

Automatisierte Logik und Programmierung

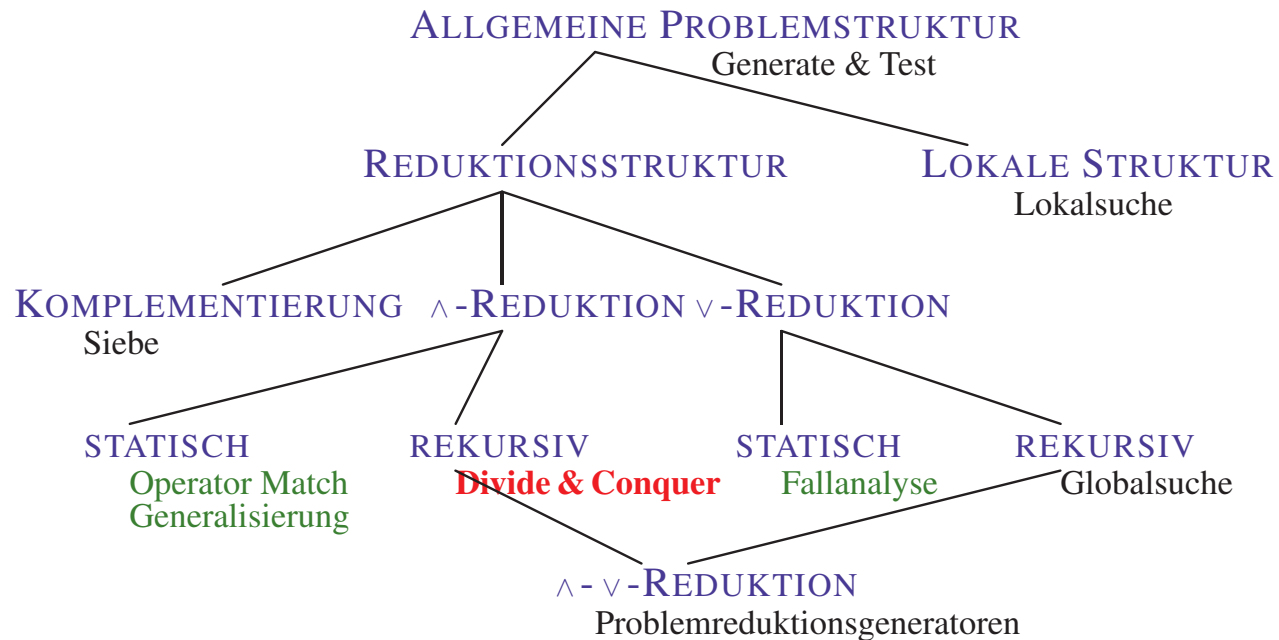
Einheit 19

Synthese von Divide & Conquer Algorithmen



1. Algorithmenschema
2. Korrektheit
3. Synthesestrategie
4. Wissensbasierte Unterstützung

DIVIDE & CONQUER ALGORITHMEN



- **Effiziente Verarbeitung strukturierter Daten**
 - Sehr gebräuchliche und einfache Programmieretechnik
 - **Aufteilen**: Zerlege Problem in kleinere Teilprobleme
 - **Erobern**: Löse Teilprobleme separat und kombiniere Lösungen
- **Rekursive \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  $L \neq []$   
  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 $\neq$ []  
  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** `hd(L)`

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

- **Maximum einer nichtleeren Liste von Zahlen**

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE L $\neq$ []  
  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** $\text{hd}(L)$

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

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

- **Maximum einer nichtleeren Liste von Zahlen**

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE L $\neq$ []  
  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** $hd(L)$

Andernfalls **Dekomposition** der Eingabe mit $HdT1: L \mapsto a.L'$

Einfache Teillösung für a $a=id(a)$

Rekursive Lösung für L' $m'=maxL(L')$

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

● Maximum einer nichtleeren Liste von Zahlen

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE L $\neq$ []  
  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** $\text{hd}(L)$

