Wissensbasierte Softwareentwicklung

18. November 2003

Christoph Kreitz

Theoretische Informatik, Raum 1.18, Telephon 3060

kreitz@cs.uni-potsdam.de

http://www.cs.uni-potsdam.de/ti/kreitz

Wissensbasierte Softwareentwicklung – wozu?

Vermeidung kostspieliger Fehler

Vermeidung kostspieliger Fehler

- **Zu viele Fehler in Wissenschaft und Praxis**
  - 40–50% aller veröffentlichten Resultate sind falsch
  - Fast alle Softwareprodukte sind unzuverlässig

Wissensbasierte Softwareentwicklung – wozu?

Vermeidung kostspieliger Fehler

- **Zu viele Fehler in Wissenschaft und Praxis**
  - 40–50% aller veröffentlichten Resultate sind falsch
  - Fast alle Softwareprodukte sind unzuverlässig
- **Informale Überprüfung ist unzureichend**
  - Fehler werden beim Reviewprozeß übersehen
  - Auch sorgfältig getestete Programme enthalten größere Fehler

Wissensbasierte Softwareentwicklung – wozu?

Vermeidung kostspieliger Fehler

- **Zu viele Fehler in Wissenschaft und Praxis**
  - 40–50% aller veröffentlichten Resultate sind falsch
  - Fast alle Softwareprodukte sind unzuverlässig
- **Informale Überprüfung ist unzureichend**
  - Fehler werden beim Reviewprozeß übersehen
  - Auch sorgfältig getestete Programme enthalten größere Fehler
- **Wir können uns Softwarefehler nicht erlauben**
  - Software ist ein integraler Bestandteil unseres Lebens

Wissensbasierte Softwareentwicklung – wozu?

Vermeidung kostspieliger Fehler

- **Zu viele Fehler in Wissenschaft und Praxis**
  - 40–50% aller veröffentlichten Resultate sind falsch
  - Fast alle Softwareprodukte sind unzuverlässig
- **Informale Überprüfung ist unzureichend**
  - Fehler werden beim Reviewprozeß übersehen
  - Auch sorgfältig getestete Programme enthalten größere Fehler
- **Wir können uns Softwarefehler nicht erlauben**
  - Software ist ein integraler Bestandteil unseres Lebens
  - Softwarefehler sind lästig (Reboot, Datenverlust, ...)

Wissensbasierte Softwareentwicklung – wozu?

Vermeidung kostspieliger Fehler

- **Zu viele Fehler in Wissenschaft und Praxis**
  - 40–50% aller veröffentlichten Resultate sind falsch
  - Fast alle Softwareprodukte sind unzuverlässig
- **Informale Überprüfung ist unzureichend**
  - Fehler werden beim Reviewprozeß übersehen
  - Auch sorgfältig getestete Programme enthalten größere Fehler
- **Wir können uns Softwarefehler nicht erlauben**
  - Software ist ein integraler Bestandteil unseres Lebens
  - Softwarefehler sind lästig (Reboot, Datenverlust, ...)
    teuer (Pentium I, Ariane V, Mars Climate Orbiter)

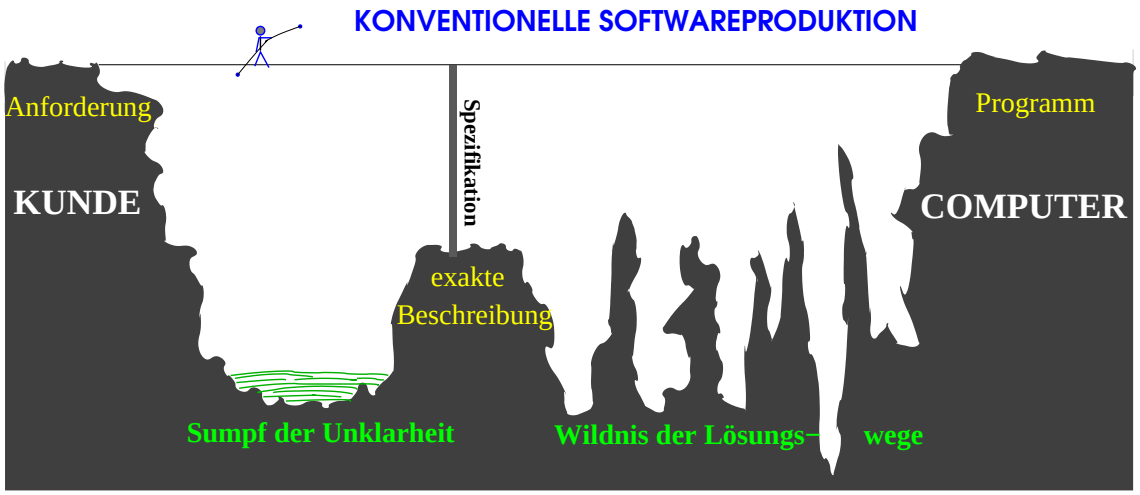
Wissensbasierte Softwareentwicklung – wozu?

Vermeidung kostspieliger Fehler

- **Zu viele Fehler in Wissenschaft und Praxis**
  - 40–50% aller veröffentlichten Resultate sind falsch
  - Fast alle Softwareprodukte sind unzuverlässig
- **Informale Überprüfung ist unzureichend**
  - Fehler werden beim Reviewprozeß übersehen
  - Auch sorgfältig getestete Programme enthalten größere Fehler
- **Wir können uns Softwarefehler nicht erlauben**
  - Software ist ein integraler Bestandteil unseres Lebens
  - Softwarefehler sind lästig (Reboot, Datenverlust, ...)
    teuer (Pentium I, Ariane V, Mars Climate Orbiter)
    fatal (Airbus-Abstürze in den frühen 1990ern)

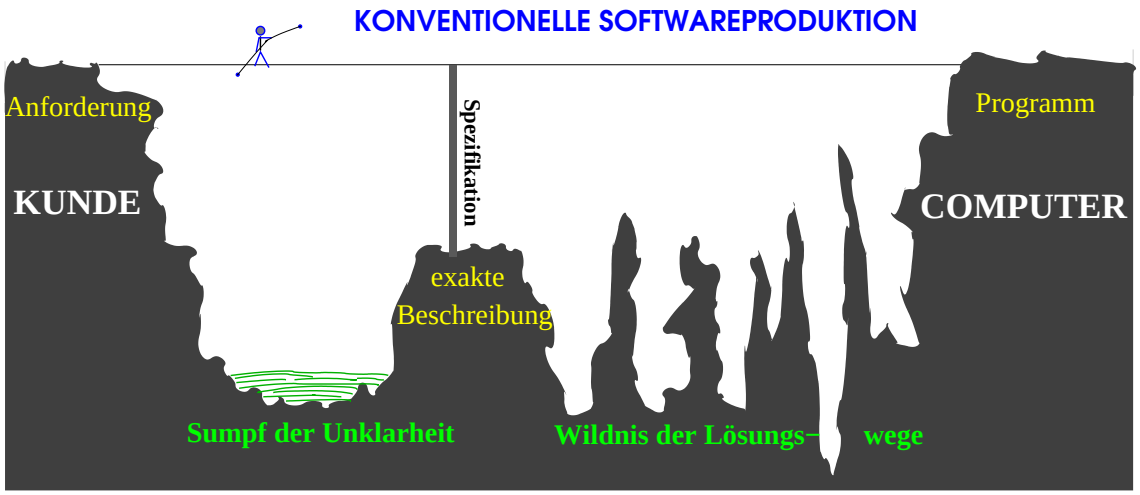
Wissensbasierte Softwareentwicklung – wozu? (II)

Erzeugung bessere Software



Wissensbasierte Softwareentwicklung – wozu? (II)

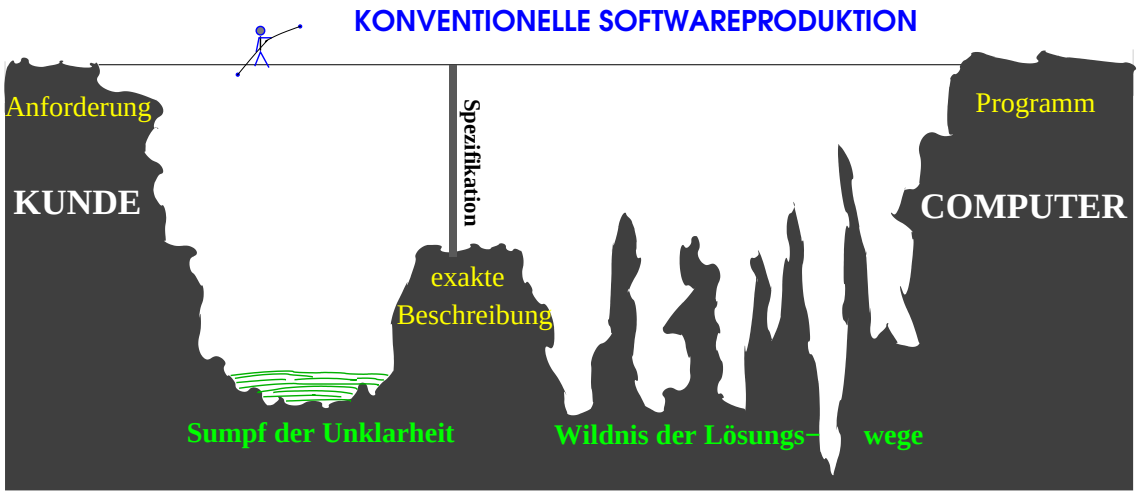
Erzeugung bessere Software



- **Konventionelle Softwareproduktion is ineffizient**
  - Codierung 'von Hand' mit hohen Kosten für Erstellung und Wartung
  - Focus auf Programmiersprache behindert Ideenreichtum
  - Resultate oft suboptimal, funktionieren selten auf Anhieb
  - Modifikationen und Erweiterungen schwierig

Wissensbasierte Softwareentwicklung – wozu? (II)

Erzeugung bessere Software



- **Konventionelle Softwareproduktion is ineffizient**
  - Codierung 'von Hand' mit hohen Kosten für Erstellung und Wartung
  - Focus auf Programmiersprache behindert Ideenreichtum
  - Resultate oft suboptimal, funktionieren selten auf Anhieb
  - Modifikationen und Erweiterungen schwierig
- **Softwareentwicklung ist mehr als Codierung**
  - Logische Verarbeitung von Wissen (schematisierbar)
  - Kreative Umsetzung von Ideen

Beispiel: Maximale Segmentsumme einer Liste

Gegeben eine Folge $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ bestimme die Summe $\Sigma_{i=p}^{q} a_i$ eines Segmentes, die maximal bezüglich aller möglicher Segmentsummen ist

2	**3**	**-6**	**4**	**5**	**-3**	**8**	**-2**	**-1**	**5**	**-9**	**2**	**3**

Beispiel: Maximale Segmentsumme einer Liste

Gegeben eine Folge $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ bestimme die Summe $\Sigma_{i=p}^{q} a_i$ eines Segmentes, die maximal bezüglich aller möglicher Segmentsummen ist

2	**3**	**-6**	**4**	**5**	**-3**	**8**	**-2**	**-1**	**5**	**-9**	**2**	**3**

16

Beispiel: Maximale Segmentsumme einer Liste

Gegeben eine Folge $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ bestimme die Summe $\Sigma_{i=p}^{q} a_i$ eines Segmentes, die maximal bezüglich aller möglicher Segmentsummen ist

2	3	-6	4	5	-3	8	-2	-1	5	-9	2	3

16

- **Direkte Lösung leicht zu programmieren**

```
let maxseg a ≡
   result := a[1]
   from p = 1 to length(a) do
      from q = i to length(a) do
         sum := 0
         from i = p to q do sum := sum+a[i] end
         if sum > result then result := sum
      end
   end
```

Beispiel: Maximale Segmentsumme einer Liste

Gegeben eine Folge $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ bestimme die Summe $\Sigma_{i=p}^{q} a_i$ eines Segmentes, die maximal bezüglich aller möglicher Segmentsummen ist

2	3	-6	4	5	-3	8	-2	-1	5	-9	2	3

16

- **Direkte Lösung leicht zu programmieren**

```
let maxseg a ≡
   result := a[1]
   from p = 1 to length(a) do
      from q = i to length(a) do
         sum := 0
         from i = p to q do sum := sum+a[i] end
         if sum > result then result := sum
      end
   end
```

- **Algorithmus ist ineffizient** ($(length(a))^3$)
  - **Wie kann man eine bessere Lösung erzeugen?**

Maximale Segmentsumme: systematische Lösung

$$\textit{Analysiere Eigenschaften von } \boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$$

