Automatisierte Logik und Programmierung

Lektion 18

Wissensbasierte Programmentwicklung

1. Algorithmenschemata
2. Globalsuche
3. Divide & Conquer
4. Lokalsuche

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 Programmierwissen verarbeiten**
  - Umsetzung von Programmiermethodik in Entwurfsstrategien
  - Schematisierung von Algorithmenstrukturen
  - Axiome für Korrektheit des schematischen Algorithmus

Aufwendiges theoretisches Fundament entlastet Syntheseprozeß

Wissensbasierte Programmentwicklung: Grundidee

- **Erzeuge Algorithmen in einem Schritt**
  - Anpassung eines schematischen Algorithmus an eine Problemstellung
  - Historisch: High-Level Transformation
- **Grundsätzliche Vorgehensweise**
  - Gegeben sei die Spezifikation
    - `FUNCTION` $\boldsymbol{f}$`(`$\boldsymbol{x}$`:`$\boldsymbol{D}$`):`$\boldsymbol{R}$ `WHERE` $\boldsymbol{I}[\boldsymbol{x}]$ `RETURNS` $\boldsymbol{y}$ `SUCH THAT` $\boldsymbol{O}[\boldsymbol{x},\boldsymbol{y}]$
  - Wähle algorithmische Grundstruktur
  - Verfeinere Basisschema der Struktur durch Bestimmung von Parametern
  - Prüfe, ob Parameter die Korrektheitsaxiome des Schemas erfüllen
  - Instantiiere schematischen Algorithmus
- **Forschungsschwerpunkte**
  - Analyse der allgemeinen Struktur einer Klasse von Algorithmen
  - Schematisierung durch Komponenten und Korrektheitsaxiome
  - Techniken zur Verfeinerung von Standardstrukturen

Divide & Conquer Synthese eines Sortieralgorithmus

- **Problemspezifikation**

  `FUNCTION sort(L:Seq(`$\mathbb{Z}$`)):Seq(`$\mathbb{Z}$`) RETURNS` S

  `SUCH THAT rearranges(`L`,`S`)` $\wedge$ `ordered(`S`)`

- **Grundstruktur von Divide & Conquer Algorithmen**

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

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

- **Komponenten für einen Sortieralgorithmus**

primitive	$\equiv$	λ`L. null?(L)`
Directly-solve	$\equiv$	λ`L. []`
Decompose	$\equiv$	λ`L. let a=L[|L|/2] in (`$L_{<a}$`,`$L_{=a}$`,`$L_{>a}$`)` ($L_{<a} \equiv$ `[x|x`$\in$`L`$\wedge$`x<`a`]`)
g	$\equiv$	`sort` $\times$ λ`S.S`
Compose	$\equiv$	$\lambda S_1,S_2,S_3.\ S_1\circ S_2\circ S_3$

- **Instantiierter Algorithmus**

  `FUNCTION sort(L:Seq(`$\mathbb{Z}$`)):Seq(`$\mathbb{Z}$`) RETURNS` S

  `SUCH THAT rearranges(`L`,`S`)` $\wedge$ `ordered(`S`)`

  $\equiv$ `if null?(L) then [] else let a = L[|L|/2]`

  `in let` L_1 `= [x|x`$\in$`L`$\wedge$`x<a]`
  `and` L_2 `= [x|x`$\in$`L`$\wedge$`x=a]`
  `and` L_3 `= [x|x`$\in$`L`$\wedge$`x>a]`
  `in sort(`L_1`)` $\circ$ L_2 $\circ$ `sort(`L_3`)`

Wissensbasierte Programmentwicklung: Literatur

Douglas R. Smith and Michael R. Lowry

Algorithm Theories and Design Tactics, Science of Computer Programming 14:305–321

Douglas R. Smith

KIDS — A Knowledge-Based Software Development system,
in: Michael R. Lowry and Robert D. McCartney, ed. Automating Software Design,
AAAI Press, 1991, p.483–514.

