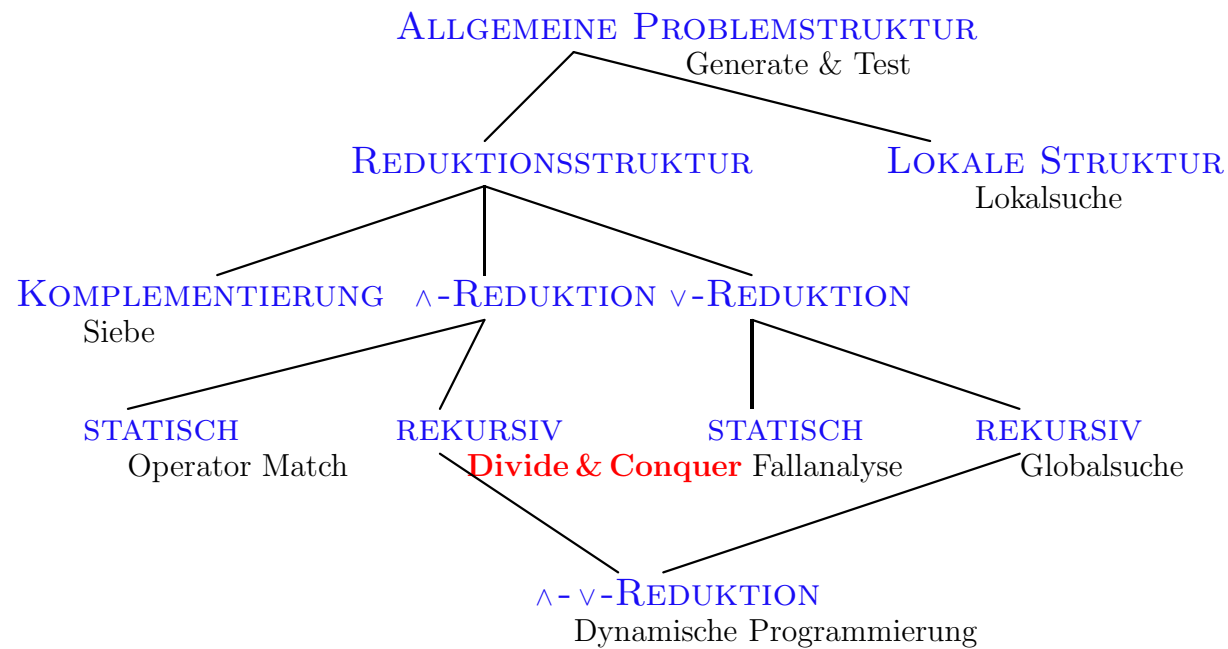
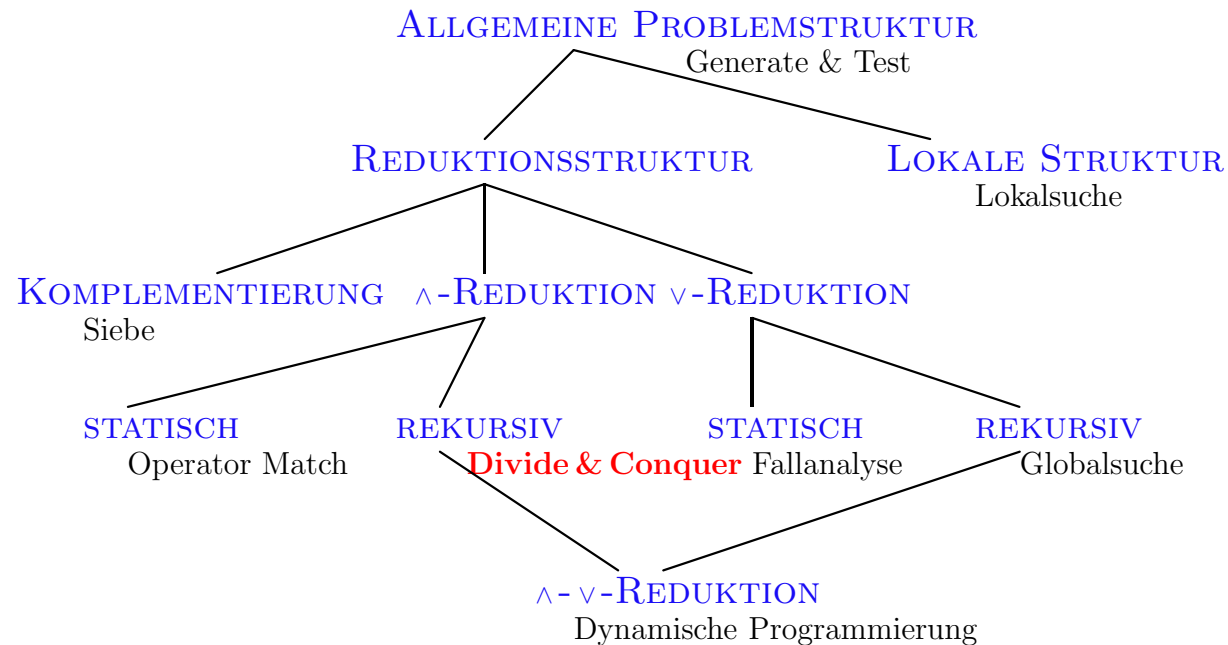


DIVIDE & CONQUER ALGORITHMEN

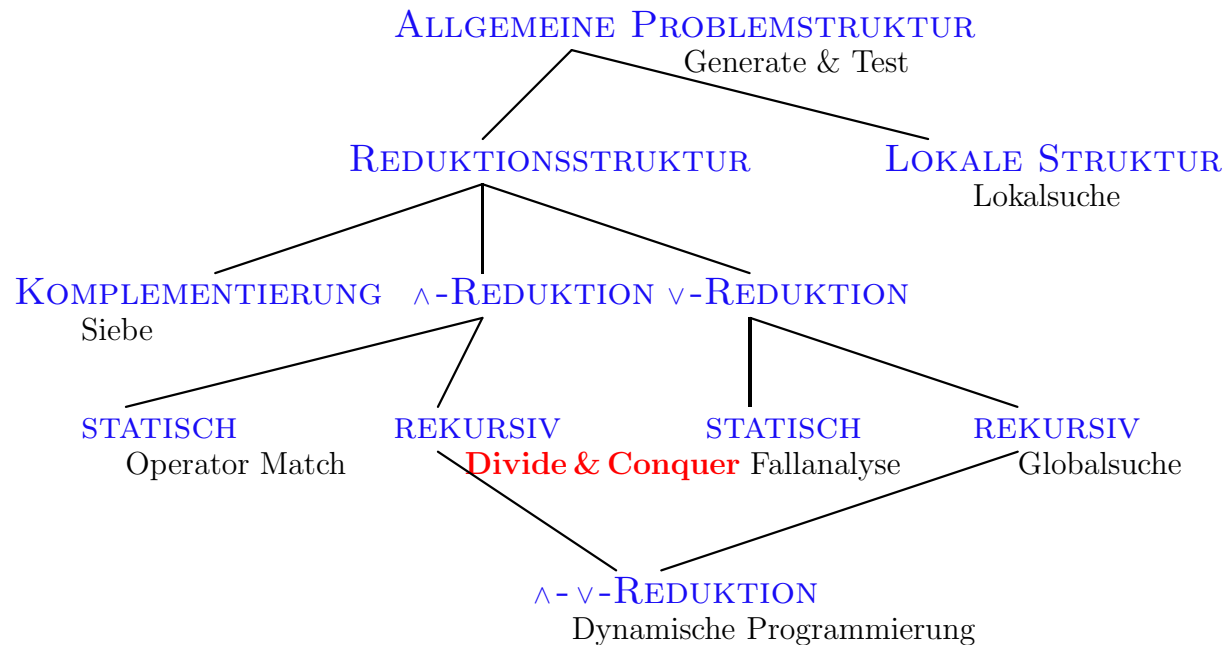


DIVIDE & CONQUER ALGORITHMEN



- **Effiziente Verarbeitung strukturierter Daten**
 - Sehr gebräuchliche und einfache Programmiertechnik