Maximale Segmentsumme: systematische Lösung

Analysiere Eigenschaften von $\boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$

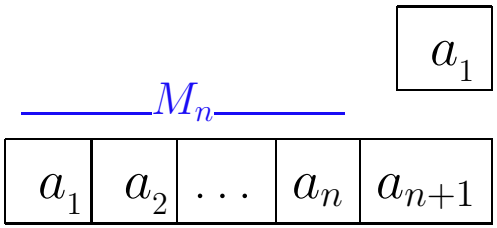
- **Induktive Analyse liefert**
  - $M_1 = a_1$

a_1

Maximale Segmentsumme: systematische Lösung

Analysiere Eigenschaften von $\boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$

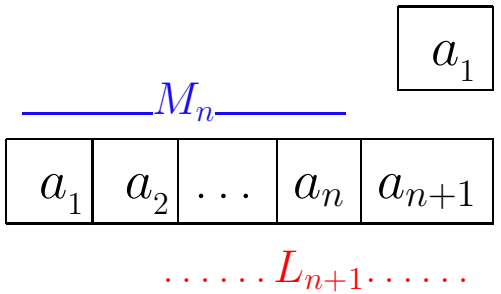
- **Induktive Analyse liefert**
  - $M_1 = a_1$
  - $M_{n+1} = max(M_n, max\{\Sigma_{i=p}^{n+1} a_i \mid 1 \leq p \leq n\})$



Maximale Segmentsumme: systematische Lösung

Analysiere Eigenschaften von $\boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$

- **Induktive Analyse liefert**
  - $M_1 = a_1$
  - $M_{n+1} = max(M_n, max\{\Sigma_{i=p}^{n+1} a_i \mid 1 \leq p \leq n\})$

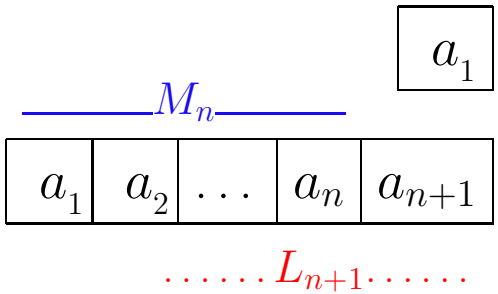


- **Definiere** $\boldsymbol{L_n} \equiv \boldsymbol{max\{\Sigma_{i=p}^{n} a_i \mid 1 \leq p \leq n\}}$

Maximale Segmentsumme: systematische Lösung

Analysiere Eigenschaften von $\boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$

- **Induktive Analyse liefert**
  - $M_1 = a_1$
  - $M_{n+1} = max(M_n, max\{\Sigma_{i=p}^{n+1} a_i \mid 1 \leq p \leq n\})$

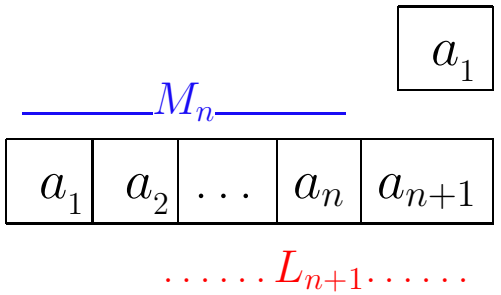


- **Definiere** $\boldsymbol{L_n} \equiv \boldsymbol{max\{\Sigma_{i=p}^{n} a_i \mid 1 \leq p \leq n\}}$
  - $L_1 = a_1$,

Maximale Segmentsumme: systematische Lösung

Analysiere Eigenschaften von $\boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$

- **Induktive Analyse liefert**
  - $M_1 = a_1$
  - $M_{n+1} = max(M_n, max\{\Sigma_{i=p}^{n+1} a_i \mid 1 \leq p \leq n\})$

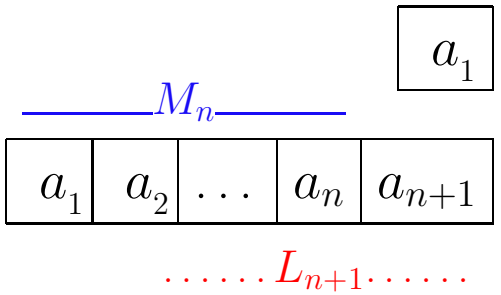


- **Definiere** $\boldsymbol{L_n} \equiv \boldsymbol{max\{\Sigma_{i=p}^{n} a_i \mid 1 \leq p \leq n\}}$
  - $L_1 = a_1\,, \qquad L_{n+1} = max(L_n + a_{n+1},\ a_{n+1})$

Maximale Segmentsumme: systematische Lösung

Analysiere Eigenschaften von $\boldsymbol{M_n} \equiv max\{\Sigma_{i=p}^{q} a_i \mid 1 \leq p \leq q \leq n\}$

- **Induktive Analyse liefert**
  - $M_1 = a_1$
  - $M_{n+1} = max(M_n, max\{\Sigma_{i=p}^{n+1} a_i \mid 1 \leq p \leq n\})$



- **Definiere** $\boldsymbol{L_n} \equiv \boldsymbol{max\{\Sigma_{i=p}^{n} a_i \mid 1 \leq p \leq n\}}$
  - $L_1 = a_1\,, \qquad L_{n+1} = max(L_n + a_{n+1},\ a_{n+1})$
- **Analyse liefert eleganten, effizienten Algorithmus**

```
let maxseg a ≡
    M_n := a[1]
    L_n := a[1]
    from n = 1 to length(a) do
       if L_n>0 then L_n := L_n+a[n] else L_n := a[n]
       if L_n>M_n then M_n := L_n
    end
```

Wissensbasierte Softwareentwicklung – Chancen

- **Logische Analyse führt zu besserer Software**
  - Wissen wird systematisch verarbeitet

Wissensbasierte Softwareentwicklung – Chancen

- **Logische Analyse führt zu besserer Software**
  - Wissen wird systematisch verarbeitet
- **Formalisierung der Argumente eliminiert Fehler**
  - Computer kontrolliert Schlüsse in logischem Kalkül
  - Kreative Steuerung des Entwicklungsprozesses durch Menschen
  - Kooperation zwischen Mensch und Maschine bei Entwicklung
  - Computer synthetisiert Code aus formaler Problemanalyse

Wissensbasierte Softwareentwicklung – Chancen

- **Logische Analyse führt zu besserer Software**
  - Wissen wird systematisch verarbeitet
- **Formalisierung der Argumente eliminiert Fehler**
  - Computer kontrolliert Schlüsse in logischem Kalkül
  - Kreative Steuerung des Entwicklungsprozesses durch Menschen
  - Kooperation zwischen Mensch und Maschine bei Entwicklung
  - Computer synthetisiert Code aus formaler Problemanalyse
- **Automatisierung reduziert Aufwand**
  - Computer wählt "trivialer" logische Schlüsse von selbst aus
  - Formales Wissen steuert Auswahl

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2					

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**				

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**			

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**		

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		
-4	**-1**	**-2**			

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		
-4	**-1**	**-2**			
-3	**1**				

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		
-4	**-1**	**-2**			
-3	**1**				
-1					

Costas Array der Ordnung 6 und seine Differenzentafel

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		
-4	**-1**	**-2**			
-3	**1**				
-1					

Costas Array der Ordnung 6 und seine Differenzentafel

– Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		
-4	**-1**	**-2**			
-3	**1**				
-1					

Costas Array der Ordnung 6 und seine Differenzentafel

– Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

Ziel: Erzeugung aller Costas Arrays der Größe n

Beispiel-Problem: Costas-Arrays

Costas Array der Größe n:

– Permutation von $\{1..n\}$ ohne doppelte Elemente in Zeilen der Differenzentafel

2	**4**	**1**	**6**	**5**	**3**
-2	**3**	**-5**	**1**	**2**	
1	**-2**	**-4**	**3**		
-4	**-1**	**-2**			
-3	**1**				
-1					

Costas Array der Ordnung 6 und seine Differenzentafel

– Hilfreich für Erzeugung leicht decodierbarer Radar- und Sonarsignale

Ziel: Erzeugung aller Costas Arrays der Größe n

Bis 1988 kein effizienter Lösungsalgorithmus bekannt

Verarbeitung von Domänen- und Programmierwissen

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**
  - Fakten: Definitionen von Begriffen, Lemmata über Eigenschaften

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**
  - Fakten: Definitionen von Begriffen, Lemmata über Eigenschaften
  - Methodenwissen: Standardalgorithmen und typische Beweistechniken

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**
  - Fakten: Definitionen von Begriffen, Lemmata über Eigenschaften
  - Methodenwissen: Standardalgorithmen und typische Beweistechniken
  - Allgemeines und Problem-spezifisches Wissen

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**
  - Fakten: Definitionen von Begriffen, Lemmata über Eigenschaften
  - Methodenwissen: Standardalgorithmen und typische Beweistechniken
  - Allgemeines und Problem-spezifisches Wissen
- **Programmierwissen $\hat{=}$ Programmiertechniken**

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**
  - Fakten: Definitionen von Begriffen, Lemmata über Eigenschaften
  - Methodenwissen: Standardalgorithmen und typische Beweistechniken
  - Allgemeines und Problem-spezifisches Wissen
- **Programmierwissen $\hat{=}$ Programmiertechniken**
  - Allgemeine algorithmische Strukturen

Verarbeitung von Domänen- und Programmierwissen

- **Domänenwissen: Wissen zu Anwendungsbereichen**
  - Fakten: Definitionen von Begriffen, Lemmata über Eigenschaften
  - Methodenwissen: Standardalgorithmen und typische Beweistechniken
  - Allgemeines und Problem-spezifisches Wissen
- **Programmierwissen $\hat{=}$ Programmiertechniken**
  - Allgemeine algorithmische Strukturen
  - Formale Programmentwicklungstechniken
    (Synthese, Optimierung, Aspekte, ...)

Allgemeines Domänenwissen

- **Grundwissen über Datenstrukturen**
  - Listen, Bäume, Graphen, Zahlentheorie, ...
  - Formalisierung von Begriffen und Notationen
  - Formal bewiesene Eigenschaften (Distributivgesetze, ...)

Allgemeines Domänenwissen

- **Grundwissen über Datenstrukturen**
  - Listen, Bäume, Graphen, Zahlentheorie, ...
  - Formalisierung von Begriffen und Notationen
  - Formal bewiesene Eigenschaften (Distributivgesetze, ...)
- **Material der Informatik Grundausbildung**
  - Mathematik 1-3
  - Konzepte aus Grundlagen der Programmierung

⋮

Allgemeines Domänenwissen

- **Grundwissen über Datenstrukturen**
  - Listen, Bäume, Graphen, Zahlentheorie, ...
  - Formalisierung von Begriffen und Notationen
  - Formal bewiesene Eigenschaften (Distributivgesetze, ...)
- **Material der Informatik Grundausbildung**
  - Mathematik 1-3
  - Konzepte aus Grundlagen der Programmierung

  ⋮
- **Wie kommen wir an das Wissen?**
  - Formalisiere Inhalt von Lehrmaterial, -büchern Forschungsergebnissen
  - Beweise Eigenschaften und verifiziere Standardalgorithmen

Problem-spezifisches Domänenwissen

- **Spezielles Wissen zum Problembereich**
  - Formalisierung spezieller Konzepte
  - Analyse der Grundeigenschaften dieser Konzepte

Problem-spezifisches Domänenwissen

- **Spezielles Wissen zum Problembereich**
  - Formalisierung spezieller Konzepte
  - Analyse der Grundeigenschaften dieser Konzepte
- **Formalisierung des Costas Arrays Problems**

  "Permutation ohne Duplikate in Zeilen der Differenzentafel"

Problem-spezifisches Domänenwissen

- **Spezielles Wissen zum Problembereich**
  - Formalisierung spezieller Konzepte
  - Analyse der Grundeigenschaften dieser Konzepte
- **Formalisierung des Costas Arrays Problems**

  "Permutation ohne Duplikate in Zeilen der Differenzentafel"

  - `perm(`π`,`S`)` $\equiv$ `nodups(`π`)` $\wedge$ `range(`π`)=`S

Problem-spezifisches Domänenwissen

- **Spezielles Wissen zum Problembereich**
  - Formalisierung spezieller Konzepte
  - Analyse der Grundeigenschaften dieser Konzepte
- **Formalisierung des Costas Arrays Problems**

  "Permutation ohne Duplikate in Zeilen der Differenzentafel"

  - `perm(`π`,`S`)` $\equiv$ `nodups(`π`)` $\wedge$ `range(`π`)=`S
  - `nodups(`L`)` $\equiv$ $\forall$`i`$\neq$`j`$\in$`dom(`π`).` L`[i]`$\neq$$L$`[j]`