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

Einfache Teillösung für a $a=\text{id}(a)$

Rekursive Lösung für L' $m'=\text{maxL}(L')$

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

EIN TYPISCHER DIVIDE & CONQUER ALGORITHMUS

● Maximum einer nichtleeren Liste von Zahlen

```
FUNCTION maxL(L:Seq( $\mathbb{Z}$ )): $\mathbb{Z}$  WHERE L $\neq$ []  
  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** $\text{hd}(L)$

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

Einfache Teillösung für a $a = \text{id}(a)$

Rekursive Lösung für L' $m' = \text{maxL}(L')$

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

● 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 (max  $\circ$  (id $\times$ maxL)  $\circ$  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 t SUCH THAT $O[x,t]$
 \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

...sowie Domains & Ein-/Ausgabebedingungen an Komponenten, Terminierungskriterium

Korrektheit folgt aus wenigen Voraussetzungen

KORREKTHEIT DES DIVIDE & CONQUER SCHEMAS

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

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 t SUCH THAT $O_C[z', z, t]$

6. Verkleinerungsrelation \succ ist wohlfundierte Ordnung auf D

Sechste Komponente der Theorie, nötig nur für Terminierungsbeweis

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIF

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

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIF

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

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

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

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

- **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: KORREKTHEITSBEWEIF

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

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

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWWEIS

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

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

DIVIDE & CONQUER SCHEMA: KORREKTHEITSBEWEIF

FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS t SUCH THAT $O[x,t]$
 \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: $I[x] \wedge \textit{primitive}[x]$

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

Nichtprimitive Eingaben: $I[x] \wedge \neg \textit{primitive}[x]$

– $f(x) = (\textit{Compose} \circ g \times f \circ \textit{Decompose})(x)$

– *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**

– *Compose* $[z', z]$ 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: KORREKTHEITSBEWEIF

FUNCTION $f(x:D) : R$ WHERE $I[x]$ RETURNS t SUCH THAT $O[x,t]$
 \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: $I[x] \wedge \textit{primitive}[x]$

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

Nichtprimitive Eingaben: $I[x] \wedge \neg \textit{primitive}[x]$

– $f(x) = (\textit{Compose} \circ g \times f \circ \textit{Decompose})(x)$

– *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**

– *Compose* $[z', z]$ 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: KORREKTHEITSBEWEIF

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

– $Compose[z', z]$ 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**

DIVIDE & CONQUER SCHEMA ALS ALGORITHMENTHEORIE

Es sind insgesamt 12 Komponenten zu bestimmen

- **Zusätzliche Datentypen**

- Ein-/Ausgabetypen der Hilfsfunktion (D' , R')

- **Zusätzliche Ein-/Ausgabebedingungen**

- Für Dekomposition (O_D), Komposition (O_C), Hilfsfunktion (I' , O')

- **Komponenten des Algorithmus**

- *Directly-solve*: $D \rightarrow R$

Axiom 1

- *primitive*: $D \rightarrow \mathbb{B}$

- *Decompose*: $D \rightarrow D' \times D$

Axiom 3

- *g*: $D' \rightarrow R'$

Axiom 4

- *Compose*: $R' \times R \rightarrow R$

Axiom 5

- **Terminierungsordnung für Rekursion**

- Wohlfundierte Ordnung \succ auf D

Axiom 6

Lösungen müssen Beweis der Axiome 1–6 liefern

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN

Spezifikation: FUNCTION $\text{minL}(L:\text{Seq}(\mathbb{Z})):\mathbb{Z}$ WHERE $L \neq []$ RETURNS m
SUCH THAT $m \in L \wedge \forall x \in L. m \leq x$

Ziel: Bestimme Komponenten des D&C Algorithmenschemas für minL

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN

Spezifikation: FUNCTION $\text{minL}(L:\text{Seq}(\mathbb{Z})):\mathbb{Z}$ WHERE $L \neq []$ RETURNS m
SUCH THAT $m \in L \wedge \forall x \in L. m \leq x$

Ziel: Bestimme Komponenten des D&C Algorithmenschemas für minL

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

- Eine Liste kann in ein Element (am Anfang oder Ende) und den Rest oder in zwei Teillisten (in der Mitte) zerlegt werden

In beiden Fällen wird die Liste hierdurch kürzer

- Für Basistypen sollten mögliche Zerlegungen samt Spezifikation und zugehörigen Ordnungen (bewiesen) in der Wissensbank gespeichert sein
- Aus den Möglichkeiten wähle Zerlegung in Anfang (hd) und Rest (tl)

Dies liefert 4 der gesuchten 12 Komponenten

- *Decompose* $\equiv \text{HdTl}: \text{Seq}(\alpha) \rightarrow \alpha \times \text{Seq}(\alpha)$ (matching liefert $\alpha \equiv \mathbb{Z}$)
- Extra-Domäne: $D' \equiv \mathbb{Z}$
- Ausgabebedingung: $O_D[L, a, L'] \equiv L = a.L'$
- und wohlfundierte Ordnung: $L \succ L' \equiv |L| > |L'| \quad \mapsto \text{Axiom 6}$

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN (2)

2. Die Hilfsfunktion g muß konstruiert werden

- Hilfsfunktion g muß auf $D' \equiv \mathbb{Z}$ operieren, also auf Elementen der Liste
- Auf Ausgabe von *Decompose* muß $g \times \text{minL}$ angewandt werden
und *Compose* muß auf Resultat von $g \times \text{minL}$ angewandt werden
- Da *Compose* ohnehin synthetisiert wird, ist “nichts tun” die beste Wahl
Dies liefert weitere 4 Komponenten

- $g \equiv id: \alpha \rightarrow \alpha$ (instantiiert mit $\alpha \equiv \mathbb{Z}$)
- Eingabebedingung $I'[a] \equiv \text{true}$
- Extra-Bildbereich $R' \equiv \mathbb{Z}$
- Ausgabebedingung $O'[a, a'] \equiv a' = a$ \mapsto **Axiom 4**

