Chapter MK:VI

VI. Planning and Configuration

- Agent Systems
- Deductive Reasoning Agents
- Planning Language

- Planning Algorithms
- State-Space Planning
- Plan Space Planning
- HTN Planning
- Complexity of Planning Problems
- Erweiterungen

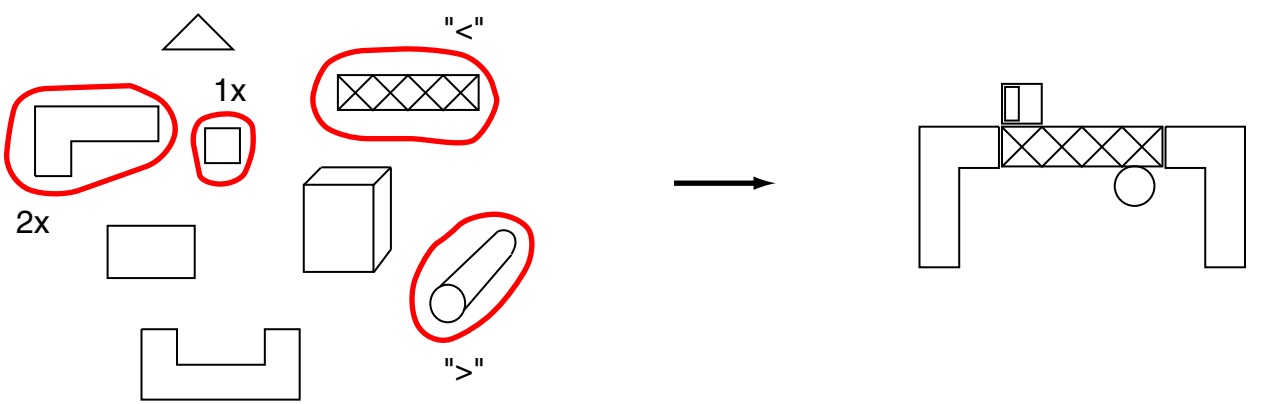
- Konfigurierungsproblemstellung
- Konfigurierungsansätze

Konfigurierungsproblemstellung

Definition 23 (Konfigurieren)

Sei D eine Menge von Anforderungen. Unter Konfigurieren versteht man die Auswahl, Parametrisierung, und Anordnung von Komponenten zu einem System (bzw. Modell eines Systems) S, so dass S alle Anforderungen D erfüllt.

Konfigurierung, Entwurf: $D \mapsto S$

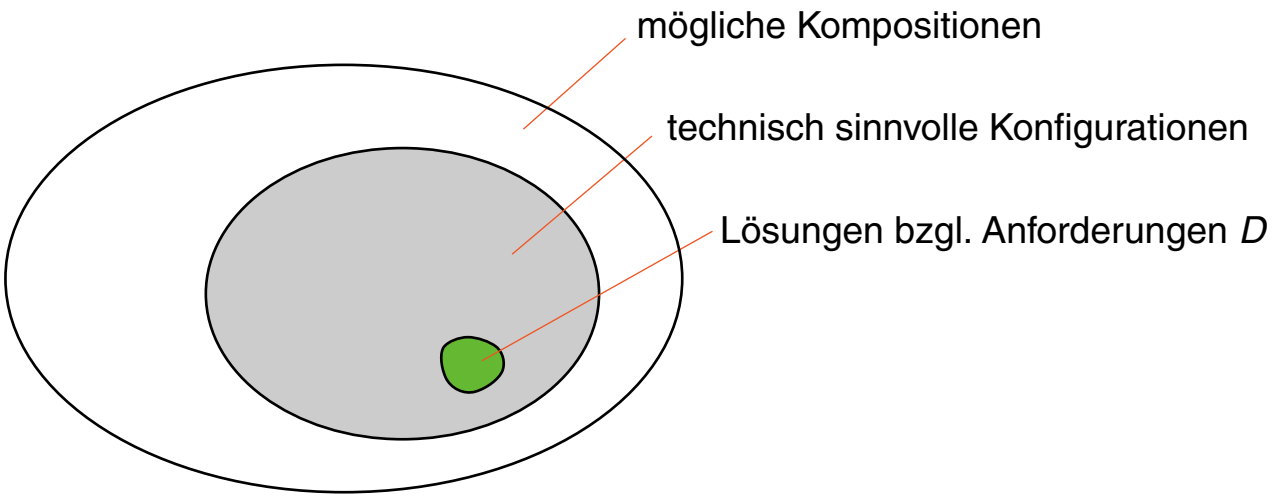


Konfigurierungsproblemstellung

Merkmale von Konfigurierungsaufgaben [vgl. Bergmann, Richter]:

- Anforderungensmenge D.
  D widersprüchlich (überspezifiziert) oder unterspezifiziert. D kann weiche Constraints, Optimierungskriterien etc. beinhalten.
- Indirekte Aufgabenstellung.
  Oft ist keine direkte Abbildung von D auf Eigenschaften von S möglich.
- System S.
  Kompositionsprinzip ist grundlegend für die Erzeugung von S.
- Constraints.
  Die Komponenten von S besitzen Eigenschaften auf denen lokale und komponentenübergreifende Constraints definiert sind.
- Zielfunktion $f(S)$.
  Optimale Lösung/System hinsichtlich eines Kriteriums gesucht.
- Suche.
  Suchraum typischerweise sehr groß und inhomogen. Verwaltung des Suchraums und Steuerung der Suche schwierig.

