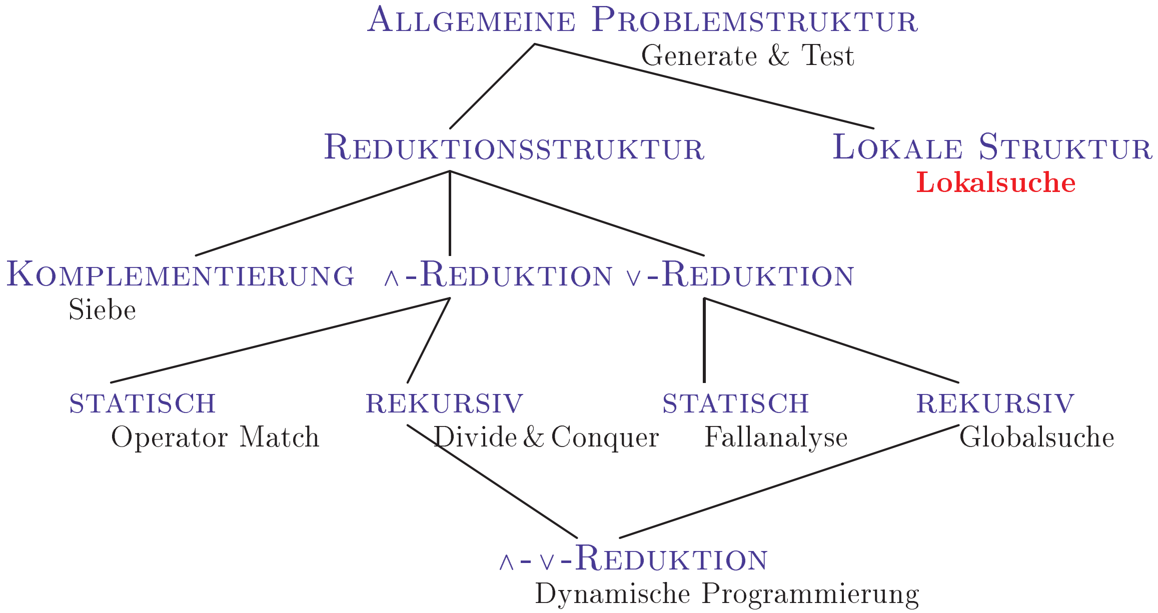
# Lokalsuch-Algorithmen

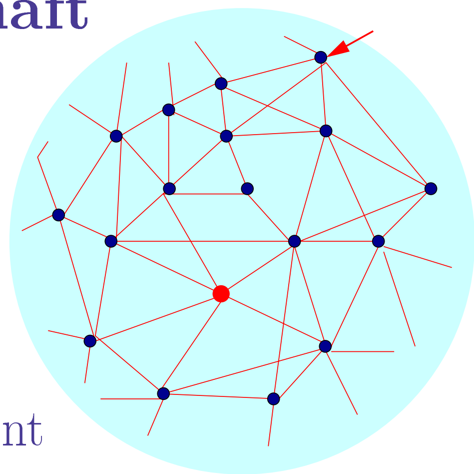



- **Problemlösung durch kleine Veränderungen**
  – Bewertung der Qualität von Elementen des Bildbereichs
  – Qualität einer (Teil-)Lösung wird schrittweise verbessert
  – Gut für Optimierungsprobleme (Travelling Salesman, Scheduling, ...)
- **Lösungsverfahren: Hillclimbing**
  – Beginne irgendwo im Lösungsraum
  – Durchsuche lokale Nachbarschaft bis keine bessere Lösung zu finden

# Lokalsuch-Algorithmen: Generelle Idee

- **Optimierung als Minimierung von Kosten**
  - Spezifikation des Problems besitzt viele mögliche "Lösungen"
  - Lösungen werden bewertet nach Nutzen, Kosten, Korrekheitsgrad, ...
  - Gesucht ist Lösung mit optimaler (o.B.d.A. minimaler) Bewertung
  - Exakte Optimierung oft $\mathcal{NP}$-vollständig