Douglas R. Smith

Top-Down Synthesis of Divide-and-Conquer Algorithms, AIJ 27:43–96, 1985

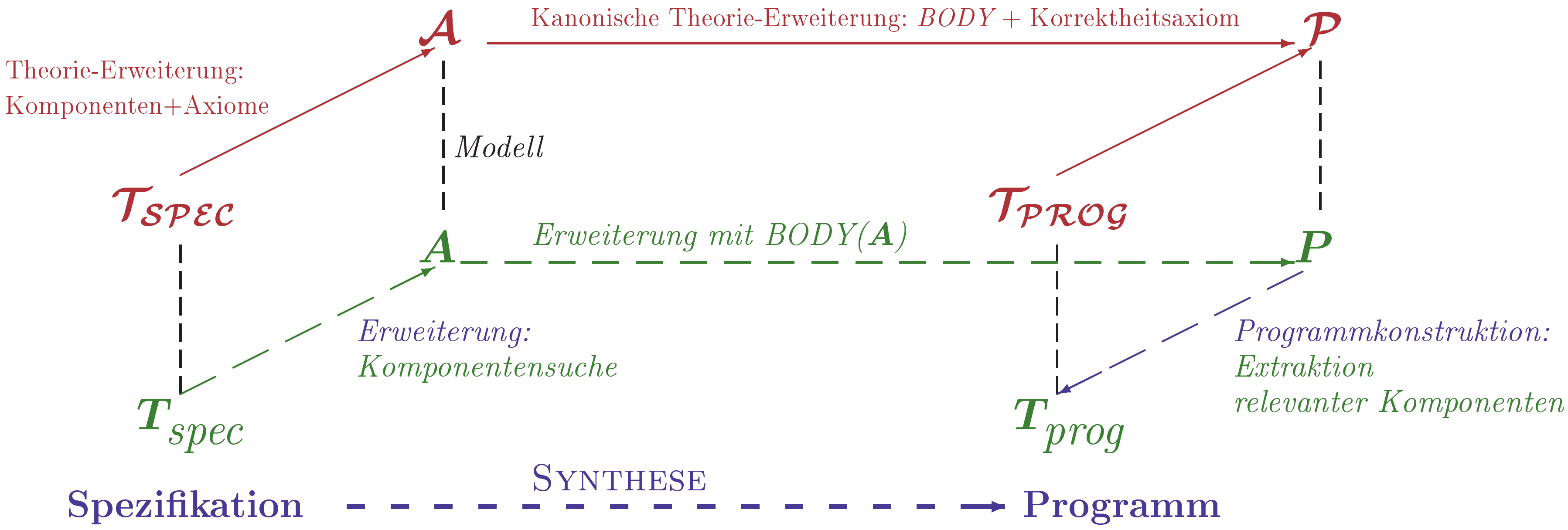
Douglas R. Smith

Structure and Design of Global Search Algorithms
Structure and Design of Problem Reduction Generators
Structure and Design of Dynamic Programming Algorithms
Technical Report, Kestrel Institute

Michael R. Lowry

Structure and Design of Local Search Algorithms
Proceedings, AAAI Workshop on Software Design, p.88–94

Algorithmenschemata als Algebraische Theorien



- **Programmentwicklung auf zwei Ebenen**
  - Allgemeine algorithmische Theorien
  - Konkrete Probleme als Instanzen der allgemeinen Theorien

- **Problemtheorie erweitert zu Programmtheorie**
  - Algorithmentheorie $\mathcal{A}$: ergänze Komponenten und Axiome eines Schemas
  - Programmtheorie $\mathcal{P}$: kanonische Erweiterung um Programmkörper
  - Synthese: Programme werden aus Programmtheorie extrahiert

Technische Grundbegriffe: Algebraische Theorien

- **Formale Theorie: Tripel $\mathcal{T} = (S, \Omega, Ax)$**
  - S: Menge von Sortennamen (Namen für Datentypen)
  - Ω: Familie von Operationsnamen (zusammen mit Typisierung)
  - Ax: Menge von Axiomen für Datentypen und Operationen
- **Theorie $\mathcal{T}_1$ erweitert $\mathcal{T}_2$**
  - Alle Sortennamen, Operationsnamen, Axiome von $\mathcal{T}_2$ existieren in $\mathcal{T}_1$
- **T Struktur für $\mathcal{T}$**
  - T ist Menge von Datentypen und Operationen, typisiert gemäß Ω

  T Modell für $\mathcal{T}$
  - T ist Struktur für $\mathcal{T}$, die alle Axiome aus $\mathcal{T}$ erfüllt
- **Struktur T_1 erweitert T_2**
  - Alle Datentypen und Operationen von T_2 existieren auch in T_1

    (gleiche Typisierung bezüglich $\mathcal{T}_2$!)

