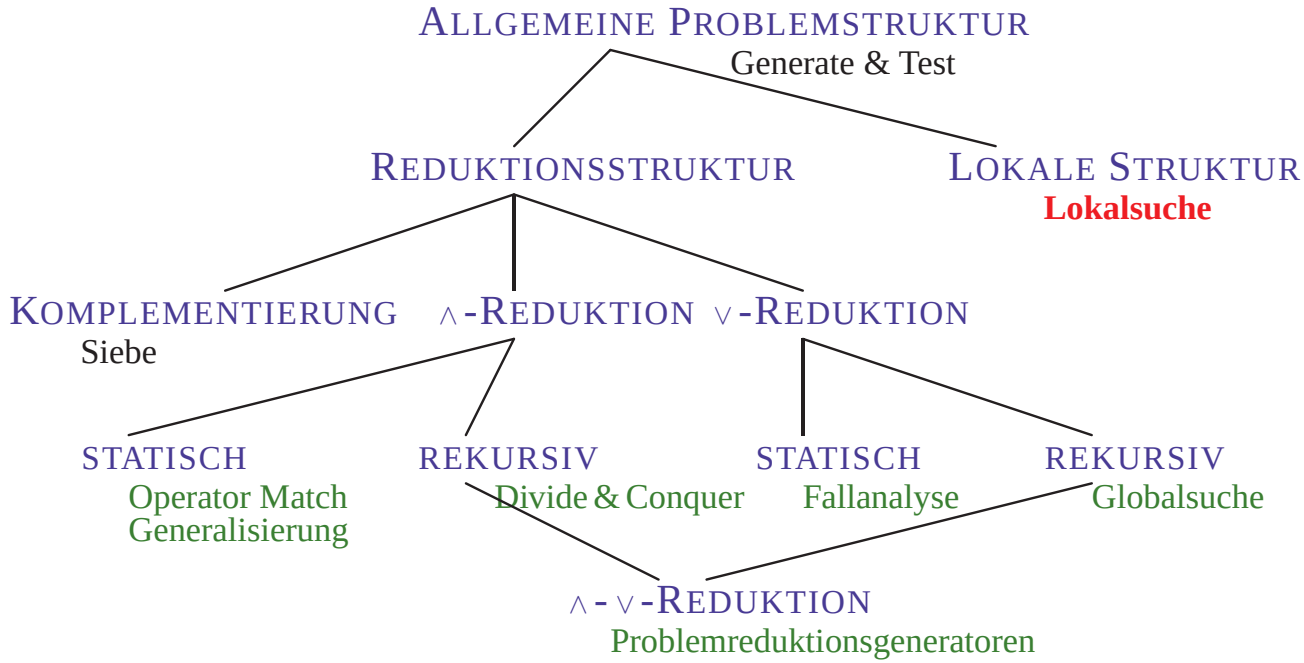
Automatisierte Logik und Programmierung

Einheit 22

Lokalsuch-Algorithmen

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

Lokalsuch-Algorithmen



- **Problemlösung durch kleine (lokale) Veränderungen**
  - Beginne Suche irgendwo im Raum akzeptabler Lösungen
  - Versuche Qualität der (Teil-)Lösung schrittweise zu verbessern
  - Gut für Optimierungsprobleme (Travelling Salesman, Scheduling, ...)
  - Erfordert Bewertung der Qualität von Elementen des Bildbereichs
- **Anderersartiger Ansatz ohne Problemreduktion**
  - Schwerpunkt liegt auf Durchsuchen einer lokale Nachbarschaft
  - Methode des Hillclimbing: Suchen, solange es noch aufwärts geht

Ein typischer Lokalsuch-Algorithmus

Minimiere $\Sigma_{i=1}^{n} c_i y_i$ **wobei** $\Sigma_{j=1}^{m} A_{i,j} y_i \leq b_j$ **und** $y_i \geq 0$

Unterbestimmtes System: Anzahl der Gleichungen (m) kleiner als Zahl der Variablen (n)

Ein typischer Lokalsuch-Algorithmus

Minimiere $\Sigma_{i=1}^{n} c_i y_i$ wobei $\Sigma_{j=1}^{m} A_{i,j} y_i \leq b_j$ und $y_i \geq 0$

Unterbestimmtes System: Anzahl der Gleichungen (m) kleiner als Zahl der Variablen (n)

- **Spezifikation als lineares Optimierungsproblem**

  FUNCTION LP(A, b:Seq(Seq($\mathbb{Q}^+$))$\times$Seq($\mathbb{Q}^+$)):Seq($\mathbb{Q}^+$)
  RETURNS y
  SUCH THAT $\Sigma A_{i,j} y_i \leq b_j \wedge \forall t$:Seq($\mathbb{Q}^+$). $\Sigma A_{i,j} t_i \leq b_j \Rightarrow \Sigma c_i y_i \leq \Sigma c_i t_i$

Ein typischer Lokalsuch-Algorithmus

Minimiere $\Sigma_{i=1}^{n} c_i y_i$ wobei $\Sigma_{j=1}^{m} A_{i,j} y_i \leq b_j$ und $y_i \geq 0$

Unterbestimmtes System: Anzahl der Gleichungen (m) kleiner als Zahl der Variablen (n)

- **Spezifikation als lineares Optimierungsproblem**

  FUNCTION LP(A, b:Seq(Seq($\mathbb{Q}^+$))$\times$Seq($\mathbb{Q}^+$)):Seq($\mathbb{Q}^+$)
  RETURNS y
  SUCH THAT $\Sigma A_{i,j} y_i \leq b_j \wedge \forall t$:Seq($\mathbb{Q}^+$). $\Sigma A_{i,j} t_i \leq b_j \Rightarrow \Sigma c_i y_i \leq \Sigma c_i t_i$