- **Lösungsverfahren durchsucht Nachbarschaft**
  - Übergang auf Nachbarn, solange Verbesserungen möglich
  - Verfahren endet in lokalen Optima
  - Nachbarschaftstruktur entscheidet über Güte der Lösung
    - · Zu fein $\Rightarrow$ Verfahren führt nicht zu globalem Optimum
    - · Zu grob $\Rightarrow$ Suche + Test auf lokale Optimalität ineffizient
  - **Bestimmung einer guten Nachbarschaftstruktur ist wichtig**



# Grundschema von Lokalsuch-Algorithmen

$$
\begin{array}{l}
\texttt{FUNCTION}\ f_{opt}(x{:}D){:}R\ \texttt{WHERE}\ I[x]\ \texttt{RETURNS}\ y \\
\quad \texttt{SUCH THAT}\ O[x,y]\ \wedge\ \forall t{:}R.\ (O[x,t]\ \Rightarrow\ c[x,y]{\leq}c[x,t]) \\
\equiv\ \texttt{let}\ f_{LS}(x,z) \\
\qquad = \texttt{if}\ \forall t\in N[x,z].\ (O[x,t]\ \Rightarrow\ c[x,z]{\leq}c[x,t])\ \texttt{then}\ z \\
\qquad\qquad \texttt{else}\ f_{LS}(x,\texttt{arb}(\{t \mid t\in N[x,z] \wedge O[x,t] \wedge c[x,t]{<}c[x,z]\})) \\
\quad \texttt{in}\ f_{LS}(x,Init[x])
\end{array}
$$

- **3 zentrale Komponenten der Algorithmentheorie**
  - $Init$: $D \to R$ — Initiallösung für Basisspezifikation $(D,R,I,O)$
  - $c$: $R \to \mathcal{R}$ — Kostenfunktion auf geordnetem Kostenraum $(\mathcal{R},\leq)$
    Zusatzspezifikation des Optimierungsproblems
  - $N$: $D \times R \to \texttt{Set}(R)$ — Nachbarschaftsstruktur
    Suchraumbeschreibung für lokale Variationen

# Korrektheit des Lokalsuch-Schemas

$$
\begin{array}{l}
\texttt{FUNCTION } f_{opt}(x{:}D){:}R \texttt{ WHERE } I[x] \texttt{ RETURNS } y \\
\quad \texttt{SUCH THAT } O[x,y] \;\wedge\; \forall t{:}R.\,(O[x,t] \;\Rightarrow\; c[x,y]{\le}c[x,t]) \\
\equiv\; \texttt{let } f_{LS}(x,z) = \texttt{if } \forall t{\in}N[x,z].\,(O[x,t] \;\Rightarrow\; c[x,z]{\le}c[x,t]) \texttt{ then } z \\
\qquad\qquad\qquad\qquad\qquad \texttt{else } f_{LS}(x, \texttt{arb}(\{t \mid t{\in}N[x,z] \wedge O[x,t] \wedge c[x,t]{<}c[x,z]\})) \\
\quad \texttt{in } f_{LS}(x, Init[x])
\end{array}
$$

**ist korrekt, wenn 4 Axiome erfüllt sind**

---

1. **$Init[x]$ berechnet gültige Initiallösung für $O$**

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

2. **Nachbarschaftsstruktur $N$ ist reflexiv**

   $\forall x{:}D.\forall y{:}R.\;\; I[x] \wedge O[x,y] \;\Rightarrow\; y \in N[x,y]$

3. **Lokale Optima sind exakt** $\mapsto$ Optimale Algorithmen

   $$\begin{array}{l} \forall x{:}D.\forall y{:}R.\;\; I[x] \wedge O[x,y] \;\Rightarrow\; (\forall t \in N[x,y].\, O[x,t] \Rightarrow c[x,y]{\le}c[x,t]) \\ \qquad\qquad\qquad\qquad\qquad\;\; \Rightarrow\; \forall z{:}R.\,(O[x,z] \Rightarrow c[x,y]{\le}c[x,z]) \end{array}$$