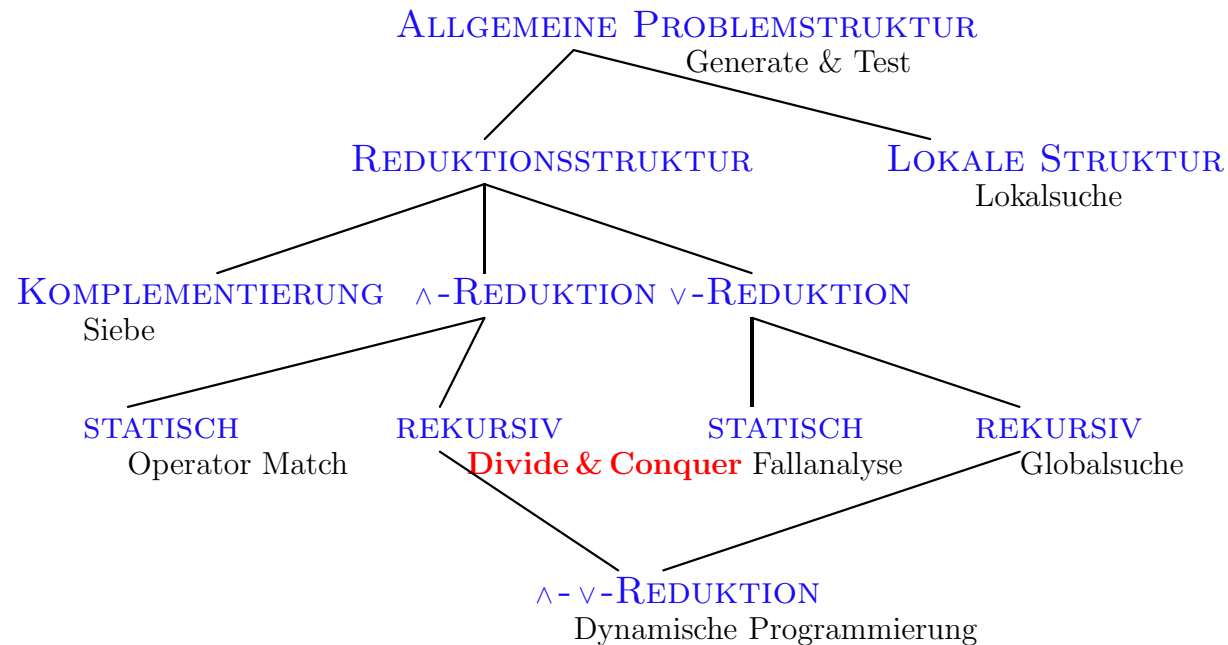
DIVIDE & CONQUER ALGORITHMEN



● Effiziente Verarbeitung strukturierter Daten

- Sehr gebräuchliche und einfache Programmiertechnik
- **Aufteilen**: Zerlege Problem in kleinere Teilprobleme

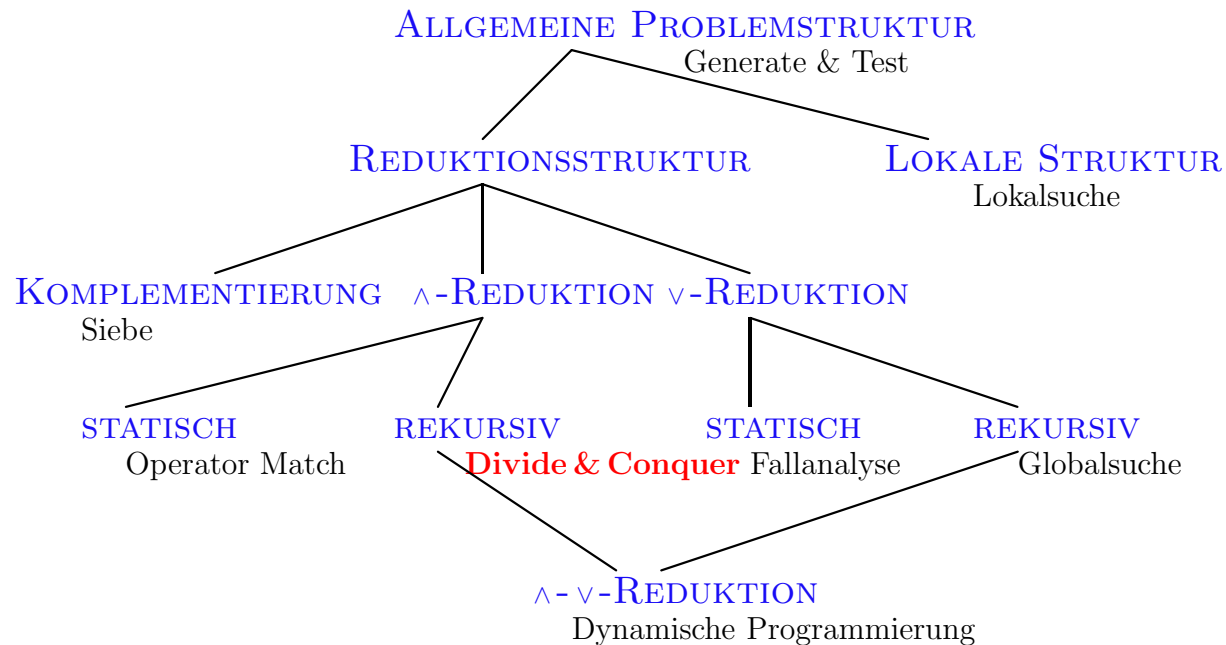
DIVIDE & CONQUER ALGORITHMEN



● Effiziente Verarbeitung strukturierter Daten

- Sehr gebräuchliche und einfache Programmiertechnik
- **Aufteilen**: Zerlege Problem in kleinere Teilprobleme
- **Erobern**: Löse Teilprobleme separat und setze Lösungen zusammen

DIVIDE & CONQUER ALGORITHMEN



● Effiziente Verarbeitung strukturierter Daten

- Sehr gebräuchliche und einfache Programmiertechnik
- **Aufteilen**: Zerlege Problem in kleinere Teilprobleme
- **Erobern**: Löse Teilprobleme separat und setze Lösungen zusammen

● $\wedge$ -Reduktion des Problems

- Lösung benötigt alle Teillösungen gleichzeitig
- Teillösungen bauen aufeinander auf

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

- **Maximum einer nichtleeren Liste von Zahlen**

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE  $L \neq []$   
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if  $|L|=1$  then hd(L)  
    else let a,L'=L  
        in let m'=maxL(L')  
        in if  $a < m'$  then m' else a
```

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

- **Maximum einer nichtleeren Liste von Zahlen**

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE L $\neq$ []  
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if |L|=1 then hd(L)  
    else let a,L'=L  
        in let m'=maxL(L')  
        in if a<m' then m' else a
```