Konfigurierungsproblemstellung



Beispiele:

- Konfigurierung von Computern oder Rechnernetzen
- Zusammstellung eines Menüs
- Gestaltung eines Abends
- Entwurf eines neuen Autos
- Chip-Design
- Entwurf fluidischer Systeme

Bemerkungen:

- Konfigurierungsproblemstellungen gehören zur Problemklasse der Synthese.
- Eine Grenze zwischen Entwurfsproblemen und Konfigurierungsproblemen lässt sich nicht scharf ziehen.

Konfigurierungsproblemstellung

1. Erstellen einer neuen Konfiguration.
   Aufgrund einer Anforderung D ist eine neue, passende Konfiguration gesucht. Oft soll die Konfiguration optimal bzgl. eines Zielkriteriums sein.

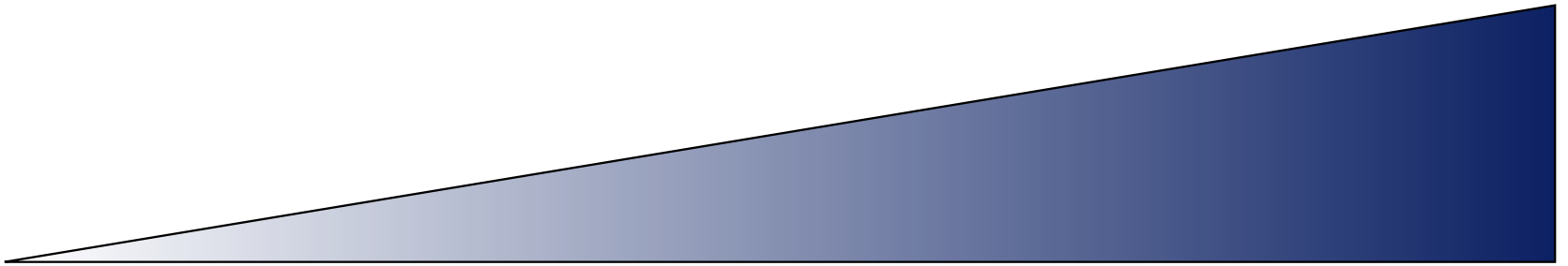
2. Parametrisieren einer Konfiguration.
   Vervollständigung der funktionalen Beschreibung einer existierenden Konfiguration. Dies kann durch einfache Berechnung oder auch durch komplexe Simulation erfolgen.

3. Überprüfen einer Konfiguration.
   Feststellung, ob eine Konfiguration S die Anforderungen D erfüllt.

4. Anpassen einer Konfiguration.
   Veränderung einer existierenden Konfiguration S derart, dass sie geänderten Anforderungen D' entspricht. Gesucht ist nicht nur eine neue, die Anforderungen erfüllende Konfiguration S', sondern auch die nötigen Operatoren, um S nach S' zu überführen.

5. Evaluierung einer Konfiguration.
   Ziel ist es, eine existierende Konfiguration S bzgl. einer gegebenen Qualitätsfunktion zu bewerten.

Konfigurierungsproblemstellung



Konfigurierung Routineentwurf Entwurf

1. Konfigurierung.
   S ist bzgl. jeder Variante durchdacht.
   Welchen Teile muss S besitzen, um D zu erfüllen?
2. Routineentwurf.
   S ist in seiner Struktur durchdacht.
   Wie sind bestimmte Teile von S zu dimensionieren, um eine Anforderung D zu erfüllen?
3. Entwurf (in fester Domäne).
   Die Struktur von S ist unbekannt.
   Wie muss S aufgebaut sein, um D zu erfüllen?

Konfigurierungsproblemstellung

Automatisierung

Typischerweise ist Konfigurieren bzw. Entwerfen ein virtueller Prozess. Die Kernprobleme bei der Automatisierung sind:

1. Modellierung.
   Entwicklung einer geeigneten Modellierung der Anwendungsdomäne.

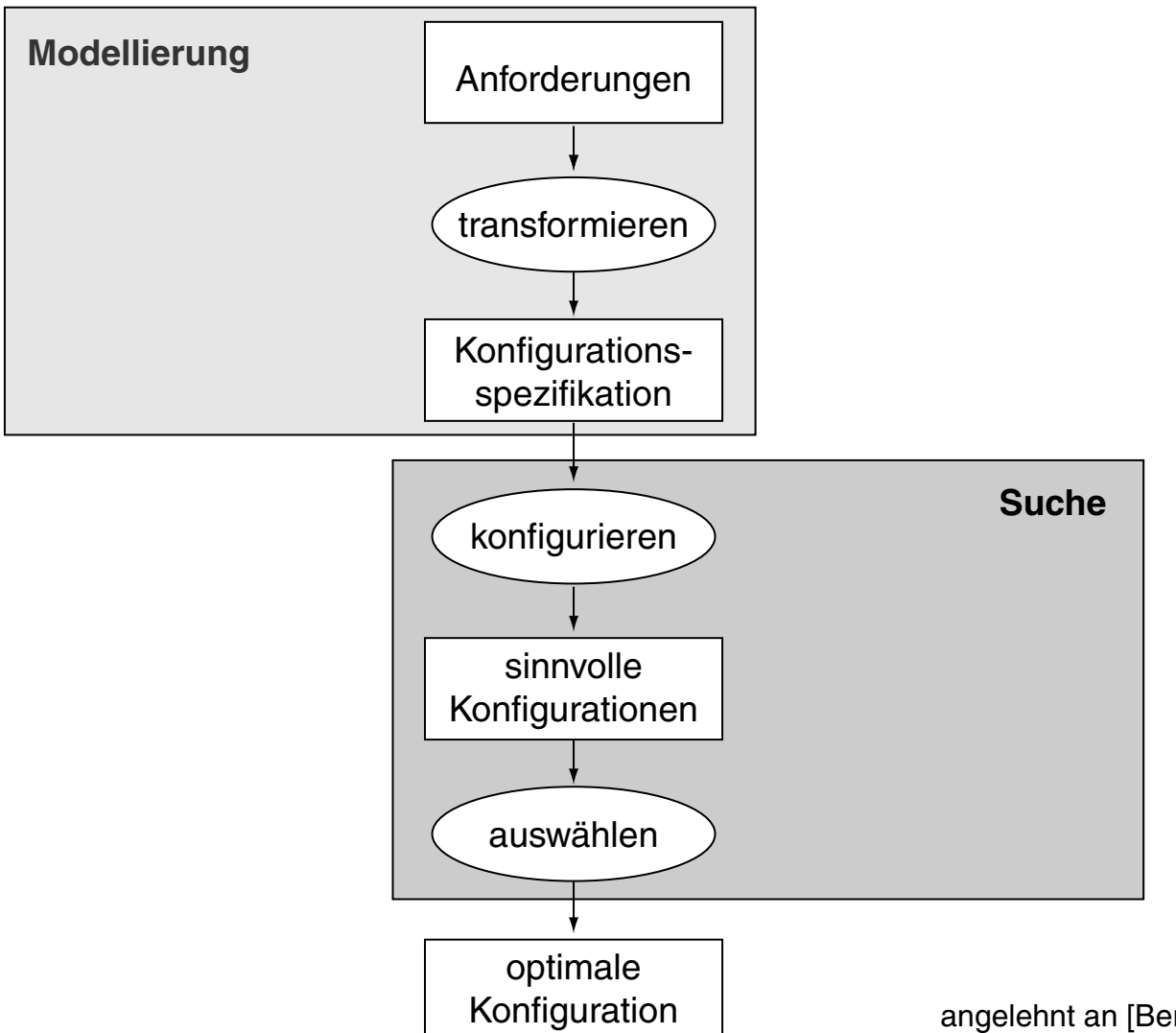
2. Suche.
   Verfahren zum Zusammensetzen und Evaluieren von Modellen.

Bemerkungen:

- Die Entwicklung einer geeigneten Modellierung ist der kritischste und schwierigste Teil bei der Automatisierung eines Konfigurierungsproblems. Sie entscheidet über Laufzeit, Akzeptanz, Wartbarkeit und Erfolg.

Konfigurierungsproblemstellung

Automatisierung (continued)



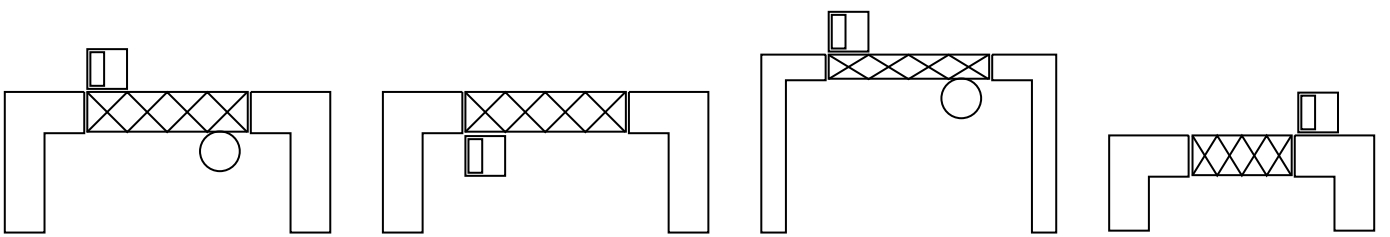
angelehnt an [Bergmann, Richter]

Konfigurierungsproblemstellung

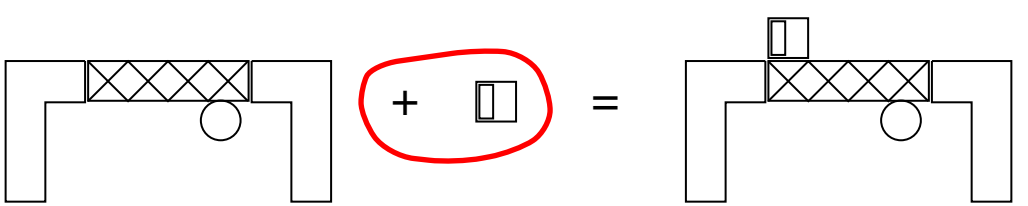
Automatisierung (continued)

Die Operationalisierung der Suche erfordert:

1. Definition des Modellraums $\mathcal{M}$.
   Welche Modelle werden betrachtet?



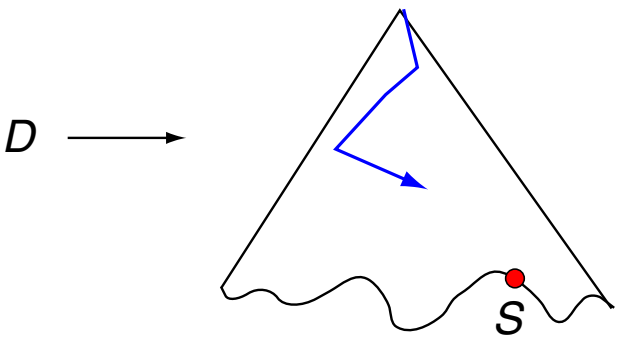
2. Definition von Operatoren.
   Wie erzeugt man aus einem Modell M ein Modell M'?