4. **Alle gültigen Lösungen sind endlich erreichbar**

   $\forall x{:}D.\forall y,z{:}R.\;\; I[x] \wedge O[x,y] \wedge O[x,z] \;\Rightarrow\; \exists k{:}\mathbb{N}.\, z \in N_O^k[x,y]$

---

$N_O^0[x,y] = \{y\} \quad N_O^{k+1}[x,y] = \bigcup\{\, N^k[x,t] \mid t \in N[x,y] \wedge O(x,t) \,\}$

# Lokalsuch-Schema: Korrektheitsbeweis

- **Abspalten und Spezifikation der Hilfsfunktion $f_{ls}$**

  $\texttt{FUNCTION } f_{opt}(x\!:\!D)\!:\!R \texttt{ WHERE } I[x] \texttt{ RETURNS } y$
  $\quad\texttt{SUCH THAT } O[x,y] \;\wedge\; \forall t\!:\!R.\,(O[x,t] \;\Rightarrow\; c[x,y]\leq c[x,t])$
  $\equiv\; f_{LS}(x, Init[x])$

  $\texttt{FUNCTION } f_{ls}(x,z\!:\!D\times R)\!:\!R \texttt{ WHERE } I[x] \;\wedge\; O[x,z] \texttt{ RETURNS } y$
  $\quad\texttt{SUCH THAT } O[x,y] \;\wedge\; \forall t\in N[x,y].\,(O[x,t] \;\Rightarrow\; c[x,y]\leq c[x,t])$
  $\equiv\; \texttt{if } \forall t\in N[x,z].\,(O[x,t] \;\Rightarrow\; c[x,z]\leq c[x,t]) \texttt{ then } z$
  $\quad\quad \texttt{else } f_{LS}(x, \texttt{arb}(\{t \mid t\in N[x,z]\wedge O[x,t]\wedge c[x,t]<c[x,z]\}))$

- **Korrektheit von $f_{opt}$ folgt aus der von $f_{ls}$ mit Axiomen 1 & 3**
  - Für den Startwert $z = Init[x]$ gilt $O[x,z]$
  - Für $y = f_{LS}(x,z)$ gilt $O[x,y] \;\wedge\; \forall t\in N[x,y].\,(O[x,t] \;\Rightarrow\; c[x,y]\leq c[x,t])$
  - Mit Axiom 3 folgt $\forall t\!:\!R.\,(O[x,t] \;\Rightarrow\; c[x,y]\leq c[x,t])$

- **Partielle Korrektheit von $f_{ls}$ folgt aus Programmkörper**
  - Hält $f_{ls}$ mit Ausgabe $z$, so gilt $\forall t\in N[x,z].\,(O[x,t] \;\Rightarrow\; c[x,z]\leq c[x,t])$

- **Terminierung von $f_{ls}$ folgt aus Ordnung $(\mathcal{R},\leq)$ und Axiom 4**

# Sortieren mit Lokalsuchalgorithmen

- **Formuliere Sortierung als Ordnungsoptimierung**
  - Einfache Basisspezifikation $O[L,S]$ = `rearranges`$(L,S)$
  - Kostenfunktion $c[L,S]$ = `#`$_>(S)$: Anzahl der Fehlstellungen $S_i{>}S_{i+1}$
  - Es gilt `#`$_>(S)\geq$`0` und `#`$_>(S)=0 \;\Rightarrow\;$ `ordered`$(S)$
  - Spezifikation `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) WHERE true RETURNS S`
    `SUCH THAT rearranges(L,S)` $\wedge$
    $\forall$`S':Seq(ℤ). rearranges(L,S')` $\Rightarrow$ `#`$_>$`(S)`$\leq$`#`$_>$`(S')`
- **Lokalsuchalgorithmus**
  - Initiallösung $Init[L] = L$ $\mapsto$ Axiom 1
  - Nachbarschaft $N[S,S']$ = `permute`$_{i,j}(S,S')$: Vertauschen von $S_i$ und $S_j$
    - Vertauschen ist reflexiv, Lokale Minima sind exakt $\mapsto$ Axiom 2,3
    - Alle Umordnungen erreichbar durch iteratives Vertauschen $\mapsto$ Axiom 4
  - Ergibt Sortieren durch beliebiges Austauschen von Elementen
    - Ineffizient, da zu viele Nachbarn zu prüfen (vermutlich $\mathcal{O}(n^3)$)
- **Lokalsuchalgorithmus mit kleinerer Nachbarschaft**
  - Restriktion auf benachbarte Komponenten
  - Nachbarschaft $N[S,S']$ = `perm`$_i(S,S')$ = `permute`$_{i,i+1}(S,S')$
  - Ergibt Bubblesort (nach algorithmischer Optimierung)

