

Automatisierte Logik und Programmierung II

Teil IV



Automatisierte Programmierung



- 1. Grundkonzepte, Paradigmen & Strategien**
- 2. Wissensbasierte Programmentwicklung**
 - Divide & Conquer, Globalsuche,
Lokalsuche, Problemreduktionsgeneratoren
- 3. Korrektheitserhaltende Optimierungen**

WOZU AUTOMATISIERTE PROGRAMMIERUNG?

- **Softwareproduktion hat viele Probleme**

- **Zeitaufwendig und teuer**

- Entwurf und Implementierung fokussiert auf **Modellierungs- und Programmiersprachen** anstatt auf Eigenschaften des Problembereichs
- Implementierung meist “von Hand” und ad hoc
- Einbeziehung der **Endanwender** zu spät

WOZU AUTOMATISIERTE PROGRAMMIERUNG?

- **Softwareproduktion hat viele Probleme**

- **Zeitaufwendig und teuer**

- Entwurf und Implementierung fokussiert auf **Modellierungs- und Programmiersprachen** anstatt auf Eigenschaften des Problembereichs
- Implementierung meist “von Hand” und ad hoc
- Einbeziehung der **Endanwender** zu spät

- **Zu viele Fehler im Endprodukt**

- **Logischer Zusammenhang** zur Aufgabenstellung **selten erkennbar**
- **Begründung für Korrektheit** des Programms fehlt meistens

WOZU AUTOMATISIERTE PROGRAMMIERUNG?

- **Softwareproduktion hat viele Probleme**

- **Zeitaufwendig und teuer**

- Entwurf und Implementierung fokussiert auf **Modellierungs- und Programmiersprachen** anstatt auf Eigenschaften des Problembereichs
- Implementierung meist “von Hand” und ad hoc
- Einbeziehung der **Endanwender** zu spät

- **Zu viele Fehler im Endprodukt**

- **Logischer Zusammenhang** zur Aufgabenstellung **selten erkennbar**
- **Begründung für Korrektheit** des Programms fehlt meistens

- **Logische Synthese von Programmen hilft**

- (Teil-)Automatisierung der Konstruktion von Algorithmen
- **Logisches Fundament erhöht Zuverlässigkeit** des erzeugten Programms
- **Automatisierung verringert Entwicklungszeit und -kosten** und ermöglicht **frühzeitige Validierung** durch Endanwender

PROGRAMMSYNTHESE: GRUNDSÄTZLICHES VORGEHEN

Erzeuge korrekte ausführbare Programme aus Spezifikationen

Erzeuge korrekte ausführbare Programme aus Spezifikationen

1. Erstellen einer formalen Spezifikation

- a) Benötigt Formalisierung des Anwendungsbereichs als **Objekttheorie**
 - Welche **Begriffe** werden benutzt und was bedeuten sie?
 - Welche **mathematischen Gesetze** gelten für diese Begriffe?} vgl. §15
- b) Präzisierung der Problemstellung in spezifischem Formalismus

Erzeuge korrekte ausführbare Programme aus Spezifikationen

1. Erstellen einer formalen Spezifikation

- a) Benötigt Formalisierung des Anwendungsbereichs als **Objekttheorie**
 - Welche **Begriffe** werden benutzt und was bedeuten sie?
 - Welche **mathematischen Gesetze** gelten für diese Begriffe?} vgl. §15
- b) Präzisierung der Problemstellung in spezifischem Formalismus

2. Entwurf eines läuffähigen, korrekten Algorithmus

- **Synthesestrategie** generiert Basisversion und Korrektheitsgarantien
- Synthesystem muß durch erfahrene Benutzer gesteuert werden

Erzeuge korrekte ausführbare Programme aus Spezifikationen

1. Erstellen einer formalen Spezifikation

- a) Benötigt Formalisierung des Anwendungsbereichs als **Objekttheorie**
 - Welche **Begriffe** werden benutzt und was bedeuten sie?
 - Welche **mathematischen Gesetze** gelten für diese Begriffe?} vgl. §15
- b) Präzisierung der Problemstellung in spezifischem Formalismus

2. Entwurf eines läuffähigen, korrekten Algorithmus

- **Synthesestrategie** generiert Basisversion und Korrektheitsgarantien
- Synthesystem muß durch erfahrene Benutzer gesteuert werden

3. Erzeugung eines effizienten, korrekten Programms

- a) Benutzergesteuerte **Optimierungstechniken** verbessern Algorithmus
- b) **Übertragung in Zielsprache** ermöglicht weitere Optimierungen
 - System garantiert jeweils die Korrektheit der Optimierungen

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2					

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2	3				

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2	3	-5			

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2	3	-5	1		

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2	3	-5	1	2	

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

Costas Array der Ordnung 6 und seine Differenzentafel

PROGRAMMSYNTHESE AM BEISPIEL: COSTAS-ARRAYS

Costas Array der Größe n :

- Permutation von $\{1..n\}$ ohne Duplikate in Zeilen der Differenzentafel
- Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

Costas Array der Ordnung 6 und seine Differenzentafel

Ziel: Berechnung aller Costas Arrays der Größe n

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

**Für $n \geq 1$ berechne alle Permutationen von $\{1..n\}$
ohne Duplikate in Zeilen der Differenzentafel**

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

**Für $n \geq 1$ berechne alle Permutationen von $\{1..n\}$
ohne Duplikate in Zeilen der Differenzentafel**

- **Formalisierung vorkommender Begriffe**

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

Für $n \geq 1$ berechne alle **Permutationen** von $\{1..n\}$
ohne Duplikate in Zeilen der Differenztafel

- **Formalisierung vorkommender Begriffe**

- $\text{perm}(p, S) \equiv \text{nodups}(p) \wedge \text{range}(p) = S$

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

**Für $n \geq 1$ berechne alle Permutationen von $\{1..n\}$
ohne Duplikate in Zeilen der Differenztafel**

- **Formalisierung vorkommender Begriffe**

- $\text{perm}(p, S) \equiv \text{nodups}(p) \wedge \text{range}(p) = S$
- $\text{nodups}(L) \equiv \forall i \neq j \leq |p|. L[i] \neq L[j]$

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

Für $n \geq 1$ berechne alle Permutationen von $\{1..n\}$
ohne Duplikate in **Zeilen der Differenztafel**

• Formalisierung vorkommender Begriffe

- $\text{perm}(p, S) \equiv \text{nodups}(p) \wedge \text{range}(p) = S$
- $\text{nodups}(L) \equiv \forall i \neq j \leq |p|. L[i] \neq L[j]$
- $\text{dtrow}(p, j) \equiv [p[i] - p[i+j] \mid i \in [1..length(p) - j]]$

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

Für $n \geq 1$ berechne alle **Permutationen** von $\{1..n\}$
ohne Duplikate in Zeilen der Differenztafel

- **Formalisierung vorkommender Begriffe**

- $\text{perm}(p, S) \equiv \text{nodups}(p) \wedge \text{range}(p) = S$
- $\text{nodups}(L) \equiv \forall i \neq j \leq |p|. L[i] \neq L[j]$
- $\text{dtrow}(p, j) \equiv [p[i] - p[i+j] \mid i \in [1..length(p) - j]]$

- **Aufstellen mathematischer Gesetze**

- Lemmas zu Eigenschaften von perm , nodups , dtrow

COSTAS-ARRAYS (1): FORMALE SPEZIFIKATION

Für $n \geq 1$ berechne alle **Permutationen** von $\{1..n\}$
ohne Duplikate in Zeilen der Differenztafel

- **Formalisierung vorkommender Begriffe**

- $\text{perm}(p, S) \equiv \text{nodups}(p) \wedge \text{range}(p) = S$
- $\text{nodups}(L) \equiv \forall i \neq j \leq |p|. L[i] \neq L[j]$
- $\text{dtrow}(p, j) \equiv [p[i] - p[i+j] \mid i \in [1.. \text{length}(p) - j]]$

- **Aufstellen mathematischer Gesetze**

- Lemmas zu Eigenschaften von perm , nodups , dtrow

- **Präzisierung der Problemstellung**

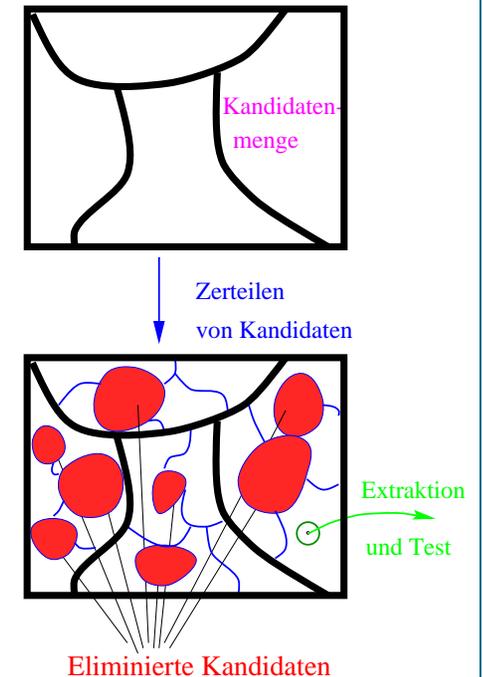
```
FUNCTION Costas (n:ℤ) WHERE n ≥ 1
  RETURNS {p:Seq(ℤ) | perm(p, {1..n})
            ∧ ∀j < |p|. nodups(dtrow(p, j))}
```

COSTAS-ARRAYS (2): ERZEUGUNG DES BASISALGORITHMUS

- **Algorithmische Struktur ist **Globalsuche****
Standardstruktur zum Durchsuchen von Lösungsräumen

COSTAS-ARRAYS (2): ERZEUGUNG DES BASISALGORITHMUS

- **Algorithmische Struktur ist **Globalsuche****
Standardstruktur zum Durchsuchen von Lösungsräumen
 - Codierung von Kandidatenmengen
 - Wiederholtes **Aufteilen** und **Filtern** auf Basis von Repräsentanten
 - **Extraktion** konkreter Lösungen aus Repräsentanten

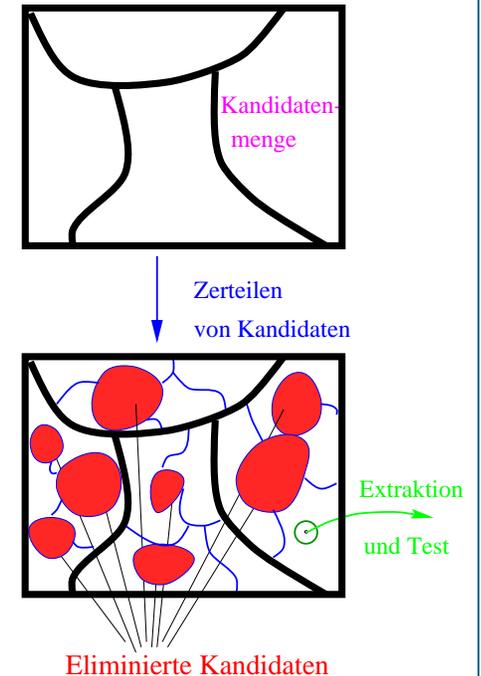


COSTAS-ARRAYS (2): ERZEUGUNG DES BASISALGORITHMUS

- **Algorithmische Struktur ist Globalsuche**
Standardstruktur zum Durchsuchen von Lösungsräumen
 - Codierung von Kandidatenmengen
 - Wiederholtes **Aufteilen** und **Filtern** auf Basis von Repräsentanten
 - **Extraktion** konkreter Lösungen aus Repräsentanten

- **Algorithmus wird aus Schema erzeugt**

```
FUNCTION Costas (n:ℤ) WHERE n ≥ 1
  RETURNS {p:Seq(ℤ) | perm(p, {1..n})
           ∧ ∀j < |p|. nodups(dtrow(p, j))}
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j < |s|. nodups(dtrow(s, j))
  = {p | p ∈ {s} ∧ perm(p, {1..n}) ∧ ∀j < n. nodups(dtrow(p, j))}
    ∪ ⋃ {aux(n, t) | t ∈ {s.i | i ∈ {1..n}}
        ∧ nodups(t) ∧ ∀j < |t|. nodups(dtrow(t, j))}
  in aux(n, [])
```



COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
  = { p | p ∈ {s} ∧ perm(p, {1..n}) ∧ ∀j<n.nodups(dtrow(p, j)) }
    ∪ ∪ { aux(n, t) | t ∈ {s.i | i ∈ {1..n}} ∧ nodups(t)
        ∧ ∀j<|t|.nodups(dtrow(t, j)) }
  in aux(n, [])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
  = { p | p ∈ {s} ∧ perm(p, {1..n}) ∧ ∀j<n.nodups(dtrow(p, j)) }
    ∪ ∪ { aux(n, t) | t ∈ {s.i | i ∈ {1..n}} ∧ nodups(t)
        ∧ ∀j<|t|.nodups(dtrow(t, j)) }
  in aux(n, [])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if perm(s,{1..n}) ∧ ∀j<n.nodups(dtrow(s,j)) then {s} else ∅
    ∪ ⋃{aux(n,t) | t∈{s.i | i∈{1..n}} ∧ nodups(t)
      ∧ ∀j<|t|.nodups(dtrow(t,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if perm(s,{1..n}) ∧ ∀j<n.nodups(dtrow(s,j)) then {s} else ∅
    ∪ ⋃{aux(n,t) | t∈{s.i|i∈{1..n}} ∧ nodups(t)
      ∧ ∀j<|t|.nodups(dtrow(t,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if perm(s,{1..n}) then {s} else ∅
    ∪ ⋃{aux(n,t) | t∈{s.i|i∈{1..n}} ∧ nodups(t)
      ∧ ∀j<|t|.nodups(dtrow(t,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if perm(s,{1..n}) then {s} else ∅
    ∪ ⋃{aux(n,t) | t∈{s.i|i∈{1..n}} ∧ nodups(t)
      ∧ ∀j<|t|.nodups(dtrow(t,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ∪{aux(n,t) | t∈{s.i | i∈{1..n}} ∧ nodups(t)
      ∧ ∀j<|t|.nodups(dtrow(t,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ⋃{aux(n,t) | t∈{s·i | i∈{1..n}} ∧ nodups(t)
      ∧ ∀j<|t|.nodups(dtrow(t,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ⋃{aux(n,s·i) | i∈{1..n} ∧ nodups(s·i)
        ∧ ∀j<|s·i|.nodups(dtrow(s·i,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ∪{aux(n,s·i) | i∈{1..n} ∧ nodups(s·i)
      ∧ ∀j<|s·i|.nodups(dtrow(s·i,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ,s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s,j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ∪{aux(n,s·i) | i∈{1..n}\s
      ∧ ∀j<|s·i|.nodups(dtrow(s·i,j)) }
  in aux(n,[])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ∪{aux(n, s·i) | i∈{1..n}\s
      ∧ ∀j<|s·i|.nodups(dtrow(s·i, j)) }
  in aux(n, [])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