Konfigurierungsproblemstellung

Automatisierung (continued)

3. Entwicklung einer systematischen Steuerungsstrategie:
   Wie kommt man effizient zum Ziel?



Chapter MK:VI

VI. Planning and Configuration

- Agent Systems
- Deductive Reasoning Agents
- Planning Language

- Planning Algorithms
- State-Space Planning
- Plan Space Planning
- HTN Planning
- Complexity of Planning Problems
- Erweiterungen

- Konfigurierungsproblemstellung
- Konfigurierungsansätze

Konfigurierungsansätze

Paradigmen zum Lösen von Konfigurierungs- und Entwurfsaufgaben verknüpfen Aspekte aus vielen Bereichen:

- Wissensakquisition
- Suchraumgröße
- Organisation des Wissens
- Tiefe der Modellierung
- Pflege und Wartung
- Einsatzbereiche des Programms
- . . .

Wichtige Paradigmen sind:

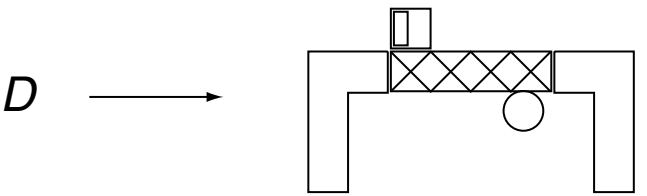
1. Skelettkonfigurieren
2. Ressourcen-basiertes Konfigurieren (Bilanzverarbeitung)
3. Fallbasiertes Konfigurieren
4. Funktionale Abstraktion

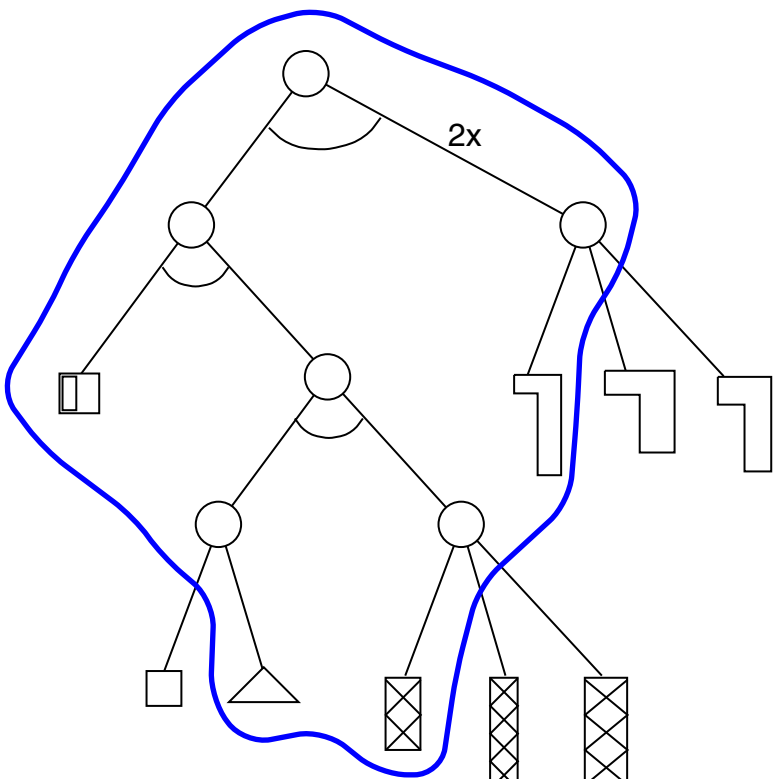
Konfigurierungsansätze

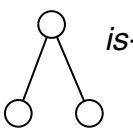
Modellierung	Paradigma
flach . . . . tief	1. Skelett-Konfigurieren ❑ Komponenten vorhanden/nicht vorhanden ❑ "'is-a"', "'has-parts"', evtl. zusätzliche Constraints 2. Ressourcen-basiertes Konfigurieren ❑ einfache funktionale Beschreibung ❑ Komponenten als Anbieter/Verbraucher von Ressourcen 3. Fallbasierter Entwurf ❑ Modell quasi ohne Einschränkung ❑ Konstruktionsspektrum stark beschränkt 4. Funktionale Abstraktion ❑ Entwurf auf einfacher funktionaler Ebene ❑ Anpassung und Simulation auf Verhaltensebene