```
FUNCTION  $g(a:\mathbb{Z}):\mathbb{Z}$  WHERE true RETURNS a' SUCH THAT a'=a  
 $\equiv a$  ist korrekt
```


SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN (3)

3. Aus Spezifikation von g kann Korrektheit von *Decompose* geprüft und dabei *primitive* konstruiert werden

FUNCTION $f_d(L: \text{Seq}(\mathbb{Z})) : \mathbb{Z} \times \text{Seq}(\mathbb{Z})$ WHERE $L \neq [] \wedge \neg \text{primitive}[L]$
 RETURNS a, L' SUCH THAT $L' \neq [] \wedge |L| > |L'| \wedge L = a.L'$
 $\equiv (\text{hd}(L), \text{tl}(L))$ *ist korrekt*

– Für nichtprimitive, nichtleere Listen muß $L' \neq [] \wedge |L| > |L'| \wedge L = a.L'$ gelten, falls $a = \text{hd}(L)$ und $L' = \text{tl}(L)$

– Transformiere Formel in hinreichende Bedingung an L

Einsetzen liefert $\text{tl}(L) \neq [] \wedge |L| > |\text{tl}(L)| \wedge L = \text{hd}(\text{tl}) . \text{tl}(L)$

· Es gilt $L \neq [] \Rightarrow L = \text{hd}(\text{tl}) . \text{tl}(L)$ und $L \neq [] \Rightarrow |L| > |\text{tl}(L)|$

· Es folgt $\neg \text{primitive}[L] \equiv \text{tl}(L) \neq []$, also $\text{primitive}[L] \equiv \text{tl}(L) = []$

· Es gilt $L \neq [] \Rightarrow \text{tl}(L) = [] \Leftrightarrow |L| = 1$

Dies liefert die Komponente $\text{primitive}[L] \equiv |L| = 1$ und \mapsto **Axiom 3**

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN (4)

4. Konstruiere Ausgabebedingung von *Compose* mit \mapsto **Axiom 2**

$$L = a.L' \wedge a' = a \wedge (m' \in L' \wedge \forall x \in L'. m' \leq x) \wedge O_C[a', m', m]$$

$$\Rightarrow (m \in L \wedge \forall x \in L. m \leq x)$$

– Formel muß in Bedingung an m transformiert werden

· Einsetzen liefert

$$m' \in L' \wedge \forall x \in L'. m' \leq x \wedge O_C[a', m', m] \Rightarrow m \in a'.L' \wedge \forall x \in a'.L'. m \leq x$$

· Es gilt $m \in a'.L' \Leftrightarrow m = a' \vee m \in L'$ und $\forall x \in a'.L'. m \leq x \Leftrightarrow m \leq a' \wedge \forall x \in L'. m \leq x$

Vereinfachen für beide Fälle liefert

$$m' \in L' \wedge \forall x \in L'. m' \leq x \wedge O_C[a', m', m] \Rightarrow m = a' \wedge \forall x \in L'. a' \leq x$$

$$\vee m \in L' \wedge \forall x \in L'. m \leq x \wedge m \leq a'$$

· Aus $m' \in L' \wedge \forall x \in L'. m' \leq x$ folgt $\forall x \in L'. z \leq x \Leftrightarrow z \leq m'$ und $y \in L \wedge y \leq m' \Leftrightarrow y = m'$

Dies führt zu folgender Bedingung, die nicht weiter vereinfacht werden kann

$$m' \in L' \wedge \forall x \in L'. m' \leq x \wedge O_C[a', m', m] \Rightarrow m = a' \wedge a' \leq m' \vee m = m' \wedge m \leq a'$$

Dies liefert $O_C[a', m', m] \equiv m = a' \wedge a' \leq m' \vee m = m' \wedge m \leq a'$

und

\mapsto **Axiom 2**

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN (5)

5. Erneute Synthese liefert Algorithmus für *Compose* \mapsto Axiom 5

FUNCTION $f_c(a', m' : \mathbb{Z} \times \mathbb{Z}) : \mathbb{Z}$ WHERE true RETURNS m
SUCH THAT $m = a' \wedge a' \leq m' \vee m = m' \wedge m \leq a'$
 \equiv *Compose*[a', m'] *ist korrekt*

– Disjunktion in O_C legt Algorithmenschema **Fallunterscheidung** nahe

Wähle $I_1(a', m') \equiv a' \leq m'$ und $I_2(a', m') \equiv m' \leq a'$

Vereinfachung unter diesen Eingabebedingungen liefert Spezifikationen

• FUNCTION $f_{c1}(a', m' : \mathbb{Z} \times \mathbb{Z}) : \mathbb{Z}$ WHERE $a' \leq m'$ RETURNS m SUCH THAT $m = a'$

• FUNCTION $f_{c2}(a', m' : \mathbb{Z} \times \mathbb{Z}) : \mathbb{Z}$ WHERE $m' \leq a'$ RETURNS m SUCH THAT $m = m'$

– Lösung ist in beiden Fällen trivial

– Synthese durch Fallanalyse liefert die Komponente

Compose[a', m'] \equiv if $a' \leq m'$ then a' else m' und \mapsto Axiom 5

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN (6)

6. Synthese liefert Algorithmus für *Directly-solve* \mapsto **Axiom 1**

FUNCTION $f_p(L: \text{Seq}(\mathbb{Z})) : \mathbb{Z}$ WHERE $|L|=1$ RETURNS m
SUCH THAT $m \in L \wedge \forall x \in L. m \leq x$
 \equiv *Directly-solve*[L]

ist korrekt

- Vereinfache $m \in L \wedge \forall x \in L. m \leq x$ im Kontext $|L|=1$
 - Es gilt $|L|=1 \Rightarrow L = [\text{hd}(L)]$ sowie $m \in [\text{hd}(L)] \Leftrightarrow m = \text{hd}(L)$
und $\forall x \in [\text{hd}(L)]. m \leq x \Leftrightarrow m \leq \text{hd}(L)$
- Liefert Komponente *Directly-solve*(L) \equiv $\text{hd}(L)$ und \mapsto **Axiom 1**

SYNTHESE EINES DIVIDE & CONQUER ALGORITHMUS

MINIMUM EINER NICHTLEEREN LISTE VON ZAHLEN (6)

6. Synthese liefert Algorithmus für *Directly-solve* ↪ **Axiom 1**

FUNCTION $f_p(L: \text{Seq}(\mathbb{Z})) : \mathbb{Z}$ WHERE $|L|=1$ RETURNS m
 SUCH THAT $m \in L \wedge \forall x \in L. m \leq x$
 \equiv *Directly-solve*[L] *ist korrekt*

- Vereinfache $m \in L \wedge \forall x \in L. m \leq x$ im Kontext $|L|=1$
 - Es gilt $|L|=1 \Rightarrow L = [\text{hd}(L)]$ sowie $m \in [\text{hd}(L)] \Leftrightarrow m = \text{hd}(L)$
 und $\forall x \in [\text{hd}(L)]. m \leq x \Leftrightarrow m \leq \text{hd}(L)$
- Liefert Komponente *Directly-solve*(L) \equiv $\text{hd}(L)$ und ↪ **Axiom 1**

7. Alle 12 Komponenten sind bestimmt, alle Axiome geprüft

- Instantiiere das Divide & Conquer Schema für minL

FUNCTION $\text{minL}(L: \text{Seq}(\mathbb{Z})) : \mathbb{Z}$ WHERE $L \neq []$ SUCH THAT $m \in L \wedge \forall x \in L. m \leq x$
 \equiv if $|L|=1$ then $\text{hd}(L)$ *(Directly-solve)*
 else let $a.L' = L$ in *(Decompose)*
 let $m' = \text{minL}(L')$ in *(Recursive call)*
 if $a' \leq m'$ then a' else m' *(Compose)*

Alle durchgeführten Schritte sind automatisierbar

PROGRAMMIERWISSEN FÜR DIVIDE & CONQUER

- **Programmsynthese benötigt formalisiertes Wissen**
 - Verfahren können nur erfolgreich sein, wenn Grundkenntnisse über Standard-Datentypen in einer Wissensbank zur Verfügung stehen
 - Wichtig für die ersten Schritte einer Synthese
- **Standard-Zerlegungen für Eingabetypen** $\mapsto \text{Decompose}, O_D, D'$
 - Endliche Listen/Folgen ($\text{Seq}(\alpha)$): **HdTl**, **FirstLast**, **ListSplit**
z.B. $\text{ListSplit}(L) \equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$
mit $O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2$ und $D' \equiv \text{Seq}(\alpha)$
 - Endliche Mengen ($\text{Set}(\alpha)$): **ArbRest**
 - Datencontainer wie Produkträume: Zerlegung in **Einzelkomponenten**
 - Natürlichen Zahlen (\mathbb{N}): **Vorgängerfunktion**
⋮
- **Standard-Wohlordnungen auf dem Typ D** $\mapsto \succ$
 - Folgen und Mengen: **Längen/Größenordnung** $L \succ L' \equiv |L| > |L'|$
 - Datencontainer wie Produkträume: **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-SYNTHESESTRATEGIE FÜR DIVIDE & CONQUER

Zerlege Problem in Spezifikationen für Teilprobleme

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

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

UNTERSTÜTZENDE TECHNIKEN DER D&C-STRATEGIE

- **Einfache Heuristik zur Bestimmung der Hilfsfunktion 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 $I'[y']:=\text{true}$