Problem-spezifisches Domänenwissen

- **Spezielles Wissen zum Problembereich**
  - Formalisierung spezieller Konzepte
  - Analyse der Grundeigenschaften dieser Konzepte
- **Formalisierung des Costas Arrays Problems**

  "Permutation ohne Duplikate in Zeilen der Differenzentafel"

  - `perm(`π`,`S`)` $\equiv$ `nodups(`π`)` $\wedge$ `range(`π`)=`S
  - `nodups(`L`)` $\equiv$ $\forall$`i`$\neq$`j`$\in$`dom(`π`).` L`[i]`$\neq$$L$`[j]`
  - `dt-row(`π`,`j`)` $\equiv$ `[`π`[i]-`π`[i+`j`] | i`$\in$`[1..length(`π`)-`j`]]`

Problem-spezifisches Domänenwissen

- **Spezielles Wissen zum Problembereich**
  - Formalisierung spezieller Konzepte
  - Analyse der Grundeigenschaften dieser Konzepte
- **Formalisierung des Costas Arrays Problems**

  "Permutation ohne Duplikate in Zeilen der Differenzentafel"

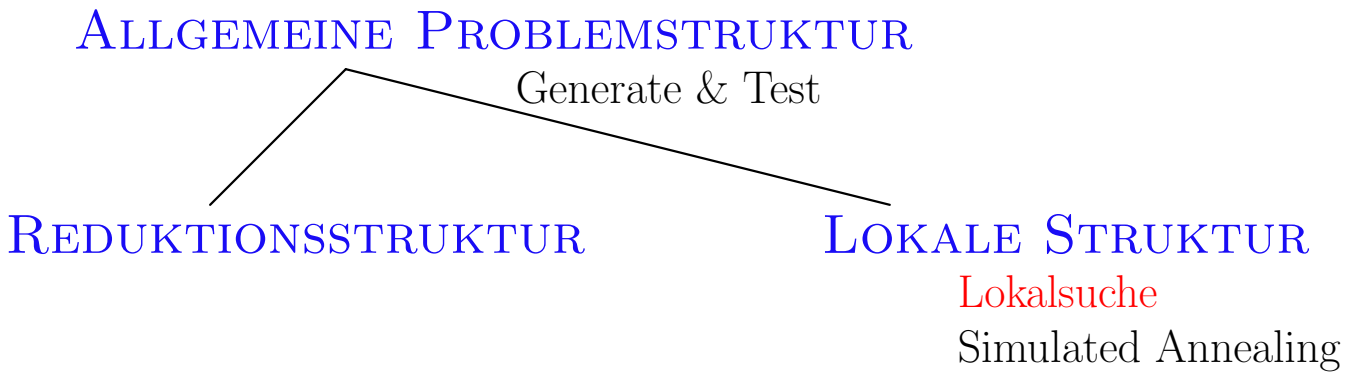
  - perm(π,S) $\equiv$ nodups(π) $\wedge$ range(π)=S
  - nodups(L) $\equiv$ $\forall$i$\neq$j$\in$dom(π). L[i]$\neq$$L$[j]
  - dt-row(π,j) $\equiv$ [π[i]-π[i+j] | i$\in$[1..length(π)-j]]
- **Gesetze der Differenzentafeln**
  - dtrow([],j)=[]
  - j$\leq$|L| $\Rightarrow$ dtrow(i.L,j) = (i-L[j]).dtrow(L,j)
  - j$\Leftrightarrow$0 $\Rightarrow$ dtrow([i],j) = []
  - L$\sqsubseteq$L' $\Rightarrow$ dtrow(L,j)$\sqsubseteq$dtrow(L',j)
  - j$\geq$|L| $\Rightarrow$ dtrow(L,j)= []
  - j$\leq$|L| $\Rightarrow$ dtrow(L$\cdot$i,j) = dtrow(L,j)$\cdot$(L[|L|+1-j]-i)

Programmierwissen: algorithmische Strukturen

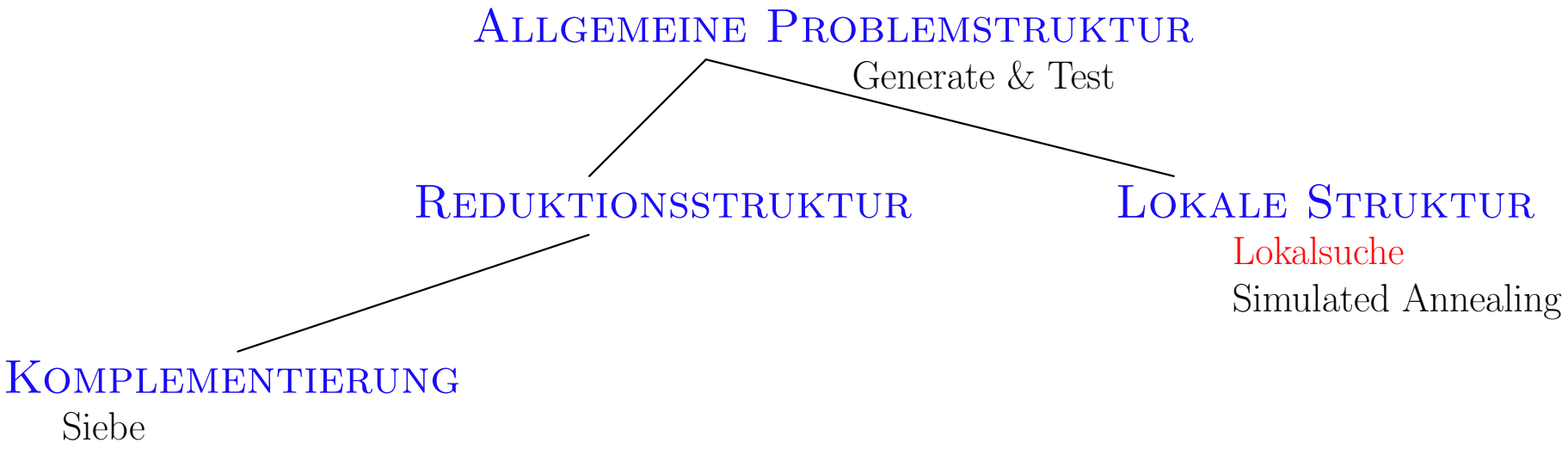
Allgemeine Problemstruktur

Generate & Test

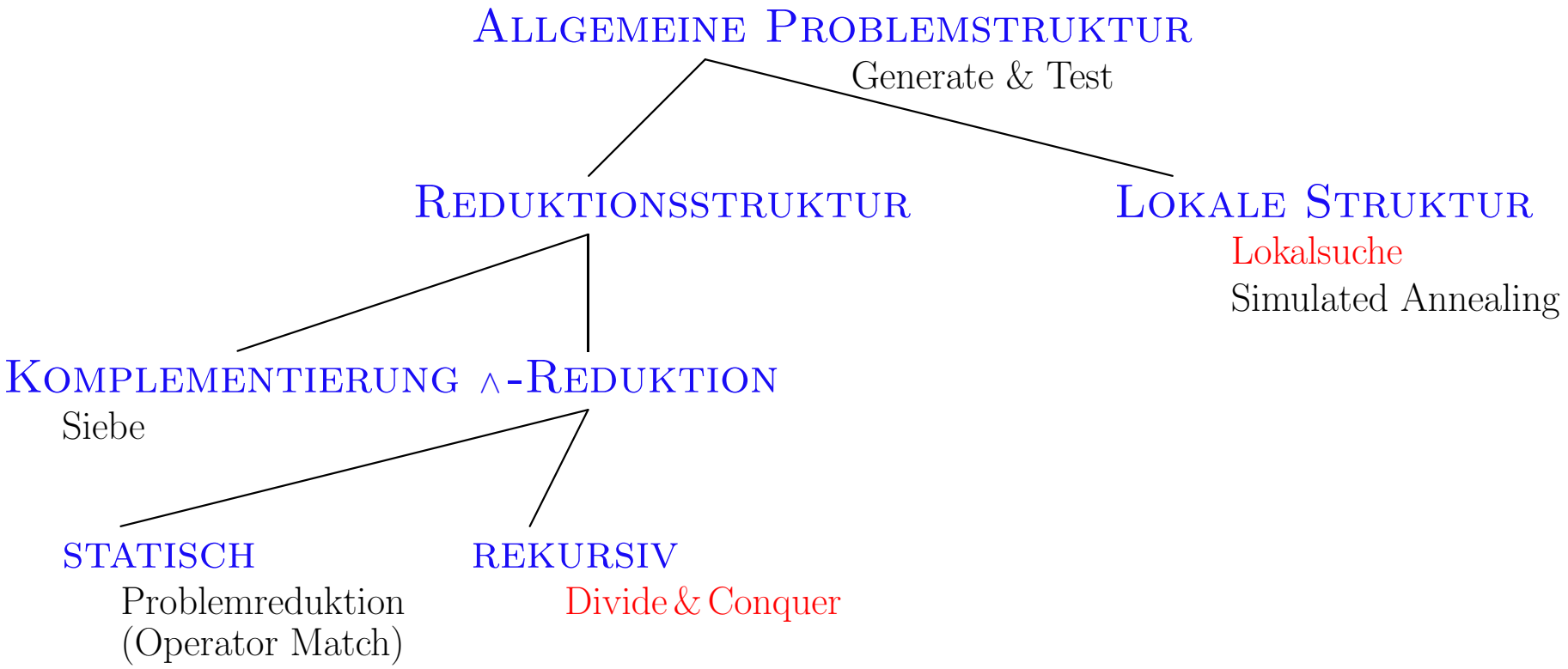
Programmierwissen: algorithmische Strukturen



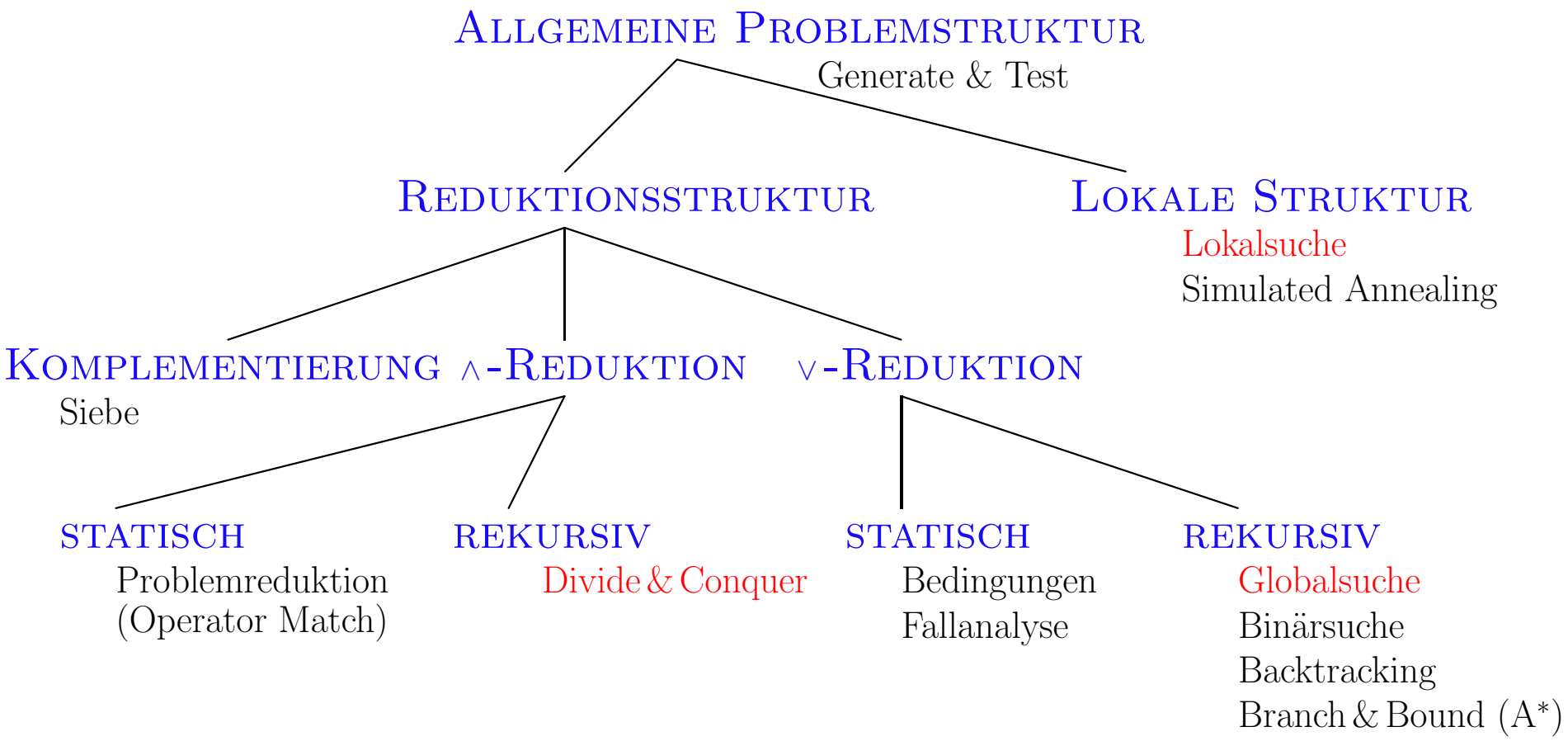
Programmierwissen: algorithmische Strukturen