ALGORITHMISCH-LOGISCHE VEREINFACHUNG

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ∪{aux(n, s·i) | i∈{1..n}\s
        ∧ ∀j<|s|. (s[|s·i|-j]-i)∉dtrow(s, j)}
  in aux(n, [])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

INKREMENTELLE BERECHNUNG VON TEILAUSDRÜCKEN

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ⋃{aux(n, s·i) | i∈{1..n}\s
        ∧ ∀j<|s|. (s[|s·i|-j]-i) ∉ dtrow(s, j)}
  in aux(n, [])
```

COSTAS-ARRAYS (3): OPTIMIERUNG

INKREMENTELLE BERECHNUNG VON TEILAUSDRÜCKEN

```
FUNCTION Costas (n:ℤ) ...
= let rec aux (n:ℤ, s:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ⋃{aux(n, s·i) | i∈{1..n}\s
          ∧ ∀j<|s|. (s[|s·i|-j]-i) ∉ dtrow(s, j)}
  in aux(n, [])
```

Ersetze ineffiziente Neuberechnung $\{1..n\}\setminus s$ durch neue Variable

COSTAS-ARRAYS (3): OPTIMIERUNG

INKREMENTELLE BERECHNUNG VON TEILAUSDRÜCKEN

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
    ∧ pool={1..n}\s
  = if {1..n}\s=∅ then {s} else ∅
    ∪ ∪{aux(n, s·i, {1..n}\(s·i)) | i∈{1..n}\s
        ∧ ∀j<|s|. (s[|s·i|-j]-i) ∉ dtrow(s, j)}
  in aux(n, [], {1..n}\[])
```

Ergänze neue Variable `pool` zu Definition und Aufruf von `aux`

COSTAS-ARRAYS (3): OPTIMIERUNG

INKREMENTELLE BERECHNUNG VON TEILAUSTRÜCKEN

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ))
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
    ∧ pool={1..n}\s
  = if pool=∅ then {s} else ∅
    ∪ ∪{aux(n, s·i, pool\{i}) | i∈pool
        ∧ ∀j<|s|. (s[|s·i|-j]-i) ∉ dtrow(s, j)}
  in aux(n, [], {1..n})
```

Vereinfachte Algorithmus mit Vorbedingung $pool = \{1..n\} \setminus s$

COSTAS-ARRAYS (3): OPTIMIERUNG

INKREMENTELLE BERECHNUNG VON TEILAUSTRÜCKEN

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
    ∧ pool={1..n}\s ∧ ssize=|s|
  = if pool=∅ then {s} else ∅
    ∪ ∪{aux(n, s·i, pool\{i}, |s·i|) | i∈pool
        ∧ ∀j<|s|. (s[|s·i|-j]-i) ∉ dtrow(s, j)}
  in aux(n, [], {1..n}, |[[]|)
```

Ersetze ineffiziente Neuberechnung $|s|$ durch neue Variable $ssize$

COSTAS-ARRAYS (3): OPTIMIERUNG

INKREMENTELLE BERECHNUNG VON TEILAUSDRÜCKEN

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
    ∧ pool={1..n}\s ∧ ssize=|s|
  = if pool=∅ then {s} else ∅
    ∪ ∪{aux(n, s.i, pool\{i}, ssize+1) | i∈pool
        ∧ ∀j<ssize. (s[ssize+1-j]-i) ∉ dtrow(s, j)}
  in aux(n, [], {1..n}, 0)
```

Vereinfachte Algorithmus mit Vorbedingung $ssize = |s|$

COSTAS-ARRAYS (3): OPTIMIERUNG

FALLANALYSE

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j<|s|.nodups(dtrow(s, j))
    ∧ pool={1..n}\s ∧ ssize=|s|
  = if pool=∅ then {s} else ∅
    ∪ ⋃{aux(n, s.i, pool\{i}, ssize+1) | i∈pool
        ∧ ∀j<ssize. (s[ssize+1-j]-i)∉dtrow(s, j)}
  in aux(n, [], {1..n}, 0)
```

Anwendung der Distributivgesetze für if .. then .. else

COSTAS-ARRAYS (3): OPTIMIERUNG

FALLANALYSE

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j<|s|. nodups(dtrow(s, j))
    ∧ pool={1..n}\s ∧ ssize=|s|
  = if pool=∅
    then {s} ∪ ⋃{aux(n, s·i, pool\{i}, ssize+1) | i∈pool
      ∧ ∀j<ssize. (s[ssize+1-j]-i)∉dtrow(s, j)}
    else ∅ ∪ ⋃{aux(n, s·i, pool\{i}, ssize+1) | i∈pool
      ∧ ∀j<ssize. (s[ssize+1-j]-i)∉dtrow(s, j)}
  in aux(n, [], {1..n}, 0)
```

Vereinfachung im Kontext der Bedingung $pool = \emptyset$

COSTAS-ARRAYS (3): OPTIMIERUNG

FALLANALYSE

```
FUNCTION Costas (n:ℤ) ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j<|s|. nodups(dtrow(s, j))
    ∧ pool={1..n}\s ∧ ssize=|s|
  = if pool=∅
    then {s}
    else ⋃{aux(n, s·i, pool\{i}, ssize+1) | i∈pool
            ∧ ∀j<ssize. (s[ssize+1-j]-i) ∉ dtrow(s, j)}
  in aux(n, [], {1..n}, 0)
```

COSTAS-ARRAYS (3): OPTIMIERUNG

WAHL DER IMPLEMENTIERUNG ABSTRAKTER DATENTYPEN

```
FUNCTION Costas (n:ℤ) WHERE n ≥ 1 ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j < |s|. nodups(dtrow(s, j))
    ∧ pool = {1..n} \ s ∧ ssize = |s|
  = if pool = ∅ then {s}
    else ⋃ {aux(n, s · i, pool \ {i}, ssize + 1) | i ∈ pool
              ∧ ∀j < ssize. (s[ssize + 1 - j] - i) ∉ dtrow(s, j)}
in aux(n, [], {1..n}, 0)
```

COSTAS-ARRAYS (3): OPTIMIERUNG

WAHL DER IMPLEMENTIERUNG ABSTRAKTER DATENTYPEN

```
FUNCTION Costas (n:ℤ) WHERE n ≥ 1 ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j < |s|. nodups(dtrow(s, j))
    ∧ pool = {1..n} \ s ∧ ssize = |s|
  = if pool = ∅ then {s}
    else ⋃ {aux(n, s·i, pool \ {i}, ssize+1) | i ∈ pool
            ∧ ∀j < ssize. (s[ssize+1-j]-i) ∉ dtrow(s, j)}
  in aux(n, [], {1..n}, 0)
```

$n:\mathbb{Z}$

↳ Standardimplementierung positiver ganzer Zahlen

$ssize:\mathbb{Z}$

↳ Standardimplementierung positiver ganzer Zahlen

COSTAS-ARRAYS (3): OPTIMIERUNG

WAHL DER IMPLEMENTIERUNG ABSTRAKTER DATENTYPEN

```
FUNCTION Costas (n:ℤ) WHERE n ≥ 1 ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j < |s|. nodups(dtrow(s, j))
    ∧ pool = {1..n} \ s ∧ ssize = |s|
  = if pool = ∅ then {s}
    else ⋃ {aux(n, s.i, pool \ {i}, ssize + 1) | i ∈ pool
            ∧ ∀j < ssize. (s[ssize + 1 - j] - i) ∉ dtrow(s, j)}
in aux(n, [], {1..n}, 0)
```

$n:\mathbb{Z}$ \mapsto Standardimplementierung positiver ganzer Zahlen

$ssize:\mathbb{Z}$ \mapsto Standardimplementierung positiver ganzer Zahlen

$s:\text{Seq}(\mathbb{Z})$, Elemente werden hinten angehängt \mapsto umgekehrt verkettete Liste

$pool:\text{Set}(\mathbb{Z})$: Elemente werden aus fester Menge entnommen \mapsto Bitvektor

COSTAS-ARRAYS (3): OPTIMIERUNG

WAHL DER IMPLEMENTIERUNG ABSTRAKTER DATENTYPEN