Verlagert einen Teil der Aufgaben von g in Kompositionsfunktion

- **Inferenzmechanismus: Abgeleitete Vorbedingungen** ↪ §18, Folie 26

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

- **Inferenzmechanismus: Fallanalyse**

↪ §18, Folie 24

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

Liefert Programmstücke mit Fallunterscheidungen

- **Inferenzmechanismus: Operator match**

↪ §18, Folie 19

- Synthese von Programmstücken durch Anpassung an bekannte Lösungen

MAN KANN AUCH IN ANDERER REIHENFOLGE VORGEHEN

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

- **Grundstrategie: Zerlege Eingaben zuerst**
 - Wähle \succ und *Decompose* aus Wissensbank
 - Konstruiere g heuristisch
 - Bestimme *primitive* durch Verifikation von *Decompose*
 - Konstruiere Spezifikation und Lösung für *Compose*
 - Konstruiere *Directly-solve*
- **Umgekehrte Strategie: “Zerlege” Ausgabe zuerst**
 - Wähle *Compose* aus Wissensbank
 - Konstruiere g heuristisch
 - Konstruiere Spezifikation und Lösung für *Decompose*; bestimme \succ
 - Bestimme *primitive* und konstruiere *Directly-solve*
- **Mischstrategie: Zerlege Eingaben, dann Ausgabe und fülle Lücke**
 - Wähle \succ und *Decompose* aus Wissensbank
 - Konstruiere *Compose* heuristisch
 - Konstruiere Spezifikation und Lösung für g
 - Bestimme *primitive* und konstruiere *Directly-solve*

Unterstützung für andere D&C Strategien

- **Standard-Kompositionen für Ausgabetypen** $\mapsto \text{Compose}, O_C, R'$

Umkehrung der Zerlegungsoperationen für den Datentyp

– Endliche Folgen: **cons** ($a.l$), **append** ($L_1 \circ L_2$)

– Endliche Mengen: **insert** ($S+a$), **union** ($S_1 \cup S_2$)

– Datencontainer wie Produkträume: **Komponentenweise Komposition**

– Natürlichen Zahlen (\mathbb{N}): **Nachfolgerfunktion**

- **Einfache Heuristik zur Bestimmung der Hilfsfunktion g**

Umkehrung der Heuristik für die Grundstrategie nach Wahl von *Compose*

– Falls $R' = R$, wähle $g := f$, $D' := D$, $O' := O$ und $I' := I$

– Falls $R' \neq R$, wähle $g := id$, $D' := R'$, $O'[y', z'] := z' = y'$ und $I'[y'] := true$

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

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

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

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

1. Wähle *Compose* \equiv cons

– Dies liefert 3 der gesuchten 12 Komponenten

- $\text{Compose}[a',S'] \equiv a'.S'$
- Extra-Bildbereich: $R' \equiv \mathbb{Z}$
- Ausgabebedingung: $O_C[a',S',S] \equiv S=a'.S'$

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

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

1. Wähle *Compose* \equiv `cons`

– Dies liefert 3 der gesuchten 12 Komponenten

- *Compose*[a', S'] $\equiv a'.S'$
- Extra-Bildbereich: $R' \equiv \mathbb{Z}$
- Ausgabebedingung: $O_C[a', S', S] \equiv S = a'.S'$

2. Konstruiere Hilfsfunktion *g* heuristisch

– Dies liefert weitere 4 Komponenten

- *g* $\equiv id: \alpha \rightarrow \alpha$ (instantiiert mit $\alpha \equiv \mathbb{Z}$)
- Eingabebedingung $I'[a] \equiv true$
- Extra-Domäne $D' \equiv Seq(\mathbb{Z})$
- Ausgabebedingung $O'[a, a'] \equiv a' = a$

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

3. Konstruiere Ausgabebedingung O_D mit Axiom 2

$$O_D[L, a, L'] \wedge a=a' \wedge \text{rearranges}(L', S') \wedge \text{ordered}(S') \wedge S=a' . S' \\ \Rightarrow \text{rearranges}(L, S) \wedge \text{ordered}(S)$$

– Formel muß in Bedingungen an L, a, L' transformiert werden

· Einsetzen liefert $O_D[L, a, L'] \wedge \text{rearranges}(L', S') \wedge \text{ordered}(S')$
 $\Rightarrow \text{rearranges}(L, a.S') \wedge \text{ordered}(a.S')$

· Es gilt $\text{ordered}(S') \Rightarrow \text{ordered}(a.S') \Leftrightarrow \forall x \in S. a \leq x$

und $\text{rearranges}(L', S') \Rightarrow \text{rearranges}(L, a.S') \Leftrightarrow \text{rearranges}(L, a.L')$

und $\text{rearranges}(L, a.L') \Leftrightarrow a \in L \wedge \text{rearranges}(L', L-a)$

– Liefert $O_D[L, a, L'] \equiv a \in L \wedge \forall x \in L. a \leq x \wedge \text{rearranges}(L', L-a)$

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

3. Konstruiere Ausgabebedingung O_D mit Axiom 2

$$O_D[L, a, L'] \wedge a=a' \wedge \text{rearranges}(L', S') \wedge \text{ordered}(S') \wedge S=a' . S' \\ \Rightarrow \text{rearranges}(L, S) \wedge \text{ordered}(S)$$

– Formel muß in Bedingungen an L, a, L' transformiert werden

· Einsetzen liefert $O_D[L, a, L'] \wedge \text{rearranges}(L', S') \wedge \text{ordered}(S')$
 $\Rightarrow \text{rearranges}(L, a.S') \wedge \text{ordered}(a.S')$

· Es gilt $\text{ordered}(S') \Rightarrow \text{ordered}(a.S') \Leftrightarrow \forall x \in S. a \leq x$
und $\text{rearranges}(L', S') \Rightarrow \text{rearranges}(L, a.S') \Leftrightarrow \text{rearranges}(L, a.L')$
und $\text{rearranges}(L, a.L') \Leftrightarrow a \in L \wedge \text{rearranges}(L', L-a)$

– Liefert $O_D[L, a, L'] \equiv a \in L \wedge \forall x \in L. a \leq x \wedge \text{rearranges}(L', L-a)$

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

– Spezifikation O_D wird erfüllt durch $a=\text{min}L(L)$ und $L'=L-a$ (Folie 7)

– Liefert als Lösung $\text{Decompose}[L] \equiv \text{let } a=\text{min}L(L) \text{ in } (a, L-a)$

– Wegen $|L| > |L-a|$ wähle wohlfundierte Ordnung $L \succ L' \equiv |L| > |L'|$

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

5. Konstruiere *primitive* mit Axiom 3

FUNCTION $f_d(L:\text{Seq}(\mathbb{Z})):\mathbb{Z}\times\text{Seq}(\mathbb{Z})$ WHERE $\neg\text{primitive}[L]$ RETURNS a,L'
SUCH THAT $|L|>|L'| \wedge a\in L \wedge \forall x\in L. a\leq x \wedge \text{rearranges}(L',L-a)$
 \equiv let $a=\text{min}L(L)$ in $(a,L-a)$ *ist korrekt*

- Vorbedingung für Anwendung $\text{min}L$ ist $L\neq []$
unter dieser Vorbedingung sind alle Ausgabebedingungen erfüllt
- Liefert $\text{primitive}[L] \equiv L=[]$

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

5. Konstruiere *primitive* mit Axiom 3

FUNCTION $f_d(L:Seq(\mathbb{Z})):\mathbb{Z}\times Seq(\mathbb{Z})$ WHERE $\neg primitive[L]$ RETURNS a,L'
SUCH THAT $|L|>|L'| \wedge a \in L \wedge \forall x \in L. a \leq x \wedge rearranges(L',L-a)$
 \equiv let $a=\min L$ in $(a,L-a)$ ist korrekt

- Vorbedingung für Anwendung $\min L$ ist $L \neq []$
unter dieser Vorbedingung sind alle Ausgabebedingungen erfüllt
- Liefert $primitive[L] \equiv L = []$

6. Synthese liefert Algorithmus für *Directly-solve* \mapsto Axiom 1

FUNCTION $f_p(L:Seq(\mathbb{Z})):Seq(\mathbb{Z})$ WHERE $L=[]$ RETURNS S
SUCH THAT $rearranges(L,S) \wedge ordered(S)$
 \equiv *Directly-solve*[L] ist korrekt

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

SYNTHESE EINES SORTIERALGORITHMUS

VERWENDUNG DER UMGEKEHRTEN DIVIDE & CONQUER SYNTHESE

5. Konstruiere *primitive* mit Axiom 3

