i..w_j}
≡   ⋃{ {A∈V | A→w_i∈P}  | i=j}
  ∪ ⋃{ {A∈V | ∃B∈aux(G,w,i,k). ∃C∈aux(G,w,k+1,j). A→BC ∈ P}
       | k∈{i..j-1} }
```

Syntaxanalyse als Problemreduktionsgenerator

- **Gesamter instantiierter Algorithmus**

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

$\equiv$ aux(G,w,1,|w|)

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 $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$

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

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

- **Schematischer Algorithmus nach Simplifikationen**

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

$\equiv$ aux(G,w,1,|w|)

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 $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$

$\equiv$ if i=j then $\{A \in V \mid A \to w_j \in P\}$
else $\{A \in V \mid \exists k \in \{i..j-1\}.\ \exists B \in aux(G,w,i,k).$
$\exists C \in aux(G,w,k+1,j).\ A \to BC \in P\}$

Korrektheit von Problemreduktionsgeneratoren

FUNCTION $f(x{:}D)$ WHERE $I[x]$ RETURNS $\{y{:}R \mid O[x,y]\}$
$\equiv \bigcup_k (\mathit{Compose}_k \circ (f_{k_1} \times .. \times f_{k_i}) \circ \mathit{Decompose}_k)(x)$
ist korrekt, wenn 5 Axiome erfüllt sind

---

1. O ist rekursiv zerlegbar in O_{D_k}, $O_{k_1} \times .. \times O_{k_i}$ und O_{C_k} *(SPRP)*

$$O[x,z] \Leftrightarrow \exists k{:}\mathbb{N}.\ \bar{y}_k{=}(y_{k_1},..,y_{k_i}){:}D_{k_1}\times..D_{k_i},\ \bar{w}_k{=}(w_{k_1},..,w_{k_i}){:}R_{k_1}\times..R_{k_i}.$$
$$O_{D_k}[x,\bar{y}_k] \wedge O_{k_1}[y_{k_1},w_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i},w_{k_i}] \wedge O_{C_k}[\bar{w}_k,z]$$

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

FUNCTION $f_{d_i}(x{:}D)$ WHERE $I[x]$
RETURNS $\{\bar{y}_k{:}D_{k_1}\times..\times D_{k_i} \mid O_{D_k}[x,\bar{y}_k] \wedge x \succ \bar{y}_k \wedge I_{k_1,..i_k}[\bar{y}_k]\}$

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

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

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

4. Kompositionen erfüllen O_{C_k}

FUNCTION $f_{c_i}(\bar{w}_k{:}R_{k_1}\times..\times R_{k_i})$ WHERE true RETURNS $\{z_k{:}R \mid O_{C_k}[\bar{w}_k,z_k]\}$

5. Verkleinerungsrelation $\succ$ ist wohlfundierte Ordnung auf D

KORREKTHEITSBEWEIS (VGL. §19: DIVIDE & CONQUER)

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

$$\equiv \bigcup_i (\mathit{Compose}_k \circ (f_{k_1} \times .. \times f_{k_i}) \circ \mathit{Decompose}_k)(x)$$

- **Partielle Korrektheit: strukturelle Induktion über $(D,\succ)$**
  - $z \in f(x) \Leftrightarrow \exists i.\ z \in (\mathit{Compose}_k \circ (f_{k_1} \times .. \times f_{k_i}) \circ \mathit{Decompose}_i)(x)$
  - $\mathit{Decompose}_k[x]$ liefert alle $\bar{y}_k = (y_{k_1}, .., y_{k_i})$ mit $O_{D_k}[x, \bar{y}_k]$ und $x \succ \bar{y}_k$ Axiom 2