- **Üblicher Ansatz: Lineare Programmierung**
  - **Initialisiere**: Setze $basic := \{1..m\}$, $x_{m+1}, ..x_n := 0$
    löse $\Sigma_{i \in basic} A_{i,j} y_i = b_j$ durch Gauß-Verfahren
  - **Optimiere $y, basic$**:
    - Wähle $i \in basic$, $k \notin basic$ so, daß $\Sigma_{i \in basic'} A_{i,j} z_i = b_j$ nach Tausch von i und k lösbar ist und $\Sigma c_i z_i < \Sigma c_i y_i$
    - Falls es ein solches Paar gibt, so optimiere $z, basic'$
    - Ansonsten terminiere mit Ausgabe y

Lokalsuch-Algorithmen: einheitliche Darstellung

FUNCTION LP(A, b:Seq(Seq($\mathbb{Q}^+$))×Seq($\mathbb{Q}^+$)):Seq($\mathbb{Q}^+$) RETURNS y
SUCH THAT $\Sigma A_{i,j} y_i \leq b_j \;\wedge\; \forall t$:Seq($\mathbb{Q}^+$). $\Sigma A_{i,j} t_i \leq b_j \;\Rightarrow\; \Sigma c_i y_i \leq \Sigma c_i t_i$

Initialisiere: Setze $basic := \{1..m\}$, $x_{m+1}, ..x_n := 0$, löse $\Sigma_{i \in basic} A_{i,j} y_i = b_j$

Optimiere $y, basic$:
- Wähle $i \in basic$, $k \notin basic$ so, daß $\Sigma_{i \in basic'} A_{i,j} z_i = b_j$ nach Tausch von i und k lösbar ist und $\Sigma c_i z_i < \Sigma c_i y_i$
- Falls es ein solches Paar gibt, so optimiere $z, basic'$
- Ansonsten terminiere mit Ausgabe y

- **Struktur der Spezifikation**

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

  Ergebnis y soll kostenoptimal unter den möglichen Lösungen der Spezifikation sein

- **Struktur des Lösungsalgorithmus**

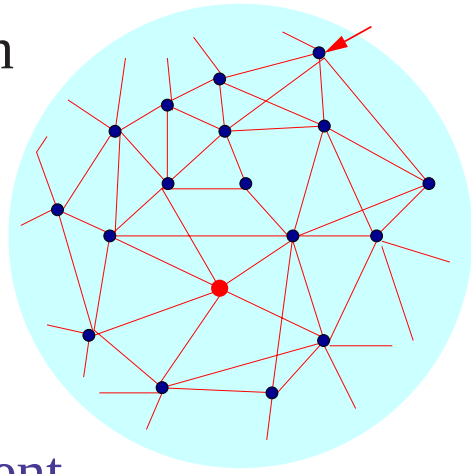
  Optimiere Initiallösung y schrittweise durch kleine Veränderungen

```
let f_LS(x,y) = if locally-optimal(y,O,cost) then y
                else let z loc-arbitrary with O[x,z] ∧ c[x,z]<c[x,y]
                     in f_LS(x,z)
in f_LS(x,Init[x])
```

  Bessere Werte z werden jeweils in einer lokal beschränkten Umgebung von y gesucht

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
    bei LP z.B. auf benachbarte Mengen von Basisvariablen
  - Verfahren endet in lokalen Optima
  - Nachbarschaftsstruktur beeinflußt Güte der Lösung
    - Zu fein $\Rightarrow$ Verfahren führt nicht zu globalem Optimum
    - Zu grob $\Rightarrow$ Suche + Test auf lokale Optimalität ineffizient



Hauptaufgabe ist Bestimmung einer guten Nachbarschaftstruktur

Grundschema von Lokalsuch-Algorithmen

- **Einfaches Programm in formaler Notation**

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

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

Korrektheit des Lokalsuch-Grundschemas

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

ist korrekt, wenn 4 Axiome erfüllt sind

---

1. $\mathit{Init}[x]$ **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

$$\forall x{:}D.\,\forall y{:}R.\;\; I[x] \wedge O[x,y] \;\Rightarrow\; (\forall t \in N[x,y].\, O[x,t] \Rightarrow c[x,y] \leq c[x,t])$$
$$\Rightarrow\; \forall z{:}R.\,(O[x,z] \Rightarrow c[x,y] \leq c[x,z])$$

4. Alle gültigen Lösungen sind endlich erreichbar

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

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

Lokalsuch-Schema: Korrektheitsbeweis

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

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

$\equiv$ $f_{LS}(x, Init[x])$

FUNCTION $f_{ls}(x,z{:}D \times R){:}R$ WHERE $I[x] \wedge O[x,z]$ RETURNS y
SUCH THAT $O[x,y] \wedge \forall t \in N[x,y].\,(O[x,t] \Rightarrow c[x,y] \leq c[x,t])$

$\equiv$ if $\forall t \in N[x,z].\,(O[x,t] \Rightarrow c[x,z] \leq c[x,t])$ then z
else $f_{LS}(x, \mathrm{arb}(\{t \mid t \in N[x,z] \wedge O[x,t] \wedge c[x,t] < c[x,z]\}))$

Lokalsuch-Schema: Korrektheitsbeweis

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

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

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

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

$$\forall x{:}D.\, I[x] \;\Rightarrow\; O[x, \mathit{Init}[x]]$$
$$\forall x{:}D.\forall z{:}R.\;\; I[x]{\wedge}O[x,z] \;\Rightarrow\; O[x, f_{ls}[x,z]]$$
$$\wedge\; \forall t{\in}N[x, f_{ls}[x,z]].\,(O[x,t] \;\Rightarrow\; c[x, f_{ls}[x,z]]{\leq}c[x,t])$$
$$\forall x{:}D.\forall y{:}R.\;\; I[x]{\wedge}O[x,y] \;\Rightarrow\; (\forall t{\in}N[x,y].\, O[x,t] \Rightarrow c[x,y]{\leq}c[x,t])$$
$$\Rightarrow\; \forall z{:}R.\,(O[x,z] \Rightarrow c[x,y]{\leq}c[x,z])$$