```
FUNCTION Costas (n:ℤ) WHERE n ≥ 1 ...
= let rec aux(n:ℤ, s:Seq(ℤ), pool:Seq(ℤ), ssize:ℤ)
  WHERE nodups(s) ∧ ∀j < |s|. nodups(dtrow(s, j))
    ∧ pool = {1..n} \ s ∧ ssize = |s|
  = if pool = ∅ then {s}
    else ⋃ {aux(n, s.i, pool \ {i}, ssize + 1) | i ∈ pool
            ∧ ∀j < ssize. (s[ssize + 1 - j] - i) ∉ dtrow(s, j)}
in aux(n, [], {1..n}, 0)
```

$n:\mathbb{Z}$ \mapsto Standardimplementierung positiver ganzer Zahlen

$ssize:\mathbb{Z}$ \mapsto Standardimplementierung positiver ganzer Zahlen

$s:\text{Seq}(\mathbb{Z})$, Elemente werden hinten angehängt \mapsto umgekehrt verkettete Liste

$pool:\text{Set}(\mathbb{Z})$: Elemente werden aus fester Menge entnommen \mapsto Bitvektor

Letzter Schritt vor Übertragung in konkrete Programmiersprache

- **Anwendung generischer Inferenztechniken**
 - **Beweise als Programme**
 - Automatischer Beweiser + Programmextraktion aus Beweisen
 - **Transformation von Formeln**
 - Rewrite-Techniken + Extraktion von Programmen aus Formeln

- **Anwendung generischer Inferenztechniken**

- **Beweise als Programme**

- Automatischer Beweiser + Programmextraktion aus Beweisen

- **Transformation von Formeln**

- Rewrite-Techniken + Extraktion von Programmen aus Formeln

- Gut zur Illustration der Prinzipien (“Synthese im Kleinen”)

- Konstruktion aufwendigerer Algorithmen verlangt Spezialstrategien

- **Anwendung generischer Inferenztechniken**

- **Beweise als Programme**

- Automatischer Beweiser + Programmextraktion aus Beweisen

- **Transformation von Formeln**

- Rewrite-Techniken + Extraktion von Programmen aus Formeln

- Gut zur Illustration der Prinzipien (“Synthese im Kleinen”)

- Konstruktion aufwendigerer Algorithmen verlangt Spezialstrategien

- **Wissensbasierte Syntheseverfahren**

- Wissen über algorithmische Grundstrukturen formalisiert

- als “**Algorithmentheorien**” (Struktur + Korrektheitsaxiome)

- Strategien verwenden Wissen zur Erzeugung effizienter Algorithmen

● Anwendung generischer Inferenztechniken

– Beweise als Programme

- Automatischer Beweiser + Programmextraktion aus Beweisen

– Transformation von Formeln

- Rewrite-Techniken + Extraktion von Programmen aus Formeln

– Gut zur Illustration der Prinzipien (“Synthese im Kleinen”)

– Konstruktion aufwendigerer Algorithmen verlangt Spezialstrategien

● Wissensbasierte Syntheseverfahren

– Wissen über algorithmische Grundstrukturen formalisiert als “**Algorithmentheorien**” (Struktur + Korrektheitsaxiome)

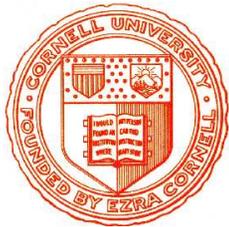
– Strategien verwenden Wissen zur Erzeugung effizienter Algorithmen

– Ziel ist Unterstützung des Programmierers (statt Ersetzung)

– Aufwendigere Vorarbeiten aber erfolgreich in der “Praxis”

Automatisierte Logik und Programmierung

Einheit 17



Syntheseparadigmen & -strategien



1. Grundkonzepte
2. Synthese im Kleinen
 - Beweise als Programme
 - Synthese durch Transformationen
3. Wissensbasierte Programmentwicklung

KOMPONENTEN EINER PROGRAMMSYNTHESE

Erzeuge korrekte ausführbare Programme aus Spezifikationen

KOMPONENTEN EINER PROGRAMMSYNTHESE

Erzeuge korrekte ausführbare Programme aus Spezifikationen

- **Formale Spezifikation als Ausgangspunkt**
 - Beschreibung von Anwendungsbereich und Problemstellung in (umfangreicher, “natürlicher”) formaler Spezifikationsprache
 - **Objekttheorie** formalisiert Eigenschaften neuer Anwendungskonzepte

KOMPONENTEN EINER PROGRAMMSYNTHESE

Erzeuge korrekte ausführbare Programme aus Spezifikationen

- **Formale Spezifikation als Ausgangspunkt**
 - Beschreibung von Anwendungsbereich und Problemstellung in (umfangreicher, “natürlicher”) formaler Spezifikationsprache
 - **Objekttheorie** formalisiert Eigenschaften neuer Anwendungskonzepte
- **Automatische Algorithmensynthese**
 - **Syntheseparadigmen**: grundsätzliche Vorgehensweisen
Theoretische Begründung der Korrektheit erzeugter Algorithmen
 - **Synthesestrategien**: Verfahren zur Steuerung der Synthese
Dokumentation getroffene Entscheidungen durch Trace der Strategie

KOMPONENTEN EINER PROGRAMMSYNTHESE

Erzeuge korrekte ausführbare Programme aus Spezifikationen

- **Formale Spezifikation als Ausgangspunkt**
 - Beschreibung von Anwendungsbereich und Problemstellung in (umfangreicher, “natürlicher”) formaler Spezifikationsprache
 - **Objekttheorie** formalisiert Eigenschaften neuer Anwendungskonzepte
- **Automatische Algorithmensynthese**
 - **Syntheseparadigmen**: grundsätzliche Vorgehensweisen
Theoretische Begründung der Korrektheit erzeugter Algorithmen
 - **Synthesestrategien**: Verfahren zur Steuerung der Synthese
Dokumentation getroffene Entscheidungen durch Trace der Strategie
- **Optimierung und Datentypverfeinerung**
 - Verbesserung des erzeugten Basisalgorithmus
 - Wahl geeigneter Implementierungen vorkommender Datentypen
 - Sprachabhängige Optimierung bei Übertragung in Programmiersprache

FORMALISMUS ZUR SPEZIFIKATION VON PROBLEMEN

- **Programme berechnen im Endeffekt Funktionen**

- Aus welchem Datentyp stammen die **Eingaben**? **Domain D**
- Zu welchem Datentyp gehören die **Ausgaben**? **Range R**
- Gibt es Beschränkungen an **zulässige Eingaben**? **Input-Bedingung I**
- Was ist der **Zusammenhang** zwischen Ein- und Ausgaben? **Output-Bedingung O**

● Programme berechnen im Endeffekt Funktionen

- Aus welchem Datentyp stammen die **Eingaben**? **Domain D**
- Zu welchem Datentyp gehören die **Ausgaben**? **Range R**
- Gibt es Beschränkungen an **zulässige Eingaben**? **Input-Bedingung I**
- Was ist der **Zusammenhang** zwischen Ein- und Ausgaben?
Output-Bedingung O
- Eine **formale Spezifikation** ist ein Quadrupel $spec = (D, R, I, O)$,
wobei D und R Datentypen, I Prädikat über D , O Prädikat über $D \times R$

FORMALISMUS ZUR SPEZIFIKATION VON PROBLEMEN

● Programme berechnen im Endeffekt Funktionen

- Aus welchem Datentyp stammen die Eingaben? Domain D
- Zu welchem Datentyp gehören die Ausgaben? Range R
- Gibt es Beschränkungen an zulässige Eingaben? Input-Bedingung I
- Was ist der Zusammenhang zwischen Ein- und Ausgaben?
Output-Bedingung O
- Eine **formale Spezifikation** ist ein Quadrupel $spec = (D, R, I, O)$,
wobei D und R Datentypen, I Prädikat über D , O Prädikat über $D \times R$

● Syntaktische Notation abhängig von Aufgabenstellung

Erhöhte Lesbarkeit durch Schlüsselwörter und instantiierte Form von I, O

- Bestimmung einer möglichen Lösung

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

- Bestimmung aller möglichen Lösungen

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

PROGRAMME UND KORREKTHEIT

- **Programm = Spezifikation + Algorithmus**
 - Algorithmen (**Programmkörper**) sind berechenbare (partielle) Funktionen auf $D \dashrightarrow R$, die auf allen zulässigen Eingaben definiert sind

PROGRAMME UND KORREKTHEIT

- **Programm = Spezifikation + Algorithmus**

- Algorithmen (**Programmkörper**) sind berechenbare (partielle) Funktionen auf $D \not\rightarrow R$, die auf allen zulässigen Eingaben definiert sind
- Eine **formales Programm** ist ein 5-Tupel $prog = (D, R, I, O, body)$ wobei (D, R, I, O) formale Spezifikation, $body: D \not\rightarrow R$ berechenbar
- Syntaktische Notation erlaubt Bezug auf Funktionsnamen in $body$

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

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

PROGRAMME UND KORREKTHEIT

● Programm = Spezifikation + Algorithmus

- Algorithmen (**Programmkörper**) sind berechenbare (partielle) Funktionen auf $D \not\rightarrow R$, die auf allen zulässigen Eingaben definiert sind
- Eine **formales Programm** ist ein 5-Tupel $prog = (D, R, I, O, body)$ wobei (D, R, I, O) formale Spezifikation, $body: D \not\rightarrow R$ berechenbar
- Syntaktische Notation erlaubt Bezug auf Funktionsnamen in $body$

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

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

● Korrektheit von Programmen

- **$prog$ ist korrekt**, falls $\forall x: D. I[x] \Rightarrow O[x, body(x)]$

PROGRAMME UND KORREKTHEIT

● Programm = Spezifikation + Algorithmus

- Algorithmen (**Programmkörper**) sind berechenbare (partielle) Funktionen auf $D \dashrightarrow R$, die auf allen zulässigen Eingaben definiert sind
- Eine **formales Programm** ist ein 5-Tupel $prog = (D, R, I, O, body)$ wobei (D, R, I, O) formale Spezifikation, $body: D \dashrightarrow R$ berechenbar
- Syntaktische Notation erlaubt Bezug auf Funktionsnamen in $body$

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

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

● Korrektheit von Programmen

- **prog ist korrekt**, falls $\forall x: D. I[x] \Rightarrow O[x, body(x)]$

● Syntheseziel: Erfüllbarkeit von Spezifikationen

- **spec ist erfüllbar** (synthetisierbar), falls es eine Funktion $body: D \dashrightarrow R$ gibt, so daß $prog = (spec, body)$ korrekt ist

ANSÄTZE ZUR PROGRAMMSYNTHESE

- **KI-Vision: Automatisches Programmieren**

- Intelligenter Agent ersetzt menschlichen Programmierer
- Ziel ist vollautomatische Erzeugung von Programmen aus Spezifikationen
- Strategien konzentrieren sich auf logisch-deduktive Verfahren

Forschungen lieferten wichtige theoretische Grundlagen

- **KI-Vision: Automatisches Programmieren**

- Intelligenter Agent ersetzt menschlichen Programmierer
- Ziel ist vollautomatische Erzeugung von Programmen aus Spezifikationen
- Strategien konzentrieren sich auf logisch-deduktive Verfahren
Forschungen lieferten wichtige theoretische Grundlagen
- Verfahren erzeugen oft nur einfache Algorithmen \mapsto **Synthese im Kleinen**

ANSÄTZE ZUR PROGRAMMSYNTHESE

- **KI-Vision: Automatisches Programmieren**

- Intelligenter Agent ersetzt menschlichen Programmierer
- Ziel ist vollautomatische Erzeugung von Programmen aus Spezifikationen
- Strategien konzentrieren sich auf logisch-deduktive Verfahren
Forschungen lieferten wichtige theoretische Grundlagen
- Verfahren erzeugen oft nur einfache Algorithmen \mapsto **Synthese im Kleinen**

- **Software-Engineering: Programmierunterstützung**

- Benutzergesteuerte Erzeugung von Programmen mit Korrektheitsgarantie
Synthesewerkzeug unterstützt den menschlichen Programmierer
- Strategien verwenden symbolisch-algebraische Techniken und theoretische Grundlagen aus der Deduktion als Fundament
Forschung liefert Formalisierung von Programmierwissen und -methoden

ANSÄTZE ZUR PROGRAMMSYNTHESE

● **KI-Vision: Automatisches Programmieren**

- Intelligenter Agent ersetzt menschlichen Programmierer
- Ziel ist vollautomatische Erzeugung von Programmen aus Spezifikationen
- Strategien konzentrieren sich auf logisch-deduktive Verfahren
Forschungen lieferten wichtige theoretische Grundlagen
- Verfahren erzeugen oft nur einfache Algorithmen \mapsto **Synthese im Kleinen**

● **Software-Engineering: Programmierunterstützung**

- Benutzergesteuerte Erzeugung von Programmen mit Korrektheitsgarantie
Synthesewerkzeug unterstützt den menschlichen Programmierer
- Strategien verwenden symbolisch-algebraische Techniken und theoretische Grundlagen aus der Deduktion als Fundament
Forschung liefert Formalisierung von Programmierwissen und -methoden
- Verfahren liefern praktisch wertvolle Algorithmen
 \mapsto **Wissensbasierte Programmentwicklung**

SYNTHESE IM KLEINEN

Anwendung generischer Inferenzmechanismen

Anwendung generischer Inferenzmechanismen

- **Beweise als Programme**
 - Extraktion aus konstruktivem Beweis eines Theorems
 - Fokus liegt auf automatischen Beweisverfahren

Anwendung generischer Inferenzmechanismen

- **Beweise als Programme**
 - Extraktion aus konstruktivem Beweis eines Theorems
 - Fokus liegt auf automatischen Beweisverfahren
- **Synthese durch Transformationen**
 - Äquivalenzumformungen in ausführbare, meist rekursive Form
 - Fokus liegt auf algebraisch motivierten Rewrite-Techniken

Anwendung generischer Inferenzmechanismen

- **Beweise als Programme**

- Extraktion aus konstruktivem Beweis eines Theorems
- Fokus liegt auf automatischen Beweisverfahren

- **Synthese durch Transformationen**

- Äquivalenzumformungen in ausführbare, meist rekursive Form
- Fokus liegt auf algebraisch motivierten Rewrite-Techniken

- **Unterschiedliche theoretische Fundamente**

- Problemrepräsentation, Lösungsmethodik, Extraktion des Algorithmus
- Gegenseitige Simulation prinzipiell möglich

Anwendung generischer Inferenzmechanismen

- **Beweise als Programme**

- Extraktion aus konstruktivem Beweis eines Theorems
- Fokus liegt auf automatischen Beweisverfahren

- **Synthese durch Transformationen**

- Äquivalenzumformungen in ausführbare, meist rekursive Form
- Fokus liegt auf algebraisch motivierten Rewrite-Techniken

- **Unterschiedliche theoretische Fundamente**

- Problemrepräsentation, Lösungsmethodik, Extraktion des Algorithmus
- Gegenseitige Simulation prinzipiell möglich

- **Nur gut zur Illustration von Prinzipien**

- Konstruktion aufwendiger Algorithmen verlangt Spezialstrategien

- **Beweise Erfüllbarkeit einer Spezifikation**

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

- Erzeuge **Spezifikationstheorem**: $\forall x:D. \exists y:R. I[x] \Rightarrow O[x,y]$
- Suche formalen Beweis in konstruktivem logischen Kalkül
- **Extrahiere** aus Beweis einen Algorithmus zur Berechnung von y aus x

BEWEISE ALS PROGRAMME

- **Beweise Erfüllbarkeit einer Spezifikation**

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

- Erzeuge **Spezifikationstheorem**: $\forall x:D. \exists y:R. I[x] \Rightarrow O[x,y]$
- Suche formalen Beweis in konstruktivem logischen Kalkül
- **Extrahiere** aus Beweis einen Algorithmus zur Berechnung von y aus x

- **Rechtfertigung durch Curry Howard Isomorphismus**

- **Konstruktiver Beweis** zeigt, wie Ausgabe aus Eingabe bestimmt wird
- Implizit enthaltenes **funktionales Programm** ist **garantiert korrekt**

BEWEISE ALS PROGRAMME

- **Beweise Erfüllbarkeit einer Spezifikation**

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

- Erzeuge **Spezifikationstheorem**: $\forall x:D. \exists y:R. I[x] \Rightarrow O[x,y]$
- Suche formalen Beweis in konstruktivem logischen Kalkül
- **Extrahiere** aus Beweis einen Algorithmus zur Berechnung von y aus x

- **Rechtfertigung durch Curry Howard Isomorphismus**

- **Konstruktiver Beweis** zeigt, wie Ausgabe aus Eingabe bestimmt wird
- Implizit enthaltenes **funktionales Programm** ist **garantiert korrekt**

- **Forschungsschwerpunkte**

- Ausdrucksstarke **Kalküle**
- Effiziente **Beweisstrategien** und **Beweisplaner** (Induktion)
- Effiziente **Extraktionsmechanismen** (Algorithmen ohne Beweisballast)

FORMALER BEWEIS: INTEGERQUADRATWURZEL

$\vdash \forall n:\mathbb{N}. \exists r:\mathbb{N}. r^2 \leq n < (r+1)^2$

BY allR

$n:\mathbb{N}$

$\vdash \exists r:\mathbb{N}. r^2 \leq n < (r+1)^2$

BY NatInd 1

.....basecase.....

$\vdash \exists r:\mathbb{N}. r^2 \leq 0 < (r+1)^2$

✓ BY existsR [0] THEN Auto

.....upcase.....

$i:\mathbb{N}^+, r:\mathbb{N}, r^2 \leq i-1 < (r+1)^2$

$\vdash \exists r:\mathbb{N}. r^2 \leq i < (r+1)^2$

BY Decide [(r+1)² ≤ i] THEN Auto

.....Case 1.....

$i:\mathbb{N}^+, r:\mathbb{N}, r^2 \leq i-1 < (r+1)^2, (r+1)^2 \leq i$

$\vdash \exists r:\mathbb{N}. r^2 \leq i < (r+1)^2$

✓ BY existsR [r+1] THEN Auto'

.....Case 2.....

$i:\mathbb{N}^+, r:\mathbb{N}, r^2 \leq i-1 < (r+1)^2, \neg((r+1)^2 \leq i)$

$\vdash \exists r:\mathbb{N}. r^2 \leq i < (r+1)^2$

✓ BY existsR [r] THEN Auto

BEWEISTERM: INTEGERQUADRATWURZEL

- **Mit Korrektheitsbeweisinformation**

```
let rec sqrt n
= if n=0 then <0, <Ax, Ax>>
  else let <r, %1> = sqrt (n-1)
        in if (r+1)2 ≤ n then <r+1, <Ax, Ax>i>
          else <r, <Ax, Ax>>
```

- **Nach Projektion und Optimierung**

```
let rec sqrt n
= if n=0 then 0
  else let r = sqrt (n-1)
        in if (r+1)2 ≤ n then r+1
          else r
```

$\lfloor \sqrt{n} \rfloor$ – SYNTHESE EINES EFFIZIENTEREN ALGORITHMUS DURCH EFFIZIENZORIENTIERTE BEWEISFÜHRUNG

$\vdash \forall n:\mathbb{N}. \exists r:\mathbb{N}. r^2 \leq n < (r+1)^2$

BY allR

$n:\mathbb{N}$

$\vdash \exists r:\mathbb{N}. r^2 \leq n < (r+1)^2$

BY NatInd4 1

.....basecase.....

$\vdash \exists r:\mathbb{N}. r^2 \leq 0 < (r+1)^2$

✓ BY existsR [0] THEN Auto

.....upcase.....

$i:\mathbb{N}, r:\mathbb{N}, r^2 \leq i \div 4 < (r+1)^2$

$\vdash \exists r:\mathbb{N}. r^2 \leq i < (r+1)^2$

BY Decide [((2*r)+1)² ≤ i] THEN Auto

.....Case 1.....

$i:\mathbb{N}, r:\mathbb{N}, r^2 \leq i \div 4 < (r+1)^2, ((2*r)+1)^2 \leq i$

$\vdash \exists r:\mathbb{N}. r^2 \leq i < (r+1)^2$

✓ BY existsR [(2*r)+1] THEN Auto'

.....Case 2.....

$i:\mathbb{N}, r:\mathbb{N}, r^2 \leq i \div 4 < (r+1)^2, \neg(((2*r)+1)^2 \leq i)$

$\vdash \exists r:\mathbb{N}. r^2 \leq i < (r+1)^2$

✓ BY existsR [2*r] THEN Auto