  - Jedes $f_{k_j}(y_{k_j})$ liefert alle w_{k_j} mit $O_{k_j}[y_{k_j}, w_{k_j}]$ Axiom 3
  - $\mathit{Compose}_k[w_{k_1}, .., w_{k_i}]$ liefert alle z_k mit $O_{C_k}[w_{k_1}, .., w_{k_i}, z_k]$ Axiom 4
  - z ist eines dieser z_k und es gilt $O[x,z]$ Axiom 1

$$O[x,z] \Leftrightarrow \exists i{:}\mathbb{N},\ \bar{y}_k = (y_{k_1}, .., y_{k_i}){:}D_{k_1} \times .. D_{k_i},\ \bar{w}_k = (w_{k_1}, .., w_{k_i}){:}R_{k_1} \times .. R_{k_i}\ .$$
$$O_{D_k}[x, \bar{y}_k] \wedge O_{k_1}[y_{k_1}, w_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i}, w_{k_i}] \wedge O_{C_k}[\bar{w}_k, z]$$

- **Terminierung: Wohlfundiertheit von $\succ$** Axiom 5

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

$\wedge$-$\vee$-Reduktionsschema als Algorithmentheorie

Es sind viele neue Komponenten zu bestimmen

- **Zusätzliche Datentypen**
  - Ein-/Ausgabetypen der Hilfsfunktionen f_{k_j} (D_{k_j}, R_{k_j})
- **Zusätzliche Ein-/Ausgabebedingungen**
  - Für Dekompositionen (O_{D_k}), Kompositionen (O_{C_k}), Hilfsfunktionen (I_{k_j}, O_{k_j})
- **Komponenten des Algorithmus**
  - *Decompose*$_k$: $D \to \text{Set}(D_{k_1} \times .. \times D_{k_i})$ Axiom 2
  - $\boldsymbol{f_{k_j}}$: $D_{k_j} \to R_{k_j}$ Axiom 3
  - *Compose*$_k$: $R_{k_1} \times .. \times R_{k_i} \to \text{Set}(R)$ Axiom 4
- **Terminierungsordnung für Rekursion**
  - Wohlfundierte Ordnung $\succ$ auf D Axiom 5

Lösungen müssen Beweis der Axiome 1–5 liefern

Synthese eines $\wedge$-$\vee$-Reduktionsalgorithmus
Syntaxanalyse

pezifikation: 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 $\{A:V \mid A \xrightarrow{*} w_i..w_j\}$

iel: Bestimme Komponenten des $\wedge$-$\vee$-Reduktionsschemas für cyk

. 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

· $Decompose_k(i,j) \equiv (\ (i,k),\ (k{+}1,j)\ )$, 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 $O_{D_k}[(i,j),(x_1,y_1),(x_2,y_2)] \equiv i{=}x_1, k{=}y_1, k{+}1{=}x_1, j{=}y_2$

$O_{D_j}[(i,j),x] \equiv x{=}i{=}j \quad \mapsto$ **Axiom 2**

· Wohlfundierte Ordnung: $(i,j)\succ(i',j')' \equiv |j{-}i|>|j'{-}i'| \quad \mapsto$ **Axiom 5**

Zerlegungen dieser Art sollten in Wissensbank gespeichert sein

2. Die Hilfsfunktionen f_{kl} 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 ↦ **Axiom 3**

```
FUNCTION f_kl(x,y:ℕ×ℕ) WHERE x,y∈{1..|w|}∧x≤y
  RETURNS {A:V | A ⟶* w_x..w_y}
 ≡ 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:ℕ):ℕ WHERE true
  RETURNS y SUCH THAT x=y
 ≡ x                                             ist korrekt
```

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

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

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

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

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 $O_{C_j}[y, A] \equiv A{\to}w_y \in P$