Algorithmenschemata als algebraische Theorien

- $\mathcal{T}_{\mathcal{SPEC}}$: **algebraische Theorie der Spezifikationen**
  – $(\ \{D,R\},\ \ \{I{:}D{\rightarrow}\mathbb{B},\ O{:}D{\times}R{\rightarrow}\mathbb{B}\},\ \ \emptyset\ )$

  $\mathcal{P}$ Problemtheorie:
  – $\mathcal{P}$ erweitert $\mathcal{T}_{\mathcal{SPEC}}$ ($\mathcal{P}$ enthält eine Programmspezifikation)

- $\mathcal{T}_{\mathcal{PROG}}$: **algebraische Theorie der Programme**
  – $(\ \{D,R\},\ \ \{I{:}D{\rightarrow}\mathbb{B},\ O{:}D{\times}R{\rightarrow}\mathbb{B},\ \mathit{body}{:}\,D{\not\rightarrow}R\ \},\ \ \{\forall x{:}D.\,I(x) \Rightarrow O(x,\mathit{body}(x))\}\ )$

  $\mathcal{P}$ Programmtheorie:
  – $\mathcal{P}$ erweitert $\mathcal{T}_{\mathcal{PROG}}$ ($\mathcal{P}$ enthält ein Programm)

- **$\mathcal{A}$ Algorithmentheorie:**
  – $\mathcal{A}$ ist Problemtheorie mit kanonischer Erweiterung zu Programmtheorie
  – Es gibt abstraktes Programmschema BODY mit der Eigenschaft
  $(\mathit{spec}_A,\ \mathit{BODY}(A))$ ist korrekt für jedes Modell A von $\mathcal{A}$
  (Für jedes Modell A existiert eine Standardlösung der Spezifikation spec_A)

$$\Downarrow$$

Synthese $\hat{=}$ Erweitere Programmtheorien zu Algorithmentheorien

DIVIDE & CONQUER SCHEMA ALS ALGORITHMENTHEORIE

$S_{D\&C}$: $\{D, R, D', R'\}$

$\Omega_{D\&C}$: $\{I\colon D\to\mathbb{B},\ O\colon D\times R\to\mathbb{B},\ O_D\colon D\times D'\times D\to\mathbb{B},\ I'\colon D'\to\mathbb{B},\ O'\colon D'\times R'\to\mathbb{B},$
$O_C\colon R'\times R\times R\to\mathbb{B},\ \succ\colon D\times D\to\mathbb{B},\ \mathit{Decompose}\colon D\to D'\times D,\ g\colon D'\to R',$
$\mathit{Compose}\colon R'\times R\to R,\ \mathit{Directly\text{-}solve}\colon D\to R,\ \mathit{primitive}\colon D\to\mathbb{B}$
$\}$

$Ax_{D\&C}$: $\{$`FUNCTION` $f_p(x{:}D){:}R$ `WHERE` $I[x]\wedge \mathit{primitive}[x]$ `RETURNS` y `SUCH THAT` $O[x,y]$
$\equiv\ \mathit{Directly\text{-}solve}[x]$ ist korrekt
$\vdots$
$\succ$ *ist wohlfundierte Ordnung auf D*
$\}$

Für $A = (\{D, R, D', R'\}, \{I, O, O_D, I', O', O_C, \succ,$
$\mathit{Decompose}, g, \mathit{Compose}, \mathit{Directly\text{-}solve}, \mathit{primitive}\})$

