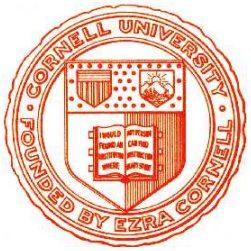
Automatisierte Logik und Programmierung

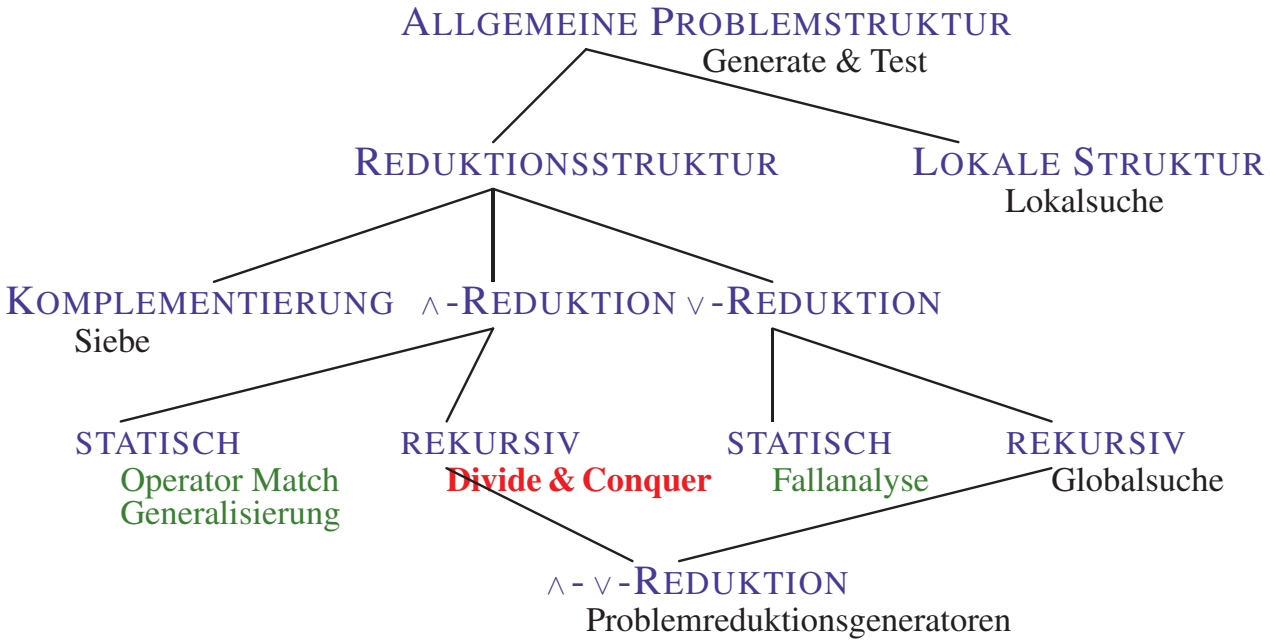
Einheit 19



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 Programmiertechnik
  - **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(ℤ)):ℤ WHERE L≠[]
    SUCH THAT  m∈L ∧ ∀x∈L. x≤m
≡ if|L|=1 then hd(L)
            else leta.L'=L
                 in letm'=maxL(L')
                     inif a<m' then m' else a
```

Einfache (primitive) Eingaben (`|L|=1`) erhalten direkte Lösung `hd(L)`

Andernfalls Dekomposition der Eingabe mit `HdTl: L ↦ a.L'`

Einfache Teillösung für `a` `a=id(a)`

Rekursive Lösung für `L'` `m'=maxL(L')`

Komposition der Teillösungen `a` und `m'` mit der Funktion `max:ℤ×ℤ→ℤ`

- **Vereinheitlichung durch separierte Beschreibung**