Synthese eines $\wedge$-$\vee$-Reduktionsalgorithmus (4)

4. Erneute Synthesen liefern Algorithmuen für $Compose_k$ $\mapsto$ **Axiom 4**

```
FUNCTION f_c(V_1,V_2:Set(V)×Set(V)) WHERE true
  RETURNS {A∈V | ∃B∈V_1. ∃C∈V_2. A→BC ∈ 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:ℕ) WHERE true
  RETURNS {A∈V | A→w_y ∈ 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×T*×ℕ×ℕ)
  WHERE i,j∈{1..|w|}∧i≤j  RETURNS {A:V | A ⟶* w_i..w_j}
≡   ⋃{ {A∈V | A→w_i ∈P} | i=j}
  ∪ ⋃{ {A∈V | ∃B∈cyk(G,w,i,k). ∃C∈cyk(G,w,k+1,j). A→BC ∈ P}
       | k∈{i..j-1}}
```

Synthesestrategie ist ähnlich zu Divide&Conquer

- **Grundstrategie**
  1. Wähle Dekompositionsstruktur $Decompose_k$ aus Wissensbank
  2. Konstruiere Hilfsfunktionen f_{k_j} (Identität oder rekursiver Aufruf von f)
  3. Konstruiere Kompositionen $Compose_k$ mit Korrektheitsaxiom 1

     $I[x] \wedge I_{k_1,..i_k}[\bar{y_k}] \wedge O_{D_k}[x, \bar{y_k}] \wedge O_{k_1}[y_{k_1}, w_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i}, w_{k_i}] \Rightarrow (O_{C_k}[\bar{w_k}, z] \Leftrightarrow O[x, z])$

     Vereinfache $O[x, z]$ im Kontext zu einer Formel über $\bar{w_k}$ und z
  4. Wähle $\succ$ aus der Wissensbank und verifiziere $Decompose_k$
  5. Verifiziere Vollständigkeitsaxiom

     $I[x] \wedge O[x, z] \wedge O_{C_k}[\bar{w_k}, z] \Rightarrow \exists \bar{y_k}{:}D_{k_1} \times .. D_{k_i}. (I_{k_1,..i_k}[\bar{y_k}] \wedge O_{k_1}[y_{k_1}, w_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i}, w_{k_i}])$
  6. Instantiiere Algorithmenschema und vereinfache das Resultat
- **Umgekehrte Strategie analog**

  – Wähle $Compose_k$ aus Wissensbank, bestimme f_{k_j}, $Decompose_k$ und $\succ$

Durchführung im Detail erheblich aufwendiger

Unterstützendes Programmierwissen

Parametrisierung von Divide & Conquer Wissen

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

Unterstützende Techniken

- **Heuristische Fixierung der Hilfsfunktionen** $\mapsto f_{k_j}$
  - Wähle f_{k_j}:=f, wo möglich, sonst f_{k_j}:=id
- **Inferenzmechanismen zur Steuerung von Vorwärtsinferenz**
  - Abgeleitete Vorbedingungen, Fallanalyse, Operator Match
- **Aufspaltung des Reduktionsprinzips in 2 Axiome**
  - **Starke Korrektheit** bzgl. Komposition und Dekomposition

$$\forall i{:}\mathbb{N}, x{:}D, \bar{y_k}{:}D_{k_1}\times..D_{k_i}, \bar{w_k}{:}R_{k_1}\times..R_{k_i}, z{:}R. \quad (1)$$
$$I[x]\wedge I_{k_1,..i_k}[\bar{y_k}]\wedge O_{k_1}[y_{k_1},w_{k_1}]\wedge..\wedge O_{k_i}[y_{k_i},w_{k_i}]\wedge O_{C_k}[\bar{w_k},z] \;\Rightarrow\; (O_{D_k}[x,\bar{y_k}]\Leftrightarrow O[x,z])$$
$$\forall i{:}\mathbb{N}, x{:}D, \bar{y_k}{:}D_{k_1}\times..D_{k_i}, \bar{w_k}{:}R_{k_1}\times..R_{k_i}, z{:}R. \quad (2)$$
$$I[x]\wedge I_{k_1,..i_k}[\bar{y_k}]\wedge O_{D_k}[x,\bar{y_k}]\wedge O_{k_1}[y_{k_1},w_{k_1}]\wedge..\wedge O_{k_i}[y_{k_i},w_{k_i}] \;\Rightarrow\; (O_{C_k}[\bar{w_k},z]\Leftrightarrow O[x,z])$$

