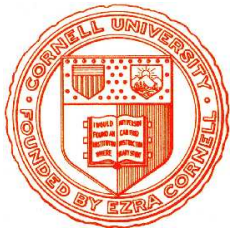
Automatisierte Logik und Programmierung

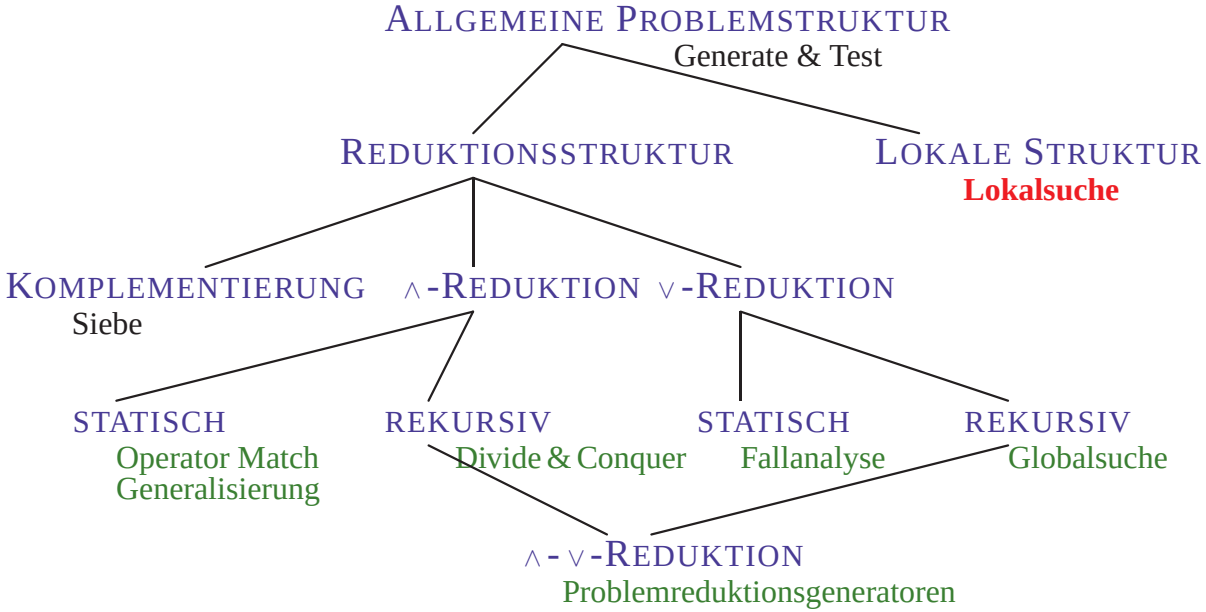
Einheit 21



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)

- **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 $basic := \{1..m\}$,
    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 $basic$:=$\{1..m\}$, 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

- **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**

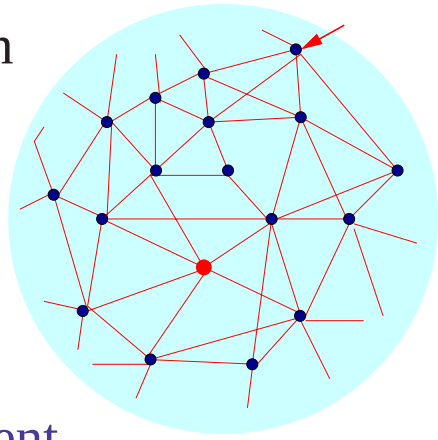
  Optimiere Initiallösung y schrittweise durch kleine Veränderungen

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

- **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-Schemas

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
$f_{LS}(x, \mathrm{arb}(\{t \mid t \in N[x,z] \wedge O[x,t] \wedge c[x,t] < c[x,z]\}))$
in $f_{LS}(x, \mathit{Init}[x])$

ist korrekt, wenn 4 Axiome erfüllt sind

---

1. $\mathit{Init}[x]$ **berechnet gültige Initiallösung für** O

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

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

  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**

Beispiel: Sortieren als Lokalsuche

- **Formuliere Sortieren als Ordnungsoptimierung**
  - Einfache Basisspezifikation $O[L,S]$ = `rearranges(`L`,`S`)`
  - Kostenfunktion $c[L,S]$ = $\#_>(S)$: Anzahl der Fehlstellungen $S_i{>}S_j$ $(i{<}j)$
    Es gilt $\#_>(S){\geq}$`0` und $\#_>(S){=}0 \Rightarrow$ `ordered(`S`)`
  - Spezifikation `FUNCTION sort(L:Seq(`$\mathbb{Z}$`)):Seq(`$\mathbb{Z}$`) RETURNS S`
    `SUCH THAT rearranges(L,S)`
    $\wedge$ `∀S':Seq(`$\mathbb{Z}$`).rearranges(L,S') ⇒ #`$_>$`(S)≤#`$_>$`(S')`
- **Einfacher Lokalsuchalgorithmus**
  - Initiallösung *Init[L]* = L $\mapsto$ Axiom 1
  - Nachbarschaft $N[S,S']$ = `permute`$_{i,j}$`(`S`,`S'`)`: vertausche 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 $N[S,S']$ = `perm`$_i$`(`S`,`S'`)` = `permute`$_{i,i+1}$`(`S`,`S'`)`
  - Ergibt Bubblesort (nach algorithmischer Optimierung)