```
FUNCTION  $f_d(L:Seq(\mathbb{Z})):\mathbb{Z}\times Seq(\mathbb{Z})$  WHERE  $\neg primitive[L]$  RETURNS  $a,L'$   
  SUCH THAT  $|L|>|L'| \wedge a\in L \wedge \forall x\in L. a\leq x \wedge rearranges(L',L-a)$   
 $\equiv$  let  $a=\min L(L)$  in  $(a,L-a)$  ist korrekt
```

- Vorbedingung für Anwendung $\min L$ ist $L \neq []$
unter dieser Vorbedingung sind alle Ausgabebedingungen erfüllt
- Liefert $primitive[L] \equiv L = []$

6. Synthese liefert Algorithmus für *Directly-solve* \mapsto Axiom 1

```
FUNCTION  $f_p(L:Seq(\mathbb{Z})):Seq(\mathbb{Z})$  WHERE  $L=[]$  RETURNS S  
  SUCH THAT  $rearranges(L,S) \wedge ordered(S)$   
 $\equiv$  Directly-solve[L] ist korrekt
```

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

7. 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  $a=\min L(L)$  in  $a.sort(L-a)$ 
```

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE

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

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE

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

1. Wähle *Decompose* \equiv ListSplit

– Dies liefert zwei weitere Komponenten

- Extra-Domäne: $D' \equiv \text{Seq}(\mathbb{Z})$
- Ausgabebedingung: $O_D[L,L_1,L_2] \equiv L=L_1 \circ L_2$
- Passend dazu die wohlfundierte Ordnung $L \succ L' \equiv |L| > |L'|$

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE

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

1. Wähle *Decompose* \equiv ListSplit

– Dies liefert zwei weitere Komponenten

- Extra-Domäne: $D' \equiv \text{Seq}(\mathbb{Z})$
- Ausgabebedingung: $O_D[L, L_1, L_2] \equiv L = L_1 \circ L_2$
- Passend dazu die wohlfundierte Ordnung $L \succ L' \equiv |L| > |L'|$