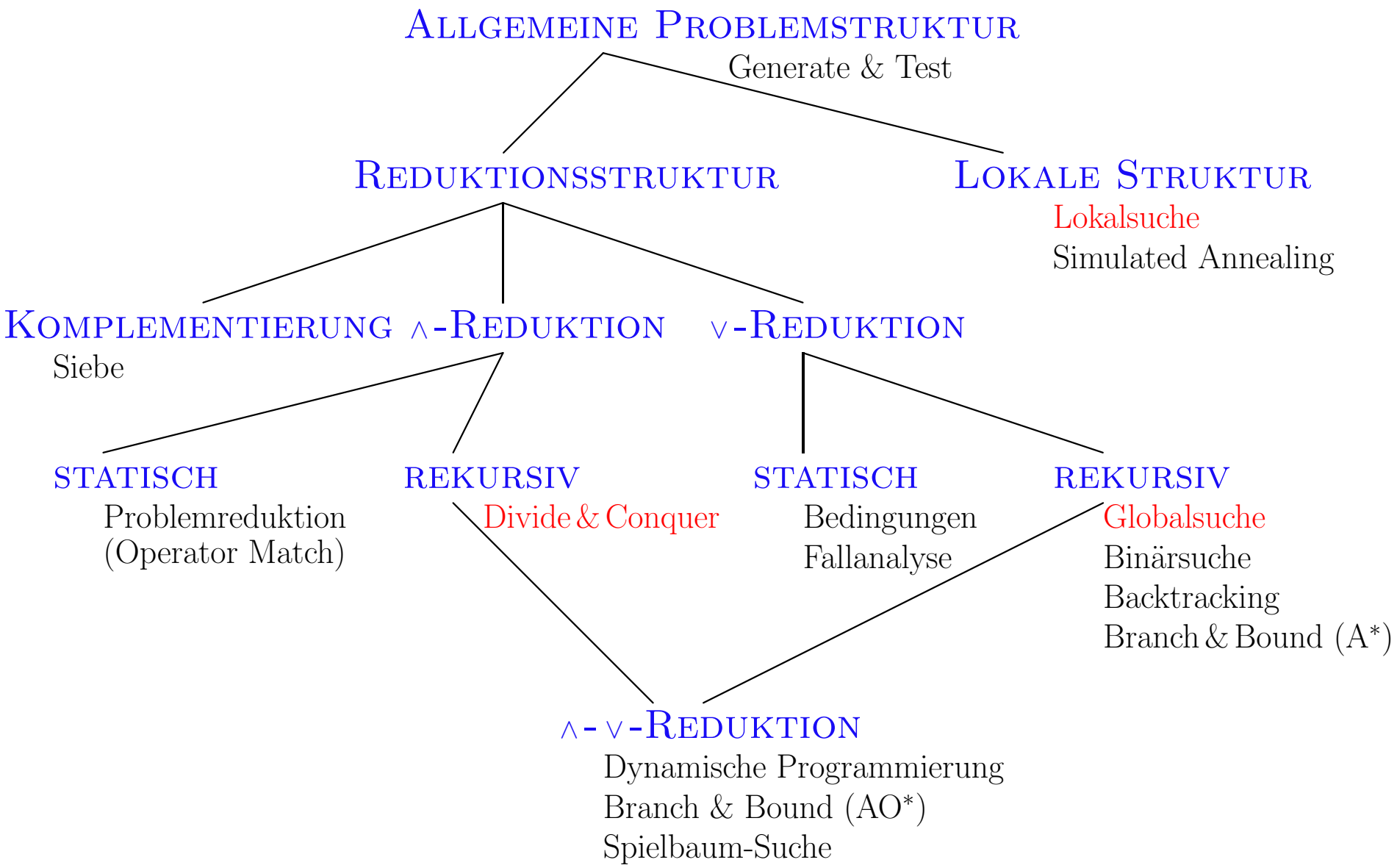
Programmierwissen: algorithmische Strukturen



Programmierwissen: algorithmische Strukturen



Programmierwissen: algorithmische Strukturen



Programmierwissen für Costas Arrays Problem

- **Spezifiziere Lösungsalgorithmus**

  FUNCTION costas(n:$\mathbb{Z}$):Set(Seq($\mathbb{Z}$)) WHERE n$\geq$1

  RETURNS $\{\pi \mid \mathtt{perm}(\pi,\{1..n\}) \wedge \forall j \in \mathtt{dom}(\pi).\mathtt{nodups}(\mathtt{dtrow}(\pi,j))\}$

Programmierwissen für Costas Arrays Problem

- **Spezifiziere Lösungsalgorithmus**

  `FUNCTION costas(n:`$\mathbb{Z}$`):Set(Seq(`$\mathbb{Z}$`)) WHERE n`$\geq$`1`

  `RETURNS {`π `| perm(`π`,{1..n}) `$\wedge$ $\forall$`j `$\in$`dom(`π`).nodups(dtrow(`π`,j))}`

  Welche Algorithmenstruktur paßt?

Programmierwissen für Costas Arrays Problem

- **Spezifiziere Lösungsalgorithmus**

  `FUNCTION costas(n:ℤ):Set(Seq(ℤ)) WHERE n≥1`

  `RETURNS {π | perm(π,{1..n}) ∧ ∀j∈dom(π).nodups(dtrow(π,j))}`

  Welche Algorithmenstruktur paßt?

- **Aufzählung und Testen ist exponentiell**
  - Wie Lösungskandidaten analysieren ohne sie aufzuzählen?

Programmierwissen für Costas Arrays Problem

- **Spezifiziere Lösungsalgorithmus**

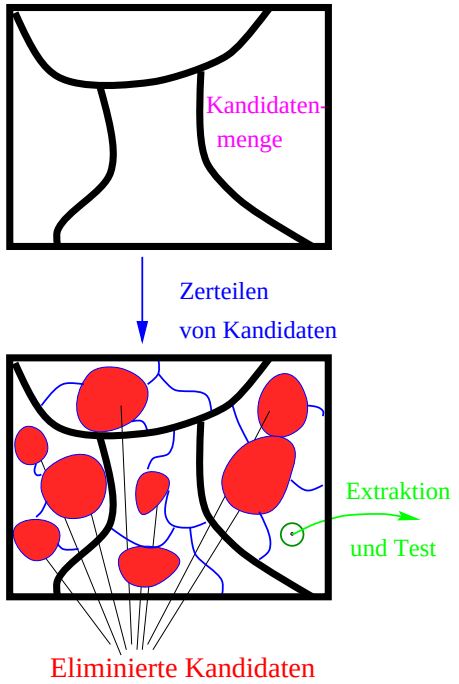
  `FUNCTION costas(n:ℤ):Set(Seq(ℤ)) WHERE n≥1`

  `RETURNS {π | perm(π,{1..n}) ∧ ∀j∈dom(π).nodups(dtrow(π,j))}`

  Welche Algorithmenstruktur paßt?

- **Aufzählung und Testen ist exponentiell**
  - Wie Lösungskandidaten analysieren ohne sie aufzuzählen?

- **Lösung benötigt Globalsuche**
  - Codierung von Kandidatenmengen
  - Wiederholtes Aufteilen und und Filtern
    auf Basis von Repräsentanten
  - Extraktion konkreter Lösungen aus Repräsentanten



Programmierwissen für Costas Arrays Problem

- **Spezifiziere Lösungsalgorithmus**

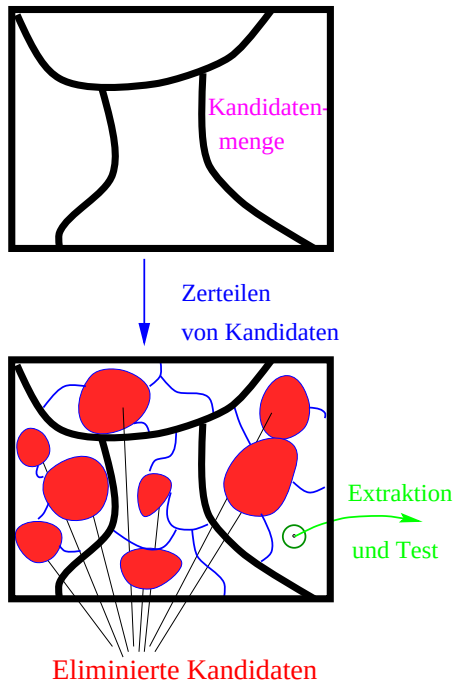
  `FUNCTION costas(n:ℤ):Set(Seq(ℤ)) WHERE n≥1`

  `RETURNS {π | perm(π,{1..n}) ∧ ∀j∈dom(π).nodups(dtrow(π,j))}`

  Welche Algorithmenstruktur paßt?

- **Aufzählung und Testen ist exponentiell**
  - Wie Lösungskandidaten analysieren ohne sie aufzuzählen?

- **Lösung benötigt Globalsuche**
  - Codierung von Kandidatenmengen
  - Wiederholtes Aufteilen und und Filtern
    auf Basis von Repräsentanten
  - Extraktion konkreter Lösungen aus Repräsentanten



Wie erzeugt man solche Algorithmen systematisch?

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

– Grundstruktur von Globalsuchalgorithmen 1a

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. **Wissen über Algorithmen**
   - Grundstruktur von Globalsuchalgorithmen 1a
   - Bedingungen für Korrektheit 1b
   - Schematische Globalsuchalgorithmen für typische Domänen 1c
2. **Wissen über Syntheseseverfahren**

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über Syntheseseverfahren

- Auswahl und Anpassung eines Algorithmenschemas 2a

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über Syntheseseverfahren

- Auswahl und Anpassung eines Algorithmenschemas 2a
- Verschärfte Filterung von Kandidaten 2b

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. **Wissen über Algorithmen**
   - Grundstruktur von Globalsuchalgorithmen 1a
   - Bedingungen für Korrektheit 1b
   - Schematische Globalsuchalgorithmen für typische Domänen 1c
2. **Wissen über Syntheseseverfahren**
   - Auswahl und Anpassung eines Algorithmenschemas 2a
   - Verschärfte Filterung von Kandidaten 2b
3. **Wissen über Optimierungstechniken**

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über Syntheseseverfahren

- Auswahl und Anpassung eines Algorithmenschemas 2a
- Verschärfte Filterung von Kandidaten 2b

3. Wissen über Optimierungstechniken

- Vereinfachung des schematisch erzeugten Algorithmus 3a

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über Syntheseseverfahren

- Auswahl und Anpassung eines Algorithmenschemas 2a
- Verschärfte Filterung von Kandidaten 2b

3. Wissen über Optimierungstechniken

- Vereinfachung des schematisch erzeugten Algorithmus 3a
- Partielle Auswertung: spart Neuberechnung konstanter Ausdrücke 3b

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über Syntheseseverfahren

- Auswahl und Anpassung eines Algorithmenschemas 2a
- Verschärfte Filterung von Kandidaten 2b

3. Wissen über Optimierungstechniken

- Vereinfachung des schematisch erzeugten Algorithmus 3a
- Partielle Auswertung: spart Neuberechnung konstanter Ausdrücke 3b
- Endliche Differenzierung: spart Neuberechnung inkrementeller Änderungen 3c

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über **Algorithmen**

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über **Syntheseseverfahren**

- Auswahl und Anpassung eines Algorithmenschemas 2a
- Verschärfte Filterung von Kandidaten 2b

3. Wissen über **Optimierungstechniken**

- Vereinfachung des schematisch erzeugten Algorithmus 3a
- Partielle Auswertung: spart Neuberechnung konstanter Ausdrücke 3b
- Endliche Differenzierung: spart Neuberechnung inkrementeller Änderungen 3c
- Datentypverfeinerung: optimale Implementierung abstrakter Datentypen 3d

Programmierwissen für Erzeugung von Globalsuchalgorithmen

1. Wissen über Algorithmen

- Grundstruktur von Globalsuchalgorithmen 1a
- Bedingungen für Korrektheit 1b
- Schematische Globalsuchalgorithmen für typische Domänen 1c