```
let rec sqrt n
= if n=0 then 0
  else let r = sqrt (n÷4)
        in if (2*r+1)2 ≤ n then 2*r+1
           else 2*r
```

- **Mathematisch elegante Konstruktionsmethode**

- Extrahiere Beweisterm t aus vollständigem Beweis für

$$\forall x : D . \exists y : R . I[x] \Rightarrow O[x, y]$$

- Konstruiere den Programmkörper $\lambda x . \text{fst}(t(x))$

- Optimiere durch Reduktion und Elimination überflüssiger Information

BEWEISE ALS PROGRAMME: EIGENSCHAFTEN

- **Mathematisch elegante Konstruktionsmethode**

- Extrahiere Beweisterm t aus vollständigem Beweis für

$$\forall x : D . \exists y : R . I[x] \Rightarrow O[x, y]$$

- Konstruiere den Programmkörper $\lambda x . \text{fst}(t(x))$

- Optimierte durch Reduktion und Elimination überflüssiger Information

- **Korrektheitsgarantien aber geringe Effizienz**

- Effizienz stark abhängig von Beweismethodik

- Gewisse Effizienzsteigerungen durch verbesserte Extraktionsverfahren

BEWEISE ALS PROGRAMME: EIGENSCHAFTEN

- **Mathematisch elegante Konstruktionsmethode**

- Extrahiere Beweisterm t aus vollständigem Beweis für

$$\forall x : D . \exists y : R . I[x] \Rightarrow O[x, y]$$

- Konstruiere den Programmkörper $\lambda x . \text{fst}(t(x))$

- Optimierte durch Reduktion und Elimination überflüssiger Information

- **Korrektheitsgarantien aber geringe Effizienz**

- Effizienz stark abhängig von Beweismethodik

- Gewisse Effizienzsteigerungen durch verbesserte Extraktionsverfahren

- **Beweisschritte zu atomar (Assemblerniveau)**

- Beweise für komplexe Programme interaktiv kaum durchführbar

- Automatisierung sehr schwierig:

- Analyse der Formel, Unifikation, Suche nach geeigneten Induktionen

BEWEISE ALS PROGRAMME: EIGENSCHAFTEN

- **Mathematisch elegante Konstruktionsmethode**

- Extrahiere Beweisterm t aus vollständigem Beweis für

$$\forall x : D . \exists y : R . I[x] \Rightarrow O[x, y]$$

- Konstruiere den Programmkörper $\lambda x . \text{fst}(t(x))$

- Optimierte durch Reduktion und Elimination überflüssiger Information

- **Korrektheitsgarantien aber geringe Effizienz**

- Effizienz stark abhängig von Beweismethodik

- Gewisse Effizienzsteigerungen durch verbesserte Extraktionsverfahren

- **Beweisschritte zu atomar (Assemblerniveau)**

- Beweise für komplexe Programme interaktiv kaum durchführbar

- Automatisierung sehr schwierig:

 - Analyse der Formel, Unifikation, Suche nach geeigneten Induktionen

Nur praktikabel in Kombination mit Definitionen und Spezialtaktiken

- **Transformiere in effektiv ausführbare Formel**
 - Spezifikation ist ineffektive Formel
 - Transformationen verbessern algorithmisches Verhalten der Formel
 - Vorwärtsschließen ohne konkret vorgegebenes Ziel

SYNTHESE DURCH TRANSFORMATION

- **Transformiere in effektiv ausführbare Formel**

- Spezifikation ist ineffektive Formel
- Transformationen verbessern algorithmisches Verhalten der Formel
- Vorwärtsschließen ohne konkret vorgegebenes Ziel

- **Verwende Spezifikation als (logisches) Initialprogramm**

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

- Transformiere Programmkörper in äquivalente Formel der Gestalt:

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

die außer f nur noch berechenbare Prädikate enthält

- Extrahiere Programm aus Formel oder interpretiere als Logik-Programm

SYNTHESE DURCH TRANSFORMATION

- **Transformiere in effektiv ausführbare Formel**

- Spezifikation ist ineffektive Formel
- Transformationen verbessern algorithmisches Verhalten der Formel
- Vorwärtsschließen ohne konkret vorgegebenes Ziel

- **Verwende Spezifikation als (logisches) Initialprogramm**

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

- Transformiere Programmkörper in äquivalente Formel der Gestalt:

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

die außer f nur noch berechenbare Prädikate enthält

- Extrahiere Programm aus Formel oder interpretiere als Logik-Programm

- **Forschungsschwerpunkte**

- Leistungsfähige Transformationsregeln
- Effiziente Rewrite Techniken und Heuristiken für Vorwärtsinferenz

- **Anwendung bedingter Ersetzungsregeln der Form**

$$\forall z:T. B[z] \Rightarrow (Q[z] \Leftrightarrow Q'[z])$$

“Ersetze Vorkommen von $Q[z]$ durch $Q'[z]$, falls $B[z]$ erfüllt ist”

– Regeln sind Äquivalenzen oder Verfeinerungen (Implikationen)

- **Anwendung bedingter Ersetzungsregeln der Form**

$$\forall z:T. B[z] \Rightarrow (Q[z] \Leftrightarrow Q'[z])$$

“*Ersetze Vorkommen von $Q[z]$ durch $Q'[z]$, falls $B[z]$ erfüllt ist*”

– Regeln sind Äquivalenzen oder Verfeinerungen (Implikationen)

- **Regeln ergeben sich aus**

– Lemmata der Wissensbasis, elementare Tautologien und Abstraktionen

– Dynamisch erzeugte Definitionen und Kombinationsformen

- **Anwendung bedingter Ersetzungsregeln der Form**

$$\forall z : T . B[z] \Rightarrow (Q[z] \Leftrightarrow Q'[z])$$

“*Ersetze Vorkommen von $Q[z]$ durch $Q'[z]$, falls $B[z]$ erfüllt ist*”

– Regeln sind Äquivalenzen oder Verfeinerungen (Implikationen)

- **Regeln ergeben sich aus**

– Lemmata der Wissensbasis, elementare Tautologien und Abstraktionen

– Dynamisch erzeugte Definitionen und Kombinationsformen

- **Mechanismus basiert auf Vorwärtsinferenz**

– Ziel ist bestimmte Struktur der Formel zu erreichen

– Starke heuristische Steuerung notwendig

- **Anwendung bedingter Ersetzungsregeln der Form**

$$\forall z : T . B[z] \Rightarrow (Q[z] \Leftrightarrow Q'[z])$$

“*Ersetze Vorkommen von $Q[z]$ durch $Q'[z]$, falls $B[z]$ erfüllt ist*”

- Regeln sind Äquivalenzen oder Verfeinerungen (Implikationen)

- **Regeln ergeben sich aus**

- Lemmata der Wissensbasis, elementare Tautologien und Abstraktionen

- Dynamisch erzeugte Definitionen und Kombinationsformen

- **Mechanismus basiert auf Vorwärtsinferenz**

- Ziel ist bestimmte Struktur der Formel zu erreichen

- Starke heuristische Steuerung notwendig

- **Gleichwertig zum Prinzip Beweise als Programme**

- Transformationen durch Beweise mit Gleichheitslemmata simulierbar

- Beweisregeln können als Rewrite-Regeln beschrieben werden

- **Nicht skalierend**: Suchraum explodiert bei nichttrivialen Problemen

- **Inferenzniveau zu niedrig** – ignoriert bekanntes Programmierwissen

Zielgerichtete Entwicklung guter Algorithmen

Zielgerichtete Entwicklung guter Algorithmen

- **Synthese im Kleinen ist zu allgemein**
 - Fokus auf Logik statt auf Programmierung
 - Steuerung durch “normale” Programmier kaum möglich
 - Keine echte Unterstützung bei der Entwicklung von Programmen

Zielgerichtete Entwicklung guter Algorithmen

- **Synthese im Kleinen ist zu allgemein**
 - Fokus auf Logik statt auf Programmierung
 - Steuerung durch “normale” Programmier kaum möglich
 - Keine echte Unterstützung bei der Entwicklung von Programmen
- **Programmiermethodik verwendet Wissen**
 - Welche grundsätzlichen Algorithmenstrukturen gibt es?
 - Welche Algorithmenstrukturen sind für ein Problem geeignet?

Zielgerichtete Entwicklung guter Algorithmen

- **Synthese im Kleinen ist zu allgemein**
 - Fokus auf Logik statt auf Programmierung
 - Steuerung durch “normale” Programmier kaum möglich
 - Keine echte Unterstützung bei der Entwicklung von Programmen
- **Programmiermethodik verwendet Wissen**
 - Welche grundsätzlichen Algorithmenstrukturen gibt es?
 - Welche Algorithmenstrukturen sind für ein Problem geeignet?
- **Synthese sollte formales Programmierwissen verarbeiten**
 - Umsetzung von Programmiermethodik in Entwurfsstrategien
 - Schematisierung von algorithmischer Grundstrukturen
 - Axiome für Korrektheit des schematischen Algorithmus

Zielgerichtete Entwicklung guter Algorithmen

- **Synthese im Kleinen ist zu allgemein**
 - Fokus auf Logik statt auf Programmierung