2. Konstruiere Hilfsfunktion *g* heuristisch

– Da $D' = D$ ist, wählen wir die zu synthetisierende Funktion

– Dies liefert weitere 4 Komponenten

- $g \equiv \text{sort}:\mathbb{Z} \rightarrow \mathbb{Z}$ (instantiiert mit $\alpha \equiv \mathbb{Z}$)
- Eingabebedingung $I'[L] \equiv \text{true}$
- Extra-Domäne $D' \equiv \text{Seq}(\mathbb{Z})$
- Ausgabebedingung $O' \equiv O'$

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE (2)

3. Verifiziere Axiom 3 für *Decompose*

FUNCTION $f_d(L:Seq(\mathbb{Z})) : Seq(\mathbb{Z}) \times Seq(\mathbb{Z})$ WHERE $\neg primitive[L]$
RETURNS L_1, L_2 SUCH THAT $|L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$
 $\equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$ *ist korrekt*

- Aus $|L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$ folgt $|L| > 1$ (§18, Folie 25)
- Dies liefert die Komponente $primitive[L] \equiv |L| \leq 1$

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE (2)

3. Verifiziere Axiom 3 für *Decompose*

FUNCTION $f_d(L:Seq(\mathbb{Z})) : Seq(\mathbb{Z}) \times Seq(\mathbb{Z})$ WHERE $\neg primitive[L]$
RETURNS L_1, L_2 SUCH THAT $|L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$
 $\equiv ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$ *ist korrekt*

– Aus $|L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$ folgt $|L| > 1$ (§18, Folie 25)

– Dies liefert die Komponente $primitive[L] \equiv |L| \leq 1$

4. Konstruiere Spezifikation für *Compose* mit Axiom 2

$|L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L \wedge SORT(L_1, S_1) \wedge SORT(L_2, S_2)$
 $\wedge O_C[S_1, S_2, S] \Rightarrow SORT(L, S)$

– wobei $SORT(L, S) \equiv rearranges(L, S) \wedge ordered(S)$

– Formel muß in Bedingung an S, S_1, S_2 transformiert werden

· Aus $L_1 \circ L_2 = L \wedge rearranges(L_1, S_1) \wedge rearranges(L_2, S_2)$ folgt $rearranges(L, S_1 \circ S_2)$

· $rearranges(L, S_1 \circ S_2) \Rightarrow rearranges(L, S)$ gilt, falls $rearranges(S, S_1 \circ S_2)$

· $ordered(S_1) \wedge ordered(S_2) \Rightarrow ordered(S)$ ist nicht zu vereinfachen

– Liefert als Ausgabebedingung $O_C \quad \mapsto$ *Synthese später auf Folie 24*

· $O_C \equiv ordered(S_1) \wedge ordered(S_2) \Rightarrow ordered(S) \wedge rearranges(S, S_1 \circ S_2)$

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE (3)

5. Konstruiere *Directly-solve* aus Axiom 1

```
FUNCTION  $f_p(L:Seq(\mathbb{Z})) : Seq(\mathbb{Z})$  WHERE  $|L| \leq 1$   
  RETURNS S SUCH THAT  $rearranges(L,S) \wedge ordered(S)$   
 $\equiv$  Directly-solve[L] ist korrekt
```

- Aus $|L| \leq 1 \wedge rearranges(L,S)$ folgt $|S| \leq 1$
 Aus $|S| \leq 1$ folgt $ordered(S)$
- Zu konstruieren bleibt nur ein S mit $rearranges(L,S)$

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

EINE KOMPLEXERE DIVIDE & CONQUER SYNTHESE (3)

5. Konstruiere *Directly-solve* aus Axiom 1

```
FUNCTION  $f_p(L:Seq(\mathbb{Z})) : Seq(\mathbb{Z})$  WHERE  $|L| \leq 1$   
  RETURNS S SUCH THAT  $rearranges(L,S) \wedge ordered(S)$   
 $\equiv$  Directly-solve[L] ist korrekt
```

- Aus $|L| \leq 1 \wedge rearranges(L,S)$ folgt $|S| \leq 1$
 Aus $|S| \leq 1$ folgt $ordered(S)$
- Zu konstruieren bleibt nur ein S mit $rearranges(L,S)$

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

6. Alle Komponenten sind bestimmt, alle Axiome geprüft

- Instantiiere das Divide & Conquer Schema für `sort`

```
FUNCTION sort(L:Seq( $\mathbb{Z}$ )) : Seq( $\mathbb{Z}$ ) WHERE true  
  RETURNS S SUCH THAT  $rearranges(L,S) \wedge ordered(S)$   
 $\equiv$  if  $|L| \leq 1$  then L (Directly-solve)  
    else let  $L_1, L_2 = ([L_i | i \in [1..|L| \div 2]], [L_i | i \in [1+|L| \div 2..|L|]])$   
      (Decompose)  
      in merge(sort( $L_1$ ), sort( $L_2$ )) (Recursive call + Compose)
```

- **Algorithmus ist korrekt per Konstruktion**

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

Spezifikation zerlegt O_C in Vorbedingung und Ausgabebedingung:

```
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

Spezifikation zerlegt O_C in Vorbedingung und Ausgabebedingung:

```
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 $\text{Compose} \equiv \text{cons}$

$(O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z})$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