Einfache (primitive) Eingaben ($|L|=1$) erhalten direkte Lösung $\text{hd}(L)$

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

● Maximum einer nichtleeren Liste von Zahlen

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE  $L \neq []$   
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if  $|L|=1$  then hd(L)  
    else let a,L'=L  
        in let  $m'=\text{maxL}(L')$   
        in if  $a < m'$  then  $m'$  else a
```

Einfache (primitive) Eingaben ($|L|=1$) erhalten direkte Lösung $\text{hd}(L)$

Andernfalls Dekomposition der Eingabe mit Funktion $\text{HdTl}: L \mapsto a, L'$

Einfache Teillösung $a=\text{id}(a)$ für a — rekursive Lösung $m'=\text{maxL}(L')$ für L'

Komposition der Teillösungen a und m' mit der Funktion $\text{max}:\mathbb{Z} \rightarrow \mathbb{Z}$

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

● Maximum einer nichtleeren Liste von Zahlen

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE  $L \neq []$   
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if  $|L|=1$  then hd(L)  
      else let a,L'=L  
            in let  $m'=\text{maxL}(L')$   
              in if  $a < m'$  then  $m'$  else a
```

Einfache (primitive) Eingaben ($|L|=1$) erhalten direkte Lösung $\text{hd}(L)$

Andernfalls Dekomposition der Eingabe mit Funktion $\text{HdTl}: L \mapsto a, L'$

Einfache Teillösung $a=\text{id}(a)$ für a — rekursive Lösung $m'=\text{maxL}(L')$ für L'

Komposition der Teillösungen a und m' mit der Funktion $\text{max}:\mathbb{Z} \rightarrow \mathbb{Z}$

● Vereinheitlichung durch separierte Beschreibung

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE  $L \neq []$   
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if  $|L|=1$  then hd(L)  
      else ( $\text{max} \circ (\text{id} \times \text{maxL}) \circ \text{HdTl}$ )(L)
```

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

● Maximum einer nichtleeren Liste von Zahlen

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE  $L \neq []$   
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if  $|L|=1$  then hd(L)  
      else let a,L'=L  
            in let  $m'=\text{maxL}(L')$   
              in if  $a < m'$  then  $m'$  else a
```

Einfache (primitive) Eingaben ($|L|=1$) erhalten direkte Lösung $\text{hd}(L)$

Andernfalls Dekomposition der Eingabe mit Funktion $\text{HdTl}: L \mapsto a, L'$

Einfache Teillösung $a=\text{id}(a)$ für a — rekursive Lösung $m'=\text{maxL}(L')$ für L'

Komposition der Teillösungen a und m' mit der Funktion $\text{max}:\mathbb{Z} \rightarrow \mathbb{Z}$