Hilfsmittel zur Konstruktion von *Decompose*$_k$ bzw. *Compose*$_k$

  - **Vollständigkeit** bzgl. Komposition

$$\forall i{:}\mathbb{N}, x{:}D, \bar{w_k}{:}R_{k_1}\times..R_{k_i}, z{:}R.$$
$$I[x]\wedge O[x,z]\wedge O_{C_k}[\bar{w_k},z] \;\Rightarrow\; \exists \bar{y_k}{:}D_{k_1}\times..D_{k_i}.\,(I_{k_1,..i_k}[\bar{y_k}]\wedge O_{k_1}[y_{k_1},w_{k_1}]\wedge..\wedge O_{k_i}[y_{k_i},w_{k_i}])$$

Synthese mit umgekehrter Strategie
Binäre Suchbäume für endliche Mengen

- **Datenstruktur mit Unterstützung für schnelle Suche**
  - Binärbaum über geordnetem Datentyp α (`nil` oder `fork(`t_1, a, t_2`)`)
    Tiefenbasierte Suche im Baum muß Ordnung widerspiegeln
  - `All-BST`: Bestimme alle legalen binären Suchbäume für Menge M

  Spezifikation:

  ```
  FUNCTION All-BST(M:Set(α))  WHERE true
    RETURNS {t:BinTree(α)|members(t)=M ∧ 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(α))  WHERE true
    RETURNS optima(cost, All-BST(M))
  ```

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

Synthese mit umgekehrter Strategie (2)

1. Wähle zwei *Compose* Operationen

– $Compose_1 \equiv$ `nil` und $Compose_2 \equiv$ `fork`

– Liefert $O_{C_1}((), t) \equiv t$=`nil` und $O_{C_2}(t_1, a, t_2, t) \equiv t$=`fork`$(t_1, a, t_2)$ und $R_1 \equiv$ `Unit`, $R_{2_{1/3}} \equiv$ `BinTree`(α), $R_{2_2} \equiv \alpha$

2. Konstruiere Hilfsfunktionen f_{k_j} heuristisch

– $f_1 \equiv$ `id`, $f_{2_{1/3}} \equiv$ `All-BST`, $f_{2_2} \equiv$ `id`

– Liefert $D_1 \equiv$ `Unit`, $D_{2_{1/3}} \equiv$ `Set`(α), $D_{2_2} \equiv \alpha$ und $I_{k_j}(y) \equiv$ `true` für alle i, j

3. Konstruiere Ausgabebedingungen O_{D_k} mit Axiom 1

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

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

– `members(t)=M` $\wedge$ `is-bst(t)`

$\Leftrightarrow$ $\exists$`y:Unit. t=fork(`t_1`,a,`t_2`)` $\wedge$ `members(`t_1`)=`M_1 $\wedge$ `is-bst(`t_1`)` $\wedge$ `members(`t_2`)=`M_2 $\wedge$ `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

$$\begin{array}{l}\texttt{FUNCTION All-BST(M:Set}(\alpha)\texttt{)}\quad \texttt{WHERE true}\\ \quad \texttt{RETURNS }\{\texttt{t:BinTree}(\alpha) \mid \texttt{members(t)=M} \wedge \texttt{is-bst(t)}\}\\ \equiv\ \{\texttt{nil} \mid \texttt{M}{=}\emptyset\}\\ \quad \cup \bigcup \{\texttt{fork(All-BST}(M_1)\texttt{,y,All-BST}(M_2)\texttt{)} \mid M_1 \cup \{y\} \cup M_2 = M \wedge M_1 < y < M_2\}\end{array}$$

Nach Vereinfachungen

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

Verallgemeinerung des Reduktionsschemas

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
  - – $\not\geqslant(p,q)$: "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 optima(c,`$\{y{:}R \mid O[x,y]\}$`)`
  $\equiv$ `optima(c,` $aux(x)$`)`

  `FUNCTION` $aux(x{:}D)$ `WHERE` $I[x]$ `RETURNS` $\{y{:}R \mid O[x,y]\}$