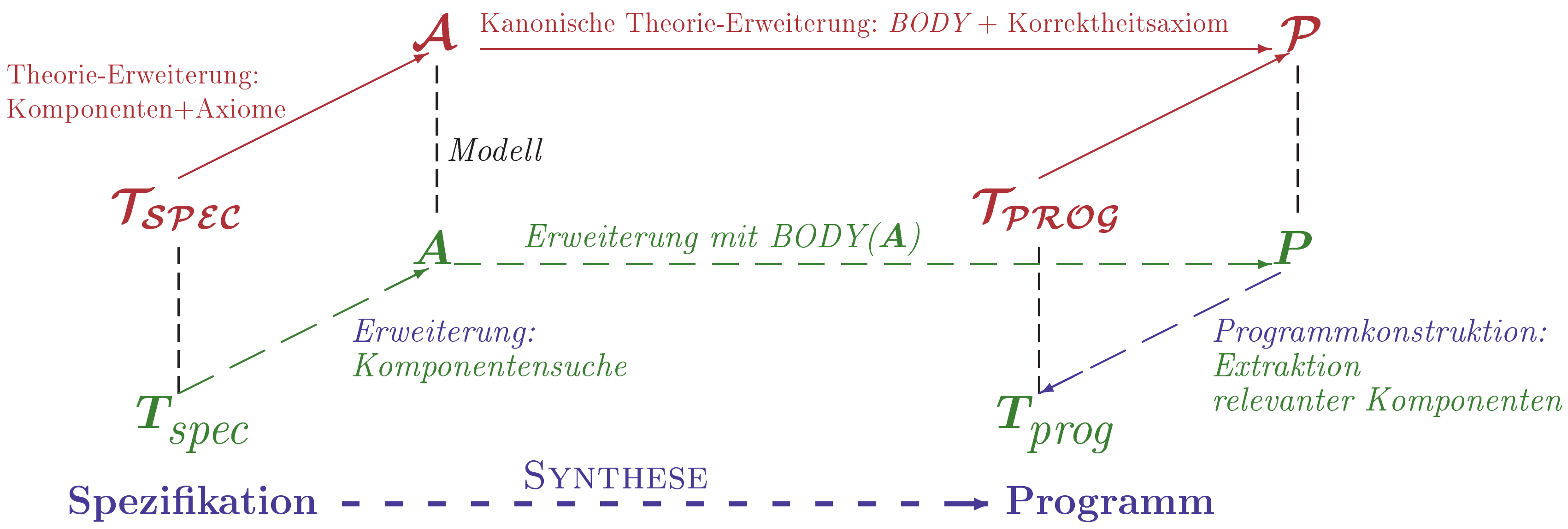
ist $\mathit{BODY(A)}\ \hat{=}$

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

$\mathit{BODY(A)}$ ist korrekt, wenn A alle Axiome in $Ax_{D\&C}$ erfüllt

Synthese mit Algorithmenschemata präzisiert

$spec = (D,R,I,O)$ ist erfüllbar, wenn es eine Algorithmentheorie $\mathcal{A}$ und ein Modell A für $\mathcal{A}$ gibt, welches die Struktur $T_{\text{spec}} = (\{D,R\},\{I,O\})$ erweitert



Methode:

- Wähle Algorithmentheorie $\mathcal{A}$,
- Erweitere T_{spec} zu Modell A von $\mathcal{A}$
- Instantiiere $BODY(A)$ und extrahiere Programmkomponenten

Algorithmenschema: Operator Match

Problemreduktion auf bekannte Algorithmentheorie

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

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

  – Eingabebedingung von *spec* (nach Transformation) stärker als *spec'*
  – Ausgabebedingung von *spec* schwächer
  – Nichtrekursive $\wedge$**-Reduktion** eines Problems

- **Reduzierbarkeit liefert Problemtransformation**

  – ***spec* $=(D,R,I,O)$ ist erfüllbar, wenn es eine Algorithmentheorie $\mathcal{A}$ und ein Modell A für $\mathcal{A}$ gibt, mit $spec \trianglelefteq spec_A$**
  – Beweis $spec \trianglelefteq spec_A$ liefert Substitutionen $\theta{:}D \rightarrow D_A$ und $\sigma{:}D_A \times R_A \rightarrow R$
  – $body(x) \;\equiv\; \sigma(x, body_A(\theta(x)))$ ist korrekter Algorithmus für *spec*

- **Syntheseverfahren**

  – Wähle Algorithmentheorie $\mathcal{A}$ und ein Modell A von $\mathcal{A}$
  – Beweise $spec \trianglelefteq spec_A$ und extrahiere Substitutionen σ und θ
  – Spezialisiere $body_A$ zu $\lambda x.\,\sigma(x, body_A(\theta(x)))$

Synthese mit Algorithmenschemata verfeinert

Integriere Operator Match in allgemeines Verfahren

$$\begin{tikzcd}[column sep=huge, row sep=huge]
D_A{,}I_A \arrow[rr, "body_A"] & & R_A{,}O_A \arrow[d, "\sigma"] \\
D{,}I \arrow[u, "\theta"] \arrow[rr, dashed, "body(x) = \sigma(\theta(x),\, body_A(\theta(x)))"] \arrow[urr, phantom, "\boxed{spec \trianglelefteq spec_A}"] & & R{,}O
\end{tikzcd}$$