 - Steuerung durch “normale” Programmier kaum möglich
 - Keine echte Unterstützung bei der Entwicklung von Programmen
- **Programmiermethodik verwendet Wissen**
 - Welche grundsätzlichen Algorithmenstrukturen gibt es?
 - Welche Algorithmenstrukturen sind für ein Problem geeignet?
- **Synthese sollte formales Programmierwissen verarbeiten**
 - Umsetzung von Programmiermethodik in Entwurfsstrategien
 - Schematisierung von algorithmischer Grundstrukturen
 - Axiome für Korrektheit des schematischen Algorithmus

Aufwendige Theoretische Grundlagenarbeit entlastet Syntheseprozess

- **Erzeuge Algorithmen in einem Schritt**
 - Anpassung eines schematischen Algorithmus an eine Problemstellung
 - Aus historischer Sicht: Anwendung einer High-Level Transformation

WISSENSBASIERTE PROGRAMMENTWICKLUNG: GRUNDIDEE

- **Erzeuge Algorithmen in einem Schritt**

- Anpassung eines schematischen Algorithmus an eine Problemstellung
- Aus historischer Sicht: Anwendung einer High-Level Transformation

- **Vorgehen formalisiert systematische Programmierung**

Gegeben sei die Spezifikation

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

- **Erzeuge Algorithmen in einem Schritt**

- Anpassung eines schematischen Algorithmus an eine Problemstellung
- Aus historischer Sicht: Anwendung einer High-Level Transformation

- **Vorgehen formalisiert systematische Programmierung**

Gegeben sei die Spezifikation

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

- Wähle eine algorithmische Grundstruktur für die Lösung
- Bestimme Komponenten eines schematischen Grundalgorithmus
- Prüfe, ob Komponenten die Korrektheitsaxiome des Schemas erfüllen
- Instantiiere Algorithmenschema

- **Erzeuge Algorithmen in einem Schritt**

- Anpassung eines schematischen Algorithmus an eine Problemstellung
- Aus historischer Sicht: Anwendung einer High-Level Transformation

- **Vorgehen formalisiert systematische Programmierung**

Gegeben sei die Spezifikation

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

- Wähle eine algorithmische Grundstruktur für die Lösung
- Bestimme Komponenten eines schematischen Grundalgorithmus
- Prüfe, ob Komponenten die Korrektheitsaxiome des Schemas erfüllen
- Instantiiere Algorithmenschema

- **Forschungsschwerpunkte**

- Analyse der allgemeinen Struktur einer Klasse von Algorithmen
- Schematisierung durch Komponenten und Korrektheitsaxiome
- Techniken zur Verfeinerung von Standardstrukturen

GLOBALSUCH SYNTHESE: COSTAS-ARRAYS PROBLEM

- **Problemspezifikation**

FUNCTION *Costas* ($n:\mathbb{Z}$) WHERE $n \geq 1$

RETURNS $\{p:\text{Seq}(\mathbb{Z}) \mid \text{perm}(p, \{1..n\}) \wedge \forall j < |p|. \text{nodups}(\text{dtrow}(p, j))\}$

GLOBAL SUCH SYNTHESE: COSTAS-ARRAYS PROBLEM

• Problemspezifikation

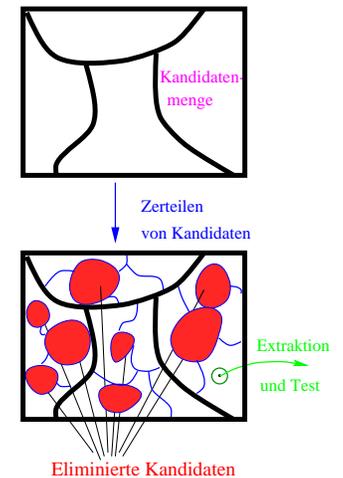
FUNCTION *Costas* ($n:\mathbb{Z}$) WHERE $n \geq 1$

RETURNS $\{p:\text{Seq}(\mathbb{Z}) \mid \text{perm}(p, \{1..n\}) \wedge \forall j < |p|. \text{nodups}(\text{dtrow}(p, j))\}$

• Grundstruktur von Globalsuch-Algorithmen

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

$\equiv \text{let rec } f_{gs}(x, s) = \{z \mid z \in \text{ext}[s] \wedge O[x, z]\} \cup \bigcup \{f_{gs}(x, t) \mid t \in \text{split}[x, s] \wedge \Phi[x, t]\}$
in $f_{gs}(x, s_0(x))$



GLOBALSUCH SYNTHESE: COSTAS-ARRAYS PROBLEM

• Problemspezifikation

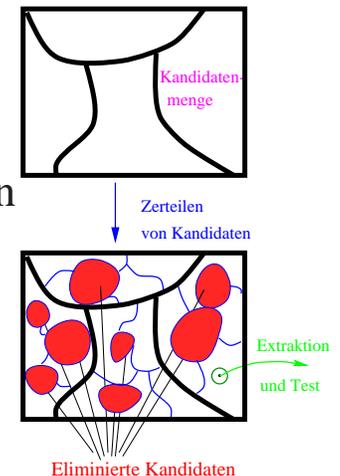
FUNCTION *Costas* ($n:\mathbb{Z}$) WHERE $n \geq 1$
RETURNS $\{p:\text{Seq}(\mathbb{Z}) \mid \text{perm}(p, \{1..n\}) \wedge \forall j < |p|. \text{nodups}(\text{dtrow}(p, j))\}$

• Grundstruktur von Globalsuch-Algorithmen

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x, y]\}$
 $\equiv \text{let rec } f_{gs}(x, s) = \{z \mid z \in \text{ext}[s] \wedge O[x, z]\} \cup \bigcup \{f_{gs}(x, t) \mid t \in \text{split}[x, s] \wedge \Phi[x, t]\}$
in $f_{gs}(x, s_0(x))$

• Bedeutung der Komponenten

- s : Deskriptor für Mengen von Lösungskandidaten
- $\text{ext}(s)$: Direkte Extraktion von Lösungskandidaten z aus Deskriptoren
- $O(x, z)$: Ausgabebedingung, verwendet zur endgültigen Selektion
- $\text{split}(x, s)$: Rekursive Aufteilung von Kandidatenmengen
- $\Phi(x, s)$: Filter zur Elimination unnötiger Deskriptoren
- $s_0(x)$: Initialdeskriptor für Eingabe x



GLOBALSUCH SYNTHESE: COSTAS-ARRAYS PROBLEM

• Problemspezifikation

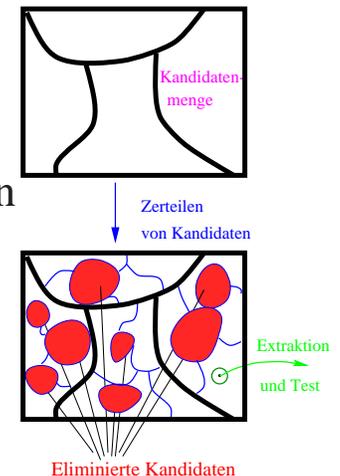
FUNCTION *Costas* ($n:\mathbb{Z}$) WHERE $n \geq 1$
 RETURNS $\{p:\text{Seq}(\mathbb{Z}) \mid \text{perm}(p, \{1..n\}) \wedge \forall j < |p|. \text{nodups}(\text{dtrow}(p, j))\}$

• Grundstruktur von Globalsuch-Algorithmen

FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x, y]\}$
 $\equiv \text{let rec } f_{gs}(x, s) = \{z \mid z \in \text{ext}[s] \wedge O[x, z]\} \cup \bigcup \{f_{gs}(x, t) \mid t \in \text{split}[x, s] \wedge \Phi[x, t]\}$
 in $f_{gs}(x, s_0(x))$

• Bedeutung der Komponenten

- s : Deskriptor für Mengen von Lösungskandidaten
- $\text{ext}(s)$: Direkte Extraktion von Lösungskandidaten z aus Deskriptoren
- $O(x, z)$: Ausgabebedingung, verwendet zur endgültigen Selektion
- $\text{split}(x, s)$: Rekursive Aufteilung von Kandidatenmengen
- $\Phi(x, s)$: Filter zur Elimination unnötiger Deskriptoren
- $s_0(x)$: Initialdeskriptor für Eingabe x



• Algorithmus nach Bestimmung der Komponenten

FUNCTION *Costas* ($n:\mathbb{Z}$) WHERE $n \geq 1$ RETURNS $\{p:\text{Seq}(\mathbb{Z}) \mid \text{perm}(p, \{1..n\}) \wedge \dots\}$
 $\equiv \text{let rec } f_{gs}(n, s)$
 $= \{p \mid p \in \{s\} \wedge \text{perm}(p, \{1..n\}) \wedge \forall j < n. \text{nodups}(\text{dtrow}(p, j))\}$
 $\cup \bigcup \{f_{gs}(x, t) \mid t \in \{s \cdot i \mid i \in \{1..n\}\} \wedge \text{nodups}(t) \wedge \forall j < |t|. \text{nodups}(\text{dtrow}(t, j))\}$
 in $f_{gs}(n, [])$

DIVIDE & CONQUER SYNTHESE: SORTIERALGORITHMEN

- **Problemspezifikation**

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

DIVIDE & CONQUER SYNTHESE: SORTIERALGORITHMEN

- **Problemspezifikation**

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

- **Grundstruktur von Divide & Conquer Algorithmen**

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

DIVIDE & CONQUER SYNTHESE: SORTIERALGORITHMEN

- **Problemspezifikation**

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

- **Grundstruktur von Divide & Conquer Algorithmen**

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

- **Komponenten für einen Sortieralgorithmus**

$\text{primitive} \equiv \lambda L. \text{null?}(L)$

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

$\text{Decompose} \equiv \lambda L. \text{let } a=L[|L|/2] \text{ in } (L_{<a}, L=a, L_{>a}) \quad (L_{<a} \equiv [x|x \in L \wedge x < a])$

$g \equiv \text{sort} \times \lambda S.S$

$\text{Compose} \equiv \lambda S_1, S_2, S_3. S_1 \circ S_2 \circ S_3$

DIVIDE & CONQUER SYNTHESE: SORTIERALGORITHMEN

- **Problemspezifikation**

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

- **Grundstruktur von Divide & Conquer Algorithmen**

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

- **Komponenten für einen Sortieralgorithmus**

$\text{primitive} \equiv \lambda L. \text{null?}(L)$

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

$\text{Decompose} \equiv \lambda L. \text{let } a=L[|L|/2] \text{ in } (L_{<a}, L_{=a}, L_{>a}) \quad (L_{<a} \equiv [x|x \in L \wedge x < a])$

$g \equiv \text{sort} \times \lambda S.S$

$\text{Compose} \equiv \lambda S_1, S_2, S_3. S_1 \circ S_2 \circ S_3$

- **Instantiierter Algorithmus**

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

\equiv if $\text{null?}(L)$ then $[]$

else let $a=L[|L|/2]$