Lokalsuch-Schema: Korrektheitsbeweis

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

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

  $\equiv$ $f_{LS}(x, \mathit{Init}[x])$

  FUNCTION $f_{ls}(x,z{:}D\times R){:}R$ WHERE $I[x] \wedge O[x,z]$ RETURNS y
  SUCH THAT $O[x,y] \wedge \forall t \in N[x,y].\,(O[x,t] \Rightarrow c[x,y] \leq c[x,t])$

  $\equiv$ if $\forall t \in N[x,z].\,(O[x,t] \Rightarrow c[x,z] \leq c[x,t])$ then z
  else $f_{LS}(x, \mathrm{arb}(\{t \mid t \in N[x,z] \wedge O[x,t] \wedge c[x,t] < c[x,z]\}))$

- **Korrektheit von f_{opt} folgt aus der von f_{ls} mit Axiomen 1 & 3**
  - Für den Startwert $z = \mathit{Init}[x]$ 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 $\forall t \in N[x,z].\,(O[x,t] \Rightarrow c[x,z] \leq c[x,t])$

Lokalsuch-Schema: Korrektheitsbeweis

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

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

  $\equiv$ $f_{LS}(x, \mathit{Init}[x])$

  FUNCTION $f_{ls}(x,z{:}D{\times}R){:}R$ WHERE $I[x] \wedge O[x,z]$ RETURNS y
  SUCH THAT $O[x,y] \wedge \forall t{\in}N[x,y].\,(O[x,t] \Rightarrow c[x,y]{\leq}c[x,t])$

  $\equiv$ if $\forall t{\in}N[x,z].\,(O[x,t] \Rightarrow c[x,z]{\leq}c[x,t])$ then z
  else $f_{LS}(x, \mathrm{arb}(\{t \mid t{\in}N[x,z] \wedge O[x,t] \wedge c[x,t]{<}c[x,z]\}))$

- **Korrektheit von f_{opt} folgt aus der von f_{ls} mit Axiomen 1 & 3**
  - Für den Startwert $z = \mathit{Init}[x]$ 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**

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

Lokalsuche als Algorithmentheorie

- **Es sind nur 2 neue Komponenten zu bestimmen**
  - Initiallösung *Init* für Basisspezifikation (D, R, I, O) wird durch andere Syntheseformen konstruiert (oft trivial)
  - Geeignete Nachbarschaftsstruktur N für lokale Suche nach Optima Kernaufgabe einer Lokalsuch-Synthese

  Lösungen müssen Beweis der Axiome 1–4 liefern
- **Wie findet man gute Nachbarschaftsstrukturen?**
  - Menge von minimalen Veränderungen des aktuellen Lösungspunktes kontrolliert durch Parameter aus einer "Verwirbelungsmenge" π
  - Dann ist $N[x, y]$ darstellbar als $\{Action[i, j, x, y] \mid i, j \in \pi[x, y]\}$
  - Bezeichnung: $\mathcal{L} = (D, R, I, O, \pi, Action)$ ist **Lokalsuchtheorie**
- **Filter können Nachbarschaftsstrukturen optimieren**
  - Nachbarschaftspunkte müssen zulässige Lösungen sein
  - Nachbarschaftspunkte müssen lokale Optima sein
  - Nachbarschaftspunkte sind, wenn möglich, globale Optima

Synthese eines Lokalsuch-Algorithmus: Sortieren

1. Beschreibe Sortieren als Ordnungsoptimierung

– Suche unter den Umordnungen einer Liste L eine geordnete Liste S
  Liefert Basisspezifikation $O[L,S] \equiv \texttt{rearranges}(L,S)$

– S ist geordnet, wenn keine Elemente in falscher Reihenfolge auftauchen
  Ziel ist also Minimierung der Anzahl von Fehlstellungen $S_i{>}S_j$ $(i{<}j)$
  Liefert Kostenfunktion $c[L,S] \equiv \#_>(S) :\equiv |\{(i,j) \mid i{<}j \wedge S_i{>}S_j\}|$

– Es gilt $\#_>(S){\geq}\texttt{0}$ und $\#_>(S){=}0 \Rightarrow \texttt{ordered}(S)$

Synthese eines Lokalsuch-Algorithmus: Sortieren

1. Beschreibe Sortieren als Ordnungsoptimierung

– Suche unter den Umordnungen einer Liste L eine geordnete Liste S
  Liefert Basisspezifikation $O[L,S] \equiv$ `rearranges(`L`,`S`)`
– S ist geordnet, wenn keine Elemente in falscher Reihenfolge auftauchen
  Ziel ist also Minimierung der Anzahl von Fehlstellungen $S_i{>}S_j$ $(i{<}j)$
  Liefert Kostenfunktion $c[L,S] \equiv \#_>(S) :\equiv |\{(i,j) \mid i{<}j \wedge S_i{>}S_j\}|$
– Es gilt $\#_>(S){\geq}$`0` und $\#_>(S){=}0 \Rightarrow$ `ordered(`S`)`

2. Erstelle formale Spezifikation:

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
  SUCH THAT rearranges(L,S)
      ∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')
```

Synthese eines Lokalsuch-Algorithmus: Sortieren

1. Beschreibe Sortieren als Ordnungsoptimierung

– Suche unter den Umordnungen einer Liste L eine geordnete Liste S
  Liefert Basisspezifikation $O[L,S] \equiv$ `rearranges(`L`,`S`)`

– S ist geordnet, wenn keine Elemente in falscher Reihenfolge auftauchen
  Ziel ist also Minimierung der Anzahl von Fehlstellungen $S_i{>}S_j$ $(i{<}j)$
  Liefert Kostenfunktion $c[L,S] \equiv \#_>(S) :\equiv |\{(i,j) \mid i{<}j \wedge S_i{>}S_j\}|$

– Es gilt $\#_>(S){\geq}$`0` und $\#_>(S){=}0 \Rightarrow$ `ordered(`S`)`

2. Erstelle formale Spezifikation:

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
  SUCH THAT rearranges(L,S)
      ∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')
```