2. Wissen über Syntheseseverfahren

- Auswahl und Anpassung eines Algorithmenschemas 2a
- Verschärfte Filterung von Kandidaten 2b

3. Wissen über Optimierungstechniken

- Vereinfachung des schematisch erzeugten Algorithmus 3a
- Partielle Auswertung: spart Neuberechnung konstanter Ausdrücke 3b
- Endliche Differenzierung: spart Neuberechnung inkrementeller Änderungen 3c
- Datentypverfeinerung: optimale Implementierung abstrakter Datentypen 3d

Wissen muß formalisiert werden

Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

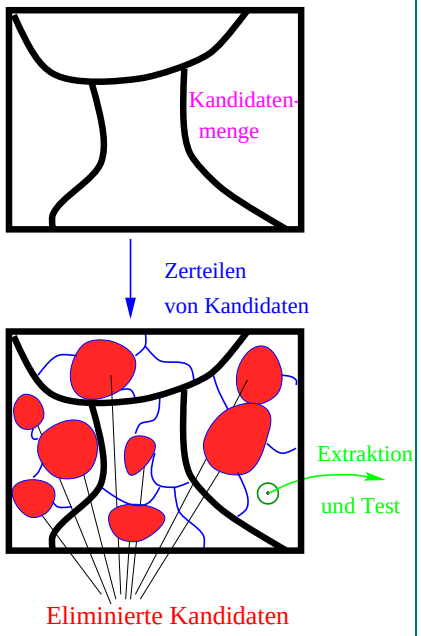
$$\texttt{in aux(x,s}_0\texttt{(x))}$$

Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,}\texttt{s}_0\texttt{(x))}$$

– Deskriptor s für Kandidatenmengen



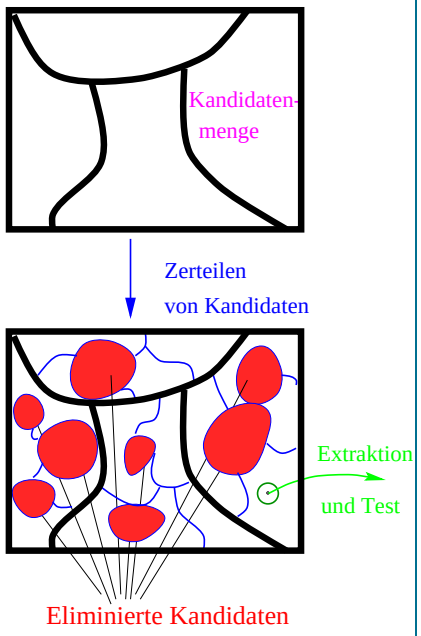
Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,}\texttt{s}_0\texttt{(x))}$$

– Deskriptor `s` für Kandidatenmengen

– Initialdeskriptor $\texttt{s}_0\texttt{(x)}$

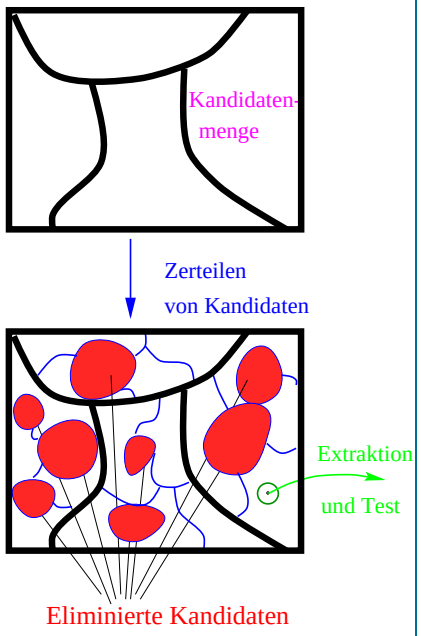


Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

- Deskriptor `s` für Kandidatenmengen
- Initialdeskriptor $\texttt{s}_0\texttt{(x)}$
- Rekursive Aufteilung `split(x,s)` von Kandidatenmengen

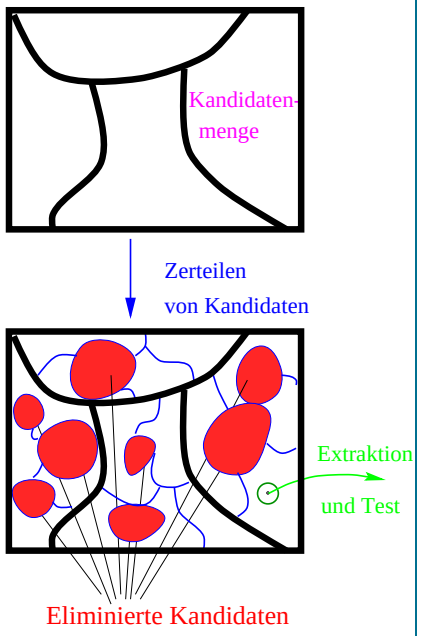


Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

- Deskriptor `s` für Kandidatenmengen
- Initialdeskriptor $\texttt{s}_0\texttt{(x)}$
- Rekursive Aufteilung `split(x,s)` von Kandidatenmengen
- Elimination unnötiger Deskriptoren durch Filter $\Phi\texttt{(x,s)}$

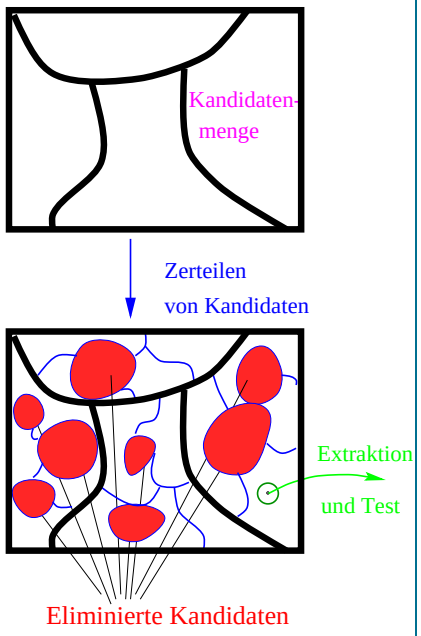


Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

- Deskriptor `s` für Kandidatenmengen
- Initialdeskriptor $\texttt{s}_0\texttt{(x)}$
- Rekursive Aufteilung `split(x,s)` von Kandidatenmengen
- Elimination unnötiger Deskriptoren durch Filter $\Phi\texttt{(x,s)}$
- Extraktion `ext(s)` von Lösungskandidaten `z` aus Deskriptoren

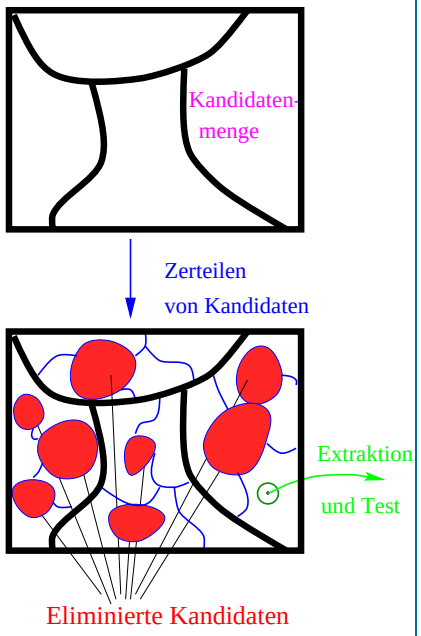


Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

- Deskriptor `s` für Kandidatenmengen
- Initialdeskriptor $\texttt{s}_0\texttt{(x)}$
- Rekursive Aufteilung `split(x,s)` von Kandidatenmengen
- Elimination unnötiger Deskriptoren durch Filter $\Phi\texttt{(x,s)}$
- Extraktion `ext(s)` von Lösungskandidaten `z` aus Deskriptoren
- Selektion mit Ausgabebedingung `O(x,z)`

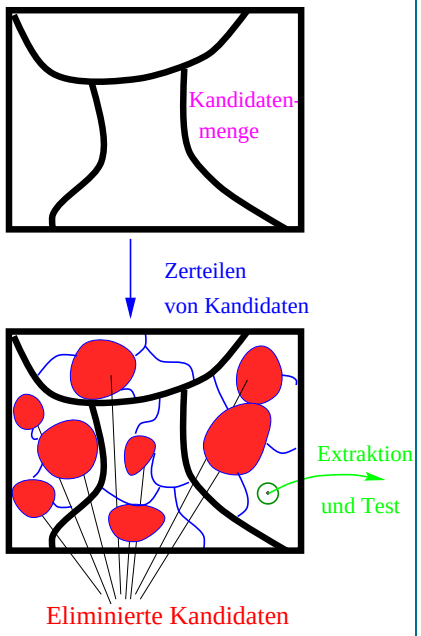


Globalsuchalgorithmen – Formale Struktur

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

- Deskriptor `s` für Kandidatenmengen
- Initialdeskriptor $\texttt{s}_0\texttt{(x)}$
- Rekursive Aufteilung `split(x,s)` von Kandidatenmengen
- Elimination unnötiger Deskriptoren durch Filter $\Phi\texttt{(x,s)}$
- Extraktion `ext(s)` von Lösungskandidaten `z` aus Deskriptoren
- Selektion mit Ausgabebedingung `O(x,z)`



Ideal für Parallelverarbeitung

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**

2. **Splitting erhält sinnvolle Deskriptoren**

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**
2. **Splitting erhält sinnvolle Deskriptoren**
3. **Initialdeskriptor enthält alle Lösungen**

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**
2. **Splitting erhält sinnvolle Deskriptoren**
3. **Initialdeskriptor enthält alle Lösungen**
4. **Alle Lösungen in endlich vielen Schritten extrahierbar**

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**
2. **Splitting erhält sinnvolle Deskriptoren**
3. **Initialdeskriptor enthält alle Lösungen**
4. **Alle Lösungen in endlich vielen Schritten extrahierbar**
5. **Splitting ist wohlfundiert (nur endlich oft durchführbar)**

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,s}_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die `O(x,z)` gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**
2. **Splitting erhält sinnvolle Deskriptoren**
3. **Initialdeskriptor enthält alle Lösungen**
4. **Alle Lösungen in endlich vielen Schritten extrahierbar**
5. **Splitting ist wohlfundiert (nur endlich oft durchführbar)**
6. **Filter ist notwendig (keine Lösung wird eliminiert)**

Globalsuche – Bedingungen für Korrektheit

$$\texttt{let rec aux(x,s)} = \{\, \texttt{z} \mid \texttt{z} \in \texttt{ext(s)} \wedge \texttt{O(x,z)} \,\} \cup \bigcup \{\, \texttt{aux(x,t)} \mid \texttt{t} \in \texttt{split(x,s)} \wedge \Phi\texttt{(x,t)} \,\}$$

$$\texttt{in aux(x,}s_0\texttt{(x))}$$

berechnet die Menge aller Lösungen z für die O(x,z) gilt, wenn

1. **Initialdeskriptor ist sinnvoll für zulässige Eingaben**
   - $\texttt{I(x)} \Rightarrow \texttt{J(x,}s_0\texttt{(x))}$
2. **Splitting erhält sinnvolle Deskriptoren**
   - $\texttt{I(x)} \wedge \texttt{J(x,s)} \Rightarrow \forall \texttt{t} \in \texttt{split(x,s)}\,.\,\texttt{J(x,t)}$
3. **Initialdeskriptor enthält alle Lösungen**
   - $\texttt{I(x)} \wedge \texttt{O(x,z)} \Rightarrow \texttt{sat(z,}s_0\texttt{(x))}$
4. **Alle Lösungen in endlich vielen Schritten extrahierbar**
   - $\texttt{I(x)} \wedge \texttt{J(x,s)} \wedge \texttt{O(x,z)} \Rightarrow \texttt{sat(z,s)} \Leftrightarrow \exists \texttt{k}:\mathbb{N}.\exists \texttt{t} \in \texttt{split}^{\texttt{k}}\texttt{(x,s)}\,.\,\texttt{z} \in \texttt{ext(t)}$
5. **Splitting ist wohlfundiert (nur endlich oft durchführbar)**
   - $\texttt{I(x)} \wedge \texttt{J(x,s)} \Rightarrow \exists \texttt{k}:\mathbb{N}.\ \texttt{split}^{\texttt{k}}\texttt{(x,s)} = \emptyset$
6. **Filter ist notwendig (keine Lösung wird eliminiert)**
   - $\texttt{I(x)} \wedge \texttt{J(x,s)} \Rightarrow \Phi\texttt{(x,s)} \Leftarrow \exists \texttt{z}:\texttt{R}.\,\texttt{sat(z,s)} \wedge \texttt{O(x,z)}$

Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

Suche $\hat{=}$ Aufzählung der Präfixe einer Liste

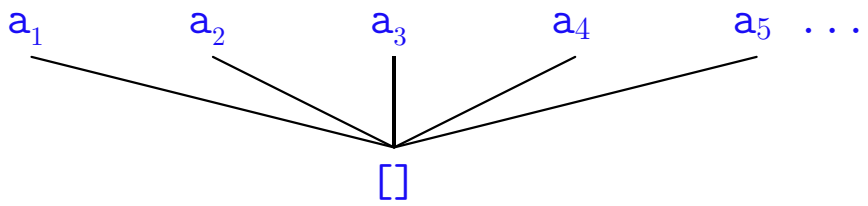
Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

Suche $\hat{=}$ Aufzählung der Präfixe einer Liste

[]

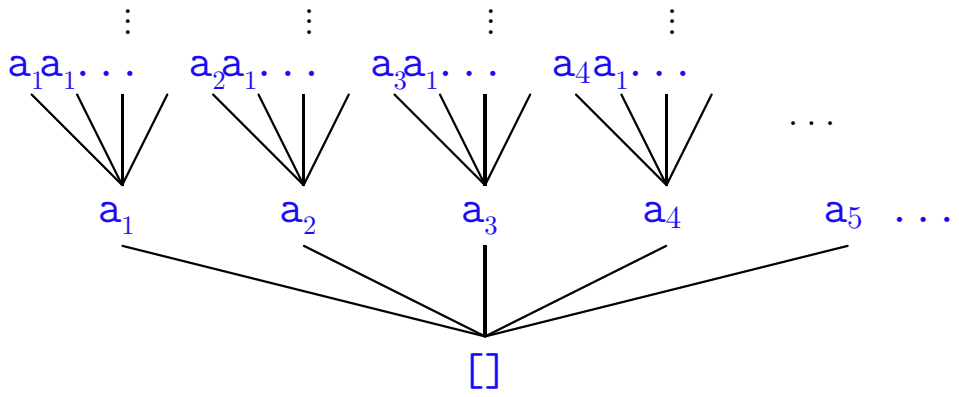
Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



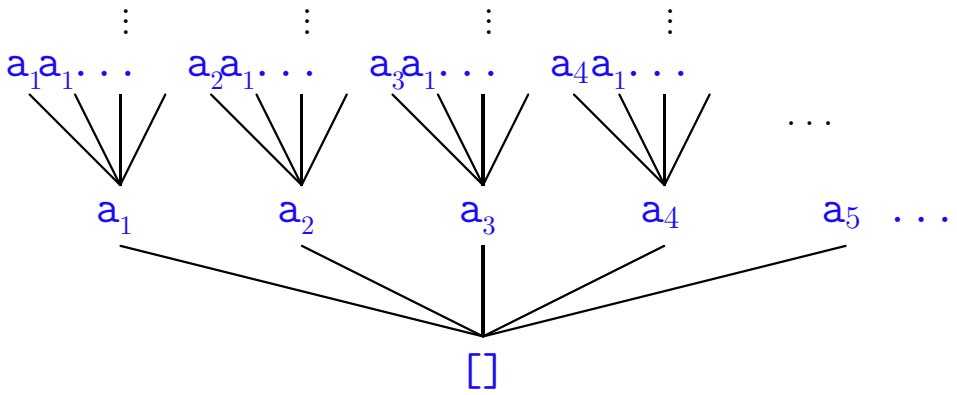
Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

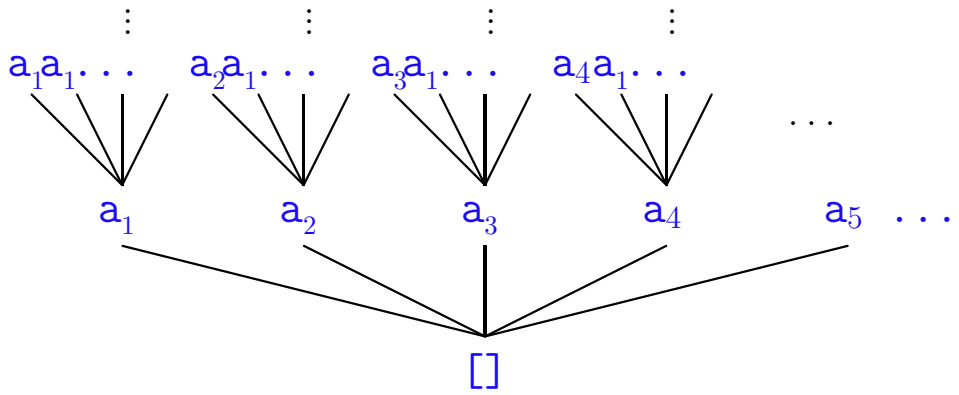
Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



- **Deskriptoren:** gemeinsamer Präfix s sat(z,s) $\equiv$ s $\sqsubseteq$ z

Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

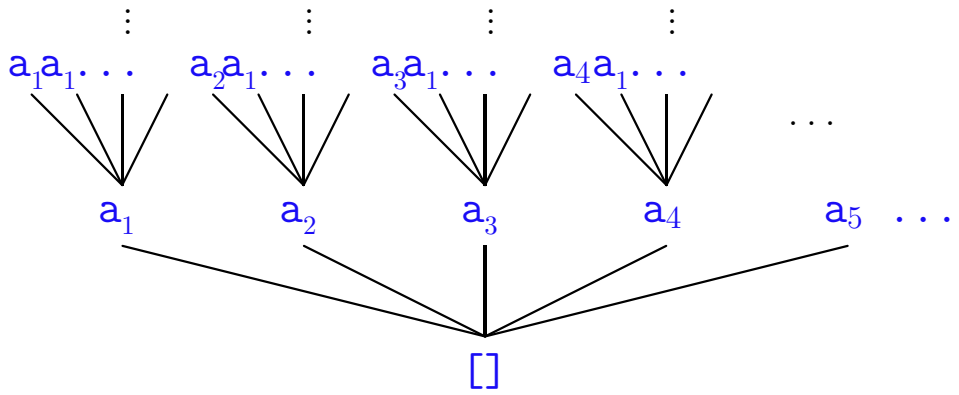
Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



- **Deskriptoren:** gemeinsamer Präfix s — sat(z,s) $\equiv$ s $\sqsubseteq$ z
- **Sinnvoll:** nur Elemente aus X — J(X,s) $\equiv$ s $\subseteq$ X

Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

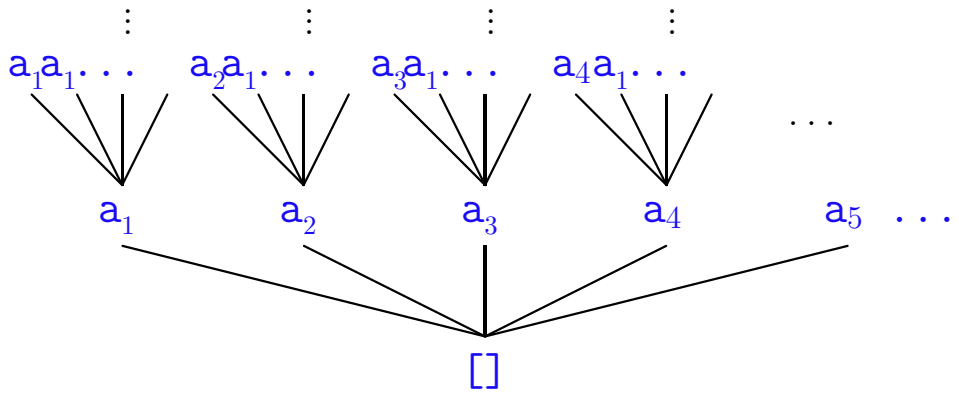
Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



- **Deskriptoren:** gemeinsamer Präfix s — $\texttt{sat(z,s)} \equiv \texttt{s} \sqsubseteq \texttt{z}$
- **Sinnvoll:** nur Elemente aus X — $\texttt{J(X,s)} \equiv \texttt{s} \subseteq \texttt{X}$
- **Initialdeskriptor:** leerer Präfix — $\texttt{s}_0\texttt{(X)} \equiv \texttt{[]}$

Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

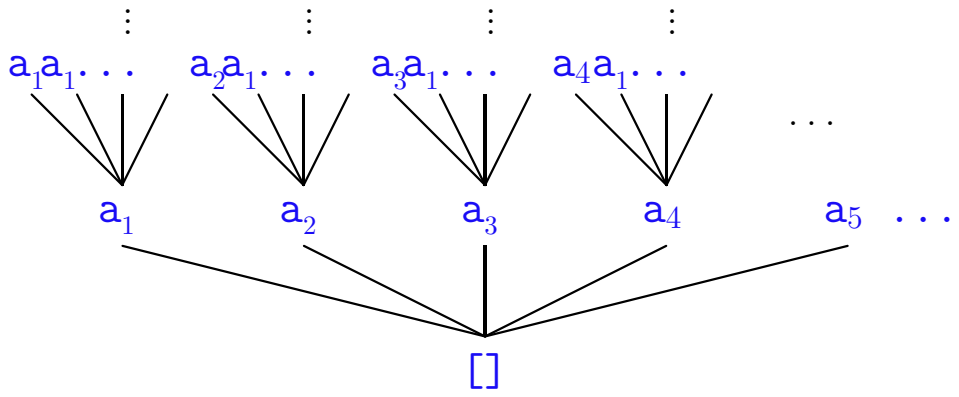
Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



- **Deskriptoren:** gemeinsamer Präfix s — $\texttt{sat(z,s)} \equiv \texttt{s} \sqsubseteq \texttt{z}$
- **Sinnvoll:** nur Elemente aus X — $\texttt{J(X,s)} \equiv \texttt{s} \subseteq \texttt{X}$
- **Initialdeskriptor:** leerer Präfix — $\texttt{s}_0\texttt{(X)} \equiv \texttt{[]}$
- **Splitting:** Verlängern des Präfix — $\texttt{split(X,s)} \equiv \{\texttt{s}\cdot\texttt{i} \mid \texttt{i} \in \texttt{X}\}$

Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

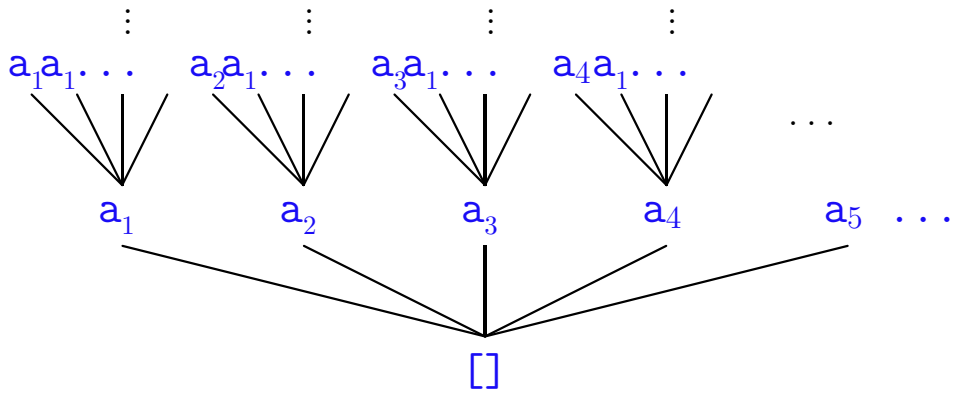
Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



- **Deskriptoren:** gemeinsamer Präfix s — $\texttt{sat(z,s)} \equiv \texttt{s} \sqsubseteq \texttt{z}$
- **Sinnvoll:** nur Elemente aus X — $\texttt{J(X,s)} \equiv \texttt{s} \subseteq \texttt{X}$
- **Initialdeskriptor:** leerer Präfix — $\texttt{s}_0\texttt{(X)} \equiv \texttt{[]}$
- **Splitting:** Verlängern des Präfix — $\texttt{split(X,s)} \equiv \{\texttt{s}\cdot\texttt{i} \mid \texttt{i} \in \texttt{X}\}$
- **Extraktion:** Gesamter Präfix — $\texttt{ext(s)} \equiv \{\texttt{s}\}$

Schematischer Globalsuchalgorithmus für Listen über endlicher Menge X

Suche $\hat{=}$ Aufzählung der Präfixe einer Liste



- **Deskriptoren:** gemeinsamer Präfix s — sat(z,s) ≡ s ⊑ z
- **Sinnvoll:** nur Elemente aus X — J(X,s) ≡ s ⊆ X
- **Initialdeskriptor:** leerer Präfix — s_0(X) ≡ []
- **Splitting:** Verlängern des Präfix — split(X,s) ≡ {s·i | i ∈ X}
- **Extraktion:** Gesamter Präfix — ext(s) ≡ {s}
- **Filter:** Test auf Duplikate — Φ(X,s) ≡ nodups(s)
  - Macht Splitting wohlfundiert (andere Filter möglich)