```
FUNCTION maxL(L:Seq(ℤ)):ℤ WHERE L≠[]
   RETURNS m SUCH THAT m∈L ∧ ∀x∈L. x≤m
≡ if |L|=1 then hd(L) else (max ∘ (id×maxL) ∘ 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*: $D \to D' \times D$ Aufspalten der Eingabe in Teilprobleme
  - Rekursiver Aufruf von f zusammen mit Hilfsfunktion g: $D' \to R'$
    · Funktionsprodukt $g \times f\,(x,y) = (g(x),\ f(y))$
  - *Compose*: $R' \times R \to R$ Zusammensetzen der Teillösungen
  - *Directly-solve*: $D \to R$: Direkte Lösung für einfache Teilprobleme
  - *primitive*: $D \to \mathbb{B}$: 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 $\mathit{primitive}[x]$ then $\mathit{Directly\text{-}solve}[x]$ else $(\mathit{Compose} \circ g \times f \circ \mathit{Decompose})\,(x)$

ist korrekt, wenn 6 Axiome erfüllt sind

---

1. Direkte Lösung korrekt für primitive Eingaben

FUNCTION $f_p(x{:}D){:}R$ WHERE $I[x] \wedge \mathit{primitive}[x]$ 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 $I[x] \wedge \neg \mathit{primitive}[x]$
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: Korrektheitsbeweis

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

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

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

  Primitive Eingaben: $I[x] \wedge \mathit{primitive}[x]$

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

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

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

  – $\mathit{Decompose}[x]$ 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

  – $\mathit{Compose}[z', z]$ 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*: $D \to R$ Axiom 1
  - *primitive*: $D \to \mathbb{B}$
  - *Decompose*: $D \to D' \times D$ Axiom 3
  - $\boldsymbol{g}$: $D' \to R'$ Axiom 4
  - *Compose*: $R' \times R \to R$ 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 minL(L:Seq(ℤ)):ℤ WHERE L≠[] RETURNS m SUCH THAT m∈L ∧ ∀x∈L. m≤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* ≡ `HdTl: Seq(α)→α×Seq(α)` (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: $L \succ L' \equiv |L| > |L'|$ ↦ **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 $g \times$`minL` angewandt werden
und *Compose* muß auf Resultat von $g \times$`minL` angewandt werden

– Da *Compose* ohnehin synthetisiert wird, ist "nichts tun" die beste Wahl

Dies liefert weitere 4 Komponenten

· $g \equiv id{:}\alpha \to \alpha$ (instantiiert mit $\alpha \equiv \mathbb{Z}$)

· Eingabebedingung $I'[a] \equiv$ `true`

· Extra-Bildbereich $R' \equiv \mathbb{Z}$

· Ausgabebedingung $O'[a, a'] \equiv a'{=}a$ ↦ **Axiom 4**

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

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

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

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

– Transformiere Formel in hinreichende Bedingung an L

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

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

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

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

Dies liefert die Komponente *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**

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

– Formel muß in Bedingung an m transformiert werden

· Einsetzen liefert

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

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

Vereinfachen für beide Fälle liefert

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

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

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

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

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

und $\mapsto$ **Axiom 2**

Synthese eines Divide & Conquer Algorithmus
Minimum einer nichtleeren Liste von Zahlen (5)

5. Erneute Synthese liefert Algorithmus für *Compose* ↦ Axiom 5

FUNCTION f_c(a',m':$\mathbb{Z}\times\mathbb{Z}$):$\mathbb{Z}$ WHERE true RETURNS m
SUCH THAT m=a' ∧ a'≤m' ∨ m=m' ∧ m≤a'
≡ *Compose*[a',m'] *ist korrekt*

– Disjunktion in O_C legt Algorithmenschema Fallunterscheidung nahe

Wähle I_1(a',m') ≡ a'≤m' und I_2(a',m') ≡ m'≤a'