3. Erzeuge Initiallösung für Basisspezifikation

– Einfachste Umordnung einer Liste ist die Liste selbst
  Liefert *Init[L]* = L

↦ Axiom 1

Synthese eines Lokalsuch-Algorithmus (2)

4. Erzeuge Nachbarschaftsstruktur

– Einfache Suchstruktur vertauscht zwei beliebige Elemente S_i und S_j
  Ergibt $N[L, S] \equiv \{\texttt{permute}_{i,j}(S) \mid i, j \leq |S|\}$

– Vertauschen ist reflexiv, Lokale Minima sind exakt $\mapsto$ Axiom 2,3

– Alle Umordnungen erreichbar durch iteratives Vertauschen $\mapsto$ Axiom 4

Synthese eines Lokalsuch-Algorithmus (2)

4. Erzeuge Nachbarschaftsstruktur

– Einfache Suchstruktur vertauscht zwei beliebige Elemente S_i und S_j
  Ergibt $N[L,S] \equiv \{\texttt{permute}_{i,j}(S) \mid i,j \leq |S|\}$

– Vertauschen ist reflexiv, Lokale Minima sind exakt $\mapsto$ Axiom 2,3

– Alle Umordnungen erreichbar durch iteratives Vertauschen $\mapsto$ Axiom 4

5. Instantiiere Algorithmenschema

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
  SUCH THAT rearranges(L,S)
                ∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')
≡ let sortLS(L,S)
   = if ∀S'∈N[L,S].rearranges(L,S') ⇒ #>(S)≤#>(S') then S
     else sortLS(L, arb{S'∈N[L,S] |
                          rearranges(L,S') ∧ #>(S)>#>(S')}))
   in sortLS(L,L)
```

Ergibt Sortieren durch beliebiges Austauschen von Elementen

– Ineffizient, da zu viele Nachbarn überprüft werden (vermutlich $\mathcal{O}(n^3)$)

Lokalsuche mit kleinerer Nachbarschaft

4. Erzeuge Nachbarschaftsstruktur

– Restriktion auf $N[L,S] \equiv \{\texttt{permute}_{i,i+1}(S) \mid i<|S|\}$

Nur unmittelbar benachbarte Elemente dürfen vertauscht werden

Lokalsuche mit kleinerer Nachbarschaft

4. Erzeuge Nachbarschaftsstruktur

– Restriktion auf $N[L,S] \equiv \{\texttt{permute}_{i,i+1}(S) \mid i<|S|\}$
  Nur unmittelbar benachbarte Elemente dürfen vertauscht werden

5. Verfeinere Struktur durch Filter

– Vertauschungen erhalten Umordnungseigenschaft (Filter unnötig)

– Vertauschungen bringen Verbesserungen, wenn $S_i > S_{i+1}$
  Liefert $N'[L,S] \equiv \{\texttt{permute}_{i,i+1}(S) \mid i<|S| \wedge S_i>S_{i+1}\}$

– Es gilt $\#_>(S)>\#_>(S')$ für alle $S' \in N'[L,S]$

Lokalsuche mit kleinerer Nachbarschaft

4. Erzeuge Nachbarschaftsstruktur

– Restriktion auf $N[L,S] \equiv \{\texttt{permute}_{i,i+1}(S) \mid i<|S|\}$

Nur unmittelbar benachbarte Elemente dürfen vertauscht werden

5. Verfeinere Struktur durch Filter

– Vertauschungen erhalten Umordnungseigenschaft (Filter unnötig)

– Vertauschungen bringen Verbesserungen, wenn $S_i > S_{i+1}$

Liefert $N'[L,S] \equiv \{\texttt{permute}_{i,i+1}(S) \mid i<|S| \wedge S_i>S_{i+1}\}$

– Es gilt $\#_>(S)>\#_>(S')$ für alle $S' \in N'[L,S]$

6. Instantiiere (und optimiere) Algorithmenschema

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
  SUCH THAT rearranges(L,S)
            ∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')
≡ let sortLS(L,S)
   = if N'[L,S]=∅ then S  else sortLS(L,arb(N'[L,S]))
   in sortLS(L,L)
```

Entspricht dem Bubblesort Algorithmus (ohne feste Reihenfolge)

Analyse liefert zwei Suchstrukturen für `sort`

- **Große Nachbarschaft: Permutation von Elementen**
  - Änderungsschritt: Vertauschung zweier Elemente der Ausgabeliste S
  - Änderungsparameter: Indizes der Liste S

$$\texttt{LS\_sort\_wide} \equiv \begin{array}{lll} D & \mapsto & \texttt{Seq}(\mathbb{Z}) \\ R & \mapsto & \texttt{Seq}(\mathbb{Z}) \\ I & \mapsto & \lambda\texttt{L}.\texttt{true} \\ O & \mapsto & \lambda\texttt{L,S}.\texttt{rearranges(L,S)} \\ \pi & \mapsto & \lambda\texttt{L,S}.\texttt{(domain(S),domain(S))} \\ \mathit{Action} & \mapsto & \lambda\texttt{i,j,L,S}.\texttt{[S}_{(i\leftrightarrow j)(k)}\texttt{|k<|S|]} \end{array}$$

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

- **Kleine Nachbarschaft: Permutation benachbarter Elemente**
  - Änderungsschritt: Vertauschung fehlgestellter Nachbarn in S

$$\texttt{LS\_sort\_narrow} \equiv \begin{array}{lll} D & \mapsto & \texttt{Seq}(\mathbb{Z}) \\ R & \mapsto & \texttt{Seq}(\mathbb{Z}) \\ I & \mapsto & \lambda\texttt{L}.\texttt{true} \\ O & \mapsto & \lambda\texttt{L,S}.\texttt{rearranges(L,S)} \\ \pi & \mapsto & \lambda\texttt{L,S}.\texttt{(domain(S),domain(S))} \\ \mathit{Action} & \mapsto & \lambda\texttt{i,j,L,S}.\texttt{if S}_i\leq\texttt{S}_{i+1}\texttt{ then S} \\ & & \texttt{else [S}_{(i\leftrightarrow i+1)(k)}\texttt{|k<|S|]} \end{array}$$

Alle 4 Axiome sind erfüllt

Wie erzeugt man effiziente Suchstrukturen?