2a Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(`π`,{1..n}) ∧ ∀j<n. nodups(dt-row(`π`,j))`

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(`π`,{1..n}) ∧ ∀j<n. nodups(dt-row(`π`,j))`

- **π ist Liste über der Menge `{1..n}`**
  - Ersetze Vorkommen von `X` durch `{1..n}`

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(`π`,{1..n}) ∧ ∀j<n. nodups(dt-row(`π`,j))`

- **π ist Liste über der Menge `{1..n}`**
  - Ersetze Vorkommen von `X` durch `{1..n}`
  - Transformierter Filter ist "notwendig" für Ausgabebedingung der Costas Arrays ✓

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

2a Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(`π`,{1..n}) ∧ ∀j<n. nodups(dt-row(`π`,j))`

- **π ist Liste über der Menge {1..n}**
  - Ersetze Vorkommen von `X` durch `{1..n}`
  - Transformierter Filter ist "notwendig" für Ausgabebedingung der Costas Arrays ✓
- **Verschärfe den Filter Φ** (2b)

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

2a Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(`π`,{1..n})` $\wedge$ $\forall$`j<n.nodups(dt-row(`π`,j))`

- **π ist Liste über der Menge {1..n}**
  - Ersetze Vorkommen von `X` durch `{1..n}`
  - Transformierter Filter ist "notwendig" für Ausgabebedingung der Costas Arrays ✓

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

- **Verschärfe den Filter Φ** (2b)
  - Für alle Präfixe `s` von π gilt $\forall$`j<|s|.nodups(dt-row(s,j))`

Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(`π`,{1..n})` $\wedge$ $\forall$`j<n.nodups(dt-row(`π`,j))`

- **π ist Liste über der Menge `{1..n}`**
  - Ersetze Vorkommen von `X` durch `{1..n}`
  - Transformierter Filter ist "notwendig" für Ausgabebedingung der Costas Arrays ✓

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

- **Verschärfe den Filter Φ** (2b)
  - Für alle Präfixe `s` von π gilt $\forall$`j<|s|.nodups(dt-row(s,j))`
  - Wähle `Φ'(n,s) := nodups(s)` $\wedge$ $\forall$`j<|s|.nodups(dt-row(s,j))`

Spezialisierung des Schemas für Costas Arrays

Suche alle π mit `perm(π,{1..n}) ∧ ∀j<n. nodups(dt-row(π,j))`

- **π ist Liste über der Menge {1..n}**
  - Ersetze Vorkommen von `X` durch `{1..n}`
  - Transformierter Filter ist "notwendig" für Ausgabebedingung der Costas Arrays ✓

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

- **Verschärfe den Filter Φ** (2b)
  - Für alle Präfixe `s` von π gilt `∀j<|s|. nodups(dt-row(s,j))`
  - Wähle `Φ'(n,s) := nodups(s) ∧ ∀j<|s|. nodups(dt-row(s,j))`

- **Instantiiere den Standard-Globalsuchalgorithmus**

```
let costas(n) =
   let rec aux(n,s)
   = { π | π∈{s} ∧ perm(π,{1..n}) ∧ ∀j<n. nodups(dt-row(π,j)) }
     ∪ ⋃{ aux(x,t) | t∈{ s·i | i∈{1..n} }
                   ∧ nodups(t) ∧ ∀j<|t|. nodups(dt-row(t,j)) }

   in aux(n,[])
```

Simplifikationen und Partielle Auswertung

- **Logische Vereinfachung von Teilausdrücken**

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke
  - Kontextunabhängig: Anwendung gerichteter Gleichungen

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke
  - Kontextunabhängig: Anwendung gerichteter Gleichungen
  - Kontextabhängig: Anwendung gerichteter Gleichungen mit Vorbedingung

    Kontext über Syntaxbaum bestimmt (Programmcode und Vorbedingung)

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke
  - Kontextunabhängig: Anwendung gerichteter Gleichungen
  - Kontextabhängig: Anwendung gerichteter Gleichungen mit Vorbedingung
    Kontext über Syntaxbaum bestimmt (Programmcode und Vorbedingung)
  - Benutzer wählt konkreten Teilausdruck und Art der Vereinfachung

Simplifikationen und Partielle Auswertung

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke
  - Kontextunabhängig: Anwendung gerichteter Gleichungen
  - Kontextabhängig: Anwendung gerichteter Gleichungen mit Vorbedingung
    Kontext über Syntaxbaum bestimmt (Programmcode und Vorbedingung)
  - Benutzer wählt konkreten Teilausdruck und Art der Vereinfachung
- **Auswertung von konstanten Teilausdrücken**

Simplifikationen und Partielle Auswertung

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke
  - Kontextunabhängig: Anwendung gerichteter Gleichungen
  - Kontextabhängig: Anwendung gerichteter Gleichungen mit Vorbedingung
    Kontext über Syntaxbaum bestimmt (Programmcode und Vorbedingung)
  - Benutzer wählt konkreten Teilausdruck und Art der Vereinfachung
- **Auswertung von konstanten Teilausdrücken**
  - Auffalten von Definitionen + Simplifikation

Simplifikationen und Partielle Auswertung

- **Logische Vereinfachung von Teilausdrücken**
  - Ziel: Elimination redundanter Teilausdrücke
  - Kontextunabhängig: Anwendung gerichteter Gleichungen
  - Kontextabhängig: Anwendung gerichteter Gleichungen mit Vorbedingung
    Kontext über Syntaxbaum bestimmt (Programmcode und Vorbedingung)
  - Benutzer wählt konkreten Teilausdruck und Art der Vereinfachung
- **Auswertung von konstanten Teilausdrücken**
  - Auffalten von Definitionen + Simplifikation
  - Benutzer wählt auszuwertenden Teilausdruck

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = { π | π ∈ {s} ∧ perm(π,{1..n}) ∧ ∀j<n. nodups(dt-row(π,j)) }
     ∪ ⋃{ aux(x,t) | t ∈ { s·i | i ∈ {1..n} } ∧ nodups(t)
                   ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = { π | π ∈ {s} ∧ perm(π,{1..n}) ∧ ∀j<n. nodups(dt-row(π,j)) }
     ∪ ⋃{ aux(x,t) | t ∈ { s·i | i ∈ {1..n} } ∧ nodups(t)
                     ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = { π | π ∈ {s} ∧ perm(π,{1..n}) ∧ ∀j<n. nodups(dt-row(π,j)) }
     ∪ ⋃{ aux(x,t) | t ∈ { s·i | i ∈ {1..n} } ∧ nodups(t)
                   ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Domänenwissen: $\{\pi \mid \pi \in \{s\} \wedge P(\pi)\} \quad \equiv \quad \text{if } P(s) \text{ then } \{s\} \text{ else } \emptyset$

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if perm(s,{1..n}) ∧ ∀j<n. nodups(dt-row(s,j)) then {s} else ∅
     ∪ ⋃{ aux(x,t) | t∈{ s·i | i∈{1..n}} ∧ nodups(t)
                   ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if perm(s,{1..n}) ∧ ∀j<n. nodups(dt-row(s,j)) then {s} else ∅
     ∪ ⋃{ aux(x,t) | t∈{ s·i|i∈{1..n}} ∧ nodups(t)
                   ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if perm(s,{1..n}) ∧ ∀j<n. nodups(dt-row(s,j)) then {s} else ∅
     ∪ ⋃{ aux(x,t) | t∈{ s·i|i∈{1..n}} ∧ nodups(t)
                   ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Domänenwissen: `perm(s,{1..n})` $\Rightarrow$ `|s|=n`

Kontext der Formel: `perm(s,{1..n})`

Kontext der Loopinvariante: `∀j<|s|. nodups(dt-row(s,j))`

Formel kann komplett entfallen

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if perm(s,{1..n}) then {s} else ∅
      ∪ ⋃{ aux(x,t) | t∈{ s·i|i∈{1..n}} ∧ nodups(t)
                    ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if perm(s,{1..n}) then {s} else ∅
      ∪ ⋃{ aux(x,t) | t∈{ s·i | i∈{1..n}} ∧ nodups(t)
                    ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if perm(s,{1..n}) then {s} else ∅
     ∪ ⋃{ aux(x,t) | t∈{ s·i|i∈{1..n}} ∧ nodups(t)
                   ∧ ∀j<|t|.nodups(dt-row(t,j)) }
   in aux(n,[])
```

Domänenwissen:

$\texttt{perm(s,\{1..n\})} \equiv \texttt{s}\subseteq\texttt{\{1..n\}} \wedge \texttt{\{1..n\}}\subseteq\texttt{s} \wedge \texttt{nodups(s)}$

$\texttt{\{1..n\}}\subseteq\texttt{s} \equiv \texttt{\{1..n\}}\setminus\texttt{s}=\emptyset$

Kontext der Loopinvariante: $\texttt{nodups(s)}$

Rahmenbedingung für Deskriptoren $\texttt{J(\{1..n\},s)}$**:** $\texttt{s}\subseteq\texttt{\{1..n\}}$

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,t) | t∈{ s·i|i∈{1..n}} ∧ nodups(t)
                    ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,t) | t∈{ s·i | i∈{1..n}} ∧ nodups(t)
                    ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,t) | t∈{ s·i | i∈{1..n}} ∧ nodups(t)
                    ∧ ∀j<|t|. nodups(dt-row(t,j)) }
   in aux(n,[])
```

Domänenwissen:

$\{\, f(x,t) \mid t \in \{g(s,i) \mid i \in S\} \wedge h(t)\} = \{\, f(x,g(s,i)) \mid i \in S \wedge h(g(s,i))\}$

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n} ∧ nodups(s·i)
                        ∧ ∀j<|s·i|. nodups(dt-row(s·i,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n} ∧ nodups(s·i)
                        ∧ ∀j<|s·i|. nodups(dt-row(s·i,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
     ∪ ⋃{ aux(x,s·i) | i∈{1..n} ∧ nodups(s·i)
                       ∧ ∀j<|s·i|. nodups(dt-row(s·i,j)) }
   in aux(n,[])
```

Domänenwissen: $i \in \{1..n\} \wedge nodups(s \cdot i) \equiv i \in \{1..n\} \setminus s$

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n}\s
                        ∧ ∀j<|s·i|. nodups(dt-row(s·i,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n}\s
                        ∧ ∀j<|s·i|. nodups(dt-row(s·i,j)) }
   in aux(n,[])
```

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n}\s
                        ∧ ∀j<|s·i|.nodups(dt-row(s·i,j)) }
   in aux(n,[])
```

Domänenwissen:

```
dt-row(s·i,j) = dt-row(s,j)·(s[|s·i|-j]-i)
nodups(t·k) ≡ nodups(t) ∧ k∉t
∀j<|s·i|.P(j) ≡ ∀j<|s|.P(j) ∧ P(|s|)
dt-row(s·i,|s|) = [s[|s·i|-|s|]-i]
nodups([s[|s·i|-|s|]-i]) ≡ true
```

2	4	1	6	5	3
-2	3	-5	1	2	
1	-2	-4	3		
-4	-1	-2			
-3	1				
-1					

Kontext der Loopinvariante: `∀j<|s|.nodups(dt-row(s,j))`

Wissensbasierte Optimierung – Vereinfachung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n}\s
                        ∧ ∀j<|s|. (s[|s·i|-j]-i)∉dt-row(s,j)}
   in aux(n,[])
```

Endliche Differenzierung

- **Inkrementelle Berechnung wiederkehrender Teilausdrücke**
  - Ersetze Teilausdrücke in Schleifen durch Variablen
  - Bestimme differentielle Veränderungen

Endliche Differenzierung

- **Inkrementelle Berechnung wiederkehrender Teilausdrücke**
  - Ersetze Teilausdrücke in Schleifen durch Variablen
  - Bestimme differentielle Veränderungen
- **Formal: Abstraktion über Teilausdruck `E(x)` in `f(x)`**
  - Bestimme neues `f'`, so daß `f(x)` = `f'(x,E(x))`
  - Ersetze Aufrufe der Art `f(g(x))` durch `f'(g(x)x,E(g(x)))`