in $\text{sort}([x|x \in L \wedge x < a]) \circ [x|x \in L \wedge x = a] \circ \text{sort}([x|x \in L \wedge x > a])$

ALGORITHMENSHEMATA: FORMALER HINTERGRUND

\mathcal{T}_{SPEC}

\mathcal{T}_{PROG}

Spezifikation

Programm

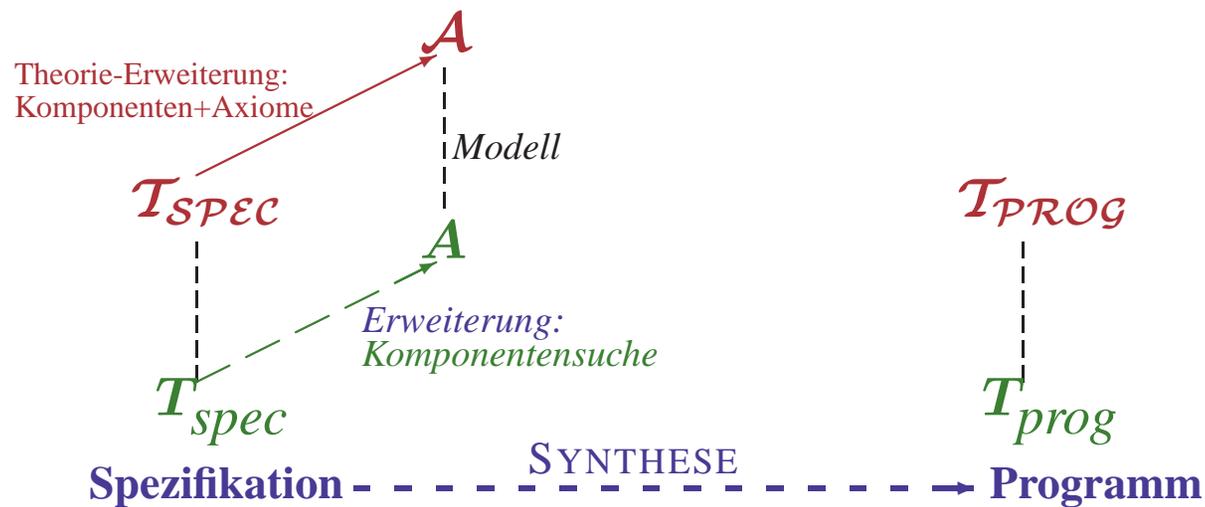
- **Algorithmische Theorien beschreiben Entwurfsmethode**
 - **Algorithmtheorie** = Komponenten+Axiome eines Algorithmenschemas
 - Formale Brücke zwischen allgemeiner Theorie der Spezifikationen und abstrakter Theorie korrekter Programme

ALGORITHMENSCHEMATA: FORMALER HINTERGRUND



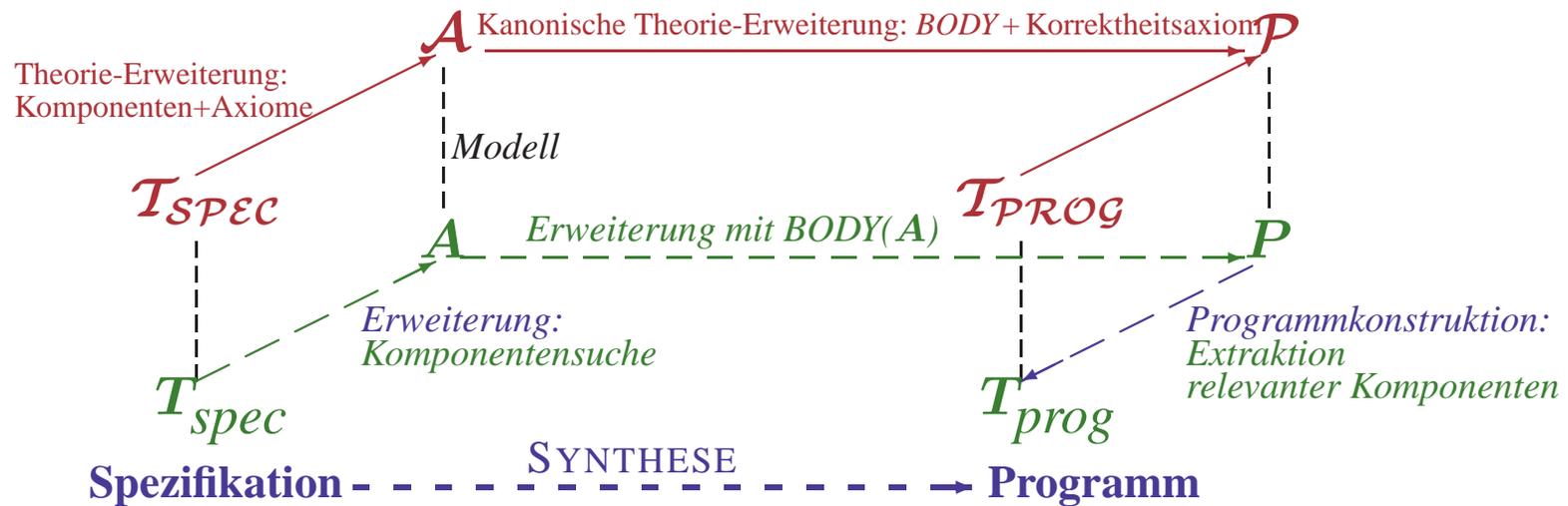
- **Algorithmische Theorien beschreiben Entwurfsmethode**
 - **Algorithmentheorie** = Komponenten+Axiome eines Algorithmenschemas
 - Formale Brücke zwischen allgemeiner Theorie der Spezifikationen und abstrakter Theorie korrekter Programme
- **Syntheseprozess instantiiert allgemeine Theorien**
 - Problem = konkrete Spezifikation + Domänenbeschreibung

ALGORITHMENSCHEMATA: FORMALER HINTERGRUND



- **Algorithmische Theorien beschreiben Entwurfsmethode**
 - **Algorithmtheorie** = Komponenten+Axiome eines Algorithmenschemas
 - Formale Brücke zwischen allgemeiner Theorie der Spezifikationen und abstrakter Theorie korrekter Programme
- **Syntheseprozess instantiiert allgemeine Theorien**
 - Problem = konkrete Spezifikation + Domänenbeschreibung
 - Synthese: bestimme Komponenten, welche die Axiome erfüllen

ALGORITHMENSCHEMATA: FORMALER HINTERGRUND



- **Algorithmische Theorien beschreiben Entwurfsmethode**

- **Algorithmentheorie** = Komponenten+Axiome eines Algorithmenschemas
- Formale Brücke zwischen allgemeiner Theorie der Spezifikationen und abstrakter Theorie korrekter Programme

- **Syntheseprozess instantiiert allgemeine Theorien**

- Problem = konkrete Spezifikation + Domänenbeschreibung
- Synthese: bestimme Komponenten, welche die Axiome erfüllen
- Korrekte Lösung ergibt sich kanonisch aus Algorithmentheorie

- **Eine formale Theorie \mathcal{T} hat drei Komponenten**

- S : Menge von Sortennamen (Namen für Datentypen)
- Ω : Familie von Operationsnamen (zusammen mit Typisierung)
- Ax : Menge von Axiomen für Datentypen und Operationen

- **Eine formale Theorie \mathcal{T} hat drei Komponenten**

- S : Menge von Sortennamen (Namen für Datentypen)
- Ω : Familie von Operationsnamen (zusammen mit Typisierung)
- Ax : Menge von Axiomen für Datentypen und Operationen

- **Modell T für Theorie \mathcal{T}**

- T ist Struktur für \mathcal{T} : Menge von Datentypen passend zu S
und Operationen typisiert gemäß Ω
- Struktur T erfüllt alle Axiome aus \mathcal{T}

- **Eine formale Theorie \mathcal{T} hat drei Komponenten**

- S : Menge von Sortennamen (Namen für Datentypen)
- Ω : Familie von Operationsnamen (zusammen mit Typisierung)
- Ax : Menge von Axiomen für Datentypen und Operationen

- **Modell T für Theorie \mathcal{T}**

- T ist Struktur für \mathcal{T} : Menge von Datentypen passend zu S und Operationen typisiert gemäß Ω
- Struktur T erfüllt alle Axiome aus \mathcal{T}

- **Theorie \mathcal{T}_1 erweitert \mathcal{T}_2**

- Alle Sortennamen, Operationsnamen, Axiome von \mathcal{T}_2 existieren in \mathcal{T}_1

- **Struktur T_1 erweitert T_2**

- Alle Datentypen und Operationen von T_2 existieren auch in T_1 und haben die gleiche Typisierung bezüglich \mathcal{T}_2

ENDLICHE MENGEN ALS ALGEBRAISCHE THEORIE

● Abstrakte Theorie = Struktur + Grundgesetze

Sorten: $\text{Set}(\alpha)$

Operationsnamen: $\emptyset: \text{Set}(\alpha)$

$+: \text{Set}(\alpha) \times \alpha \rightarrow \text{Set}(\alpha)$

$\in: \alpha \times \text{Set}(\alpha) \rightarrow \text{Bool}$

Axiome:

$a \notin \emptyset$

$x \in (S+a) \Leftrightarrow (x=a \vee x \in S)$

$(S+a)+x = (S+x)+a$

$(S+a)+a = S+a$

$P(\emptyset) \wedge (\forall S: \text{Set}(\alpha). P(S) \Rightarrow \forall a: \alpha. P(S+a))$

$\Rightarrow \forall S: \text{Set}(\alpha). P(S)$

ENDLICHE MENGEN ALS ALGEBRAISCHE THEORIE

● Abstrakte Theorie = Struktur + Grundgesetze

Sorten: $\text{Set}(\alpha)$

Operationsnamen: $\emptyset: \text{Set}(\alpha)$

$+: \text{Set}(\alpha) \times \alpha \rightarrow \text{Set}(\alpha)$

$\in: \alpha \times \text{Set}(\alpha) \rightarrow \text{Bool}$

Axiome:

$a \notin \emptyset$

$x \in (S+a) \Leftrightarrow (x=a \vee x \in S)$

$(S+a)+x = (S+x)+a$

$(S+a)+a = S+a$

$P(\emptyset) \wedge (\forall S: \text{Set}(\alpha). P(S) \Rightarrow \forall a: \alpha. P(S+a))$

$\Rightarrow \forall S: \text{Set}(\alpha). P(S)$

● Modell = Implementierung durch existierende Konzepte

$\emptyset \equiv \text{nil}$

$+ \equiv \lambda a, S. a.S$

$\in \equiv \lambda a, S. \exists x \in S. x =_b a$

$=_{\text{Set}} \equiv \lambda S, T. (\forall a \in S. a \in T) \wedge (\forall a' \in T. a' \in S)$

$\text{Set}(\alpha) \equiv (S, T): \alpha \text{ list} // S =_{\text{Set}} T$

Alternative Modelle sind möglich (Bitvektoren, sortierte Listen,...)

- **\mathcal{T}_{SPEC} : Theorie der Spezifikationen**

- Sorten: D, R ; Operationen: $I:D \rightarrow \mathbb{B}$, $O:D \times R \rightarrow \mathbb{B}$; keine Axiome

- **Problemtheorie**: Erweiterung von \mathcal{T}_{SPEC}

Spezifikation eines Programms durch Bedingungen an I und O

- **\mathcal{T}_{SPEC} : Theorie der Spezifikationen**
 - Sorten: D, R ; Operationen: $I:D \rightarrow \mathbb{B}$, $O:D \times R \rightarrow \mathbb{B}$; keine Axiome
 - **Problemtheorie**: Erweiterung von \mathcal{T}_{SPEC}
Spezifikation eines Programms durch Bedingungen an I und O
- **\mathcal{T}_{PROG} : Theorie korrekter Programme**
 - Sorten: D, R ; Operationen: $I:D \rightarrow \mathbb{B}$, $O:D \times R \rightarrow \mathbb{B}$, $body: D \nrightarrow R$
 - Axiome: $\forall x:D. I(x) \Rightarrow O(x, body(x))$
 - **Programmtheorie**: Erweiterung von \mathcal{T}_{PROG}
Beschreibung einer Klasse korrekter Programme

- **\mathcal{T}_{SPEC} : Theorie der Spezifikationen**
 - Sorten: D, R ; Operationen: $I:D \rightarrow \mathbb{B}$, $O:D \times R \rightarrow \mathbb{B}$; keine Axiome
 - **Problemtheorie**: Erweiterung von \mathcal{T}_{SPEC}
Spezifikation eines Programms durch Bedingungen an I und O
- **\mathcal{T}_{PROG} : Theorie korrekter Programme**
 - Sorten: D, R ; Operationen: $I:D \rightarrow \mathbb{B}$, $O:D \times R \rightarrow \mathbb{B}$, $body: D \not\rightarrow R$
 - Axiome: $\forall x:D. I(x) \Rightarrow O(x, body(x))$
 - **Programmtheorie**: Erweiterung von \mathcal{T}_{PROG}
Beschreibung einer Klasse korrekter Programme
- **Algorithmtheorie: Theorie einer Algorithmenstruktur**
 - Problemtheorie \mathcal{A} mit kanonischer Erweiterung zu Programmtheorie:
 - Es gibt ein abstraktes Programmschema $BODY$, so daß für jedes Modell A von \mathcal{A} die Spezifikation $spec_A$ zusammen mit $BODY(A)$ korrekt ist
(Für jedes Modell A von \mathcal{A} hat $spec_A$ eine Standardlösung)

DIVIDE & CONQUER SCHEMA ALS ALGORITHMMENTHEORIE

Sorten : D, R, D', R'

Operationen : $I: D \rightarrow \mathbb{B}, O: D \times R \rightarrow \mathbb{B}, \succ: D \times D \rightarrow \mathbb{B},$
 $Decompose: D \rightarrow D' \times D, O_D: D \times D' \times D \rightarrow \mathbb{B}, I': D' \rightarrow \mathbb{B},$
 $Compose: R' \times R \rightarrow R, O': D' \times R' \rightarrow \mathbb{B}, O_C: R' \times R \times R \rightarrow \mathbb{B},$
 $g: D' \rightarrow R', Directly-solve: D \rightarrow R, primitive: D \rightarrow \mathbb{B}$

Axiome : FUNCTION $f_p(x: D) : R$ WHERE $I[x] \wedge primitive[x]$
 RETURNS y SUCH THAT $O[x, y]$
 $\equiv Directly-solve[x]$
 :
 \succ ist wohlfundierte Ordnung auf D

ist korrekt

DIVIDE & CONQUER SCHEMA ALS ALGORITHMENTHEORIE

Sorten : D, R, D', R'

Operationen : $I: D \rightarrow \mathbb{B}, O: D \times R \rightarrow \mathbb{B}, \succ: D \times D \rightarrow \mathbb{B},$