**Lokalsuche $\hat{=}$ Nachbarschaft + Suchfilter**

- **Bestimme effektive Nachbarschaftsstruktur**
  - Beschreibe Nachbarschaft als Perturbation (Verwirbelung)
  - Inkrementelle Veränderung von Werten aus $R$
    - Numerik: $\delta$-Vektoren, Kombinatorik: Austausch von Komponenten
  - Formalisiert als $N[x,y] \hat{=} \{ \mathit{Action}[i,j,x,y] \mid i,j \in \pi[x,y] \}$
    - Änderungsaktion $\mathit{Action}[i,j,x,y]$ modifiziert Lösungspunkt $(x,y) \in D \times R$
    - Parameter $i,j \in \pi[x,y]$ sind minimale Bestandteile von $(x,y)$

- **Bestimme effiziente Suchfilter**
  - Optimiere Nachbarschaftsstruktur durch frühzeitiges Abschneiden
    - Feasibility Constraint für $O[x,y]$
    - Optimality Constraint für $\forall t \in N[x,z].\,(O[x,t] \;\Rightarrow\; c[x,z] \leq c[x,t])$

# Wissensbasierte Synthese von Lokalsuch-Algorithmen

## Spezialisiere vorformuliertes Programmierwissen

- **Lokalsuchtheorie: allgemeine Suchstruktur für $R$**
  - Vorgefertigte Nachbarschaftsstruktur, die Axiome 2–4 erfüllt
  - Formalisiert als Objekt $\mathcal{L} = (D,R,I,O,\pi,Action)$
  - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen

- **Spezialisierungsmechanismen**
  - Synthetisiere Initiallösung $Init[x]$ für Spezifikation $spec = (D,R,I,O)$
  - Wähle $\mathcal{L}$ für Bildbereich $R$, so daß $spec \ll spec_{\mathcal{L}}$ beweisbar
  - Extrahiere Substitution $\theta{:}D{\rightarrow}D_{\mathcal{L}}$ und spezialisiere $\mathcal{L}$ mit $\theta$
    - Ergibt Nachbarschaftsstruktur $N$ für Problemstellung
  - Generiere Filter zur Beschränkung auf optimale Lösungen

- **Eventuell Verzicht auf Exaktheit**
  - Liefert effizienteren, aber suboptimalen Algorithmus

# Standard-Lokalsuchtheorien

- **Umordnung von Folgen**
  - Suche $\hat{=}$ Permutation einzelner Elemente einer Folge
  - Änderungsparameter: Indizes der Eingabeliste `L`
  - Perturbation: Vertauschung zweier Elemente einer Ausgabeliste `S`

$$
\texttt{LS\_seq\_re}(\alpha) \equiv
\begin{array}{lll}
D & \mapsto & \texttt{Seq}(\alpha) \\
R & \mapsto & \texttt{Seq}(\alpha) \\
I & \mapsto & \lambda\texttt{L}.\,\texttt{true} \\
O & \mapsto & \lambda\texttt{L,S}.\,\texttt{rearranges(L,S)} \\
\pi & \mapsto & \lambda\texttt{L,S}.\,\texttt{(domain(S),domain(S))} \\
\mathit{Action} & \mapsto & \lambda\texttt{i,j,L,S}.\,[\texttt{S}_{(i\leftrightarrow j)(k)} \mid \texttt{k} \in \texttt{domain(S)}]
\end{array}
$$

$\texttt{(i}\leftrightarrow\texttt{j)(k)}$ $\hat{=}$ `if k=i then j  else if k=j then i  else k`