Vereinfachung unter diesen Eingabebedingungen liefert Spezifikationen

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

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

– Lösung ist in beiden Fällen trivial

– Synthese durch Fallanalyse liefert die Komponente

Compose[a',m'] ≡ if a'≤m' then a' else m' und ↦ Axiom 5

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:Seq(ℤ)):ℤ WHERE |L|=1 RETURNS m
  SUCH THAT m∈L ∧ ∀x∈L. m≤x
≡ Directly-solve[L]                                   ist korrekt
```

– Vereinfache `m∈L ∧ ∀x∈L. m≤x` im Kontext `|L|=1`

· Es gilt `|L|=1 ⇒ L=[hd(L)]` sowie `m∈[hd(L)] ⇔ m=hd(L)`
und `∀x∈[hd(L)].m≤x ⇔ m≤hd(L)`

– Liefert Komponente *Directly-solve(L)* ≡ `hd(L)` und ↦ **Axiom 1**

7. Alle 12 Komponenten sind bestimmt, alle Axiome geprüft

– Instantiiere das Divide & Conquer Schema für `minL`

```
FUNCTION minL(L:Seq(ℤ)):ℤ WHERE L≠[] SUCH THAT  m∈L ∧ ∀x∈L. m≤x
≡ if |L|=1 then hd(L)                                  (Directly-solve)
            else let a.L'=L in                         (Decompose)
                 let m'= minL(L') in                   (Recursive call)
                   if a'≤m' then a' else m'            (Compose)
```

Alle durchgeführten Schritte sind automatisierbar

Programmierwissen für Divide & Conquer

- **Programmsynthese benötigt formalisiertes Wissen**
  - Verfahren könen 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** $\mapsto$ *Decompose*,O_D,D'
  - Endliche Listen/Folgen (`Seq(`α`)`): `HdTl`, `FirstLast`, `ListSplit`
    z.B. `ListSplit(L)` $\equiv$ `([`L_i`|i`$\in$`[1..|L|÷2]],[`L_i`|i`$\in$`[1+|L|÷2..|L|]])`
    mit $O_D[L, L_1, L_2] \equiv L{=}L_1{\circ}L_2$ und $D' \equiv \mathrm{Seq}(\alpha)$
  - Endliche Mengen (`Set(`α`)`): `ArbRest`
  - Datencontainer wie Produkträume: Zerlegung in Einzelkomponenten
  - Natürlichen Zahlen ($\mathbb{N}$): Vorgängerfunktion
  - $\vdots$
- **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 $I'[y']{:=}\texttt{true}$

    Verlagert einen Teil der Aufgaben von g in Kompoitionsfunktion
- **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 $(\mathit{Compose} \circ g \times f \circ \mathit{Decompose})(x)$

- **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** $\mapsto$ *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(ℤ)):Seq(ℤ) WHERE true`
`RETURNS S SUCH THAT rearranges(L,S) ∧ ordered(S)`

1. Wähle *Compose* ≡ `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

· $g \equiv id{:}\alpha{\to}\alpha$ (instantiiert mit $\alpha \equiv \mathbb{Z}$)

· Eingabebedingung $I'[a] \equiv$ `true`

· Extra-Domäne $D' \equiv \mathrm{Seq}(\mathbb{Z})$

· Ausgabebedingung $O'[a, a'] \equiv a'{=}a$

Synthese eines Sortieralgorithmus
Verwendung der umgekehrten Divide & Conquer Synthese (2)

3. Konstruiere Ausgabebedingung O_D mit Axiom 2

O_D[L,a,L'] ∧ a=a' ∧ rearranges(L',S') ∧ ordered(S') ∧ S=a'.S'
⇒ rearranges(L,S) ∧ ordered(S)

– Formel muß in Bedingungen an L,a,L' transformiert werden
  · Einsetzen liefert O_D[L,a,L'] ∧ rearranges(L',S') ∧ ordered(S')
    ⇒ rearranges(L,a.S') ∧ ordered(a.S')
  · Es gilt ordered(S') ⇒ ordered(a.S') ⇔ ∀x∈S. a≤x
    und rearranges(L',S') ⇒ rearranges(L,a.S') ⇔ rearranges(L,a.L')
    und rearranges(L,a.L') ⇔ a∈L ∧ rearranges(L',L-a)

– Liefert O_D[L,a,L'] ≡ a∈L ∧ ∀x∈L. a≤x ∧ rearranges(L',L-a)

4. Konstruiere *Decompose* und ≻ mit Axiom 3

– Spezifikation O_D wird erfüllt durch a=minL(L) und L'=L-a (Folie 7)

– Liefert als Lösung *Decompose*[L] ≡ let a=minL(L) in (a,L-a)

– Wegen |L|>|L-a| wähle wohlfundierte Ordnung L≻L' ≡ |L|>|L'|

Synthese eines Sortieralgorithmus
Verwendung der umgekehrten Divide & Conquer Synthese (3)

5. Konstruiere *primitive* mit Axiom 3