- **Speichere generische Modelle von $\mathcal{A}$**
  - Standardkomponenten der Algorithmentheorie für wichtige Domänen
  - Axiome von $\mathcal{A}$ sind für diese Modelle ein für alle Mal bewiesen
- **Reduziere Spezifikation auf ein Modell A**
  - Auswahl passend zur Grunddomäne von *spec* (Liste, Menge, Bäume, ...)
  - Versuche, $spec \trianglelefteq spec_A$ automatisch zu beweisen
  - Spezialisierung von $body_A$ korrekt für extrahierte Substitutionen σ und θ

Keine weiteren Inferenzen erforderlich!

Problemreduktion für mengenwertige Spezifikationen

- ***spec* spezialisiert *spec′*** $\quad$ **(*spec* ≪ *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′*

  spec′* generalisiert *spec

- **Spezialisierung liefert Problemtransformation**
  – $\boldsymbol{spec} =$ `FUNCTION` $\boldsymbol{f(x{:}D)}$ `WHERE` $\boldsymbol{I[x]}$ `RETURNS` $\boldsymbol{\{y{:}R \mid O[x,y]\}}$
  ist erfüllbar, wenn es eine Algorithmentheorie $\mathcal{A}$ und ein Modell A für $\mathcal{A}$ gibt, mit *spec* ≪ $spec_A$
  – Beweis *spec* ≪ $spec_A$ liefert Substitution $\theta{:}D\to D_A$
  – $body(x) \;\equiv\; \{\, y \mid y \in body_A(\theta(x)) \wedge O(x,y) \,\}$ ist korrekt für *spec*

- **Syntheseverfahren**
  – Wähle Algorithmentheorie $\mathcal{A}$ und ein Modell A von $\mathcal{A}$
  – Beweise *spec* ≪ $spec_A$ und extrahiere σ
  – Spezialisiere $body_A$ zu $\lambda x.\{\, y \mid y \in body_A(\theta(x)) \wedge O(x,y) \,\}$

Integriere Generalisierung in allgemeines Verfahren

$$
\begin{tikzcd}
D_A,I_A \arrow[drr, "body_A"] & & \\
 & \boxed{spec \ll spec_A} & R_A,O_A \arrow[d] \\
D,I \arrow[uu, "\theta"] \arrow[rr, dashed, "body(x)=\{\, y \mid y \in body_A(\theta(x)) \wedge O(x,y)\}"] & & R,O
\end{tikzcd}
$$

- **Speichere generische Modelle von $\mathcal{A}$**
  - Standardkomponenten der Algorithmentheorie für wichtige Domänen
- **Reduziere Spezifikation auf ein Modell A**
  - Auswahl passend zur Grunddomäne von *spec* (Liste, Menge, Bäume, ...)
  - Versuche, $spec \ll spec_A$ automatisch zu beweisen
  - Spezialisierung von $body_A$ korrekt für extrahierte Substitution θ