- **Teilmengen fester Größe**
  - Suche $\hat{=}$ Austausch einzelner Elemente einer Menge
  - Änderungsparameter: Elemente der Ein- und Ausgabemenge
  - Perturbation: Austausch zweier Elemente in Ausgabemenge

$$
\texttt{LS\_subsets}(\alpha) \equiv
\begin{array}{lll}
D & \mapsto & \texttt{Set}(\alpha)\times\mathbb{N} \\
R & \mapsto & \texttt{Set}(\alpha) \\
I & \mapsto & \lambda\texttt{S,m}.\,\texttt{m}\leq\texttt{|S|} \\
O & \mapsto & \lambda\texttt{S,m,S'}.\,\texttt{S'}\subseteq\texttt{S} \;\wedge\; \texttt{|S'|=m} \\
\pi & \mapsto & \lambda\texttt{S,m,S'}.\,\texttt{(S}\backslash\texttt{S', S')} \\
\mathit{Action} & \mapsto & \lambda\texttt{i,j,S,m,S'}.\,\texttt{S' = (S}\cup\{\texttt{i}\}\texttt{)-y}
\end{array}
$$

# Synthesestrategie für Lokalsuch-Algorithmen

**Start:** FUNCTION $f_{opt}(x{:}D){:}R$ WHERE $I[x]$ RETURNS $y$
SUCH THAT $O[x,y] \land \forall t{:}R.\ (O[x,t] \Rightarrow c[x,y]{\leq}c[x,t])$

**1. Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp $R$ aus Wissensbank**

**2. Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$**

– Extrahiere Substitution $\theta$ und setze $\mathcal{L}_\theta = (D, R, I, O, \pi_\theta, Action_\theta)$

**3. Generiere Lösungs-Filter $FC$ für $\mathcal{L}_\theta$** *(Feasibility Constraint)*

– Filter eliminiert Punkte, die keine gültigen Lösungen bzgl. $O$ sind

· $I[x] \land O[x,y] \land O[x, Action[i,j,\theta(x),y]] \Rightarrow FC[i,j,x,y]$

– Vorwärtsinferenz: Vereinfachung der linken Seite ergibt $FC$

**4. Generiere Optimalitäts-Filter $OC$ für $\mathcal{L}_\theta$** *(Optimality Constraint)*

– Filter eliminiert kostenungünstigere Punkte

· $I[x] \land O[x,y] \land O[x, Action[i,j,\theta(x),y]] \land c[x,y]{\leq}c[x, Action[i,j,\theta(x),y]]$
$\Rightarrow OC[i,j,x,y]$

– Lokale Optima müssen Bedingung $\forall i,j \in \pi(x,y).\ OC(i,j,x,y)$ erfüllen

**5. Synthetisiere Initiallösung $Init$ für Spezifikation**

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

## Synthesestrategie für Lokalsuch-Algorithmen

**6. Instantiiere Schema für suboptimale Lokalsuch Algorithmen**

$$\texttt{FUNCTION}\ f_{opt}(x{:}D){:}R\ \texttt{WHERE}\ I[x]\ \texttt{RETURNS}\ y$$
$$\quad \texttt{SUCH THAT}\ O[x,y]\ \wedge\ \forall t{:}R.\ (O[x,t]\ \Rightarrow\ c[x,y]\leq c[x,t])$$
$$\equiv\ f_{LS}(x, Init[x])$$

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

$$\texttt{FUNCTION}\ f_{ls}(x,z{:}D\times R){:}R\ \texttt{WHERE}\ I[x]\ \wedge\ O[x,z]\ \texttt{RETURNS}\ y$$
$$\quad \texttt{SUCH THAT}\ O[x,y]\ \wedge\ \forall t\in N[x,y].\ (O[x,t]\ \Rightarrow\ c[x,y]\leq c[x,t])$$
$$\equiv\ \texttt{if}\ \forall i,j\in\pi[\theta(x),z].\ FC[i,j,x,z]\wedge O[x,Action[i,j,\theta(x),y]]$$
$$\Rightarrow\ OC[i,j,x,z]\wedge c[x,z]\leq c[x,Action[i,j,\theta(x),y]]\ \texttt{then z}$$
$$\texttt{else}\ f_{LS}(x,\ \texttt{arb}(\{\ Action[i,j,\theta(x),y]\ \mid\ i,j\in\pi[\theta(x),z].$$
$$\wedge\ FC[i,j,x,z]\wedge O[x,Action[i,j,\theta(x),y]]$$
$$\wedge\ \neg OC[i,j,x,z]\vee c[x,z]>c[x,Action[i,j,\theta(x),y]]\ \}))$$