● Vereinheitlichung durch separierte Beschreibung

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE  $L \neq []$   
  RETURNS m SUCH THAT  $m \in L \wedge \forall x \in L. x \leq m$   
 $\equiv$  if  $|L|=1$  then hd(L)  
      else ( $\text{max} \circ (\text{id} \times \text{maxL}) \circ \text{HdTl}$ )(L)
```

Algorithmus zur Verarbeitung der Liste folgt festem Schema

ALLGEMEINES DIVIDE & CONQUER SCHEMA

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

ALLGEMEINES DIVIDE & CONQUER SCHEMA

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

- 5 zentrale Komponenten der Algorithmentheorie

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

- **5 zentrale Komponenten der Algorithmentheorie**
 - $Decompose: D \rightarrow D' \times D$ Aufspalten der Eingabe in Teilprobleme

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

• 5 zentrale Komponenten der Algorithmentheorie

- $Decompose: D \rightarrow D' \times D$ Aufspalten der Eingabe in Teilprobleme
- Rekursiver Aufruf von f zusammen mit Hilfsfunktion $g: D \rightarrow R'$
 - Funktionsprodukt $g \times f(x, y) = (g(x), f(y))$

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

• 5 zentrale Komponenten der Algorithmentheorie

- *Decompose*: $D \rightarrow D' \times D$ Aufspalten der Eingabe in Teilprobleme
- Rekursiver Aufruf von f zusammen mit Hilfsfunktion $g: D \rightarrow R'$
 - Funktionsprodukt $g \times f(x, y) = (g(x), f(y))$
- *Compose*: $R' \times R \rightarrow R$ Zusammensetzen der Teillösungen

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

• 5 zentrale Komponenten der Algorithmentheorie

- *Decompose*: $D \rightarrow D' \times D$ Aufspalten der Eingabe in Teilprobleme
- Rekursiver Aufruf von f zusammen mit Hilfsfunktion $g: D \rightarrow R'$
 - Funktionsprodukt $g \times f(x, y) = (g(x), f(y))$
- *Compose*: $R' \times R \rightarrow R$ Zusammensetzen der Teillösungen
- *Directly-solve*: $D \rightarrow R$: Direkte Lösung für einfache Teilprobleme

ALLGEMEINES DIVIDE & CONQUER SCHEMA

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

• 5 zentrale Komponenten der Algorithmentheorie

- $Decompose: D \rightarrow D' \times D$ Aufspalten der Eingabe in Teilprobleme
- Rekursiver Aufruf von f zusammen mit Hilfsfunktion $g: D \rightarrow R'$
 - Funktionsprodukt $g \times f(x, y) = (g(x), f(y))$
- $Compose: R' \times R \rightarrow R$ Zusammensetzen der Teillösungen
- $Directly-solve: D \rightarrow R:$ Direkte Lösung für einfache Teilprobleme
- $primitive: D \rightarrow \mathbb{B}:$ Test, ob Eingabe “einfach” ist

Grundstruktur aller Divide & Conquer Algorithmen

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$
 else $(Compose \circ g \times f \circ Decompose)(x)$

• 5 zentrale Komponenten der Algorithmentheorie

- *Decompose*: $D \rightarrow D' \times D$ Aufspalten der Eingabe in Teilprobleme
- Rekursiver Aufruf von f zusammen mit Hilfsfunktion $g: D \rightarrow R'$
 - Funktionsprodukt $g \times f(x, y) = (g(x), f(y))$
- *Compose*: $R' \times R \rightarrow R$ Zusammensetzen der Teillösungen
- *Directly-solve*: $D \rightarrow R$: Direkte Lösung für einfache Teilprobleme
- *primitive*: $D \rightarrow \mathbb{B}$: Test, ob Eingabe “einfach” ist

Korrektheit folgt aus wenigen Voraussetzungen

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

2. Ausgabebedingung O rekursiv zerlegbar in O_D , O' und O_C

$O_D[x, y', y] \wedge O'[y', z'] \wedge O[y, z] \wedge O_C[z', z, t] \Rightarrow O[x, t]$

(Strong Problem Reduction Principle)

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

2. Ausgabebedingung O rekursiv zerlegbar in O_D , O' und O_C

$O_D[x,y',y] \wedge O'[y',z'] \wedge O[y,z] \wedge O_C[z',z,t] \Rightarrow O[x,t]$

(Strong Problem Reduction Principle)

3. Dekomposition erfüllt O_D und ‘verkleinert’ Problem

FUNCTION $f_d(x:D):D' \times D$ WHERE $I[x] \wedge \neg primitive[x]$

RETURNS y',y SUCH THAT $I'[y'] \wedge I[y] \wedge x \succ y \wedge O_D[x,y',y]$

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

2. Ausgabebedingung O rekursiv zerlegbar in O_D , O' und O_C

$O_D[x, y', y] \wedge O'[y', z'] \wedge O[y, z] \wedge O_C[z', z, t] \Rightarrow O[x, t]$

(Strong Problem Reduction Principle)

3. Dekomposition erfüllt O_D und ‘verkleinert’ Problem

FUNCTION $f_d(x:D):D' \times D$ WHERE $I[x] \wedge \neg primitive[x]$
RETURNS y', y SUCH THAT $I'[y'] \wedge I[y] \wedge x \succ y \wedge O_D[x, y', y]$

4. Hilfsfunktion g erfüllt O'

FUNCTION $g(y':D'):R'$ WHERE $I'[y']$ RETURNS z' SUCH THAT $O'[y', z']$

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

2. Ausgabebedingung O rekursiv zerlegbar in O_D , O' und O_C

$O_D[x,y',y] \wedge O'[y',z'] \wedge O[y,z] \wedge O_C[z',z,t] \Rightarrow O[x,t]$

(Strong Problem Reduction Principle)

3. Dekomposition erfüllt O_D und ‘verkleinert’ Problem

FUNCTION $f_d(x:D):D' \times D$ WHERE $I[x] \wedge \neg primitive[x]$
RETURNS y',y SUCH THAT $I'[y'] \wedge I[y] \wedge x \succ y \wedge O_D[x,y',y]$

4. Hilfsfunktion g erfüllt O'

FUNCTION $g(y':D'):R'$ WHERE $I'[y']$ RETURNS z' SUCH THAT $O'[y',z']$

5. Komposition erfüllt O_C

FUNCTION $f_c(z',z:R' \times R):R$ WHERE true RETURNS y SUCH THAT $O_C[z',z,t]$

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

2. Ausgabebedingung O rekursiv zerlegbar in O_D , O' und O_C

$O_D[x,y',y] \wedge O'[y',z'] \wedge O[y,z] \wedge O_C[z',z,t] \Rightarrow O[x,t]$

(Strong Problem Reduction Principle)

3. Dekomposition erfüllt O_D und ‘verkleinert’ Problem

FUNCTION $f_d(x:D):D' \times D$ WHERE $I[x] \wedge \neg primitive[x]$
RETURNS y',y SUCH THAT $I'[y'] \wedge I[y] \wedge x \succ y \wedge O_D[x,y',y]$

4. Hilfsfunktion g erfüllt O'

FUNCTION $g(y':D'):R'$ WHERE $I'[y']$ RETURNS z' SUCH THAT $O'[y',z']$

5. Komposition erfüllt O_C

FUNCTION $f_c(z',z:R' \times R):R$ WHERE true RETURNS y SUCH THAT $O_C[z',z,t]$

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

Sechste Komponente in Algorithmentheorie, nötig für Terminierungsbeweis

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

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

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

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

- **Primitive Eingaben:** $I[x] \wedge primitive[x]$

- $f(x) = Directly-solve[x]$ Korrektheit folgt direkt aus **Axiom 1**

FUNCTION $f_p(x:D):R$ WHERE $I[x] \wedge primitive[x]$ RETURNS y SUCH THAT $O[x,y]$

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

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

- **Primitive Eingaben:** $I[x] \wedge primitive[x]$

- $f(x) = Directly-solve[x]$ Korrektheit folgt direkt aus **Axiom 1**

- **Nichtprimitive Eingaben:** $I[x] \wedge \neg primitive[x]$

- $f(x) = (Compose \circ g \times f \circ Decompose)(x)$

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

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

- **Primitive Eingaben:** $I[x] \wedge \text{primitive}[x]$

- $f(x) = \text{Directly-solve}[x]$ Korrektheit folgt direkt aus Axiom 1

- **Nichtprimitive Eingaben:** $I[x] \wedge \neg \text{primitive}[x]$

- $f(x) = (\text{Compose} \circ g \times f \circ \text{Decompose})(x)$

- *Decompose* $[x]$ liefert y', y mit $O_D[x, y', y]$ und $x \succ y$ Axiom 3

FUNCTION $f_d(x:D):D' \times D$ WHERE $I[x] \wedge \neg \text{primitive}[x]$
RETURNS y', y SUCH THAT $I'[y'] \wedge I[y] \wedge x \succ y \wedge O_D[x, y', y]$

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

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

- **Primitive Eingaben:** $I[x] \wedge \text{primitive}[x]$

- $f(x) = \text{Directly-solve}[x]$ Korrektheit folgt direkt aus Axiom 1

- **Nichtprimitive Eingaben:** $I[x] \wedge \neg \text{primitive}[x]$

- $f(x) = (\text{Compose} \circ g \times f \circ \text{Decompose})(x)$

- *Decompose* $[x]$ liefert y', y mit $O_D[x, y', y]$ und $x \succ y$ Axiom 3

- $g(y')$ liefert z' mit $O'[y', z']$ Axiom 4

FUNCTION $g(y':D'):R'$ WHERE $I'[y']$ RETURNS z' SUCH THAT $O'[y', z']$

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

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

- **Primitive Eingaben:** $I[x] \wedge \text{primitive}[x]$

- $f(x) = \text{Directly-solve}[x]$ Korrektheit folgt direkt aus Axiom 1

- **Nichtprimitive Eingaben:** $I[x] \wedge \neg \text{primitive}[x]$

- $f(x) = (\text{Compose} \circ g \times f \circ \text{Decompose})(x)$

- $\text{Decompose}[x]$ liefert y', y mit $O_D[x, y', y]$ und $x \succ y$ Axiom 3

- $g(y')$ liefert z' mit $O'[y', z']$ Axiom 4

- $f(y)$ liefert z mit $O[y, z]$ Induktionsannahme

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

● Partielle Korrektheit: strukturelle Induktion über $(D, \succ)$

- Primitive Eingaben: $I[x] \wedge primitive[x]$
 - $f(x) = Directly-solve[x]$ Korrektheit folgt direkt aus Axiom 1
- Nichtprimitive Eingaben: $I[x] \wedge \neg primitive[x]$
 - $f(x) = (Compose \circ g \times f \circ Decompose)(x)$
 - $Decompose[x]$ liefert y', y mit $O_D[x, y', y]$ und $x \succ y$ Axiom 3
 - $g(y')$ liefert z' mit $O'[y', z']$ Axiom 4
 - $f(y)$ liefert z mit $O[y, z]$ Induktionsannahme
 - $Decompose[z', z]$ liefert t mit $O_C[z', z, t]$ Axiom 5

FUNCTION $f_c(z', z:R' \times R):R$ WHERE true RETURNS y SUCH THAT $O_C[z', z, t]$

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

● Partielle Korrektheit: strukturelle Induktion über $(D, \succ)$

- **Primitive Eingaben:** $I[x] \wedge \text{primitive}[x]$
 - $f(x) = \text{Directly-solve}[x]$ Korrektheit folgt direkt aus Axiom 1
- **Nichtprimitive Eingaben:** $I[x] \wedge \neg \text{primitive}[x]$
 - $f(x) = (\text{Compose} \circ g \times f \circ \text{Decompose})(x)$
 - $\text{Decompose}[x]$ liefert y', y mit $O_D[x, y', y]$ und $x \succ y$ Axiom 3
 - $g(y')$ liefert z' mit $O'[y', z']$ Axiom 4
 - $f(y)$ liefert z mit $O[y, z]$ Induktionsannahme
 - $\text{Decompose}[z', z]$ liefert t mit $O_C[z', z, t]$ Axiom 5
 - t ist das Ergebnis ($f(x) = t$) und es gilt $O[x, t]$ Axiom 2

$$O_D[x, y', y] \wedge O'[y', z'] \wedge O[y, z] \wedge O_C[z', z, t] \Rightarrow O[x, t]$$

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIS

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

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

- **Primitive Eingaben:** $I[x] \wedge \text{primitive}[x]$

- $f(x) = \text{Directly-solve}[x]$ Korrektheit folgt direkt aus Axiom 1

- **Nichtprimitive Eingaben:** $I[x] \wedge \neg \text{primitive}[x]$

- $f(x) = (\text{Compose} \circ g \times f \circ \text{Decompose})(x)$

- $\text{Decompose}[x]$ liefert y', y mit $O_D[x, y', y]$ und $x \succ y$ Axiom 3

- $g(y')$ liefert z' mit $O'[y', z']$ Axiom 4

- $f(y)$ liefert z mit $O[y, z]$ Induktionsannahme