```
FUNCTION f_d(L:Seq(ℤ)):ℤ×Seq(ℤ) WHERE ¬primitive[L] RETURNS a,L'
  SUCH THAT |L|>|L'| ∧ a∈L ∧ ∀x∈L. a≤x ∧ rearranges(L',L-a)
≡ let a=minL(L) in (a,L-a)                                  ist korrekt
```

– Vorbedingung für Anwendung `minL` ist $L \neq$ `[]`
unter dieser Vorbedingung sind alle Ausgabebedingungen erfüllt

– Liefert *primitive*$[L] \equiv L$=`[]`

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

```
FUNCTION f_p(L:Seq(ℤ)):Seq(ℤ) WHERE L=[] RETURNS S
  SUCH THAT rearranges(L,S) ∧ ordered(S)
≡ Directly-solve[L]                                         ist korrekt
```

– Liefert *Directly-solve*[L] ≡ `[]`

7. Instantiiere das Divide & Conquer Schema

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) WHERE true
  RETURNS S SUCH THAT rearranges(L,S) ∧ ordered(S)
≡ if L=[] then [] else let a=minL(L) in a.sort(L-a)
```

Eine komplexere Divide & Conquer Synthese

Spezifikation: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) WHERE true`
`RETURNS S SUCH THAT rearranges(L,S) ∧ ordered(S)`

1. Wähle *Decompose* ≡ `ListSplit`

– Dies liefert zwei weitere Komponenten

· Extra-Domäne: D' ≡ Seq(ℤ)

· Ausgabebedingung: $O_D[\mathtt{L},\mathtt{L}_1,\mathtt{L}_2] \equiv \mathtt{L}{=}\mathtt{L}_1{\circ}\mathtt{L}_2$

· Passend dazu die wohlfundierte Ordnung $\mathtt{L}{\succ}\mathtt{L'} \equiv \mathtt{|L|}{>}\mathtt{|L'|}$

2. Konstruiere Hilfsfunktion *g* heuristisch

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

– Dies liefert weitere 4 Komponenten

· $g \equiv \mathtt{sort}{:}\mathbb{Z}{\to}\mathbb{Z}$ (instantiiert mit $\alpha \equiv \mathbb{Z}$)

· Eingabebedingung $I'[\mathtt{L}] \equiv \mathtt{true}$

· Extra-Domäne D' ≡ Seq(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($\mathbb{Z}$)×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 ↦ *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(ℤ)):Seq(ℤ) WHERE |L|≤1
RETURNS S SUCH THAT rearranges(L,S) ∧ ordered(S)
≡ *Directly-solve*[L] *ist korrekt*

– Aus |L|≤1 ∧ rearranges(L,S) folgt |S|≤1
Aus |S|≤1 folgt ordered(S)
– Zu konstruieren bleibt nur ein S mit rearranges(L,S)

Liefert *Directly-solve*[L] ≡ L

6. Alle Komponenten sind bestimmt, alle Axiome geprüft

– Instantiiere das Divide & Conquer Schema für sort

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) WHERE true
 RETURNS S SUCH THAT rearranges(L,S) ∧ ordered(S)
 ≡ if |L|≤1 then L                                        (Directly-solve)
     else let L1,L2 = ( [Li|i∈[1..|L|÷2]], [Li|i∈[1+|L|÷2..|L|]] )
                                                          (Decompose)
          in  merge(sort(L1),sort(L2))                    (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:Seq($\mathbb{Z}$)$\times$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$)