Lösungs- und Optimalitätstests notwendig für Korrektheit

Algorithmus testet nur Parameter, welche die Filter $FC$ und $OC$ passieren

Effiziente Abarbeitung nutzt **andthen/orelse** Semantik von $\wedge$ und $\vee$

**7. Generiere Bedingungen für Exaktheit** *(Global Optimality Constraint)*

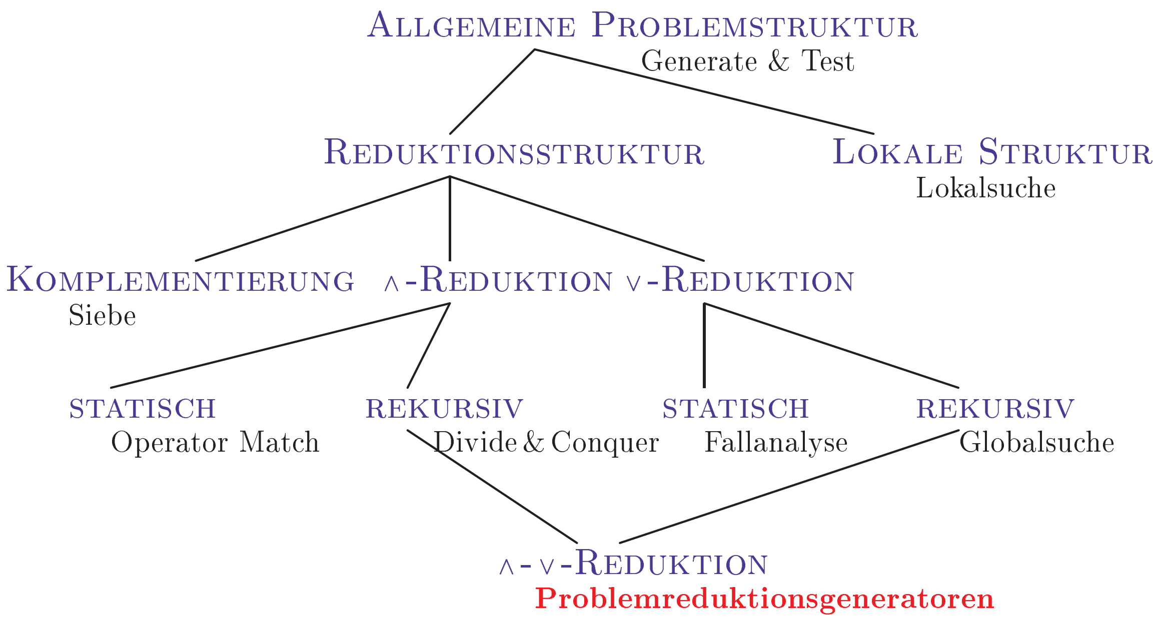
– Zusätzlicher (optionaler) Filter eliminiert suboptimale Lösungen

· $I[x]\wedge O[x,y]\wedge O[x,Action[i,j,\theta(x),y]]\wedge c[x,y]\leq c[x,Action[i,j,\theta(x),y]]$
$\wedge\forall t{:}R.\ O[x,t]\Rightarrow c[x,y]\leq c[x,t]\ \Rightarrow\ GOC[x,y]$