- **Formuliere fundamentale Nachbarschaftsstrukturen auf R**
  - Beschreibe Nachbarschaft durch inkrementelle Veränderungen in R
    - Numerik: δ-Vektoren,
    - Kombinatorik: Austausch von Komponenten (in Listen, Mengen, ...)
  - Formalisiere $N[x,y]$ als $\{\mathit{Action}[i,j,x,y] \mid i,j \in \pi[x,y]\}$
    - Änderung $\mathit{Action}[i,j,x,y]$ modifiziert Lösungspunkt $(x,y) \in D \times R$
    - Parameter $i,j \in \pi[x,y]$ sind "minimale Bestandteile" von (x,y)
  - Wissensbank speichert Lokalsuchtheorien für Grunddatentypen

    Synthese spezialisiert generische Suchstrukturen auf spezifisches Problem
- **Optimiere Nachbarschaftsstruktur durch Suchfilter**
  - Verenge Suchraum durch Abschneiden unnötiger Nachbarn
    - Feasibility Constraint für $O[x,y]$
    - (Local) Optimality Constraint für $\forall t \in N[x,z].\,(O[x,t] \Rightarrow c[x,t] < c[x,z])$
    - ggf. Global Optimality Constraint für exakte Lösungen
  - Filter können durch Vorwärtsinferenz aus Bedingungen abgeleitet werden

Erzeugung geeigneter Nachbarschaftsstruktur

Spezialisiere vorformulierte Suchstrukturen

- **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
- **z.B. Standard-Suche auf Teilmengen fester Größe**
  - Suche $\hat{=}$ Austausch einzelner Elemente einer Menge
  - Änderungsparameter: Elemente der Ein- und Ausgabemenge

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

Erzeugung einer Suchstruktur für `sort`

Wähle Struktur $\mathcal{L}$ für R so daß $spec \ll spec_{\mathcal{L}}$ beweisbar

- **Standard-Umordnung von Folgen über Datentyp α**
  - Suche $\hat{=}$ Permutation einzelner Elemente einer Folge
  - Änderungsparameter: Indizes der Ausgabeliste `L`

`LS_seq_re(`α`)` $\equiv$

D	$\mapsto$	`Seq(`α`)`
R	$\mapsto$	`Seq(`α`)`
I	$\mapsto$	λ`L.true`
O	$\mapsto$	λ`L,S.rearranges(L,S)`
π	$\mapsto$	λ`L,S.(domain(S),domain(S))`
Action	$\mapsto$	λ`i,j,L,S.[ S`$_{(i\leftrightarrow j)(k)}$` | k`$\in$`domain(S)]`