- $\text{Decompose}[z', z]$ liefert t mit $O_C[z', z, t]$ Axiom 5

- t ist das Ergebnis ($f(x) = t$) und es gilt $O[x, t]$ Axiom 2

- **Terminierung:** Wohlfundiertheit von $\succ$ Axiom 6

SYNTHESESTRATEGIE FÜR DIVIDE & CONQUER ALGORITHMEN

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

SYNTHESESTRATEGIE FÜR DIVIDE & CONQUER ALGORITHMEN

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

1. Wähle $\succ$ und *Decompose* aus Wissensbank $\mapsto O_D, D', \text{Axiom 6}$

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

1. Wähle $\succ$ und *Decompose* aus Wissensbank $\mapsto O_D, D', \text{Axiom 6}$

2. Konstruiere Hilfsfunktion g : $\mapsto O', I', \text{Axiom 4}$

– Heuristik: $g:=f$, falls $D'=D$, sonst $g:=id$

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

1. Wähle $\succ$ und *Decompose* aus Wissensbank $\mapsto O_D, D', \text{Axiom 6}$
2. Konstruiere Hilfsfunktion g : $\mapsto O', I', \text{Axiom 4}$
 - Heuristik: $g:=f$, falls $D'=D$, sonst $g:=id$
3. Verifiziere *Decompose*, generiere Vorbedingung $\mapsto \text{primitive}, \text{Axiom 3}$
 - Heuristik: abgeleitete zusätzliche Vorbedingung für O_D ist $\neg \text{primitive}[x]$

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

1. Wähle $\succ$ und *Decompose* aus Wissensbank $\mapsto O_D, D', \text{Axiom 6}$
2. Konstruiere Hilfsfunktion g : $\mapsto O', I', \text{Axiom 4}$
 - Heuristik: $g:=f$, falls $D'=D$, sonst $g:=id$
3. Verifiziere *Decompose*, generiere Vorbedingung $\mapsto \text{primitive}, \text{Axiom 3}$
 - Heuristik: abgeleitete zusätzliche Vorbedingung für O_D ist $\neg \text{primitive}[x]$
4. Konstruiere *Compose* $\mapsto O_C, \text{Axiome 5 \& 2}$
 - Heuristik: Erzeuge O_C mit Axiom 2;
Synthetisiere *Compose* gemäß Axiom 5

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

1. Wähle $\succ$ und *Decompose* aus Wissensbank $\mapsto O_D, D', \text{Axiom 6}$
2. Konstruiere Hilfsfunktion g : $\mapsto O', I', \text{Axiom 4}$
 - Heuristik: $g:=f$, falls $D'=D$, sonst $g:=id$
3. Verifiziere *Decompose*, generiere Vorbedingung $\mapsto \text{primitive}, \text{Axiom 3}$
 - Heuristik: abgeleitete zusätzliche Vorbedingung für O_D ist $\neg \text{primitive}[x]$
4. Konstruiere *Compose* $\mapsto O_C, \text{Axiome 5 \& 2}$
 - Heuristik: Erzeuge O_C mit Axiom 2;
Synthetisiere *Compose* gemäß Axiom 5
5. Konstruiere *Directly-solve* $\mapsto \text{Axiome 1}$
 - Heuristik: Suche nach vorgefertigten Lösungen, sonst neue Synthese
 - Falls dies nicht möglich ist, konstruiere eingeschränkte Vorbedingung $\hat{I}$

Zerlege Problem in Spezifikationen für Teilprobleme

Start: FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

1. Wähle $\succ$ und *Decompose* aus Wissensbank $\mapsto O_D, D', \text{Axiom 6}$
2. Konstruiere Hilfsfunktion g : $\mapsto O', I', \text{Axiom 4}$
 - Heuristik: $g:=f$, falls $D'=D$, sonst $g:=id$
3. Verifiziere *Decompose*, generiere Vorbedingung $\mapsto \text{primitive}, \text{Axiom 3}$
 - Heuristik: abgeleitete zusätzliche Vorbedingung für O_D ist $\neg \text{primitive}[x]$
4. Konstruiere *Compose* $\mapsto O_C, \text{Axiome 5 \& 2}$
 - Heuristik: Erzeuge O_C mit Axiom 2;
Synthetisiere *Compose* gemäß Axiom 5
5. Konstruiere *Directly-solve* $\mapsto \text{Axiome 1}$
 - Heuristik: Suche nach vorgefertigten Lösungen, sonst neue Synthese
 - Falls dies nicht möglich ist, konstruiere eingeschränkte Vorbedingung $\hat{I}$
6. Instantiiere das Divide & Conquer Schema

ALTERNATIVE REIHENFOLGEN DER SYNTHESESTRATEGIE

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if $primitive[x]$ then $Directly-solve[x]$ else $(Compose \circ g \times f \circ Decompose)(x)$

ALTERNATIVE REIHENFOLGEN DER SYNTHESESTRATEGIE

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

$\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

● Grundstrategie

- Wähle $\succ$ und *Decompose* aus Wissensbank
- Konstruiere g heuristisch
- Bestimme *primitive* durch Verifikation von *Decompose*
- Konstruiere Spezifikation und Lösung für *Compose*
- Konstruiere *Directly-solve*

ALTERNATIVE REIHENFOLGEN DER SYNTHESESTRATEGIE

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

● Grundstrategie

- Wähle $\succ$ und *Decompose* aus Wissensbank
- Konstruiere g heuristisch
- Bestimme *primitive* durch Verifikation von *Decompose*
- Konstruiere Spezifikation und Lösung für *Compose*
- Konstruiere *Directly-solve*

● Variante

- Wähle $\succ$ und *Decompose* aus Wissensbank
- Konstruiere *Compose* heuristisch
- Konstruiere Spezifikation und Lösung für g
- Bestimme *primitive* und konstruiere *Directly-solve*

ALTERNATIVE REIHENFOLGEN DER SYNTHESESTRATEGIE

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
 $\equiv$ if *primitive* $[x]$ then *Directly-solve* $[x]$ else (*Compose* $\circ g \times f \circ$ *Decompose*) (x)

● Grundstrategie

- Wähle $\succ$ und *Decompose* aus Wissensbank
- Konstruiere g heuristisch
- Bestimme *primitive* durch Verifikation von *Decompose*
- Konstruiere Spezifikation und Lösung für *Compose*
- Konstruiere *Directly-solve*

● Variante

- Wähle $\succ$ und *Decompose* aus Wissensbank
- Konstruiere *Compose* heuristisch
- Konstruiere Spezifikation und Lösung für g
- Bestimme *primitive* und konstruiere *Directly-solve*

● Umgekehrte Strategie

- Wähle *Compose* aus Wissensbank
- Konstruiere g heuristisch
- Konstruiere Spezifikation und Lösung für *Decompose* und bestimme $\succ$
- Bestimme *primitive* und konstruiere *Directly-solve*

- **Standard-Zerlegungen für Typen (D)** $\mapsto \text{Decompose}, O_D, D'$
 - Endliche Listen / Folgen ($\text{Seq}(\alpha)$): **HdTl**, **ListSplit**
 - $\text{ListSplit}(L) \equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$
mit $O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2$ $D' \equiv \text{Seq}(\alpha)$
 - Endliche Mengen ($\text{Set}(\alpha)$): **ArbRest**
 - Produkträume (+ Mengen, Folgen, ...): Zerlegung in **Einzelkomponenten**
 - Natürlichen Zahlen ($\mathbb{N}$): **Vorgängerfunktion**

● **Standard-Zerlegungen für Typen (D)** $\mapsto \text{Decompose}, O_D, D'$

- Endliche Listen / Folgen ($\text{Seq}(\alpha)$): **HdTl**, **ListSplit**
 - $\text{ListSplit}(L) \equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$
mit $O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2$ $D' \equiv \text{Seq}(\alpha)$
- Endliche Mengen ($\text{Set}(\alpha)$): **ArbRest**
- Produkträume (+ Mengen, Folgen, ...): Zerlegung in **Einzelkomponenten**
- Natürlichen Zahlen ($\mathbb{N}$): **Vorgängerfunktion**

● **Standard-Wohlordnungen auf Typen (D)** $\mapsto \succ$

- Folgen und Mengen: **Längen/Größenordnung** $L \succ L' \equiv |L| > |L'|$
- Produkträume (+ Mengen, Folgen, ...): **Lexikographische Ordnung**
 - $(a_1, b_1) \succ (a_2, b_2) \equiv a_1 > a_2 \vee (a_1 = a_2 \wedge b_1 > b_2)$
- Zahlen ($\mathbb{N}/\mathbb{Z}$): **Zahlenordnung** bzw. **Absolutordnung**

PROGRAMMIERWISSEN FÜR DIVIDE & CONQUER

- **Standard-Zerlegungen für Typen (D)** $\mapsto \text{Decompose}, O_D, D'$
 - Endliche Listen / Folgen ($\text{Seq}(\alpha)$): **HdTl**, **ListSplit**
 - $\text{ListSplit}(L) \equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$
mit $O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2$ $D' \equiv \text{Seq}(\alpha)$
 - Endliche Mengen ($\text{Set}(\alpha)$): **ArbRest**
 - Produkträume (+ Mengen, Folgen, ...): Zerlegung in **Einzelkomponenten**
 - Natürlichen Zahlen ($\mathbb{N}$): **Vorgängerfunktion**
- **Standard-Wohlordnungen auf Typen (D)** $\mapsto \succ$
 - Folgen und Mengen: **Längen/Größenordnung** $L \succ L' \equiv |L| > |L'|$
 - Produkträume (+ Mengen, Folgen, ...): **Lexikographische Ordnung**
 - $(a_1, b_1) \succ (a_2, b_2) \equiv a_1 > a_2 \vee (a_1 = a_2 \wedge b_1 > b_2)$
 - Zahlen ($\mathbb{N}/\mathbb{Z}$): **Zahlenordnung** bzw. **Absolutordnung**
- **Standard-Kompositionen für Typen (R)** $\mapsto \text{Compose}, O_C, R'$
 - Endliche Folgen: **cons** ($a.l$), **append** ($L_1 \circ L_2$)
 - Endliche Mengen: **insert** ($S+a$), **union** ($S_1 \cup S_2$)
 - Produkträume (+ Mengen, Folgen, ...): **Komponentenweise Komposition**
 - Natürlichen Zahlen ($\mathbb{N}$): **Nachfolgerfunktion**

● Heuristische Fixierung von g

- Falls $D' = D$, wähle $g := f$ $R' := R$, $O' := O$ und $I' := I$
 - Falls $D' \neq D$, wähle $g := id$ $R' := D'$, $O[y'z'] := z' = y'$ und $I'[y'] := \text{true}$
- Analoge Entscheidungen, falls $R' = R$ durch Wahl von *Compose*

- **Heuristische Fixierung von g**

- Falls $D' = D$, wähle $g := f$ $R' := R$, $O' := O$ und $I' := I$
 - Falls $D' \neq D$, wähle $g := id$ $R' := D'$, $O[y'z'] := z' = y'$ und $I'[y'] := \text{true}$
- Analoge Entscheidungen, falls $R' = R$ durch Wahl von *Compose*

- **Inferenzmechanismus: Abgeleitete Vorbedingungen**

- Bestimme Voraussetzungen für Gültigkeit einer Formel durch zielgerichteten Einsatz von Lemmas entsprechend vorkommender Begriffe
- Voraussetzungen sind verbleibende Vorbedingungen beim Beweisversuch

● Heuristische Fixierung von g

- Falls $D'=D$, wähle $g:=f$ $R':=R$, $O':=O$ und $I':=I$
 - Falls $D'\neq D$, wähle $g:=id$ $R':=D'$, $O[y'z']':=z' = y'$ und $I'[y']:=\text{true}$
- Analoge Entscheidungen, falls $R'=R$ durch Wahl von *Compose*

● Inferenzmechanismus: Abgeleitete Vorbedingungen

- Bestimme Voraussetzungen für Gültigkeit einer Formel durch zielgerichteten Einsatz von Lemmas entsprechend vorkommender Begriffe
- Voraussetzungen sind verbleibende Vorbedingungen beim Beweisversuch

● Inferenzmechanismus: Fallanalyse

- Erzeugung von Alternativen über existierende Prädikate
- Partielle Auswertung der Einzelfälle

Liefert Programmstücke mit Fallunterscheidungen

● Heuristische Fixierung von g

- Falls $D'=D$, wähle $g:=f$ $R':=R$, $O':=O$ und $I':=I$
 - Falls $D'\neq D$, wähle $g:=id$ $R':=D'$, $O[y'z']':=z' = y'$ und $I'[y']:=\text{true}$
- Analoge Entscheidungen, falls $R'=R$ durch Wahl von *Compose*

● Inferenzmechanismus: Abgeleitete Vorbedingungen

- Bestimme Voraussetzungen für Gültigkeit einer Formel durch zielgerichteten Einsatz von Lemmas entsprechend vorkommender Begriffe
- Voraussetzungen sind verbleibende Vorbedingungen beim Beweisversuch

● Inferenzmechanismus: Fallanalyse

- Erzeugung von Alternativen über existierende Prädikate
- Partielle Auswertung der Einzelfälle

Liefert Programmstücke mit Fallunterscheidungen

● Inferenzmechanismus: Operator match

- Synthese von Programmstücken durch Anpassung an bekannte Lösungen

DIVIDE & CONQUER SYNTHESE VON SORTIERALGORITHMEN

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) WHERE true  
  RETURNS S SUCH THAT rearranges(L,S)  $\wedge$  ordered(S)
```