# Lokalsuche: Anwendungsbeispiele

- **Minimal Spanning Tree**
  - Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände,…)
  - Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar
  - Initialwert: Erzeuge spannenden Baum
  - Perturbation: Ergänze neue Kante, entferne eine andere
  - Feasibility Constraint: Entfernte Kante muß redundant sein
  - Optimality Constraint: Hinzugefügte Kante ist teurer als bisheriger Weg
- **Lineare Programmierung**
  - Minimiere lineare Funktion $f(x_1,..x_n)=\Sigma c_i x_i$ unter Restriktionen $A_j[x_1,..x_n]$
    Standard Darstellung: Minimiere $c \star x$ unter $A \star x = b \wedge \forall i \leq n. x_i \geq 0$
  - Initiallösung: Setze $x_{m+1},..x_n := 0$ und löse $A_{1..m} \star x_{1..m}$ mit $\forall i \leq m. x_i > 0$
    durch Gauß-Verfahren
  - Perturbation: Setze ein $\mathtt{x}_i\mathtt{:=0}$, wähle neues $\mathtt{x}_j \neq \mathtt{0}$
  - Feasibility Constraint: Alte + neue $x$-Komponenten nach Lösung positiv
  - Optimality Constraint: Relative Kosten steigen durch Veränderung

# Problemreduktionsgeneratoren




- **∨-∧-Reduktion von Problemen**
  - Problem besitzt mehrere Lösungen
  - Gesamtlösung ist Summe unabhängiger Einzellösungen (∨-Reduktion)
  - Einzellösungen aus Teillösungen zusammengesetzt (∧-Reduktion)
  - Verallgemeinert Dynamisches Programmieren, Spielbaumsuche, ...
- **Synthese ähnlich zu Divide & Conquer Techniken**

# Problemreduktionsgeneratoren: Grundidee

- **Verallgemeinertes Divide & Conquer Schema**
  - Unabhängige Divide & Conquer Algorithmen für jede Einzellösung
    - Dekompositionen und Kompositionen verschieden
    - Teilprobleme können einander überlappen
    - Basisfall primitiver Eingaben wird einfaches Divide & Conquer
  - Lösung durch Vereinigung aller Einzellösungen berechnen

- **Allgemeines Algorithmenschema**

  $\texttt{FUNCTION}\ \boldsymbol{f}(x{:}D)\ \texttt{WHERE}\ I[x]\ \texttt{RETURNS}\ \{y{:}R\,|\,O[x,y]\}$
  $\equiv\ \bigcup_{i,k}(Compose_i \circ (f_{i_1}\times..\times f_{i_k}) \circ Decompose_i)\,(x)$

- **4 zentrale Komponenten der Algorithmentheorie**
  - $Decompose_i$: $D \rightarrow D_{i_1}\times..\times D_{i_k}$ — Aufspalten der Eingabe in Teilprobleme
  - Hilfsfunktionen $\boldsymbol{f_{i_j}}$: $D_{i_j} \rightarrow R_{i_j}$ — evtl. rekursiver Aufruf von $\boldsymbol{f}$
  - $Compose_i$: $R_{i_1}\times..\times R_{i_k} \rightarrow R$ — Zusammensetzen der Teillösungen
  - Wohlfundierte Ordnung $\succ$ für Terminierungsgarantiee
  - **Erweitertes Strong Problem Reduction Principle**

# Problemreduktionsgeneratoren: Korrektheit

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

$\equiv \bigcup_{i,k}(Compose_i \circ (f_{i_1}\times..\times f_{i_k}) \circ Decompose_i)(x)$

**ist korrekt, wenn 5 Axiome erfüllt sind**

---

1. **$O$ rekursiv zerlegbar in $O_{D_i}$, $O_{i_1}\times..\times O_{i_k}$ und $O_{C_i}$** *(SPRP)*

   $O[x,z] \Leftrightarrow \exists i{:}\mathbb{N}, \bar{y}_i{:}D_{i_1}\times..D_{i_k}, \bar{w}_i{:}R_{i_1}\times..R_{i_k}.\ O_{D_i}[x,\bar{y}_i] \wedge O_{i_1,..i_k}[\bar{y}_i,\bar{w}_i] \wedge O_{C_i}[\bar{w}_i,z]$