Konfigurierungsansätze

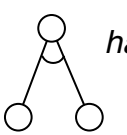
Paradigma 1: Skelett-Konfigurieren







is-a–Relation



has-parts–Relation

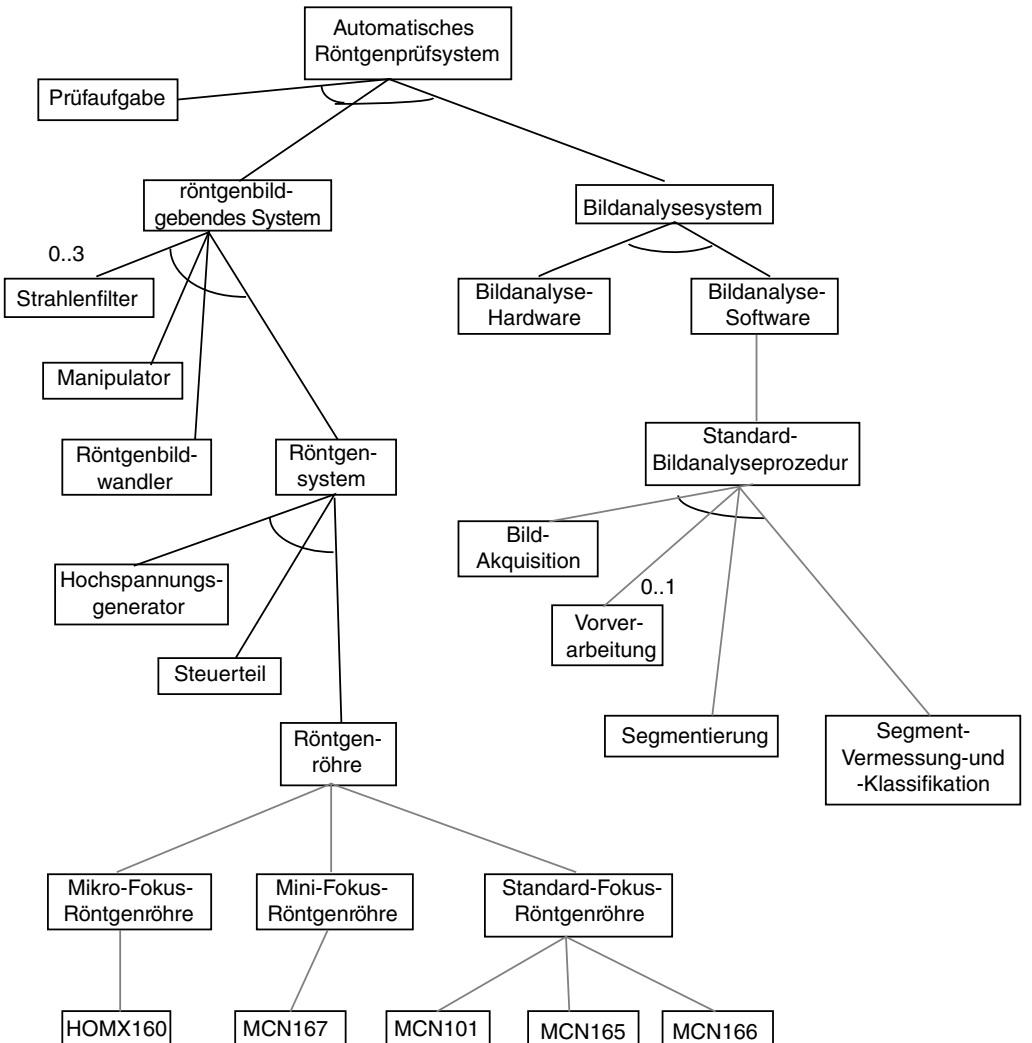
Bemerkungen:

- Der Suchraum ist ein Und-Oder-Graph.
- Und-Knoten: kompositionelle Relation
- Oder-Knoten: taxonomische Relation

Konfigurierungsansätze

Paradigma 1: Skelett-Konfigurieren (continued)

Plakon:



Bemerkungen:

- Verarbeitung des Und-Oder-Graphen mittels General Best First, GBF.
- Skelett-Konfigurieren ist sinnvoll, falls Strukturinformation die Hauptrolle spielt.
- Kann flexibel um Synthese-Constraints erweitert werden – z. B.:

```
f(Komponente_X) > 300  ->  #(Komponente_Y) = 2
```

Konfigurierungsansätze

Paradigma 1: Skelett-Konfigurieren (continued)

General Best First Search im UND-ODER-Graph, GBF. `[Collection: Suche, Part: Informierte Suchverfahren]`

Wiederholung:

- Lösungskandidaten sind Teillösungsgraphen, die in der OPEN-Liste verwaltet werden.
- Ein Teillösungsgraph kann mehrere Expansionskandidaten besitzen.
- Ein Knoten kann in mehreren Teillösungsgraphen auftauchen.

➜ Das Best-First-Prinzip wird in zwei Stufen angewandt:

1. Bewertungsfunktion f_1 für Graphen, um den erfolgverspechendsten Teillösungsgraph G_0 zu bestimmen.
2. Bewertungsfunktion f_2 für Knoten, um aus G_0 den für die Suche informativsten Knoten zu bestimmen.

Bemerkungen:

- Der Bewertungsfunktion f_1 kommt in der Regel die größere Bedeutung zu; sie basiert auf einer Schätzfunktion (Heuristik) h.

  $h(n)$: Kostenschätzung für die Lösung eines Teilproblems n. Ideal wäre eine Berücksichtigung der Wahrscheinlichkeit, ob n lösbar ist.

  $f_1(G)$: Verrechnung der Kostenschätzung für einen Teillösungsgraph G.

- Bei der Best-First-Search im Zustandsraumgraph existiert eine 1:1-Beziehung zwischen Expansionskandidaten und Lösungskandidaten: Lösungskandidaten sind die Pfade, die bei s starten und mit einem Knoten in der OPEN-Liste enden.

Konfigurierungsansätze

Paradigma 1: Skelett-Konfigurieren (continued)

Die Überprüfung, ob ein erzeugter UND-ODER-Graph G einen Lösungsgraphen enthält, kann durch rekursive Anwendung folgender Propagierungsregeln für gelöste Probleme geschehen.

Definition 24 (solved-labeling-procedure)

Gegeben sei ein UND-ODER-Graph G mit Startknoten (Wurzelknoten) s.