DIVIDE & CONQUER SYNTHESE VON SORTIERALGORITHMEN

FUNCTION **sort**($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE true
RETURNS S SUCH THAT rearranges(L,S) $\wedge$ ordered(S)

1. Wähle $L \succ L' \equiv |L| > |L'|$

Decompose $\equiv$ ListSplit $(O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2, \quad D' \equiv \text{Seq}(\alpha))$

DIVIDE & CONQUER SYNTHESE VON SORTIERALGORITHMEN

FUNCTION **sort**($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE true
RETURNS S SUCH THAT rearranges(L,S) $\wedge$ ordered(S)

1. Wähle $L \succ L' \equiv |L| > |L'|$

$Decompose \equiv \text{ListSplit} \quad (O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2, \quad D' \equiv \text{Seq}(\alpha))$

2. Konstruiere Hilfsfunktion $g \equiv \text{sort} \quad \mapsto O' \equiv O, \quad I'[L'] \equiv \text{true}$

DIVIDE & CONQUER SYNTHESE VON SORTIERALGORITHMEN

FUNCTION **sort**($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE true
RETURNS S SUCH THAT rearranges(L,S) $\wedge$ ordered(S)

1. Wähle $L \succ L' \equiv |L| > |L'|$

$Decompose \equiv \text{ListSplit} \quad (O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2, \quad D' \equiv \text{Seq}(\alpha))$

2. Konstruiere Hilfsfunktion $g \equiv \text{sort} \quad \mapsto O' \equiv O, \quad I'[L'] \equiv \text{true}$

3. Verifiziere $Decompose$

FUNCTION $f_d(L:\text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ WHERE $\neg \text{primitive}[L]$