---

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

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

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

$O_D[S_1,S_2,a',S_1',S_2'] \wedge a{=}a' \wedge \mathtt{MERGE}(S_1',S_2',S') \wedge S{=}a.S' \Rightarrow \mathtt{MERGE}(S_1,S_2,S)$

liefert als Spezifikation und Lösung für *Decompose*

FUNCTION f_d(S_1,S_2:Seq($\mathbb{Z}$)$\times$Seq($\mathbb{Z}$)):$\mathbb{Z}\times$Seq($\mathbb{Z}$)$\times$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$)
$\wedge$ $\forall x{\in}S_1'\circ S_2'\,.\,a{\leq}x$ $\wedge$ $(S_1,S_2){\succ}(S_1',S_2')$
$\equiv$ let $x_1.S_1'{=}S_1$, $x_2.S_2'{=}S_2$ in if $x_1{\leq}x_2$ then (x_1,S_1',S_2) else (x_2,S_1,S_2')

(Lösung durch Fallanalyse: a' muß Minimum von hd(S_1) und hd(S_2) sein)

Divide & Conquer Synthese der merge-Funktion (2)

```
FUNCTION merge(S1,S2:Seq(ℤ)×Seq(ℤ)):Seq(ℤ)
    WHERE ordered(S1) ∧ ordered(S2)
    RETURNS S SUCH THAT ordered(S) ∧ rearranges(S, S1∘S2)
```

---

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

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

```
FUNCTION fp(S1,S2:Seq(ℤ)×Seq(ℤ)):Seq(ℤ)
   WHERE ordered(S1) ∧ ordered(S2) ∧ (S1=[] ∨ S2=[])
   RETURNS S SUCH THAT ordered(S) ∧ rearranges(S, S1∘S2)
= if S1=[] then S2 else S1
```