- **Spezialisierung auf das Sortierproblem**

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
SUCH THAT rearranges(L,S)
          ∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')
```

  - LS-Theorie $\mathcal{L}$ `= LS_seq_re(`$\mathbb{Z}$`)` paßt zum Bildbereich `Seq(`$\mathbb{Z}$`)`
  - Wegen $O_{\mathcal{L}}(L,S)$`= rearranges(L,S)` folgt $spec \ll spec_{\mathcal{L}}$ mit ϑ`=id`
    Liefert Grundstruktur für die Lokalsuche

Verfeinerung von Suchstrukturen durch Filter

Elimiere überflüssige Kandidaten aus Nachbarschaft

- **Lösungs-Filter FC für $\mathcal{L}$** *(Feasibility Constraint)*
  - Eliminiere Punkte aus der Nachbarschaft, die keine Lösungen bzgl. O sind, aber erhalte alle akzeptablen Lösungen
  - Ist y akzeptable Lösung, dann erhalte jedes $t \in N[x,y]$ mit $O(x,t)$
    $\cdot\ I[x] \wedge O[x,y] \wedge t \in N[x,y] \Rightarrow (O[x,t] \Rightarrow FC(x,y,t))$
  - Entspricht notwendigen Filtern der Globalsuche
- **Optimalitäts-Filter OC für $\mathcal{L}$** *(Optimality Constraint)*
  - Eliminiere kostenungünstigere Punkte aus der Nachbarschaft, aber erhalte jedes bessere $t \in N[x,y]$ (mit $c[x,t] < c[x,y]$)
    $\cdot\ I[x] \wedge O[x,y] \wedge t \in N[x,y] \wedge O[x,t] \Rightarrow (c[x,t] < c[x,y] \Rightarrow OC(x,y,t))$
  - y ist ein lokales Optimum, wenn $\{t \in N[x,y] \mid OC(x,y,t)\}$ leer ist
- **Exaktheits-Filter GOC** *(Global Optimality Constraint)*
  - Eliminiere suboptimale Lösungen aber erhalte alle optimalen
    $\cdot\ I[x] \wedge O[x,y] \Rightarrow ((\forall t{:}R.\, O[x,t] \Rightarrow c[x,y] \leq c[x,t]) \Rightarrow GOC(x,y)$
  - Test oft aufwendig, macht Algorithmus ggf. weniger effizient

Formuliere Filter als Parameterconstraints

- **Lösungs-Filter FC für $\mathcal{L}$**
  - $I[x] \wedge O[x,y] \wedge t \in N[x,y] \Rightarrow (O[x,t] \Rightarrow FC(x,y,t))$
  - Filter auch direkt auf Parameter der Änderungsaktion angewandt werden, da $N[x,y] = \{Action[i,j,x,y] \mid i,j \in \pi[x,y]\}$
  - Führt zu $I[x] \wedge O[x,y] \Rightarrow (O[x, Action[i,j,x,y]] \Rightarrow FC[i,j,x,y])$
  - FC kann durch Vereinfachung von $O[x, Action[i,j,x,y]]$ im Kontext der rechten Seite abgeleitet werden
- **Optimalitäts-Filter OC für $\mathcal{L}$**
  - Umformulieren von
    $$I[x] \wedge O[x,y] \wedge t \in N[x,y] \wedge O[x,t] \Rightarrow (c[x,t] < c[x,y] \Rightarrow OC(x,y,t))$$
    ergibt $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,x,y]]$
    $$\Rightarrow (c[x, Action[i,j,x,y]] < c[x,y] \Rightarrow OC[i,j,x,y])$$
- **Integriere Filter in Basis-Nachbarschaftsstruktur**
  - $N_c[x,y] = \{Action[i,j,x,y] \mid i,j \in \pi[x,y] \wedge FC[i,j,x,y] \wedge OC[i,j,x,y]\}$
  - Exaktheits-Filter $GOC[x,y]$ wird direkt auf Lösungspunkte angewandt

Unterstützende Techniken für eine Synthese

- **Spezialisierungsmechanismus für LS-Theorien** ↦ §18, Folie 20
  - Wähle $\mathcal{L}$ passend zum Bildbereich der Spezifikation $spec = (D,R,I,O)$
  - Beweise $spec \ll spec_{\mathcal{L}}$

    $$R \subseteq R_{\mathcal{G}} \;\wedge\; \forall x{:}D\,.\,I[x] \;\Rightarrow\; \exists x'{:}D_{\mathcal{L}}\,.\,(I_{\mathcal{L}}[x'] \;\wedge\; \forall y{:}R\,.\,O[x,y] \Rightarrow O_{\mathcal{L}}[x',y])$$

  - Extrahiere Substitution $\vartheta{:}D \to D_{\mathcal{L}}$ aus dem Beweis
  - Modifiziere $\mathcal{L}$ mit ϑ zu wohlfundierter Lokalsuchtheorie für $spec$

    Ergibt Basis-Nachbarschaftsstruktur N für Problemstellung

- **Vorwärtsinferenz (heuristisch) für Filterung** ↦ Effizienzsteigerung
  - Leite Folgerungen aus den Vorbedingungen der Filter ab
    - $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,x,y]]$ (FC)
    - $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,x,y]] \wedge c[x, Action[i,j,x,y]] < c[x,y]$ (OC)
    - $I[x] \wedge O[x,y] \;\wedge \forall t{:}R.\, O[x,t] \Rightarrow c[x,y] \leq c[x,t]$ (GOC)
  - Verwende nur Folgerungen, die auch leicht zu testen sind

Synthesestrategie für Lokalsuch-Algorithmen

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

Synthesestrategie für Lokalsuch-Algorithmen

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

1. Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R (Wissensbank)

Synthesestrategie für Lokalsuch-Algorithmen

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

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** (Wissensbank)
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$**
   – Extrahiere Substitution ϑ und setze $\mathcal{L}_\vartheta = (D, R, I, O, \pi_\vartheta, Action_\vartheta)$

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

$\pi_\vartheta[x,y] :\equiv \pi[\vartheta(x),y]$

$Action_\vartheta[i,j,x,y] :\equiv Action[i,j,\vartheta(x),y]$

Synthesestrategie für Lokalsuch-Algorithmen

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

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** (Wissensbank)
2. **Beweise (D,R,I,O) $\ll spec_{\mathcal{L}}$**
   – Extrahiere Substitution ϑ und setze $\mathcal{L}_\vartheta = (D, R, I, O, \pi_\vartheta, Action_\vartheta)$
3. **Synthetisiere Initiallösung *Init* für Spezifikation** (Standardsynthese)
   FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$

Synthesestrategie für Lokalsuch-Algorithmen

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

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** (Wissensbank)
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$**
   – Extrahiere Substitution ϑ und setze $\mathcal{L}_\vartheta = (D, R, I, O, \pi_\vartheta, Action_\vartheta)$
3. **Synthetisiere Initiallösung *Init* für Spezifikation** (Standardsynthese)
   $\texttt{FUNCTION}\ f(x:D):R\ \texttt{WHERE}\ I[x]\ \texttt{RETURNS}\ y\ \texttt{SUCH THAT}\ O[x,y]$
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$ durch Vorwärtsinferenz**

$I[x] \wedge O[x,y] \Rightarrow (O[x, Action[i,j,x,y]] \Rightarrow FC[i,j,x,y])$
FC ergibt sich durch Vereinfachung von $O[x, Action[i,j,x,y]]$ im Kontext

Synthesestrategie für Lokalsuch-Algorithmen

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

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** (Wissensbank)
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$**
   – Extrahiere Substitution ϑ und setze $\mathcal{L}_\vartheta = (D, R, I, O, \pi_\vartheta, Action_\vartheta)$
3. **Synthetisiere Initiallösung *Init* für Spezifikation** (Standardsynthese)
   FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$ durch Vorwärtsinferenz**
5. **Generiere Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*

$$I[x] \wedge O[x,y] \wedge O[x, Action[i,j,x,y]] \Rightarrow (c[x, Action[i,j,x,y]] < c[x,y]) \Rightarrow OC[i,j,x,y])$$

OC ergibt sich durch Vereinfachung von $c[x, Action[i,j,x,y]] < c[x,y]$ im Kontext

Synthesestrategie für Lokalsuch-Algorithmen

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

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** (Wissensbank)
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$**
   – Extrahiere Substitution ϑ und setze $\mathcal{L}_\vartheta = (D, R, I, O, \pi_\vartheta, Action_\vartheta)$
3. **Synthetisiere Initiallösung *Init* für Spezifikation** (Standardsynthese)
   FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$ durch Vorwärtsinferenz**
5. **Generiere Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*
6. **Integriere Filter in Nachbarschaftsstruktur**