 $Decompose: D \rightarrow D' \times D, O_D: D \times D' \times D \rightarrow \mathbb{B}, I': D' \rightarrow \mathbb{B},$
 $Compose: R' \times R \rightarrow R, O': D' \times R' \rightarrow \mathbb{B}, O_C: R' \times R \times R \rightarrow \mathbb{B},$
 $g: D' \rightarrow R', Directly-solve: D \rightarrow R, primitive: D \rightarrow \mathbb{B}$

Axiome : FUNCTION $f_p(x: D) : R$ WHERE $I[x] \wedge primitive[x]$

RETURNS y SUCH THAT $O[x, y]$

$\equiv Directly-solve[x]$

ist korrekt

⋮

\succ ist wohlfundierte Ordnung auf D

Für jedes Modell A der Divide & Conquer Theorie ist

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

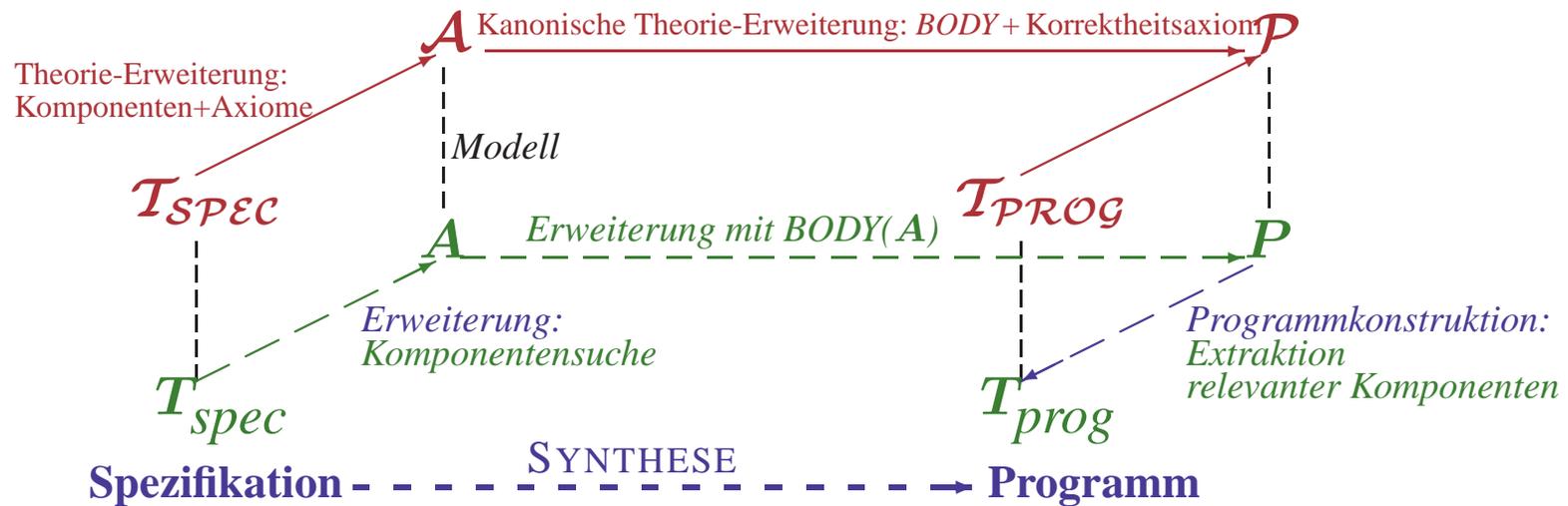
ein korrektes Programm

(Details in Einheit 18)

SYNTHESE MIT ALGORITHMENSHEMATA PRÄZISIERT

- **Satz:** Eine Spezifikation $spec$ ist erfüllbar, wenn es eine Algorithmentheorie \mathcal{A} und ein Modell A für \mathcal{A} gibt, das die Struktur T_{spec} erweitert

SYNTHESE MIT ALGORITHMENSHEMATA PRÄZISIERT



- **Satz:** Eine Spezifikation $spec$ ist erfüllbar, wenn es eine Algorithmentheorie \mathcal{A} und ein Modell A für \mathcal{A} gibt, das die Struktur T_{spec} erweitert
- **Synthese erweitert Problemtheorien zu Algorithmentheorien**
 - Wähle eine Algorithmentheorie \mathcal{A}
 - Erweitere die Spezifikation zu einem Modell A von \mathcal{A}
Bestimme Komponenten und beweise, daß Axiome erfüllt sind
 - Instantiierung von $BODY(\mathcal{A})$ liefert Modell einer Programmtheorie
 - Extrahiere Programm als Lösung der Spezifikation

EINFACHES ALGORITHMENSHEMA: OPERATOR MATCH

Problemreduktion auf bekannte Algorithmen

Problemreduktion auf bekannte Algorithmen

- ***spec* reduzierbar auf *spec'* ($spec \trianglelefteq spec'$)**

$$\forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y':R'. O'[x', y'] \Rightarrow \exists y:R. O[x, y])$$

- Transformation von Ein- und Ausgaben einer gegebenen Spezifikation
- *spec* hat stärkere Eingabebedingung, schwächere Ausgabebedingung

Problemreduktion auf bekannte Algorithmen

- ***spec* reduzierbar auf *spec'*** ($spec \trianglelefteq spec'$)

$$\forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y':R'. O'[x', y'] \Rightarrow \exists y:R. O[x, y])$$

- Transformation von Ein- und Ausgaben einer gegebenen Spezifikation
- *spec* hat stärkere Eingabebedingung, schwächere Ausgabebedingung

- **Reduzierbarkeit liefert statische Problemtransformation**

- $spec = (D, R, I, O)$ ist erfüllbar, wenn es eine erfüllbare Spezifikation $spec' = (D', R', I', O')$ gibt mit $spec \trianglelefteq spec'$

- Beweis für $spec \trianglelefteq spec'$ liefert Substitutionen $\vartheta:D \rightarrow D'$, $\sigma:D \times R' \rightarrow R$

- Ist $body'$ korrekter Algorithmus für $spec'$, dann definiert

$$body(x) \equiv \sigma(x, body'(\vartheta(x))) \text{ einen korrekten Algorithmus für } spec$$

Problemreduktion auf bekannte Algorithmen

- ***spec* reduzierbar auf *spec'* ($spec \trianglelefteq spec'$)**

$$\forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y':R'. O'[x', y'] \Rightarrow \exists y:R. O[x, y])$$

- Transformation von Ein- und Ausgaben einer gegebenen Spezifikation
- *spec* hat stärkere Eingabebedingung, schwächere Ausgabebedingung

- **Reduzierbarkeit liefert statische Problemtransformation**

- *spec* = (D, R, I, O) ist erfüllbar, wenn es eine erfüllbare Spezifikation *spec'* = (D', R', I', O') gibt mit $spec \trianglelefteq spec'$

- Beweis für $spec \trianglelefteq spec'$ liefert Substitutionen $\vartheta:D \rightarrow D'$, $\sigma:D \times R' \rightarrow R$
- Ist *body'* korrekter Algorithmus für *spec'*, dann definiert

$$body(x) \equiv \sigma(x, body'(\vartheta(x))) \text{ einen korrekten Algorithmus für } spec$$

- **Einfaches Syntheseverfahren**

- Suche erfüllbare Spezifikation *spec'* und beweise $spec \trianglelefteq spec'$
- Extrahiere σ und ϑ und spezialisiere *body'* zu $\lambda x. \sigma(x, body'(\vartheta(x)))$

Auch geeignet zur Verfeinerung anderer Algorithmentheorien

↪ später

EINFACHES ALGORITHMENSHEMA: GENERALISIERUNG

Problemreduktion für mengenwertige Spezifikationen

Problemreduktion für mengenwertige Spezifikationen

- *spec* spezialisiert *spec'* ($spec \ll spec'$)

$$R \subseteq R' \wedge \forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y:R. O[x, y] \Rightarrow O'[x', y])$$

- Ein- und Ausgabebedingungen von *spec* stärker als *spec'*
- Andere Bezeichnungsart: *spec'* generalisiert *spec*

Problemreduktion für mengenwertige Spezifikationen

- *spec* spezialisiert *spec'* ($spec \ll spec'$)

$$R \subseteq R' \wedge \forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y:R. O[x, y] \Rightarrow O'[x', y])$$

- Ein- und Ausgabebedingungen von *spec* stärker als *spec'*
- Andere Bezeichnungsart: *spec'* generalisiert *spec*

- **Spezialisierung liefert Problemtransformation**

- *spec* = FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x, y]\}$ *ist erfüllbar, wenn $spec \ll spec'$ für eine erfüllbare Spezifikation $spec'$ gilt*
- Beweis für $spec \ll spec_A$ liefert Substitution $\vartheta:D \rightarrow D'$
- *body*(x) $\equiv \{y \mid y \in body'(\vartheta(x)) \wedge O(x, y)\}$ ist korrekt für *spec*

Problemreduktion für mengenwertige Spezifikationen

- ***spec* spezialisiert *spec'* ($spec \ll spec'$)**

$$R \subseteq R' \wedge \forall x:D. I[x] \Rightarrow \exists x':D'. (I'[x'] \wedge \forall y:R. O[x, y] \Rightarrow O'[x', y])$$

- Ein- und Ausgabebedingungen von *spec* stärker als *spec'*
- Andere Bezeichnungsart: *spec'* generalisiert *spec*

- **Spezialisierung liefert Problemtransformation**

- *spec* = FUNCTION $f(x:D)$ WHERE $I[x]$ RETURNS $\{y:R \mid O[x, y]\}$ *ist erfüllbar*, wenn $spec \ll spec'$ für eine erfüllbare Spezifikation *spec'* gilt
- Beweis für $spec \ll spec_A$ liefert Substitution $\vartheta:D \rightarrow D'$
- *body*(x) $\equiv \{y \mid y \in body'(\vartheta(x)) \wedge O(x, y)\}$ ist korrekt für *spec*

- **Syntheseverfahren**

- Suche erfüllbare Spezifikation *spec'* und beweise $spec \ll spec'$
- Extrahiere ϑ , spezialisiere *body'* zu $\lambda x. \{y \mid y \in body'(\vartheta(x)) \wedge O(x, y)\}$

Geeignet zur Verfeinerung mengenwertiger Algorithmentheorien \mapsto später

EINFACHES ALGORITHMENSHEMA: FALLUNTERSCHIEDUNG

Statische \vee -Reduktion durch Zerlegung des Problems

Statische \vee -Reduktion durch Zerlegung des Problems

- ***spec* zerlegbar in $spec_1..spec_n$ ($spec = \cup_i spec_i$)**
 $\forall x:D . I[x] \Rightarrow I_1[x] \vee .. \vee I_n[x] \quad \wedge \quad \forall i \leq n . \forall y:R . O_i[x, y] \Rightarrow O[x, y]$
 - Eingabebedingung zerlegbar in Eingabebedingungen der $spec_i$
 - Ausgabebedingung der $spec_i$ stärker

Statische \vee -Reduktion durch Zerlegung des Problems

- ***spec* zerlegbar in $spec_1..spec_n$ ($spec = \cup_i spec_i$)**
$$\forall x:D. I[x] \Rightarrow I_1[x] \vee .. \vee I_n[x] \quad \wedge \quad \forall i \leq n. \forall y:R. O_i[x, y] \Rightarrow O[x, y]$$
 - Eingabebedingung zerlegbar in Eingabebedingungen der $spec_i$
 - Ausgabebedingung der $spec_i$ stärker
- **Zerlegbarkeit liefert Fallunterscheidung**
 - *spec* ist erfüllbar, wenn es erfüllbare Spezifikationen $spec_i$ gibt mit $spec = \cup_i spec_i$
 - Sind $body_i$ korrekte Algorithmen für $spec_i$, dann ist der Algorithmus $body(x) \equiv \text{if } I_1[x] \text{ then } body_{A_1}[x] .. \text{else } body_{A_n}[x]$ korrekt für $spec$

Statische \vee -Reduktion durch Zerlegung des Problems

- ***spec* zerlegbar in $spec_1..spec_n$ ($spec = \cup_i spec_i$)**

$$\forall x:D. I[x] \Rightarrow I_1[x] \vee .. \vee I_n[x] \quad \wedge \quad \forall i \leq n. \forall y:R. O_i[x, y] \Rightarrow O[x, y]$$