Spezifikation zerlegt O_C in Vorbedingung und Ausgabebedingung:

```
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 $Compose \equiv \text{cons}$ $(O_C[a, S', S] \equiv S = a.S', R' \equiv \mathbb{Z})$
2. Wegen $R' \neq R$ wähle $g \equiv \text{id}$ $(O'[a, a'] \equiv a = a')$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

Spezifikation zerlegt O_C in Vorbedingung und Ausgabebedingung:

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

2. Wegen $R' \neq R$ wähle $g \equiv \text{id}$ $(O'[a, a'] \equiv a = a')$

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

– Kombiniere lexikographische Ordnung für Produkte mit Längenordnung für Listen

$$(S_1, S_2) \succ (S'_1, S'_2) \equiv |S_1| > |S'_1| \vee (|S_1| = |S'_1| \wedge |S_2| > |S'_2|)$$

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

Spezifikation zerlegt O_C in Vorbedingung und Ausgabebedingung:

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

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

2. Wegen $R' \neq R$ wähle *g* \equiv id $(O'[a, a'] \equiv a = a')$

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

– Kombiniere lexikographische Ordnung für Produkte mit Längenordnung für Listen

$$(S_1, S_2) \succ (S'_1, S'_2) \equiv |S_1| > |S'_1| \vee (|S_1| = |S'_1| \wedge |S_2| > |S'_2|)$$

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 ordered( $S_1$ )  $\wedge$  ordered( $S_2$ )  $\wedge$   $S_1 \neq []$   $\wedge$   $S_2 \neq []$ 
```

```
  RETURNS  $a', S'_1, S'_2$ 
```

```
  SUCH THAT 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 rearranges($S'_1 \circ S'_2, S'$), $S = a.S'$ und rearranges($S_1 \circ S_2, S$)

(2) folgt aus (1) und aus ordered(S_1) \wedge ordered(S_2)

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

Spezifikation zerlegt O_C in Vorbedingung und Ausgabebedingung:

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. Wegen $R' \neq R$ wähle $g \equiv \text{id}$ $(O'[a, a'] \equiv a = a')$

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

– Kombiniere lexikographische Ordnung für Produkte mit Längenordnung für Listen

$(S_1, S_2) \succ (S'_1, S'_2) \equiv |S_1| > |S'_1| \vee (|S_1| = |S'_1| \wedge |S_2| > |S'_2|)$

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

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

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

(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 `ordered`(S_1) \wedge `ordered`(S_2)

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

DIVIDE & CONQUER SYNTHESE DER merge-FUNKTION

```
FUNCTION merge( $S_1, S_2: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z})$ ) : 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. Vorbedingung für Korrektheit von *Decompose* ist $S_1 \neq [] \wedge S_2 \neq []$

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

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

6. Instantiiertes Divide & Conquer Algorithmus

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

ERZEUGUNG ALTERNATIVER SORTIERALGORITHMEN

- **Wähle einfache Dekomposition** (Sortieren durch Einfügen)
 - *Decompose* \equiv HdTl, $g \equiv$ id
 - *primitive*[L] \equiv L=[], *Directly-solve*[L] \equiv []
 - *Compose*[a, S] \equiv ordered_insert(a, S)
- **Wähle einfache Komposition** (Sortieren durch Auswahl)
 - *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)
- **Wähle binäre Komposition** (Naives Quicksort \mapsto Übung)
 - *Compose*[S₁, S₂] \equiv S₁°S₂, $g \equiv$ sort
 - *primitive*[L] \equiv |L| \leq 1, *Directly-solve*[L] \equiv L
 - *Decompose*[L] \equiv let let a=L[|L|/2] in (L<a, L \geq a]

Flexible Strategie mit vielfältigen Anwendungen

• Formuliere und beweise D&C-Synthesetheorem

$\forall D, R: \mathbb{U}. \forall I: D \rightarrow \mathbb{P}. \forall O: D \times R \rightarrow \mathbb{P}.$

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

\Leftarrow

$\exists D', R': \mathbb{U}. \exists I': D' \rightarrow \mathbb{B}. \exists O': D' \times R' \rightarrow \mathbb{B}. \exists O_D: D \times D' \times D \rightarrow \mathbb{B}. \exists O_C: R' \times R \times R \rightarrow \mathbb{B}$

$\exists \text{primitive}: D \rightarrow \mathbb{B}. \exists \text{Directly-solve}: D \rightarrow R. \exists \text{Decompose}: D \rightarrow D' \times D.$

$\exists g: D' \rightarrow R'. \exists \text{Compose}: R' \times R \rightarrow R. \exists \succ: D \times D \rightarrow \mathbb{B}.$

FUNCTION $f_p(x:D):R$ WHERE $I(x) \wedge \text{primitive}(x)$ RETURNS y SUCH THAT $O(x,y)$
 $\equiv \text{Directly-solve}(x)$ korrekt

$\wedge \forall x, y, y', z, z', t. O_D(x, y', y) \wedge O(y, z) \wedge O'(y', z') \wedge O_C(z, z', t) \Rightarrow O(x, t)$

\wedge FUNCTION $F_d(x:D):D' \times D$ WHERE $I(x) \wedge \neg \text{primitive}(x)$ RETURNS y', y
 SUCH THAT $I'(y') \wedge I(y) \wedge x \succ y \wedge O_D(x, y', y) \equiv \text{Decompose}(x)$ korrekt

\wedge FUNCTION $F_c(z, z': R \times R'): R$ RETURNS t SUCH THAT $O_C(z, z', t)$
 $\equiv \text{Compose}(z, z')$ korrekt

\wedge FUNCTION $F_G(y': D'): R'$ WHERE $I'(y')$ RETURNS z' SUCH THAT $O'(y', z')$
 $\equiv g(y')$ korrekt

$\wedge \succ$ ist wohlfundierte Ordnung auf D

• Wende Synthesetheorem auf konkrete Probleme an

- Matching liefert Instanzen für D, R, I, O
- Bestimme Instanzen für D', R', \dots heuristisch und beweise Bedingungen
- Extrahiere Divide & Conquer Algorithmus aus dem Beweis

SYNTHESE VON `sort` MIT EINEM BEWEISSYSTEM

Formuliere Spezifikation als Beweisziel und wende D&C Theorem an

\vdash FUNCTION `sort`($L:\text{Seq}(\mathbb{Z})$): $\text{Seq}(\mathbb{Z})$ WHERE `true`
RETURNS S SUCH THAT `SORT(L,S)` erfüllbar

BY lemma 'Divide & Conquer Synthese'

1.