   – Setze $N_c[x,y] \equiv \{Action[i,j,\vartheta(x),y] \mid i,j \in \pi_c[\vartheta(x),y]\}$
   wobei $\pi_c[x,y] \equiv \{i,j \in \pi[x,y] \mid FC[i,j,x,y] \wedge OC[i,j,x,y]\}$

Synthesestrategie für Lokalsuch-Algorithmen

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

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** (Wissensbank)
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$**
   – Extrahiere Substitution ϑ und setze $\mathcal{L}_\vartheta = (D, R, I, O, \pi_\vartheta, Action_\vartheta)$
3. **Synthetisiere Initiallösung *Init* für Spezifikation** (Standardsynthese)
   FUNCTION $f(x:D):R$ WHERE $I[x]$ RETURNS y SUCH THAT $O[x,y]$
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$ durch Vorwärtsinferenz**
5. **Generiere Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*
6. **Integriere Filter in Nachbarschaftsstruktur**
   – Setze $N_c[x,y] \equiv \{Action[i,j,\vartheta(x),y] \mid i,j \in \pi_c[\vartheta(x),y]\}$
   wobei $\pi_c[x,y] \equiv \{i,j \in \pi[x,y] \mid FC[i,j,x,y] \wedge OC[i,j,x,y]\}$
7. **Instantiiere & Optimiere Schema für Lokalsuch Algorithmen**
   – Ggf. generiere und integriere Exaktheitsfilter

Synthese eines Lokalsuch-Sortieralgorithmus

Start: FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
SUCH THAT rearranges(L,S)
$\wedge$ $\forall$S′:Seq(ℤ).rearranges(L,S′) $\Rightarrow$ $\#_{>}$(S)$\leq\#_{>}$(S′)

Synthese eines Lokalsuch-Sortieralgorithmus

Start: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S`
`SUCH THAT rearranges(L,S)`
$\wedge$ `∀S′:Seq(ℤ).rearranges(L,S′)` $\Rightarrow$ `#>(S)≤#>(S′)`

1. Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R $\mathcal{L}$ = `LS_seq_re(ℤ)`

Synthese eines Lokalsuch-Sortieralgorithmus

Start: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S`
`SUCH THAT rearranges(L,S)`
$\wedge$ `∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')`

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** $\mathcal{L}$ = `LS_seq_re(ℤ)`
2. **Beweise (D,R,I,O)** $\ll spec_{\mathcal{L}}$ (Folie 15) liefert ϑ=`id`

Synthese eines Lokalsuch-Sortieralgorithmus

Start: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S`
`SUCH THAT rearranges(L,S)`
$\wedge$ `∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')`

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** — $\mathcal{L}$ = `LS_seq_re(ℤ)`
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$** (Folie 15) — liefert ϑ=`id`
3. **Synthetisiere Initiallösung *Init* für Spezifikation** — setze `Init(L)=L`

Synthese eines Lokalsuch-Sortieralgorithmus

Start: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S`
`SUCH THAT rearranges(L,S)`
$\wedge$ `∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')`

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** `ℒ = LS_seq_re(ℤ)`
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$** (Folie 15) liefert ϑ=`id`
3. **Synthetisiere Initiallösung *Init* für Spezifikation** setze `Init(L)=L`
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$** *(Feasibility Constraint)*

   `rearranges(L,S)` $\Rightarrow$ `(rearranges(L,S`$_{(i\leftrightarrow j)}$`)⇒FC[i,j,L,S])`

   Vereinfachung im Kontext ergibt `FC[i,j,L,S]=true`

Synthese eines Lokalsuch-Sortieralgorithmus

Start: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S`
`SUCH THAT rearranges(L,S)`
$\wedge$ `∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')`

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** `ℒ = LS_seq_re(ℤ)`
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$** (Folie 15) liefert ϑ=`id`
3. **Synthetisiere Initiallösung *Init* für Spezifikation** setze `Init(L)=L`
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$** *(Feasibility Constraint)*
   `rearranges(L,S)` ⇒ `(rearranges(L,S`$_{(i\leftrightarrow j)}$`)⇒FC[i,j,L,S])`
   Vereinfachung im Kontext ergibt `FC[i,j,L,S]=true`
5. **Generiere Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*
   `rearranges(L,S)` $\wedge$ `rearranges(L,S`$_{(i\leftrightarrow j)}$`)`
   ⇒ `(#>(S`$_{(i\leftrightarrow j)}$ `< #>(S)) ⇒ OC[i,j,L,S])`
   Vereinfachung im Kontext ergibt `OC[i,j,L,S] = i<j` $\Leftrightarrow$ `S`$_i$`>S`$_j$

Synthese eines Lokalsuch-Sortieralgorithmus

Start: `FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S`
`SUCH THAT rearranges(L,S)`
`∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')`

1. **Wähle Lokalsuchtheorie $\mathcal{L}$ mit Ausgabetyp R** `ℒ = LS_seq_re(ℤ)`
2. **Beweise $(D,R,I,O) \ll spec_{\mathcal{L}}$** (Folie 15) liefert ϑ=`id`
3. **Synthetisiere Initiallösung *Init* für Spezifikation** setze `Init(L)=L`
4. **Generiere Lösungs-Filter FC für $\mathcal{L}_\vartheta$** *(Feasibility Constraint)*
   `rearranges(L,S) ⇒ (rearranges(L,S`$_{(i\leftrightarrow j)}$`)⇒FC[i,j,L,S])`
   Vereinfachung im Kontext ergibt `FC[i,j,L,S]=true`
5. **Generiere Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*
   `rearranges(L,S) ∧ rearranges(L,S`$_{(i\leftrightarrow j)}$`)`
   `⇒ (#>(S`$_{(i\leftrightarrow j)}$` < #>(S)) ⇒ OC[i,j,L,S])`
   Vereinfachung im Kontext ergibt `OC[i,j,L,S] = i<j ⇔ S`$_i$`>S`$_j$
6. **Instantiiere & Optimiere Schema für Lokalsuch Algorithmen**
   – $N_c[L,S] \equiv \{S_{(i\leftrightarrow j)} \mid i<j<|S| \wedge S_i > S_j\}$
   – Kein Exaktheitstest erforderlich

Synthese eines Lokalsuch-Sortieralgorithmus

7. Instantiiertes Lokalsuch-Schema

$$
\begin{array}{l}
\texttt{FUNCTION sort(L:Seq}(\mathbb{Z})\texttt{):Seq}(\mathbb{Z})\texttt{ RETURNS S} \\
\quad \texttt{SUCH THAT rearranges(L,S)} \\
\qquad\qquad \wedge\ \forall \texttt{S'}:\texttt{Seq}(\mathbb{Z}).\texttt{rearranges(L,S')} \Rightarrow \#_{>}\texttt{(S)} \leq \#_{>}\texttt{(S')} \\
\equiv\ \mathit{sort}_{LS}\texttt{(L,L)} \\
\texttt{FUNCTION } \mathit{sort}_{LS}\texttt{(L,S:Seq}(\mathbb{Z}) \times \texttt{Seq}(\mathbb{Z})\texttt{):Seq}(\mathbb{Z}) \\
\quad \texttt{WHERE rearranges(L,S) RETURNS Y} \\
\quad \texttt{SUCH THAT rearranges(L,Y)} \\
\qquad \wedge\ \forall \texttt{T} \in \{\texttt{Y}_{(i \leftrightarrow j)} \mid \texttt{i<j<|Y|} \wedge \texttt{Y}_i > \texttt{Y}_j\}.(\texttt{rearranges(L,T)} \Rightarrow \#_{>}\texttt{(Y)} \leq \#_{>}\texttt{(T))} \\
\equiv\ \texttt{if } \forall \texttt{T} \in \{\texttt{S}_{(i \leftrightarrow j)} \mid \texttt{i<j<|S|} \wedge \texttt{S}_i > \texttt{S}_j\}.(\texttt{rearranges(L,T)} \Rightarrow \#_{>}\texttt{(S)} \leq \#_{>}\texttt{(T))} \\
\qquad \texttt{then S} \\
\qquad \texttt{else } \mathit{sort}_{LS}\texttt{(L, arb(\{ T} \in \{\texttt{S}_{(i \leftrightarrow j)} \mid \texttt{i<j<|S|} \wedge \texttt{S}_i > \texttt{S}_j\} \mid \\
\qquad\qquad\qquad\qquad \texttt{rearranges(L,T)} \wedge \#_{>}\texttt{(T)<}\#_{>}\texttt{(S)\}))}
\end{array}
$$

Synthese eines Lokalsuch-Sortieralgorithmus

7. Instantiiertes Lokalsuch-Schema

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ) RETURNS S
  SUCH THAT rearranges(L,S)
            ∧ ∀S':Seq(ℤ).rearranges(L,S') ⇒ #>(S)≤#>(S')
≡ sort_LS(L,L)
FUNCTION sort_LS(L,S:Seq(ℤ)×Seq(ℤ)):Seq(ℤ)
  WHERE rearranges(L,S) RETURNS Y
  SUCH THAT rearranges(L,Y)
    ∧ ∀T∈{Y_(i↔j) | i<j<|Y| ∧ Y_i>Y_j}.(rearranges(L,T)⇒#>(Y)≤#>(T))
≡ if ∀T∈{S_(i↔j) | i<j<|S| ∧ S_i>S_j}.(rearranges(L,T)⇒#>(S)≤#>(T))
     then S
     else sort_LS(L, arb({T∈{S_(i↔j) | i<j<|S| ∧ S_i>S_j} |
                          rearranges(L,T)∧#>(T)<#>(S)}))
```