1. Rekursionsanfang: $n = s$
2. Ein Problem eines Knotens n in G ist gelöst, wenn eine der folgenden Bedingungen erfüllt ist.
   (a) n ist ein Blattknoten; d. h., es repräsentiert ein primitives Problem.
   (b) n ist ein (nicht-terminaler) ODER-Knoten, wobei mindestens eine seiner ODER-Kanten auf einen mit "'gelöst"' (solved) markierten Knoten zeigt.
   (c) n ist ein (nicht-terminaler) UND-Knoten, wobei alle seiner UND-Kanten auf mit "'gelöst"' (solved) markierte Knoten zeigen.

Konfigurierungsansätze

Paradigma 1: Skelett-Konfigurieren (continued)

Algorithm: GBF

Input: s. Configuration of the initial problem.
successors(n). Provides the successors of a node n.
$\perp\ (n)$. *True* if n is unsolvable.
$\star(n)$. *True* if n is solved.
$h(n)$. Heuristic cost estimation for problem n.
$f_1(G)$. Evaluation function for solution bases in explored graph.
$f_2(G_0)$. Selection function for the OPEN-nodes in a solution base.

Output: A solution graph or the symbol *'Failure'*.

Konfigurierungsansätze

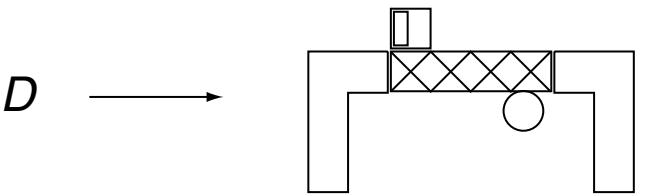
Paradigma 1: Skelett-Konfigurieren (continued)

GBF$(s, \textit{successors}, \bot, \star, h, f_1, f_2)$

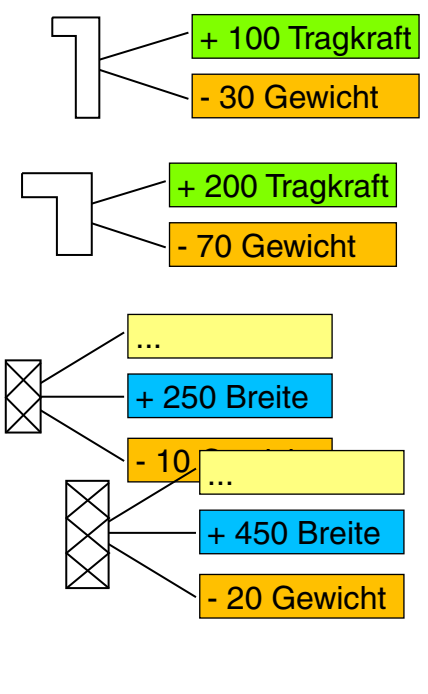
1. *insert*$(s, \texttt{OPEN})$;
   $H =$ *init_graph*(s);
2. **LOOP**
3. $G_0 =$ *min_solution_base*(H, f_1); // Most promising solution base.
4. $n = f_2(G_0)$; // Identify most informative node in G_0.
   remove$(n, \texttt{OPEN})$;
   push$(n, \texttt{CLOSED})$;
   $H =$ *expand_graph*$(\textit{successors}(n))$;
5. **FOREACH** n' IN *successors*(n) **DO**
   set_backpointer(n', n);
   IF $(\star(n')$ OR $\bot(n'))$
   THEN
   solved_labeling(H); // Apply solved-labeling-procedure to H.
   IF $\star(s)$ THEN RETURN(G_0);
   IF $\bot(s)$ THEN RETURN('*Failure*');
   cleanup_graph(H);
   ELSE *insert_node*$(n', \texttt{OPEN})$;
   ENDDO
6. **ENDLOOP**

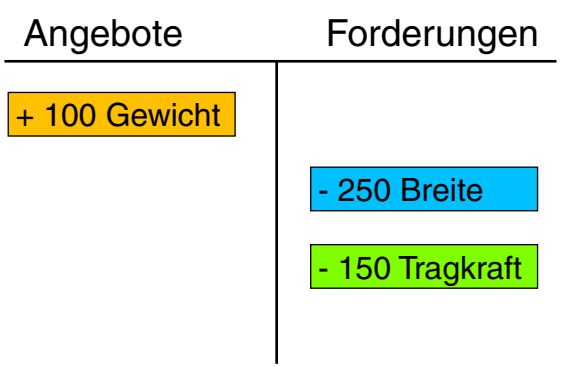
Konfigurierungsansätze

Paradigma 2: Ressourcen-basiertes Konfigurieren



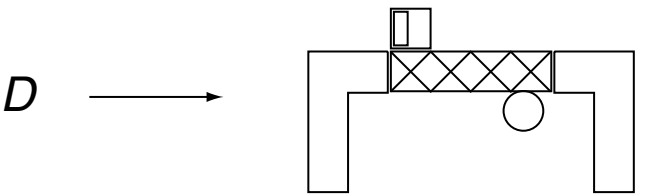
Bilanz-Initialisierung mit den Anforderungen D:

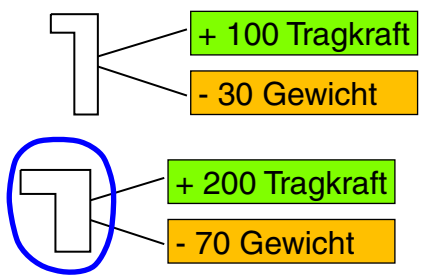


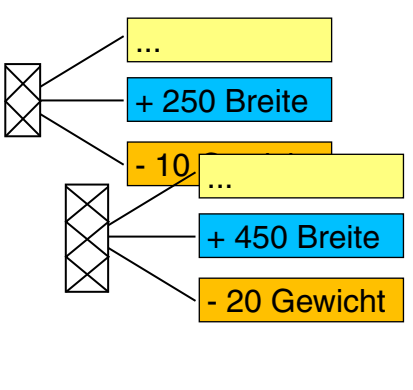


Konfigurierungsansätze

Paradigma 2: Ressourcen-basiertes Konfigurieren (continued)







...