6. Instantiierter Divide & Conquer Algorithmus

```
FUNCTION merge(S1,S2:Seq(ℤ)×Seq(ℤ)):Seq(ℤ)
   WHERE ordered(S1) ∧ ordered(S2)
   RETURNS S SUCH THAT ordered(S) ∧ rearranges(S, S1∘S2)
= if S1=[] then S2                                          (Directly-solve)
     elseif S2=[] then S1                                   (Directly-solve)
       else let x1.S1'=S1 and x2.S2'=S2                     (Decompose)
            in if x1≤x2 then x1.merge(S1',S2) else x2.merge(S1,S2')
                                   (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*$[S_1, S_2] \equiv S_1 \circ S_2$, $g \equiv$ `sort`
  - *primitive*`[L]` $\equiv$ `|L|`≤ 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

Integration der Methodik in Beweisumgebung

- **Formuliere und beweise D&C-Synthesetheorem**

$\forall D,R:\mathbb{U}.\ \forall I:D\to\mathbb{P}.\ \forall O:D\times R\to\mathbb{P}.$
FUNCTION f(x:D):R WHERE I(x) RETURNS y SUCH THAT O(x,y) erfüllbar
$\Leftarrow$
$\exists D',R':\mathbb{U}.\ \exists I':D'\to\mathbb{B}.\ \exists O':D'\times R'\to\mathbb{B}.\ \exists O_D:D\times D'\times D\to\mathbb{B}.\ \exists O_C:R'\times R\times R\to\mathbb{B}$
$\exists$primitive:D$\to\mathbb{B}$. $\exists$Directly-solve:D$\to$R. $\exists$Decompose:D$\to$D'$\times$D.
$\exists g:D'\to R'.\ \exists$Compose:R'$\times$R$\to$R. $\exists\succ:D\times D\to\mathbb{B}$.
FUNCTION f_p(x:D):R WHERE I(x)$\wedge$primitive(x) RETURNS y SUCH THAT O(x,y)
$\equiv$ 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)$\wedge\neg$primitive(x) RETURNS y',y
SUCH THAT I'(y')$\wedge$I(y)$\wedge x\succ y\wedge O_D(x,y',y)\ \equiv$ Decompose(x) korrekt
$\wedge$ FUNCTION F_c(z,z':R$\times$R'):R RETURNS t SUCH THAT O_C(z,z',t)
$\equiv$ 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',... 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:Seq($\mathbb{Z}$)):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'$\to\mathbb{B}$. $\exists$O':D'$\times$R'$\to\mathbb{B}$. $\exists O_D$:Seq($\mathbb{Z}$)$\times$D'$\times$Seq($\mathbb{Z}$)$\to\mathbb{B}$.
$\exists O_C$:R'$\times$Seq($\mathbb{Z}$)$\times$Seq($\mathbb{Z}$)$\to\mathbb{B}$. $\exists$primitive:Seq($\mathbb{Z}$)$\to\mathbb{B}$.
$\exists$Directly-solve:Seq($\mathbb{Z}$)$\to$Seq($\mathbb{Z}$). $\exists$Decompose:Seq($\mathbb{Z}$)$\to$D'$\times$Seq($\mathbb{Z}$).
$\exists$g:D'$\to$R'. $\exists$Compose:R'$\times$Seq($\mathbb{Z}$)$\to$Seq($\mathbb{Z}$). $\exists\succ$:Seq($\mathbb{Z}$)$\times$Seq($\mathbb{Z}$)$\to\mathbb{B}$.
FUNCTION f_p(L:Seq($\mathbb{Z}$)):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')$\wedge$SORT(L',z')$\wedge$O'(y,z)$\wedge O_C$(z,z',t)
$\Rightarrow$ SORT(L,t)
$\wedge$ FUNCTION F_d(L:Seq($\mathbb{Z}$)):D'$\times$Seq($\mathbb{Z}$) WHERE $\neg$primitive(L) RETURNS y,L'
SUCH THAT I'(y)$\wedge$L$\succ$L'$\wedge O_D$(L,y,L') $\equiv$ Decompose(L) korrekt
$\wedge$ FUNCTION F_c(z,z':Seq($\mathbb{Z}$)$\times$R'):Seq($\mathbb{Z}$) RETURNS t SUCH THAT O_C(z,z',t)
$\equiv$ 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 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'$\to\mathbb{B}$. $\exists$O':D'$\times$R'$\to\mathbb{B}$. $\exists$O$_D$:Seq($\mathbb{Z}$)$\times$D'$\times$Seq($\mathbb{Z}$)$\to\mathbb{B}$.....

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

2.

$\vdash$ FUNCTION f$_p$(L:Seq($\mathbb{Z}$)):Seq($\mathbb{Z}$) WHERE L=[] RETURNS S
SUCH THAT SORT(L,S) $\equiv$ [] korrekt
$\wedge$ $\forall$L,L',y,z,z',t. L=y.L'$\wedge$SORT(L',z')$\wedge$y=z$\wedge$(ordered(z)$\Rightarrow$SORT(z.z',t))
$\Rightarrow$ SORT(L,t)
$\wedge$ FUNCTION F$_d$(L:Seq($\mathbb{Z}$)):$\mathbb{Z}\times$Seq($\mathbb{Z}$) WHERE L$\neq$[] RETURNS y,L'
SUCH THAT |L|>|L'| $\wedge$L=y.L' $\equiv$ (hd(L),tl(L)) korrekt
$\wedge$ FUNCTION F$_c$(z,z':Seq($\mathbb{Z}$)$\times\mathbb{Z}$):Seq($\mathbb{Z}$) RETURNS t
SUCH THAT ordered(z)$\Rightarrow$SORT(z.z',t) $\equiv$ ordered_insert(z,z') korrekt
$\wedge$ FUNCTION F$_G$(y:$\mathbb{Z}$):$\mathbb{Z}$ WHERE true RETURNS z SUCH THAT y=z $\equiv$ y korrekt
$\wedge$ λL,L'.|L|>|L'| ist wohlfundierte Ordnung auf Seq($\mathbb{Z}$)

BY Repeat andR 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 formalisierem 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