Endliche Differenzierung

- **Inkrementelle Berechnung wiederkehrender Teilausdrücke**
  - Ersetze Teilausdrücke in Schleifen durch Variablen
  - Bestimme differentielle Veränderungen
- **Formal: Abstraktion über Teilausdruck `E(x)` in `f(x)`**
  - Bestimme neues `f'`, so daß `f(x)` = `f'(x,E(x))`
  - Ersetze Aufrufe der Art `f(g(x))` durch `f'(g(x)x,E(g(x)))`
  - Vereinfache Ausdrücke der Form `E(g(x))` zur Form `g'(E(x))`
  - Ersetze alle Vorkommen von `E(x)` durch neue Variable

Endliche Differenzierung

- **Inkrementelle Berechnung wiederkehrender Teilausdrücke**
  - Ersetze Teilausdrücke in Schleifen durch Variablen
  - Bestimme differentielle Veränderungen
- **Formal: Abstraktion über Teilausdruck `E(x)` in `f(x)`**
  - Bestimme neues `f'`, so daß `f(x)` = `f'(x,E(x))`
  - Ersetze Aufrufe der Art `f(g(x))` durch `f'(g(x)x,E(g(x)))`
  - Vereinfache Ausdrücke der Form `E(g(x))` zur Form `g'(E(x))`
  - Ersetze alle Vorkommen von `E(x)` durch neue Variable

  Benutzer wählt konkreten Teilausdruck `E(x)`

Endliche Differenzierung

- **Inkrementelle Berechnung wiederkehrender Teilausdrücke**
  - Ersetze Teilausdrücke in Schleifen durch Variablen
  - Bestimme differentielle Veränderungen
- **Formal: Abstraktion über Teilausdruck `E(x)` in `f(x)`**
  - Bestimme neues `f'`, so daß `f(x)` = `f'(x,E(x))`
  - Ersetze Aufrufe der Art `f(g(x))` durch `f'(g(x)x,E(g(x)))`
  - Vereinfache Ausdrücke der Form `E(g(x))` zur Form `g'(E(x))`
  - Ersetze alle Vorkommen von `E(x)` durch neue Variable

  Benutzer wählt konkreten Teilausdruck `E(x)`

Effizienter als ständige Neuberechnung

Optimierung – Endliche Differenzierung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n}\s
                        ∧ ∀j<|s|. (s[|s·i|-j]-i)∉dt-row(s,j)}
   in aux(n,[])
```

Optimierung – Endliche Differenzierung

```
let costas(n) =
   let rec aux(n,s)
   = if {1..n}\s=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈{1..n}\s
                        ∧ ∀j<|s|. (s[|s·i|-j]-i)∉dt-row(s,j)}
   in aux(n,[])
```

Ersetze ineffiziente Neuberechnung durch neue Variablen:

$\{1..n\}\backslash s \mapsto$ pool

$|s \cdot i| \mapsto$ ssize

Integriere Variablen in Definition von aux

Optimierung – Endliche Differenzierung

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈pool
                        ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[])
```

Optimierung – Endliche Differenzierung

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i) | i∈pool
                        ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Initiierung der Variablen im ersten Aufruf

Optimierung – Endliche Differenzierung

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Inkrementelle Veränderung im rekursiven Aufruf

Optimierung – Fallanalyse

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅ then {s} else ∅
      ∪ ⋃{ aux(x,s·i,pool\{i},ssize+1) | i ∈pool
                                ∧ ∀j<|s|. (s[ssize-j]-i) ∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Optimierung – Fallanalyse

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅ then {s} else ∅
     ∪ ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                              ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Domänenwissen:

$$(\texttt{if pool=}\emptyset\texttt{ then \{s\} else }\emptyset) \cup S = \texttt{if pool=}\emptyset\texttt{ then \{s\}} \cup S \texttt{ else } S$$

Optimierung – Fallanalyse

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅
      then {s} ∪ ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
      else ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Optimierung – Fallanalyse

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅
      then {s} ∪ ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
      else ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Domänenwissen: $\bigcup\{ f(i) \mid i \in \emptyset \} = \emptyset$

Kontext der Formel: $pool = \emptyset$

Optimierung – Fallanalyse

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅
      then {s}
      else ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Datentypverfeinerung

Ersetze abstrakte Definitionen von Datentypen durch effiziente konkrete Implementierungen

Ersetze abstrakte Definitionen von Datentypen durch effiziente konkrete Implementierungen

- **Endliche Mengen**
  - Listen: Standardimplementierung
  - Bitvektor: Mengen über endlichem Domain
  - Charakteristische Funktion: (effiziente Elementrelation)

Datentypverfeinerung

Ersetze abstrakte Definitionen von Datentypen durch effiziente konkrete Implementierungen

- **Endliche Mengen**
  - Listen: Standardimplementierung
  - Bitvektor: Mengen über endlichem Domain
  - Charakteristische Funktion: (effiziente Elementrelation)
- **Folgen**
  - Verkettete Liste: Standardimplementierung
  - Umgekehrt verkettete Liste: (gut für append-Operation $\cdot$)

Datentypverfeinerung

Ersetze abstrakte Definitionen von Datentypen durch effiziente konkrete Implementierungen

- **Endliche Mengen**
  - Listen: Standardimplementierung
  - Bitvektor: Mengen über endlichem Domain
  - Charakteristische Funktion: (effiziente Elementrelation)
- **Folgen**
  - Verkettete Liste: Standardimplementierung
  - Umgekehrt verkettete Liste: (gut für append-Operation $\cdot$)
- **System stellt diverse Implementierungen bereit**

Datentypverfeinerung

Ersetze abstrakte Definitionen von Datentypen durch effiziente konkrete Implementierungen

- **Endliche Mengen**
  - Listen: Standardimplementierung
  - Bitvektor: Mengen über endlichem Domain
  - Charakteristische Funktion: (effiziente Elementrelation)
- **Folgen**
  - Verkettete Liste: Standardimplementierung
  - Umgekehrt verkettete Liste: (gut für append-Operation $\cdot$)
- **System stellt diverse Implementierungen bereit**
  - Benutzer wählt Nichtstandard-Implementierung für jede einzelne Variable

Optimierung – Datentypverfeinerung

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅
      then {s}
      else ⋃{ aux(x,s·i,pool\{i},ssize+1) | i ∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i) ∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Implementierung der Variablen

Optimierung – Datentypverfeinerung

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅
      then {s}
      else ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Implementierung der Variablen

`s`: Folge als umgekehrt verkettete Liste

Optimierung – Datentypverfeinerung

```
let costas(n) =
   let rec aux(n,s,pool,ssize)
   = if pool=∅
      then {s}
      else ⋃{ aux(x,s·i,pool\{i},ssize+1) | i∈pool
                                  ∧ ∀j<|s|. (s[ssize-j]-i)∉dt-row(s,j)}
   in aux(n,[],{1..n},1)
```

Implementierung der Variablen

`s`: Folge als umgekehrt verkettete Liste

`pool`: Menge als Bitvektor

Wissensbasierte Methoden liefern schematische Techniken zur Softwareentwicklung

- **Erzeugte Algorithmen sind korrekt und effizient**

Wissensbasierte Methoden liefern schematische Techniken zur Softwareentwicklung

- **Erzeugte Algorithmen sind korrekt und effizient**
- **Vergleichbare Techniken für andere Strukturen**
  - Divide-and Conquer
  - Lokalsuche
  - Dynamic Programming

Wissensbasierte Methoden liefern schematische Techniken zur Softwareentwicklung

- **Erzeugte Algorithmen sind korrekt und effizient**
- **Vergleichbare Techniken für andere Strukturen**
  - Divide-and Conquer
  - Lokalsuche
  - Dynamic Programming
- **Techniken sind automatisierbar**
  - Jeder Schritt basiert auf logischer Inferenz
  - Wissen steuert alle Strategien des Algorithmenentwurfs
  - Mathematische Notation übersetzbar in Programmiersprachen

Wissensbasierte Methoden liefern schematische Techniken zur Softwareentwicklung

- **Erzeugte Algorithmen sind korrekt und effizient**
- **Vergleichbare Techniken für andere Strukturen**
  - Divide-and Conquer
  - Lokalsuche
  - Dynamic Programming
- **Techniken sind automatisierbar**
  - Jeder Schritt basiert auf logischer Inferenz
  - Wissen steuert alle Strategien des Algorithmenentwurfs
  - Mathematische Notation übersetzbar in Programmiersprachen
- **Techniken sind erfolgreich**
  - KIDS erzeugt korrekte Scheduling Algorithmen in wenigen Stunden
  - Erzeugter Lisp Code 2000 mal schneller als existierende ADA Software

Weitere Erfolge wissensbasierter Programmierung

Netzwerkverifikation, -optimierung und -entwurf

- **Logisches Schließen über "echte" Kommunikationssysteme**
  - Ensemble Toolkit: Börseninfrastruktur, Flugverkehrskontrolle, ...
  - Implementierung hat über 100 OCaml Module mit mehr als 100.000 Zeilen
  - Repräsentation in der Logik des Nuprl Beweissystems

Weitere Erfolge wissensbasierter Programmierung

Netzwerkverifikation, -optimierung und -entwurf

- **Logisches Schließen über "echte" Kommunikationssysteme**
  - Ensemble Toolkit: Börseninfrastruktur, Flugverkehrskontrolle, ...
  - Implementierung hat über 100 OCaml Module mit mehr als 100.000 Zeilen
  - Repräsentation in der Logik des Nuprl Beweissystems
- **Verifikation findet subtilen Fehler im Total-Order Protokoll**

Weitere Erfolge wissensbasierter Programmierung

Netzwerkverifikation, -optimierung und -entwurf

- **Logisches Schließen über "echte" Kommunikationssysteme**
  - Ensemble Toolkit: Börseninfrastruktur, Flugverkehrskontrolle, ...
  - Implementierung hat über 100 OCaml Module mit mehr als 100.000 Zeilen
  - Repräsentation in der Logik des Nuprl Beweissystems
- **Verifikation findet subtilen Fehler im Total-Order Protokoll**
- **Optimierungstechnik verbessert Performanz um Faktor 3–10**
  - Strategien verarbeiten Wissen über Code der Protokolle und Komposition

Weitere Erfolge wissensbasierter Programmierung

Netzwerkverifikation, -optimierung und -entwurf

- **Logisches Schließen über "echte" Kommunikationssysteme**
  - Ensemble Toolkit: Börseninfrastruktur, Flugverkehrskontrolle, ...
  - Implementierung hat über 100 OCaml Module mit mehr als 100.000 Zeilen
  - Repräsentation in der Logik des Nuprl Beweissystems
- **Verifikation findet subtilen Fehler im Total-Order Protokoll**
- **Optimierungstechnik verbessert Performanz um Faktor 3–10**
  - Strategien verarbeiten Wissen über Code der Protokolle und Komposition
- **Formaler Entwurf eines verifizierten adaptiven Protokolls**

Relevante Studien- und Forschungsgebiete

- **Formale Logiken**
  - *"Natürliche" Darstellung komplexer Software und ihrer Eigenschaften?*

Relevante Studien- und Forschungsgebiete

- **Formale Logiken**
  - *"Natürliche" Darstellung komplexer Software und ihrer Eigenschaften?*
- **Beweissysteme**
  - Beweiseditoren und vollautomatische Beweisprozeduren
  - Strategien für Synthese, Verifikation und logische Optimierung
  - Logische Wissensbanken, Kooperierende & verteilte Beweiser, ...

Relevante Studien- und Forschungsgebiete

- **Formale Logiken**
  - *"Natürliche" Darstellung komplexer Software und ihrer Eigenschaften?*
- **Beweissysteme**
  - Beweiseditoren und vollautomatische Beweisprozeduren
  - Strategien für Synthese, Verifikation und logische Optimierung
  - Logische Wissensbanken, Kooperierende & verteilte Beweiser, ...
- **Anwendungen**
  - Verifiziertes Informatikwissen: Datenstrukturen, Graphentheory, ...
  - Synthese, Verifikation, Optimierung konkreter Softwareprodukte

⋮

Relevante Studien- und Forschungsgebiete

- **Formale Logiken**
  - *"Natürliche" Darstellung komplexer Software und ihrer Eigenschaften?*
- **Beweissysteme**
  - Beweiseditoren und vollautomatische Beweisprozeduren
  - Strategien für Synthese, Verifikation und logische Optimierung
  - Logische Wissensbanken, Kooperierende & verteilte Beweiser, ...
- **Anwendungen**
  - Verifiziertes Informatikwissen: Datenstrukturen, Graphentheory, ...
  - Synthese, Verifikation, Optimierung konkreter Softwareprodukte

⋮

- **Angebote an der Universität Potsdam**
  - Lehrveranstaltungen (Automatische Logik und Programmierung)
  - Mitarbeit in Forschungsprojekten
  - Kooperationen mit anderen Universitäten, z.B. CORNELL UNIVERSITY