Erzeuge effiziente Nachbarschaftsstrukturen

- **Bestimme fundamentale Nachbarschaftsstruktur auf R**
  - Beschreibe Nachbarschaft als Perturbation (Verwirbelung)
  - Inkrementelle Veränderung von Werten aus R
    - Numerik: δ-Vektoren,
    - Kombinatorik: Austausch von Komponenten
  - 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)
- **Optimiere Nachbarschaftsstruktur durch Suchfilter**
  - Optimiere Nachbarschaftsstruktur durch frühzeitiges Abschneiden
    - Feasibility Constraint für $O[x,y]$
    - Optimality Constraint für $\forall t \in N[x,z].\,(O[x,t] \Rightarrow c[x,z] \leq c[x,t])$
    - ggf. Global Optimality Constraint für exakte Lösungen

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

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

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

Standard-Lokalsuchtheorie für Mengen

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

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

Spezialisiere Lokalsuchtheorien der Wissensbank

- **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 $\vartheta: D \rightarrow D_{\mathcal{L}}$ und spezialisiere $\mathcal{L}$ mit ϑ
    - Ergibt Nachbarschaftsstruktur N für Problemstellung
- **Beispiel einer Spezialisierung, z.B. Bubblesort**

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

  - Wähle LS-Theorie $\mathcal{L}$ = `LS_seq_re(ℤ)`
  - Wegen $O_{\mathcal{L}}(L,S)$= `rearranges(L,S)` folgt $spec \ll spec_{\mathcal{L}}$ mit ϑ=`id`

Optimierung von Suchstrukturen durch Filter

Elimiere überflüssige Kandidaten aus Nachbarschaft

- **Lösungs-Filter FC für $\mathcal{L}_\vartheta$** *(Feasibility Constraint)*
  - Eliminiert Punkte, die keine akzeptablen Lösungen bzgl. O sind
    - $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,\vartheta(x),y]] \Rightarrow FC[i,j,x,y]$
  - Entspricht notwendigen Filtern der Globalsuche
- **Lokale Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*
  - Eliminiert kostenungünstigere Punkte von weiterer Betrachtug
    - $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,\vartheta(x),y]]$
      $\wedge\ c[x,y] \leq c[x, Action[i,j,\vartheta(x),y]] \Rightarrow OC[i,j,x,y]$
  - Lokale Optima müssen Bedingung $\forall i,j \in \pi(x,y).\, OC(i,j,x,y)$ erfüllen
- **Exaktheits-Filter GOC** *(Global Optimality Constraint)*
  - Eliminiere suboptimale Lösungen (Algorithmus wird ggf. weniger effizient)
    - $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,\vartheta(x),y]]$
      $\wedge\ c[x,y] \leq c[x, Action[i,j,\vartheta(x),y]] \wedge \forall t{:}R.\, O[x,t] \Rightarrow c[x,y] \leq c[x,t]$
      $\Rightarrow GOC[x,y]$

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 $\mathcal{L}_\vartheta$** *(Feasibility Constraint)*
   $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,\vartheta(x),y]] \Rightarrow FC[i,j,x,y]$
   – FC ergibt sich durch Vereinfachung der linken Seite (Vorwärtsinferenz)

5. **Generiere Optimalitäts-Filter OC für $\mathcal{L}_\vartheta$** *(Optimality Constraint)*
   $I[x] \wedge O[x,y] \wedge O[x, Action[i,j,\vartheta(x),y]] \wedge c[x,y] \leq c[x, Action[i,j,\vartheta(x),y]]$
   $\Rightarrow OC[i,j,x,y]$
   – OC ergibt sich durch Vereinfachung der linken Seite (Vorwärtsinferenz)

Synthesestrategie für Lokalsuch-Algorithmen

6. Instantiiere Schema für suboptimale Lokalsuch Algorithmen

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

Nachgeschaltete Lösungs- und Optimalitätstests notwendig für Korrektheit

– Abarbeitung nutzt andthen/orelse Semantik von $\wedge$ und $\vee$ aus

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

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

– Oft Verzicht auf Exaktheit zugunsten effizienterer Algorithmen

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 $(D,R,I,O) \ll spec_{\mathcal{L}}$** 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)` $\wedge$ `rearranges(L,S`$_{(i\leftrightarrow j)}$`)` $\Rightarrow$ `FC[i,j,L,S]`
   – Vereinfachung der linken Seite 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`$_{(i\leftrightarrow j)}$`)`
   $\wedge$ `#>(S)≤#>(S`$_{(i\leftrightarrow j)}$`)` $\Rightarrow$ `OC[i,j,L,S]`
   – Vereinfachung der linken Seite ergibt `OC[i,j,L,S] = i<j ⇒ S`$_i \leq$ `S`$_j$
6. **Kein Exaktheitstest erforderlich**

Synthese eines Lokalsuch-Sortieralgorithmus

6. Instantiieres Schema

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

7. Nach Optimierung

$$\begin{array}{l}
\texttt{FUNCTION sort(L:Seq(}\mathbb{Z}\texttt{)):Seq(}\mathbb{Z}\texttt{) ...} \\
\equiv\ \mathit{sort}_{LS}\texttt{(L,L)} \\
\texttt{FUNCTION } \mathit{sort}_{LS}\texttt{(L,S:Seq(}\mathbb{Z}\texttt{)}\times\texttt{Seq(}\mathbb{Z}\texttt{)):Seq(}\mathbb{Z}\texttt{) ...} \\
\equiv\ \texttt{if } \forall \texttt{i<j<|S|. S}_i \leq \texttt{S}_j \texttt{ then S} \\
\qquad \texttt{else } \mathit{sort}_{LS}\texttt{(L, arb(\{ S}_{(i\leftrightarrow j)}\texttt{ | i<j<|S|} \wedge \texttt{ S}_i > \texttt{S}_j \texttt{ \}))}
\end{array}$$

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

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