RETURNS L_1, L_2 SUCH THAT $L \succ L_1 \wedge L \succ L_2 \wedge L_1 \circ L_2 = L$

Vorbedingung für Korrektheit $\text{primitive}[L] \equiv L = []$

DIVIDE & CONQUER SYNTHESE VON SORTIERALGORITHMEN

FUNCTION **sort**($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE true
RETURNS S SUCH THAT rearranges(L,S) $\wedge$ ordered(S)

1. Wähle $L \succ L' \equiv |L| > |L'|$

$Decompose \equiv \text{ListSplit} \quad (O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2, \quad D' \equiv \text{Seq}(\alpha))$

2. Konstruiere Hilfsfunktion $g \equiv \text{sort} \quad \mapsto O' \equiv O, \quad I'[L] \equiv \text{true}$

3. Verifiziere $Decompose$

FUNCTION $f_d(L:\text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ WHERE $\neg \text{primitive}[L]$
RETURNS L_1, L_2 SUCH THAT $L \succ L_1 \wedge L \succ L_2 \wedge L_1 \circ L_2 = L$

Vorbedingung für Korrektheit $\text{primitive}[L] \equiv L = []$

4. Konstruiere $Compose$

$L \succ L_1 \wedge L \succ L_2 \wedge L_1 \circ L_2 = L \wedge \text{SORT}(L_1, S_1) \wedge \text{SORT}(L_2, S_2) \wedge O_C[S_1, S_2, S] \Rightarrow \text{SORT}(L, S)$

liefert als Spezifikation für $Compose \quad \mapsto \text{Synthese folgt später}$

FUNCTION $f_c(S_1, S_2:\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z})$ WHERE ordered(S_1) $\wedge$ ordered(S_2)
RETURNS S SUCH THAT ordered(S) $\wedge$ rearranges($S, S_1 \circ S_2$)

DIVIDE & CONQUER SYNTHESE VON SORTIERALGORITHMEN

FUNCTION **sort**($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE true
RETURNS S SUCH THAT rearranges(L,S) $\wedge$ ordered(S)

1. Wähle $L \succ L' \equiv |L| > |L'|$

$Decompose \equiv \text{ListSplit} \quad (O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2, \quad D' \equiv \text{Seq}(\alpha))$

2. Konstruiere Hilfsfunktion $g \equiv \text{sort} \quad \mapsto O' \equiv O, \quad I'[L] \equiv \text{true}$

3. Verifiziere $Decompose$

FUNCTION $f_d(L:\text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ WHERE $\neg \text{primitive}[L]$
RETURNS L_1, L_2 SUCH THAT $L \succ L_1 \wedge L \succ L_2 \wedge L_1 \circ L_2 = L$

Vorbedingung für Korrektheit $\text{primitive}[L] \equiv L = []$

4. Konstruiere $Compose$

$L \succ L_1 \wedge L \succ L_2 \wedge L_1 \circ L_2 = L \wedge \text{SORT}(L_1, S_1) \wedge \text{SORT}(L_2, S_2) \wedge O_C[S_1, S_2, S] \Rightarrow \text{SORT}(L, S)$

liefert als Spezifikation für $Compose \quad \mapsto \text{Synthese folgt später}$

FUNCTION $f_c(S_1, S_2:\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z})$ WHERE ordered(S_1) $\wedge$ ordered(S_2)
RETURNS S SUCH THAT ordered(S) $\wedge$ rearranges($S, S_1 \circ S_2$)

5. Konstruiere $Directly-solve$

FUNCTION $f_p(L:\text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z})$ WHERE $L = []$
RETURNS S SUCH THAT rearranges(L,S) $\wedge$ ordered(S)

Liefert $Directly-solve[L] \equiv []$

Ermittelte Komponenten

$\text{Decompose}[L] \equiv \text{ListSplit}[L]$
 $\equiv ([L_i \mid i \in [1..|L| \div 2]], [L_i \mid i \in [1+|L| \div 2..|L|]])$

$g \equiv \text{sort}$

$\text{Compose}[S_1, S_2] \equiv \text{merge}(S_1, S_2) \quad \mapsto \text{nächste Folie}$

$\text{Directly-solve}[L] \equiv []$

$\text{primitive}[L] \equiv L = []$

Ermittelte Komponenten

$Decompose[L] \equiv ListSplit[L]$
 $\equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$
 $g \equiv sort$
 $Compose[S_1, S_2] \equiv merge(S_1, S_2) \quad \mapsto \text{nächste Folie}$
 $Directly-solve[L] \equiv []$
 $primitive[L] \equiv L = []$

6. Instantiiere das Divide & Conquer Schema

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) WHERE true  
  RETURNS S SUCH THAT rearranges(L,S)  $\wedge$  ordered(S)  
 $\equiv$  if L=[] then []  
    else let L1.L2 = ([Li | i  $\in$  [1..|L| $\div$ 2]],  
                        [Li | i  $\in$  [1+|L| $\div$ 2..|L|]])  
    in merge(sort(L1),sort(L2))
```

Ermittelte Komponenten

$Decompose[L] \equiv ListSplit[L]$
 $\equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$
 $g \equiv sort$
 $Compose[S_1, S_2] \equiv merge(S_1, S_2) \quad \mapsto \text{nächste Folie}$
 $Directly-solve[L] \equiv []$
 $primitive[L] \equiv L = []$

6. Instantiiere das Divide & Conquer Schema

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )):Seq( $\mathbb{Z}$ ) WHERE true
  RETURNS S SUCH THAT rearranges(L,S)  $\wedge$  ordered(S)
 $\equiv$  if L=[] then []
      else let L1.L2 = ([Li | i  $\in$  [1..|L| $\div$ 2]],
                          [Li | i  $\in$  [1+|L| $\div$ 2..|L|]])
      in merge(sort(L1),sort(L2))
```

Algorithmus ist korrekt durch Konstruktion

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION **merge**($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION **merge**($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

1. Wähle *Compose* $\equiv$ **cons** ($O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z}$)

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION **merge**($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

1. Wähle *Compose* $\equiv$ **cons** $(O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z})$
2. *g* kann nicht **merge** sein Wähle *g* $\equiv$ **id** $(O'[a, a'] \equiv a = a')$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION $\text{merge}(S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})) : \text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

1. Wähle $\text{Compose} \equiv \text{cons}$ $(O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z})$
2. g kann nicht merge sein Wähle $g \equiv \text{id}$ $(O'[a, a'] \equiv a = a')$
3. Konstruiere $\succ$ auf $\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$:
 - $(S_1, S_2) \succ (S'_1, S'_2) \equiv |S_1| > |S'_1| \vee (|S_1| = |S'_1| \wedge |S_2| > |S'_2|)$
 - Kombination Längenordnung für Listen + lexikographische Ordnung für Produkte

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION $\text{merge}(S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})) : \text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
 RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

1. Wähle $\text{Compose} \equiv \text{cons}$ $(O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z})$

2. g kann nicht merge sein Wähle $g \equiv \text{id}$ $(O'[a, a'] \equiv a = a')$

3. Konstruiere $\succ$ auf $\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$:

- $(S_1, S_2) \succ (S'_1, S'_2) \equiv |S_1| > |S'_1| \vee (|S_1| = |S'_1| \wedge |S_2| > |S'_2|)$
- Kombination Längenordnung für Listen + lexikographische Ordnung für Produkte