  $\equiv$ $\bigcup_k$`(`$Compose_k \circ (f_{k_1} \times .. \times f_{k_i}) \circ Decompose_k$`)(`$x$`)`
- **Integration von Aufruf/Optimierung in die Hilfsfunktion**

  `FUNCTION` $f(x{:}D)$ `WHERE` $I[x]$ `RETURNS optima(c,`$\{y{:}R \mid O[x,y]\}$`)`
  $\equiv$ `optima(c,` $\bigcup_k$`(`$Compose_k' \circ (f'_{k_1} \times .. \times f'_{k_i}) \circ Decompose_k$`)(`$x$`))`

  wobei $f'_{k_j}(y) \mathrel{\hat{=}} opt(c, f_{k_j}(y))$ und $Compose_k'(.., opt(c, z_k), .) \mathrel{\hat{=}} Compose_k(z_1, .., z_k)$

  Voraussetzungen für Korrektheit sind ähnlich zum einfachen Schema

Korrektheit des allgemeineren Reduktionsschemas

FUNCTION $f(x{:}D)$ WHERE $I[x]$
  RETURNS optima(c,$\{y{:}R \mid O[x,y]\}$)
$\equiv$ optima(c,$\bigcup_k(Compose_k \circ (f_{k_1} \times .. \times f_{k_i}) \circ Decompose_i)(x)$)

ist korrekt, wenn 5 Axiome erfüllt sind

---

1. O ist optimal rekursiv zerlegbar in O_{D_k}, $O_{k_1} \times .. \times O_{k_i}$ und O_{C_k} (SPRP)

$$O[x,z] \Leftrightarrow \exists i{:}\mathbb{N},\ \bar{y}_k{=}(y_{k_1},..,y_{k_i}){:}D_{k_1}\times..D_{k_i},\ \bar{w}_k{=}(w_{k_1},..,w_{k_i}){:}R_{k_1}\times..R_{k_i}.$$
$$O_{D_k}[x,\bar{y}_k] \wedge O_{k_1}[y_{k_1},w_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i},w_{k_i}] \wedge O_{C_k}[\bar{w}_k,z]$$
$$\forall i,\bar{y}_k,\bar{w}_k,\bar{w}'_k,z,z'.\ O_{D_k}[x,\bar{y}_k] \wedge O_{k_1}[y_{k_1},w_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i},w_{k_i}] \wedge O_{C_k}[\bar{w}_k,z]$$
$$\wedge O_{k_1}[y_{k_1},w'_{k_1}] \wedge .. \wedge O_{k_i}[y_{k_i},w'_{k_i}] \wedge O_{C_k}[\bar{w}'_k,z'] \wedge c(\bar{w}_k){<}c(\bar{w}'_k)$$
$$\Rightarrow c(z){<}c(z')$$

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

FUNCTION $f_{d_i}(x{:}D)$ WHERE $I[x]$
RETURNS $\{\bar{y}_k{:}D_{k_1}\times..\times D_{k_i} \mid O_{D_k}[x,\bar{y}_k] \wedge x \succ \bar{y}_k \wedge I_{k_1,..i_k}[\bar{y}_k]\}$

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

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

4. Kompositionen erfüllen O_{C_k}

FUNCTION $f_{c_i}(\bar{w}_k{:}R_{k_1}\times..\times R_{k_i})$ WHERE true RETURNS $\{z_k{:}R \mid O_{C_k}[\bar{w}_k,z_k]\}$

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

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

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

2. Integriere Optimierung in Komponenten

- $f'_{2_{1/3}} \equiv$ `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:Set(`α`)) WHERE true`
`RETURNS optima(cost,{t:BinTree(`α`)|members(t)=M` $\wedge$ `is-bst(t)})`
$\equiv$ `if M=`$\emptyset$ `then {nil}`
`else optima(cost,{fork(O-BST(M`$_{<y}$`),y,O-BST(M`$_{>y}$`))|y`$\in$`M})`

Problemreduktionsgeneratoren im Rückblick

- **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 vorgespeichertem 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