   $O_{i_1,..i_k}[y_{i_1},..y_{i_k}, w_{i_1},..w_{i_k}] \equiv O_{i_1}(y_{i_1},w_{i_1}) \wedge .. \wedge O_{i_k}(y_{i_k},w_{i_k})$

2. **Dekompositionen erfüllen $O_{D_i}$ und 'verkleinern' Problem**

   FUNCTION $f_{d_i}(x{:}D)$ WHERE $I[x]$ RETURNS $\{\bar{y}_i{:}D_{i_1}\times..D_{i_k} \mid O_{D_i}[x,\bar{y}_i] \wedge x \succ \bar{y}_i \wedge I_{i_1,..i_k}[\bar{y}_i]\}$

   $x \succ \bar{y}_i \ \equiv\ x \succ y_{i_j}$ für alle $j$ mit $D_{i_j}{=}D$

3. **Hilfsfunktionen $f_{i_j}$ erfüllen $O_{i_j}$**

   FUNCTION $f_{i_j}(y_{i_j}{:}D_{i_j})$ WHERE $I_{i_j}[y_{i_j}]$ RETURNS $\{w_{i_j}{:}R_{i_j} \mid O_{i_j}[y_{i_j},w_{i_j}]\}$

4. **Kompositionen erfüllen $O_{C_i}$**

   FUNCTION $f_{c_i}(\bar{w}_i{:}R_{i_1}\times..\times R_{i_k})$ WHERE true RETURNS $\{z_i{:}R \mid O_{C_i}[\bar{w}_i,z_i]\}$

5. **Verkleinerungsrelation $\succ$ ist wohlfundierte Ordnung auf $D$**

# Synthese von Problemreduktionsgeneratoren

- **Aufspaltung des Reduktionsprizips in 2 Axiome**
  - **Starke Korrektheit** bzgl. Komposition und Dekomposition

$$\forall i{:}\mathbb{N}, x{:}D, \bar{y}_i{:}D_{i_1}\times..D_{i_k}, \bar{w}_i{:}R_{i_1}\times..R_{i_k}, z{:}R. \qquad (1)$$
$$I[x] \wedge I_{i_1,..i_k}[\bar{y}_i] \wedge O_{i_1,..i_k}[\bar{y}_i, \bar{w}_i] \wedge O_{C_i}[\bar{w}_i, z] \;\Rightarrow\; (O_{D_i}[x, \bar{y}_i] \Leftrightarrow O[x, z])$$

$$\forall i{:}\mathbb{N}, x{:}D, \bar{y}_i{:}D_{i_1}\times..D_{i_k}, \bar{w}_i{:}R_{i_1}\times..R_{i_k}, z{:}R. \qquad (2)$$
$$I[x] \wedge I_{i_1,..i_k}[\bar{y}_i] \wedge O_{D_i}[x, \bar{y}_i] \wedge O_{i_1,..i_k}[\bar{y}_i, \bar{w}_i] \;\Rightarrow\; (O_{C_i}[\bar{w}_i, z] \Leftrightarrow O[x, z])$$

  - **Vollständigkeit** bzgl. Komposition

$$\forall i{:}\mathbb{N}, x{:}D, \bar{w}_i{:}R_{i_1}\times..R_{i_k}, z{:}R.$$
$$I[x] \wedge O[x, z] \wedge O_{C_i}[\bar{w}_i, z] \;\Rightarrow\; \exists \bar{y}_i{:}D_{i_1}\times..D_{i_k}.\, (I_{i_1,..i_k}[\bar{y}_i] \wedge O_{i_1,..i_k}[\bar{y}_i, \bar{w}_i])$$

- **Strategie analog zu Divide & Conquer Verfahren**
  1. Wähle $Decompose_i$ aus Wissensbank
  2. Konstruiere Hilfsfunktionen $f_{i_j}$ ($id$ oder rekursiver Aufruf von $f$)
  3. Konstruiere Dekompositionen $Compose_i$ mit Korrektheitsaxiom 1
  4. Wähle $\succ$ aus der Wissensbank und verifiziere $Decompose_i$
  5. Verifiziere Vollständigkeitsaxiom
  6. Instantiiere Algorithmenschema