8. Nach Optimierung der Hilfsfunktion

Es gilt $\forall T \in \{Y_{(i\leftrightarrow j)} \mid i<j<|Y| \wedge Y_i>Y_j\}$. rearranges(Y,T) ∧ #>(T)<#>(Y)

```
FUNCTION sort_LS(L,S:Seq(ℤ)×Seq(ℤ)):Seq(ℤ)
  WHERE rearranges(L,S) RETURNS Y
  SUCH THAT rearranges(L,Y) ∧ {Y_(i↔j) | i<j<|Y| ∧ Y_i>Y_j}=∅
≡ if ∀i<j<|S|. S_i≤S_j then S
     else sort_LS(L, arb({S_(i↔j) | i<j<|S| ∧ S_i>S_j}))
```

Lokalsuche: Anwendungsbeispiele

- **Sortieren mit engeren Suchstrukturen**
- **Minimal Spanning Tree**
  - Gegeben: Graph mit gewichteten Kanten (Zugriffszeiten, Abstände,...)
  - Gesucht: Baum, auf dem alle Knoten mit minimalen Kosten erreichbar
  - Initialwert: Erzeuge spannenden Baum
  - Änderungsaktion: 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
  - Änderungsaktion: Setze ein $\texttt{x}_i\texttt{:=0}$, wähle neues $\texttt{x}_j \neq \texttt{0}$
  - Feasibility Constraint: Alte + neue x-Komponenten nach Lösung positiv
  - Optimality Constraint: Relative Kosten steigen durch Veränderung

Rückblick: Synthese mit Algorithmenschemata

Wissensbasierte Techniken zur Softwareentwicklung

- **Erzeugte Algorithmen sind korrekt und effizient**
  - Formales theoretisches Fundament sichert Korrektheit
  - Gute algorithmische Struktur liefert Effizienz
  - Nachträgliche Optimierung des schematischen Algorithmus möglich/nötig
  - Mathematische Notation übersetzbar in Programmiersprachen
- **Synthesetechniken sind automatisierbar**
  - Jeder Schritt basiert auf logischer Inferenz
  - Wissen steuert alle Strategien des Algorithmenentwurfs
  - Ähnliche Techniken für Entwurf verschiedener Algorithmenstrukturen
- **Techniken sind praktisch erfolgreich**
  - KIDS erzeugt korrekte Scheduling Algorithmen in wenigen Stunden
  - Erzeugter Lisp Code 2000 mal schneller als existierende ADA Software