Angebote	Forderungen	
+ 100 Gewicht	- 70 Gewicht	✓
	- 250 Breite	
+ 200 Tragkraft	- 150 Tragkraft	✓

Bemerkungen:

- Ressourcen-basiertes Konfigurieren ist Generate-and-Test mit Bilanz-Abgleich.
- Besonderheit: Lokale Modellierung ➜ Suchraum implizit gegeben.
- Sinnvoll bei modularen Systemen mit einfachen funktionalen Constraints.
- Ansatz sehr flexibel bzgl. der Wissensakquisition und -pflege.

Konfigurierungsansätze

Paradigma 2: Ressourcen-basiertes Konfigurieren (continued)

Vereinfachtes formales Modell:

- Menge von n Komponenten: $\mathbf{c_1}, \ldots, \mathbf{c_n}$

- Jede Komponente $\mathbf{c_i}$ ist charakterisiert durch m Eigenschaften:
  $\mathbf{c_i} = (c_{i_1}, \ldots, c_{i_m})^T$ wobei $c_{i_j} \in \mathbf{Z},\ i = 1, \ldots, n,\ j = 1, \ldots, m$

  Semantik:
  Komponente i bietet c_{i_j} Einheiten von Funktionalität j, falls $c_{i_j} > 0$.
  Komponente $i$fordert c_{i_j} Einheiten von Funktionalität j, falls $c_{i_j} \leq 0$.

- Eine Konfiguration $\mathbf{s} = (s_1, \ldots, s_n)$ ist ein Vektor aus $\mathbf{N}^n$.

  Semantik:
  s_i definiert, wie oft Komponente i Teil der Konfiguration ist.

Konfigurierungsansätze

Paradigma 2: Ressourcen-basiertes Konfigurieren (continued)

Vereinfachtes formales Modell (continued):

- Sei $C = (\mathbf{c_1}, \ldots, \mathbf{c_n})$. Eine Konfiguration $\mathbf{s}$ ist *korrekt*, falls gilt:

$$\sum_{i=1}^{n} s_i \cdot \mathbf{c_i} = C \cdot \mathbf{s} \geq \mathbf{0}$$

- Eine Anforderungsmenge $\mathbf{d} = (d_1, \ldots, d_m)$ ist ein Vektor aus $\mathbf{N}^m$. Eine Konfiguration $\mathbf{s}$ ist *zulässig*, falls gilt:

$$C \cdot \mathbf{s} \geq \mathbf{d}$$

Konfigurierungsansätze

Paradigma 2: Ressourcen-basiertes Konfigurieren (continued)

Vereinfachtes formales Modell (continued):

- Ein Preisvektor $\mathbf{p} = (p_1, \ldots, p_n)$ ist ein Vektor aus $\mathbf{N}^n$.

  Semantik:
  p_i definiert, wie teuer Komponente i ist. Dann definiert das Skalarprodukt

  $$\langle \mathbf{s}, \mathbf{p} \rangle = \sum_{i=1}^{n} s_i \cdot p_i$$

  den Preis einer Konfiguration $\mathbf{s}$.

- Eine Konfiguration $\mathbf{s}^*$ ist *optimal* bzgl. einer Anforderungsmenge $\mathbf{d}$, falls gilt:

  $$C \cdot \mathbf{s}^* \geq \mathbf{d} \quad \wedge \quad \forall \mathbf{s} \in \mathbf{N}^n : \; C \cdot \mathbf{s} \geq \mathbf{d} \; \rightarrow \; \langle \mathbf{s}, \mathbf{p} \rangle \geq \langle \mathbf{s}^*, \mathbf{p} \rangle$$

Bemerkungen:

- Das vereinfachte formale Modell kann in vieler Hinsicht erweitert werden. Insbesondere ist die einfache Addition von Funktionalitäten zur Modellierung realer Anwendungen zu schwach.

Konfigurierungsansätze

Paradigma 2: Ressourcen-basiertes Konfigurieren (continued)

Beispiel:

- Vier Funktionalitäten: f_1, f_2, f_3, f_4
- Drei Komponentenbeschreibungen:
  $((f_1, -1), (f_3, 2), (f_4, -1))$,
  $((f_1, 2), (f_2, 1), (f_3, -1), (f_4, 2))$,
  $((f_2, 2), (f_3, 1))$

$$\rightarrow \qquad C = \begin{pmatrix} -1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & -1 & 1 \\ -1 & 2 & 0 \end{pmatrix}$$

- Preisvektor der Komponenten: $\mathbf{p} = (1, 2, 4)^T$

Dann gilt u. a.:

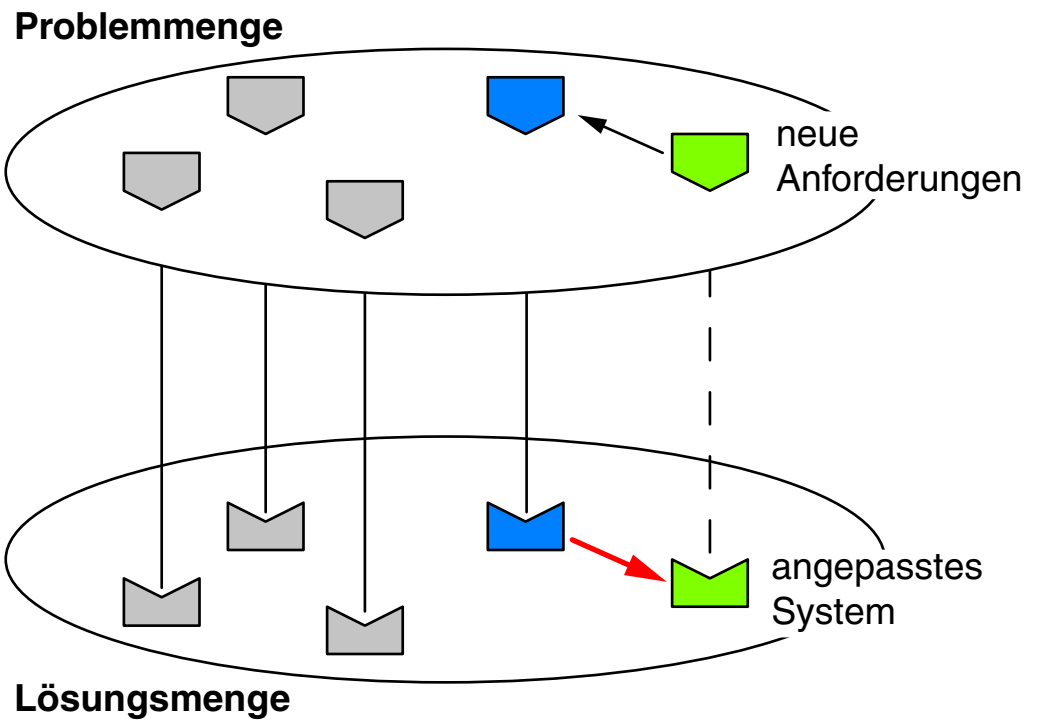
Die Konfiguration $\mathbf{s} = (3, 1, 0)^T$ ist nicht korrekt, da $C \cdot \mathbf{s} = (-1, 1, 5, -1)^T \not\geq \mathbf{0}$.
Die Konfiguration $\mathbf{s} = (1, 1, 0)^T$ ist korrekt und kostet 4.

Konfigurierungsansätze

Paradigma 3: Fallbasiertes Konfigurieren

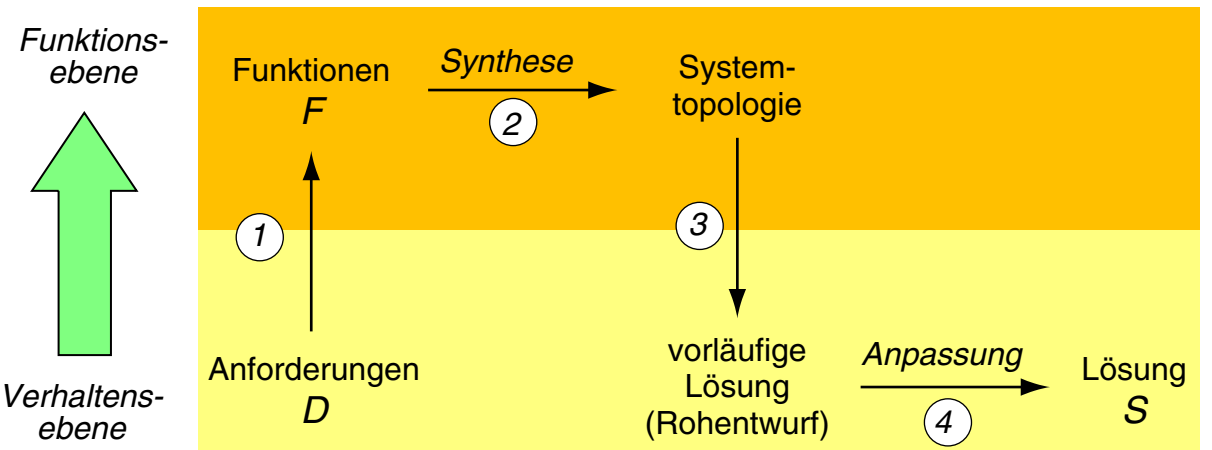
Ein Fall besteht aus

1. einer Anforderungsdefinition D und
2. einer optimalen Konfiguration S für D.



Konfigurierungsansätze

Paradigma 4: Funktionale Abstraktion



1. Abstraktion des Entwurfsproblems:
   "'Leite aus den Anforderungen D abstrakte Funktionen F ab."'
2. Lösung von $F \longrightarrow S$ in einem stark vereinfachtem Syntheseraum:
   "'Entwerfe Topologie für das System S auf Basis von F."'
3. Rücktransformation in den ursprünglichen Syntheseraum:
   "'Erzeuge das zugehörige Verhaltensmodell."'
4. Anpassung der Lösung:
   "'Analysiere Verhalten und führe Verbesserungen durch."'

Konfigurierungsansätze

Realisierungsaspekte

Konfigurierungssystem

Technische Sicht

- optimale Lösungen
- effiziente Algorithmen
- organisatorische Aspekte

Wartung

- Wissen über Komponenten
- Konsistenz der Daten
- Terminologie der Domäne

Interface

- Schnittstelle zum Anwender
- Spezifikation neuer Probleme
- Generierung von Erklärungen