$\vdash \exists D', R': \mathbb{U}. \exists I': D' \rightarrow \mathbb{B}. \exists O': D' \times R' \rightarrow \mathbb{B}. \exists O_D: \text{Seq}(\mathbb{Z}) \times D' \times \text{Seq}(\mathbb{Z}) \rightarrow \mathbb{B}.$

$\exists O_C: R' \times \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z}) \rightarrow \mathbb{B}. \exists \text{primitive}: \text{Seq}(\mathbb{Z}) \rightarrow \mathbb{B}.$

$\exists \text{Directly-solve}: \text{Seq}(\mathbb{Z}) \rightarrow \text{Seq}(\mathbb{Z}). \exists \text{Decompose}: \text{Seq}(\mathbb{Z}) \rightarrow D' \times \text{Seq}(\mathbb{Z}).$

$\exists g: D' \rightarrow R'. \exists \text{Compose}: R' \times \text{Seq}(\mathbb{Z}) \rightarrow \text{Seq}(\mathbb{Z}). \exists \succ: \text{Seq}(\mathbb{Z}) \times \text{Seq}(\mathbb{Z}) \rightarrow \mathbb{B}.$

FUNCTION $f_p(L:\text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z})$ WHERE `primitive(L)` RETURNS S

SUCH THAT `SORT(L,S) \equiv Directly-solve(L)` korrekt

$\wedge \forall L, L', y, z, z', t. O_D(L, y, L') \wedge \text{SORT}(L', z') \wedge O'(y, z) \wedge O_C(z, z', t)$
 $\Rightarrow \text{SORT}(L, t)$

\wedge FUNCTION $F_d(L:\text{Seq}(\mathbb{Z})):\text{Seq}(\mathbb{Z})$ WHERE $\neg \text{primitive}(L)$ RETURNS y, L'
SUCH THAT $I'(y) \wedge L \succ L' \wedge O_D(L, y, L') \equiv \text{Decompose}(L)$ korrekt

\wedge FUNCTION $F_c(z, z': \text{Seq}(\mathbb{Z}) \times R'):\text{Seq}(\mathbb{Z})$ RETURNS t SUCH THAT $O_C(z, z', t)$
 $\equiv \text{Compose}(z, z')$ korrekt

\wedge FUNCTION $F_G(y:D'):R'$ WHERE $I'(y)$ RETURNS z' SUCH THAT $O'(y, z')$
 $\equiv g(y)$ korrekt

$\wedge \succ$ ist wohlfundierte Ordnung auf $\text{Seq}(\mathbb{Z})$

SYNTHESE VON `sort` MIT EINEM BEWEISSYSTEM (2)

Bestimme Instanzen für Parameter (mit D&C Metalevel Strategie)

1.

$\vdash \exists D', R' : \mathbb{U}. \exists I' : D' \rightarrow \mathbb{B}. \exists O' : D' \times R' \rightarrow \mathbb{B}. \exists O_D : \text{Seq}(\mathbb{Z}) \times D' \times \text{Seq}(\mathbb{Z}) \rightarrow \mathbb{B} \dots$

BY `exIon` [$\mathbb{Z}; \mathbb{Z}; \lambda y. \text{true}; \lambda y, y'. y = y'; \lambda L, y, L'. L = y.L'$;
 $\lambda z, z', t. \text{ordered}(z) \Rightarrow \text{SORT}(z.z', t)$; $\lambda L. L = []$;
 $\lambda L. []$; `HdTl` ; `id` ; `ordered_insert` ; $\lambda L, L'. |L| > |L'|$]

2.

\vdash FUNCTION `fp`(`L`:`Seq`(`Z`)):`Seq`(`Z`) WHERE `L` = `[]` RETURNS `S`

SUCH THAT `SORT`(`L`,`S`) \equiv `[]` korrekt

$\wedge \forall L, L', y, z, z', t. L = y.L' \wedge \text{SORT}(L', z') \wedge y = z \wedge (\text{ordered}(z) \Rightarrow \text{SORT}(z.z', t))$
 $\Rightarrow \text{SORT}(L, t)$

\wedge FUNCTION `Fd`(`L`:`Seq`(`Z`)):`Z` × `Seq`(`Z`) WHERE `L` \neq `[]` RETURNS `y`,`L'`

SUCH THAT $|L| > |L'| \wedge L = y.L' \equiv (\text{hd}(L), \text{tl}(L))$ korrekt

\wedge FUNCTION `Fc`(`z`,`z'`:`Seq`(`Z`) × `Z`):`Seq`(`Z`) RETURNS `t`

SUCH THAT `ordered`(`z`) \Rightarrow `SORT`(`z.z'`,`t`) \equiv `ordered_insert`(`z`,`z'`) korrekt

\wedge FUNCTION `FG`(`y`:`Z`):`Z` WHERE `true` RETURNS `z` SUCH THAT `y` = `z` \equiv `y` korrekt

$\wedge \lambda L, L'. |L| > |L'|$ ist wohlfundierte Ordnung auf `Seq`(`Z`)

BY `Repeat` and `R` THEN ...

Ergibt “formal verifizierte” Anwendung der D&C Synthesestrategie

DIVIDE&CONQUER SYNTHESE IM RÜCKBLICK

- **Wissensbasierte Erzeugung effizienter Grundalgorithmen**

- Sinnvoll, wenn Problem rekursiv zerlegbar in Basisfall und Konjunktion von Bedingungen im Schrittfall
- Schematischer Algorithmus verwendet 5 Basiskomponenten
- Komponenten lassen sich mit formalisiertem Wissen über verwendete Datenstrukturen heuristisch bestimmen
- Bekannte Dekompositions-/Kompositionsarten von Standard-Datenstrukturen müssen in Wissensbank vorgespeichert sein
- Heuristiken stützen sich auf Vorwärts- und Rückwärtsinferenz (Rewriting)

- **Erfolgreich in der Praxis**

- Schematische Algorithmen lassen sich in wenigen Schritten generieren
- Nachträgliche Optimierungen möglich durch formale Vereinfachungen
- Korrektheit der Algorithmen ist durch theoretische Vorarbeiten garantiert
- Korrektheit der **implementierten** Methode kann durch Integration in Beweissysteme gesichert werden