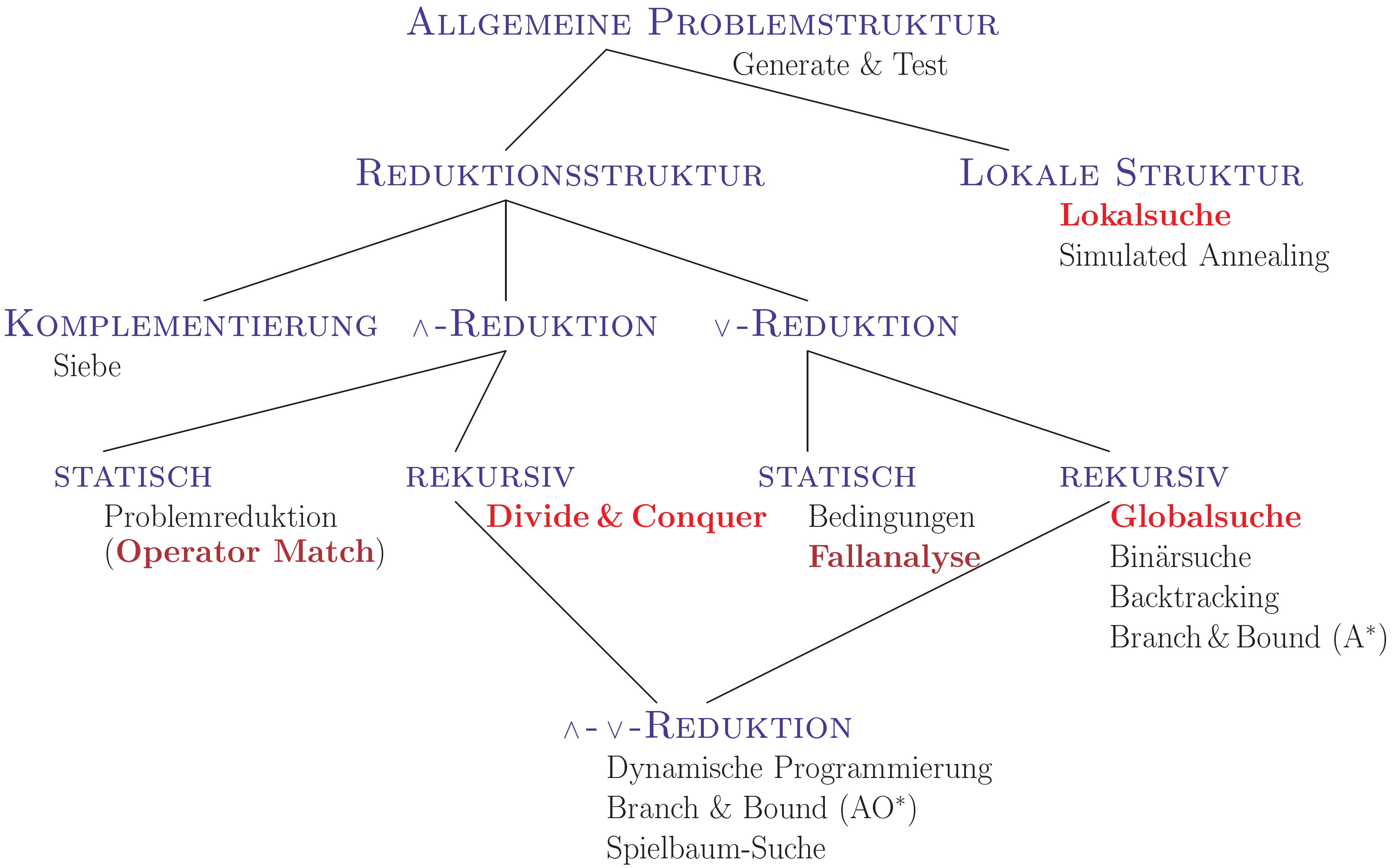
Keine weiteren Inferenzen erforderlich!
Optimierte Version für spezielle Algorithmentheorien möglich

Algorithmenschema: Fallunterscheidung

Zerlegung in bekannte Lösungen

- ***spec* zerlegbar in $spec_1..spec_n$** $\quad$ ($spec = \cup_i spec_i$)
  $\forall x{:}D\,.\,I[x] \Rightarrow I_1[x] \vee .. \vee I_n[x] \;\wedge\; \forall i{\leq}n\,.\forall y{:}R\,.\,O_i[x,y] \Rightarrow O[x,y]$)
  - Eingabebedingung zerlegbar in Eingabebedingungen der $spec_i$
  - Ausgabebedingung der $spec_i$ stärker
  - Nichtrekursive $\vee$-**Reduktion** eines Problems
- **Zerlegbarkeit liefert Fallunterscheidung**
  - ***spec* ist erfüllbar, wenn es Algorithmentheorien $\mathcal{A}_1..\mathcal{A}_n$ Modelle A_i für $\mathcal{A}_i$ gibt, so daß $spec = \cup_i spec_{A_i}$**
  - $body(x) \equiv$ `if` $I_1[x]$ `then` $body_{A_1}[x]$...`else` $body_{A_n}[x]$ ist korrekt für *spec*
- **Syntheseverfahren "Derived Antecedants"**
  - Wähle Algorithmentheorie $\mathcal{A}_1$ und ein Modell A_1
  - Bestimme Eingabebedingung I_1 für die $\mathcal{A}_1$ "korrekt arbeitet"
  - Wiederhole Verfahren für $spec' = (D, R, I \wedge \neg I_1, O)$ bis Zerlegung komplett
  - Setze Lösungen durch Fallunterscheidung zusammen

Hierarchie algorithmischer Strukturen



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