- Eingabebedingung zerlegbar in Eingabebedingungen der $spec_i$
- Ausgabebedingung der $spec_i$ stärker

- **Zerlegbarkeit liefert Fallunterscheidung**

- *spec ist erfüllbar, wenn es erfüllbare Spezifikationen $spec_i$ gibt mit $spec = \cup_i spec_i$*
- Sind $body_i$ korrekte Algorithmen für $spec_i$, dann ist der Algorithmus $body(x) \equiv \text{if } I_1[x] \text{ then } body_{A_1}[x] .. \text{else } body_{A_n}[x]$ korrekt für $spec$

- **Syntheseverfahren “Derived Antecedants”**

- Suche erfüllbare Spezifikation $spec_1$ mit stärkerer Ausgabebedingung
- **Wiederhole** mit $spec'=(D, R, I \wedge \neg I_1, O)$ bis Zerlegung komplett
- **Setze einzelne Lösungen** durch Fallunterscheidung **zusammen**

Auch geeignet zur Verfeinerung anderer Algorithmentheorien

↪ später

- **Bestimme Vorbedingungen für Gültigkeit einer Formel**

- Formel F darf Quantoren enthalten

- Vorbedingung P darf evtl. manche Variablen von F nicht enthalten

- z.B. Vorbedingung für $F \equiv \exists L_1, L_2 \ |L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$

- darf nur die Variable L verwenden

(Lösung $P[L] \equiv |L| > 1$)

- **Bestimme Vorbedingungen für Gültigkeit einer Formel**

- Formel F darf Quantoren enthalten
- Vorbedingung P darf evtl. manche Variablen von F nicht enthalten
- z.B. Vorbedingung für $F \equiv \exists L_1, L_2 \ |L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$
darf nur die Variable L verwenden (Lösung $P[L] \equiv |L| > 1$)

- **Herleitung durch Anwendung von Reduktionsregeln**

- Rewriting mit Äquivalenzumformungen und Implikationen
z.B. $\text{True} \Rightarrow \exists L_1, L_2 \ L_1 \circ L_2$ und $L_1 \circ L_2 = L \Rightarrow |L_1| + |L_2| = |L|$ und $i + j > i \Leftrightarrow j > 0$
- Inferenzsystem muß hunderte von Rewrite Regeln verwalten

- **Bestimme Vorbedingungen für Gültigkeit einer Formel**

- Formel F darf Quantoren enthalten
- Vorbedingung P darf evtl. manche Variablen von F nicht enthalten
- z.B. Vorbedingung für $F \equiv \exists L_1, L_2 \ |L| > |L_1| \wedge |L| > |L_2| \wedge L_1 \circ L_2 = L$
darf nur die Variable L verwenden (Lösung $P[L] \equiv |L| > 1$)

- **Herleitung durch Anwendung von Reduktionsregeln**

- Rewriting mit Äquivalenzumformungen und Implikationen
z.B. $\text{True} \Rightarrow \exists L_1, L_2 \ L_1 \circ L_2$ und $L_1 \circ L_2 = L \Rightarrow |L_1| + |L_2| = |L|$ und $i + j > i \Leftrightarrow j > 0$
- Inferenzsystem muß hunderte von Rewrite Regeln verwalten

- **Steuerung ist heuristisch**

- Anwendung von Regeln auf quantorenfreien Teil der Formel
- **Vorwärtsinferenz**: Ziel ist syntaktischer (intensionaler) Natur
z.B. Beschränkung der erlaubten Variablen, Größe der Formel etc.

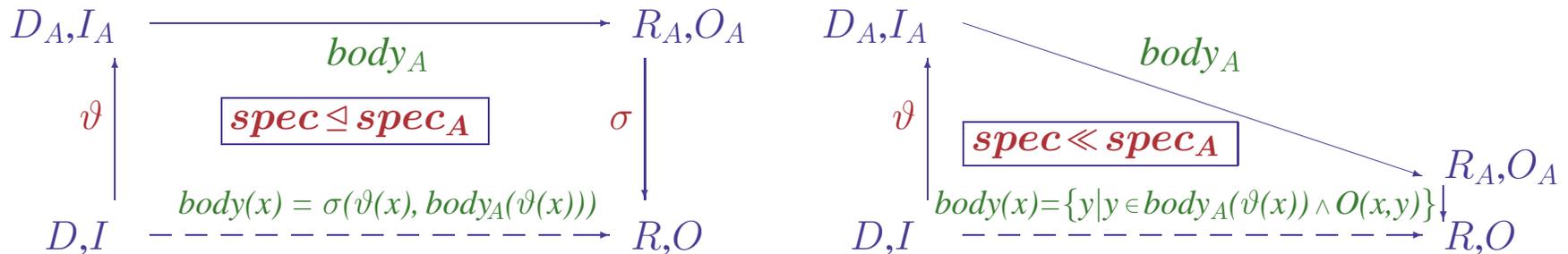
SYNTHESE MIT INTEGRIERTER PROBLEMREDUKTION

Modellbildung durch statische Reduktion auf Teilmodelle

Modellbildung durch statische Reduktion auf Teilmodelle

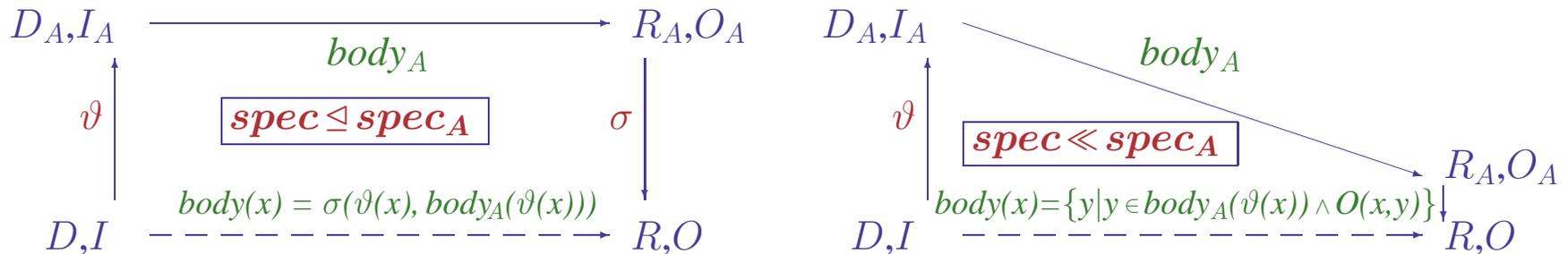
- **Speichere generische Modelle von Algorithmentheorien**
 - Standardkomponenten der Algorithmentheorie für wichtige Domänen
 - Axiome von \mathcal{A} sind für diese Modelle a-priori bewiesen

Modellbildung durch statische Reduktion auf Teilmodelle



- **Speichere generische Modelle von Algorithmentheorien**
 - Standardkomponenten der Algorithmentheorie für wichtige Domänen
 - Axiome von \mathcal{A} sind für diese Modelle a-priori bewiesen
- **Synthese reduziert Spezifikation auf gespeichertes Modell**
 - Wähle eine Algorithmentheorie \mathcal{A}
 - Wähle Modell A passend zur Grunddomäne von $spec$ (Liste, Menge, Baum, ...)
 - Versuche $spec \sqsubseteq spec_A$ automatisch zu beweisen (proofs-as-programs)
 - Mengenwertige Synthese analog mit Generalisierung $spec \ll spec_A$
 - Spezialisierung von $body_A$ korrekt für extrahierte Substitutionen σ und ϑ

Modellbildung durch statische Reduktion auf Teilmodelle



- **Speichere generische Modelle von Algorithmentheorien**
 - Standardkomponenten der Algorithmentheorie für wichtige Domänen
 - Axiome von \mathcal{A} sind für diese Modelle a-priori bewiesen
- **Synthese reduziert Spezifikation auf gespeichertes Modell**
 - Wähle eine Algorithmentheorie \mathcal{A}
 - Wähle Modell A passend zur Grunddomäne von spec (Liste, Menge, Baum, ...)
 - Versuche $\text{spec} \sqsubseteq \text{spec}_A$ automatisch zu beweisen (proofs-as-programs)
 - Mengenwertige Synthese analog mit Generalisierung $\text{spec} \ll \text{spec}_A$
 - Spezialisierung von body_A korrekt für extrahierte Substitutionen σ und ϑ

Syntheseverfahren erfordert keine weiteren Inferenzen!

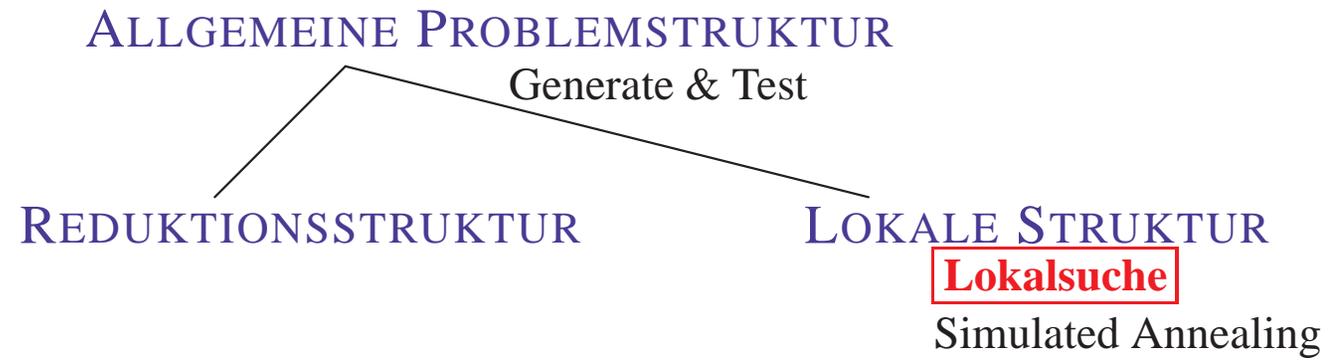
↪ Beispiele später

HIERARCHIE ALGORITHMISCHER STRUKTUREN

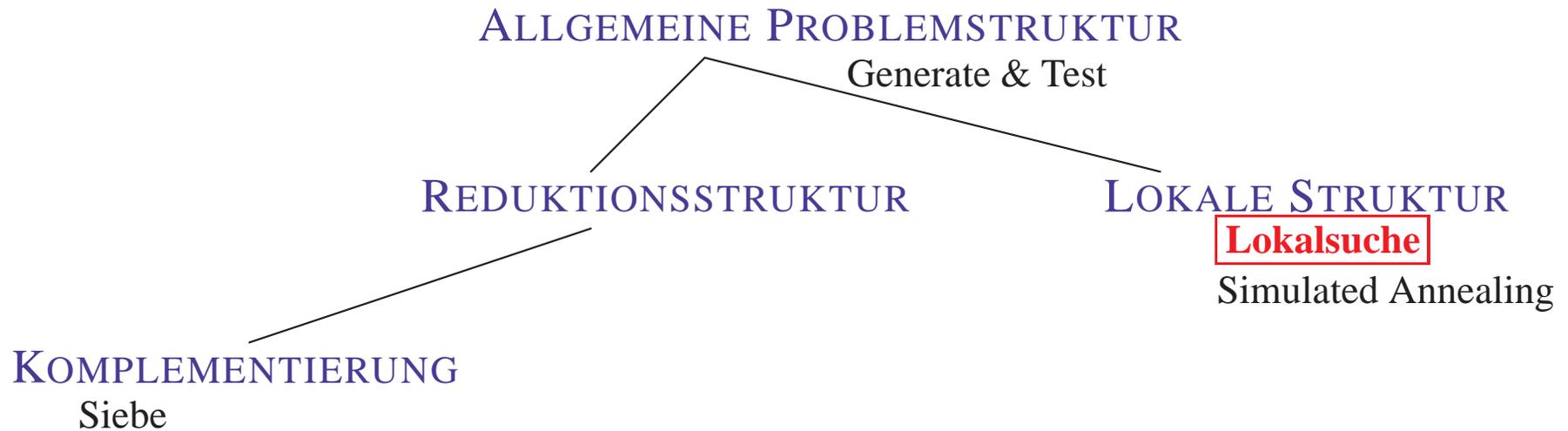
ALLGEMEINE PROBLEMSTRUKTUR

Generate & Test

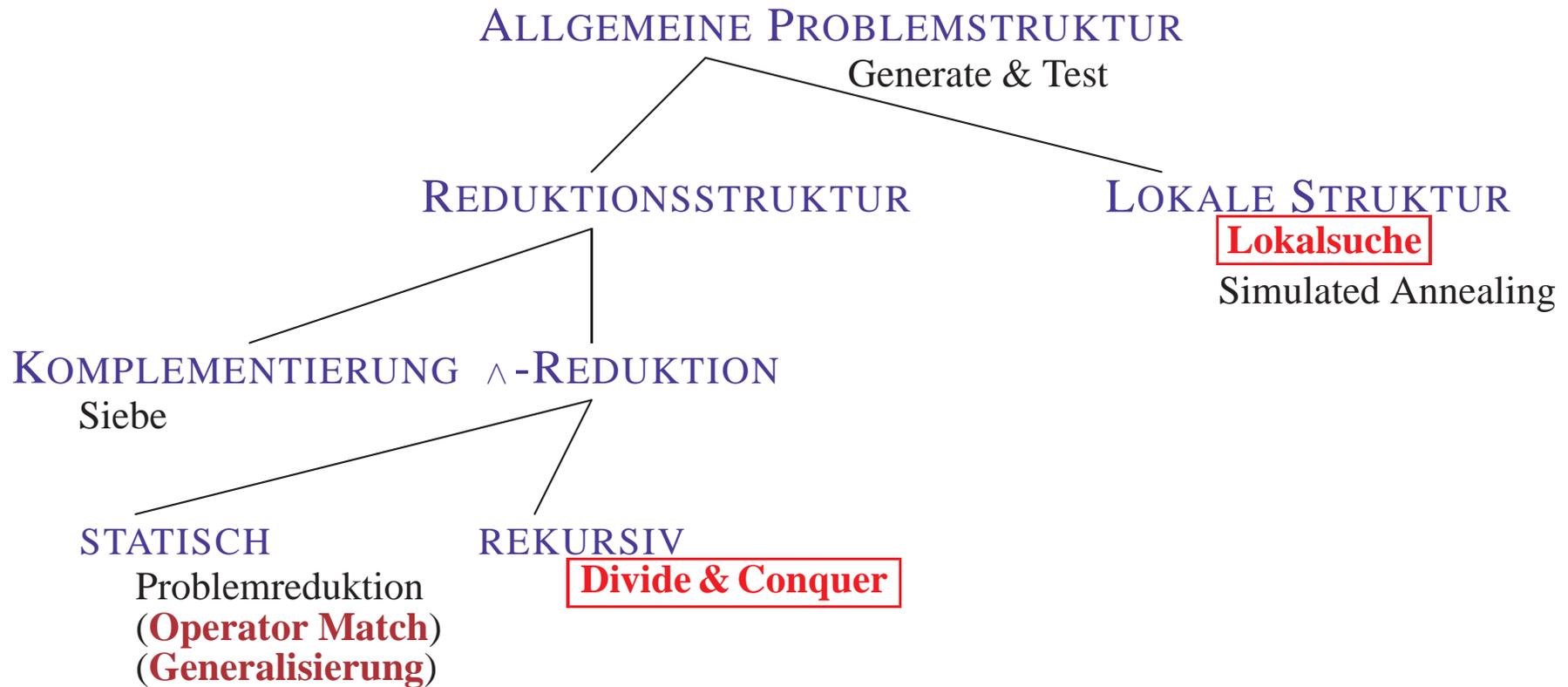
HIERARCHIE ALGORITHMISCHER STRUKTUREN



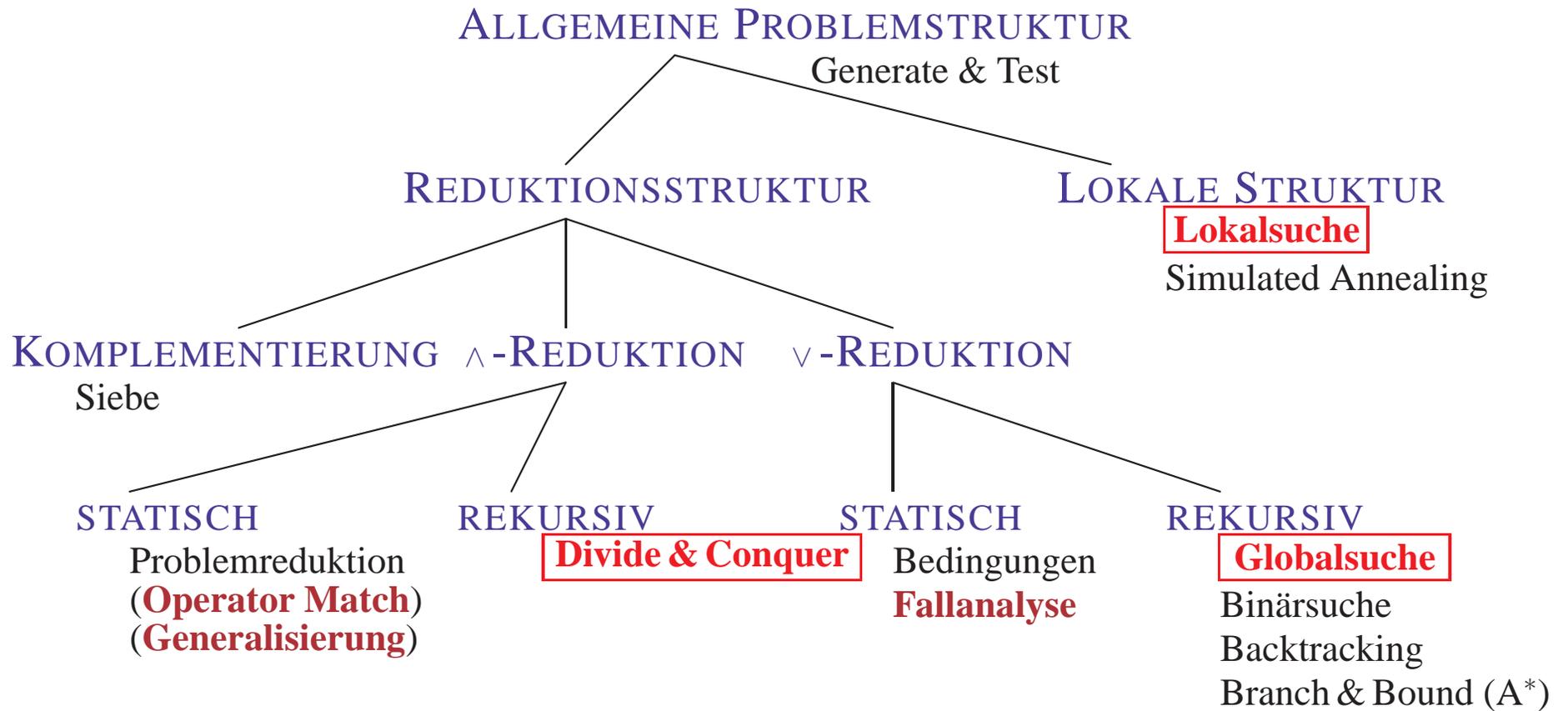
HIERARCHIE ALGORITHMISCHER STRUKTUREN



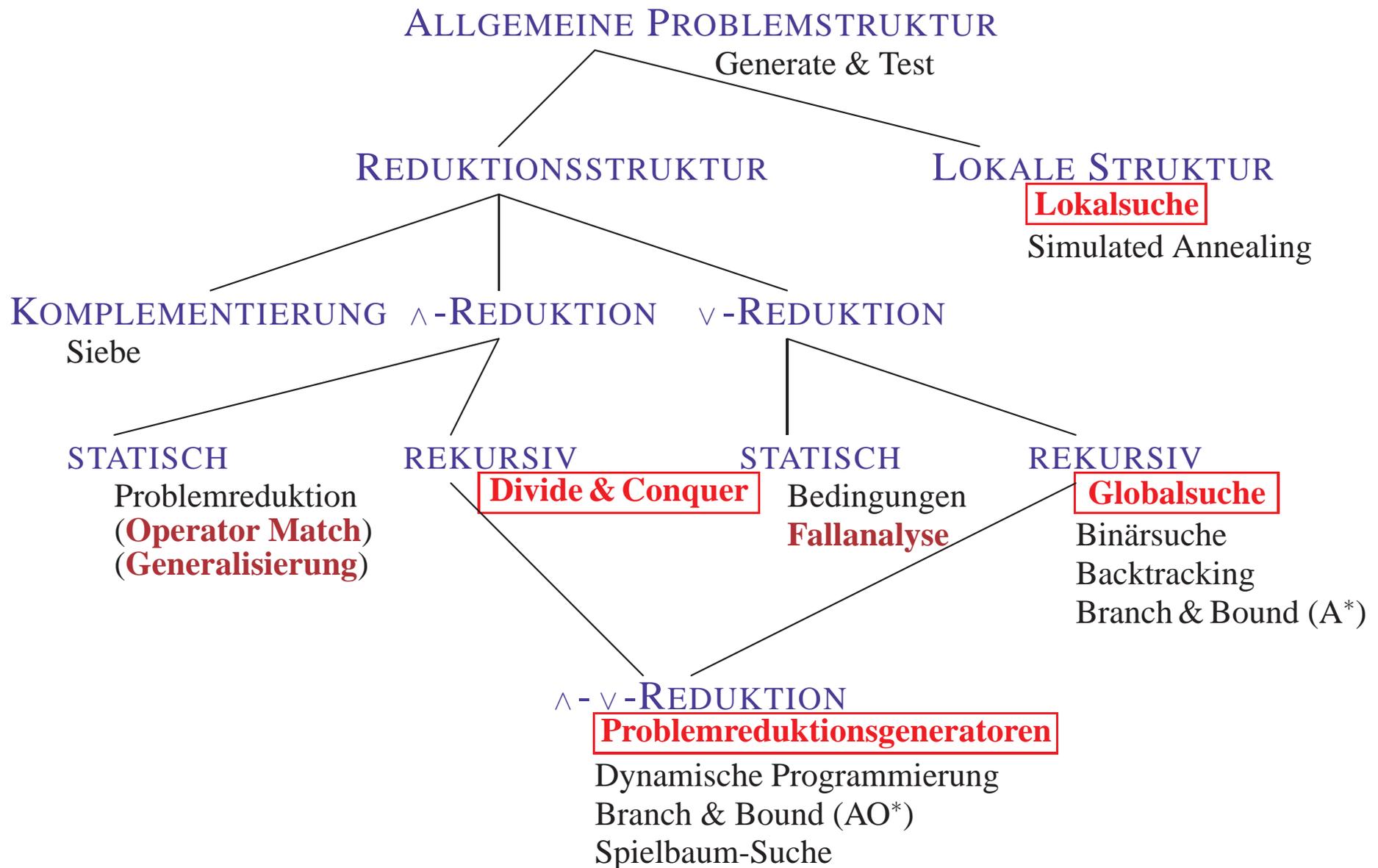
HIERARCHIE ALGORITHMISCHER STRUKTUREN



HIERARCHIE ALGORITHMISCHER STRUKTUREN



HIERARCHIE ALGORITHMISCHER STRUKTUREN



ALGORITHMENSHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

ALGORITHMENSCHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

- **Erzeugung effizienter Algorithmen**

- Vorgabe einer **effizienten Grundstruktur** durch Theoreme
- Individuelle Optimierung nachträglich

ALGORITHMENSHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

- **Erzeugung effizienter Algorithmen**

- Vorgabe einer **effizienten Grundstruktur** durch Theoreme
- Individuelle Optimierung nachträglich

- **Wissensbasiertes Vorgehen**

- Erkenntnisse über Algorithmen als Theoreme verwendbar

ALGORITHMENSCHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

- **Erzeugung effizienter Algorithmen**

- Vorgabe einer **effizienten Grundstruktur** durch Theoreme
- Individuelle Optimierung nachträglich

- **Wissensbasiertes Vorgehen**

- Erkenntnisse über Algorithmen als Theoreme verwendbar

- **Formales theoretisches Fundament**

- Leicht in das Konzept beweisbasierter Systeme integrierbar

ALGORITHMENSCHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

- **Erzeugung effizienter Algorithmen**

- Vorgabe einer **effizienten Grundstruktur** durch Theoreme
- Individuelle Optimierung nachträglich

- **Wissensbasiertes Vorgehen**

- Erkenntnisse über Algorithmen als Theoreme verwendbar

- **Formales theoretisches Fundament**

- Leicht in das Konzept beweisbasierter Systeme integrierbar

ALGORITHMENSCHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

- **Erzeugung effizienter Algorithmen**

- Vorgabe einer **effizienten Grundstruktur** durch Theoreme
- Individuelle Optimierung nachträglich

- **Wissensbasiertes Vorgehen**

- Erkenntnisse über Algorithmen als Theoreme verwendbar

- **Formales theoretisches Fundament**

- Leicht in das Konzept beweisbasierter Systeme integrierbar

ALGORITHMENSCHEMATA: VORTEILE FÜR SYNTHESE

- **Effizientes Syntheseverfahren**

- **Voruntersuchungen entlasten Syntheseprozess** zur Laufzeit
 - Beweislast verlagert in Entwurf und Beweis der Algorithmentheorien
- **Verfeinerung vorgefertigter Teillösungen** (Modelle) möglich
 - zielgerichtetes Vorgehen, Verifikation der Axiome entfällt
- Echte **Kooperation zwischen Mensch und Computer**
 - Mensch: Entwurfsentscheidungen — Computer: formale Details

- **Erzeugung effizienter Algorithmen**

- Vorgabe einer **effizienten Grundstruktur** durch Theoreme
- Individuelle Optimierung nachträglich

- **Wissensbasiertes Vorgehen**

- Erkenntnisse über Algorithmen als Theoreme verwendbar

- **Formales theoretisches Fundament**

- Leicht in das Konzept beweisbasierter Systeme integrierbar