4. Konstruiere Decompose mit Axiom 2 und 3

$O_D(S_1, S_2, a', S'_1, S'_2) \wedge a = a' \wedge \text{MERGE}(S'_1, S'_2, S') \wedge S = a.S' \Rightarrow \text{MERGE}(S_1, S_2, S)$

liefert als Spezifikation und Lösung für Decompose

FUNCTION $f_d(S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})) : \mathbb{Z} \times \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$

WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2) \wedge S_1 \neq [] \wedge S_2 \neq []$

RETURNS a', S'_1, S'_2

SUCH THAT $\text{rearranges}(a.(S'_1 \circ S'_2), S_1 \circ S_2)$ (1)

$\wedge \forall x \in S'_1 \circ S'_2. a \leq x \wedge (S_1, S_2) \succ (S'_1, S'_2)$ (2)

(1) folgt aus $\text{rearranges}(S'_1 \circ S'_2, S')$, $S = a.S'$ und $\text{rearranges}(S_1 \circ S_2, S)$

(2) folgt aus (1) und aus $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION *merge*($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
 RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

1. Wähle *Compose* $\equiv$ *cons* $(O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z})$

2. g kann nicht *merge* sein Wähle $g \equiv$ *id* $(O'[a, a'] \equiv a = a')$

3. Konstruiere $\succ$ auf $\text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$:

- $(S_1, S_2) \succ (S'_1, S'_2) \equiv |S_1| > |S'_1| \vee (|S_1| = |S'_1| \wedge |S_2| > |S'_2|)$
- Kombination Längenordnung für Listen + lexikographische Ordnung für Produkte

4. Konstruiere *Decompose* mit Axiom 2 und 3

$O_D(S_1, S_2, a', S'_1, S'_2) \wedge a = a' \wedge \text{MERGE}(S'_1, S'_2, S') \wedge S = a.S' \Rightarrow \text{MERGE}(S_1, S_2, S)$

liefert als Spezifikation und Lösung für *Decompose*

FUNCTION *f_d*($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\mathbb{Z} \times \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$

WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2) \wedge S_1 \neq [] \wedge S_2 \neq []$

RETURNS a', S'_1, S'_2

SUCH THAT $\text{rearranges}(a.(S'_1 \circ S'_2), S_1 \circ S_2)$

$\wedge \forall x \in S'_1 \circ S'_2. a \leq x \wedge (S_1, S_2) \succ (S'_1, S'_2)$

$= \text{let } x_1.S'_1 = S_1 \text{ and } x_2.S'_2 = S_2 \text{ in if } x_1 \leq x_2 \text{ then } (x_1, S'_1, S'_2) \text{ else } (x_2, S'_1, S'_2)$

(Lösung durch Fallanalyse: a' muß Minimum von $\text{hd}(S_1)$ und $\text{hd}(S_2)$ sein)

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION **merge**($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

FUNCTION *merge*($S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$

5. Vorbedingung für Korrektheit von *Decompose* ist $S_1 \neq [] \wedge S_2 \neq []$

– Liefert *primitive* und Spezifikation für *Directly-solve*

FUNCTION $f_p(S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})) : \text{Seq}(\mathbb{Z})$
WHERE $\text{ordered}(S_1) \wedge \text{ordered}(S_2) \wedge (S_1 = [] \vee S_2 = [])$
RETURNS S SUCH THAT $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$
= if $S_1 = []$ then S_2 else S_1

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

```
FUNCTION merge( $S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ ):  $\text{Seq}(\mathbb{Z})$  WHERE  $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$   
RETURNS S SUCH THAT  $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$ 
```

5. Vorbedingung für Korrektheit von *Decompose* ist $S_1 \neq [] \wedge S_2 \neq []$

- Liefert *primitive* und Spezifikation für *Directly-solve*

```
FUNCTION  $f_p(S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ ):  $\text{Seq}(\mathbb{Z})$   
WHERE  $\text{ordered}(S_1) \wedge \text{ordered}(S_2) \wedge (S_1 = [] \vee S_2 = [])$   
RETURNS S SUCH THAT  $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$   
= if  $S_1 = []$  then  $S_2$  else  $S_1$ 
```

6. Instantiiierter Divide & Conquer Algorithmus

```
FUNCTION merge( $S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ ):  $\text{Seq}(\mathbb{Z})$   
WHERE  $\text{ordered}(S_1) \wedge \text{ordered}(S_2)$   
RETURNS S SUCH THAT  $\text{ordered}(S) \wedge \text{rearranges}(S, S_1 \circ S_2)$   
= if  $S_1 = []$  then  $S_2$   
   elseif  $S_2 = []$  then  $S_1$   
   else let  $x_1.S_1' = S_1$  and  $x_2.S_2' = S_2$   
        in if  $x_1 \leq x_2$  then  $x_1.\text{merge}(S_1', S_2)$  else  $x_2.\text{merge}(S_1, S_2')$ 
```

- Wähle einfachere Dekomposition

- *Decompose* $\equiv$ HdTl, $g \equiv$ id
- *primitive*[L] $\equiv$ L=[], *Directly-solve*[L] $\equiv$ []
- *Compose*[a,S] $\equiv$ ordered_insert(a,S)

Sortieren durch Einfügen

● Wähle einfachere Dekomposition

- *Decompose* $\equiv$ HdTl, $g \equiv$ id
- *primitive*[L] $\equiv$ L=[], *Directly-solve*[L] $\equiv$ []
- *Compose*[a,S] $\equiv$ ordered_insert(a,S)

Sortieren durch Einfügen

● Wähle einfache Komposition

- *Compose*[a,S] $\equiv$ a.S, $g \equiv$ id
- *primitive*[L] $\equiv$ L=[], *Directly-solve*[L] $\equiv$ []
- *Decompose*[L] $\equiv$ let m=min(L) in (m, L-m)

Sortieren durch Auswahl

● Wähle einfachere Dekomposition

- $Decompose \equiv HdTl, \quad g \equiv id$
- $primitive[L] \equiv L=[], \quad Directly-solve[L] \equiv []$
- $Compose[a, S] \equiv ordered_insert(a, S)$

Sortieren durch Einfügen

● Wähle einfache Komposition

- $Compose[a, S] \equiv a.S, \quad g \equiv id$
- $primitive[L] \equiv L=[], \quad Directly-solve[L] \equiv []$
- $Decompose[L] \equiv \text{let } m = \min(L) \text{ in } (m, L - m)$

Sortieren durch Auswahl

● Wähle binare Komposition

- $Compose[S_1, S_2] \equiv S_1 \circ S_2, \quad g \equiv \text{sort}$
- $primitive[L] \equiv L=[], \quad Directly-solve[L] \equiv []$
- $Decompose[L] \equiv \text{let } a = L_{[|L|/2]} \text{ in } (L_{<a}, L_{\geq a})$

Naives Quicksort

● Wähle einfachere Dekomposition

- $Decompose \equiv HdTl, \quad g \equiv id$
- $primitive[L] \equiv L=[], \quad Directly-solve[L] \equiv []$
- $Compose[a, S] \equiv ordered_insert(a, S)$

Sortieren durch Einfügen

● Wähle einfache Komposition

- $Compose[a, S] \equiv a.S, \quad g \equiv id$
- $primitive[L] \equiv L=[], \quad Directly-solve[L] \equiv []$
- $Decompose[L] \equiv let \ m=min(L) \ in \ (m, L-m)$

Sortieren durch Auswahl

● Wähle binare Komposition

- $Compose[S_1, S_2] \equiv S_1 \circ S_2, \quad g \equiv sort$
- $primitive[L] \equiv L=[], \quad Directly-solve[L] \equiv []$
- $Decompose[L] \equiv let \ let \ a=L_{\lfloor |L|/2 \rfloor} \ in \ (L_{<a}, L_{\geq a})$

Naives Quicksort



Flexible Strategie mit vielfältigen Anwendungen