

# Offline Bewegungsplanung: Reine Translation

Elmar Langetepe  
University of Bonn

# Folgerungen!! **Theorem 2.16**

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ ,

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!
- Trianguliere alle  $P_j$ ,

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!
- Trianguliere alle  $P_j, T_1, T_2, \dots, T_l$  mit  $O(n)$  Kanten

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!
- Trianguliere alle  $P_j, T_1, T_2, \dots, T_l$  mit  $O(n)$  Kanten
- $CT_i = T_i \oplus -R(0, 0)$  Familie von Pseudokreisen **Lem. 2.12**,



## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!
- Trianguliere alle  $P_j, T_1, T_2, \dots, T_l$  mit  $O(n)$  Kanten
- $CT_i = T_i \oplus -R(0, 0)$  Familie von Pseudokreisen **Lem. 2.12**,  
 $O((m + 3)n)$  Kanten **Lem. 2.15**

## Folgerungen!! **Theorem 2.16**

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!
- Trianguliere alle  $P_j, T_1, T_2, \dots, T_l$  mit  $O(n)$  Kanten
- $CT_i = T_i \oplus -R(0, 0)$  Familie von Pseudokreisen **Lem. 2.12**,  
 $O((m + 3)n)$  Kanten **Lem. 2.15**

$$C_{\text{verb}} = \underbrace{\bigcup_i T_i \oplus -R(0, 0)}_{\text{Komplexität } O(mn)}.$$

## Folgerungen!! Theorem 2.16

- Konvexer Roboter  $R$  mit  $|R| = m$ , polygonale Szene,  $n$  Kanten
- Komplexität von  $C_{\text{verb}}$  in  $O(mn)$ , Beweis!!
- Trianguliere alle  $P_j, T_1, T_2, \dots, T_l$  mit  $O(n)$  Kanten
- $CT_i = T_i \oplus -R(0, 0)$  Familie von Pseudokreisen **Lem. 2.12**,  $O((m + 3)n)$  Kanten **Lem. 2.15**

$$C_{\text{verb}} = \underbrace{\bigcup_i T_i \oplus -R(0, 0)}_{\text{Komplexität } O(mn)}.$$

Benutze **Theorem 2.13**

# Folgerungen!! **Theorem 2.16**

# Folgerungen!! **Theorem 2.16**

- Algorithmus 2.2: Berechnung von  $C_{\text{verb}}$

## Folgerungen!! **Theorem 2.16**

- Algorithmus 2.2: Berechnung von  $C_{\text{verb}}$ 
  - Unterteile Hindernisse in gleichgroße Teilmengen von Dreiecken  $CP_1$  und  $CP_2$

## Folgerungen!! Theorem 2.16

- Algorithmus 2.2: Berechnung von  $C_{\text{verb}}$ 
  - Unterteile Hindernisse in gleichgroße Teilmengen von Dreiecken  $CP_1$  und  $CP_2$
  - Berechne rek.  $C_{\text{verb}}(CP_1)$  und  $C_{\text{verb}}(CP_2)$ , je  $O(mn)$  Kanten

## Folgerungen!! Theorem 2.16

- Algorithmus 2.2: Berechnung von  $C_{\text{verb}}$ 
  - Unterteile Hindernisse in gleichgroße Teilmengen von Dreiecken  $CP_1$  und  $CP_2$
  - Berechne rek.  $C_{\text{verb}}(CP_1)$  und  $C_{\text{verb}}(CP_2)$ , je  $O(mn)$  Kanten
  - Berechne Vereinigung  $C_{\text{verb}}(CP_1) \cup C_{\text{verb}}(CP_2)$  in Zeit  $O(mn \log(mn))$  mit Sweep (Alg. Geom.)



## Folgerungen!! Theorem 2.16

- Algorithmus 2.2: Berechnung von  $C_{\text{verb}}$ 
  - Unterteile Hindernisse in gleichgroße Teilmengen von Dreiecken  $CP_1$  und  $CP_2$
  - Berechne rek.  $C_{\text{verb}}(CP_1)$  und  $C_{\text{verb}}(CP_2)$ , je  $O(mn)$  Kanten
  - Berechne Vereinigung  $C_{\text{verb}}(CP_1) \cup C_{\text{verb}}(CP_2)$  in Zeit  $O(mn \log(mn))$  mit Sweep (Alg. Geom.)

Laufzeit insgesamt:  $O(mn \log^2 mn)$

# Laufzeitanalyse: **Theorem 2.16**

# Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$T(m, n) \leq 2T\left(m, \frac{n}{2}\right)$$

## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$T(m, n) \leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)}$$

## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$T(m, n) \leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)} + C \times mn \log mn$$

## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$T(m, n) \leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)} + C \times mn \log mn \text{ (Merge)}$$

## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$\begin{aligned} T(m, n) &\leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)} + C \times mn \log mn \text{ (Merge)} \\ &\leq 2\left(2T\left(m, \frac{n}{4}\right) + C\frac{mn}{2} \log \frac{mn}{2}\right) + C \times mn \log mn \end{aligned}$$



## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$\begin{aligned} T(m, n) &\leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)} + C \times mn \log mn \text{ (Merge)} \\ &\leq 2\left(2T\left(m, \frac{n}{4}\right) + C\frac{mn}{2} \log \frac{mn}{2}\right) + C \times mn \log mn \\ &\vdots \quad \vdots \end{aligned}$$

## Laufzeitanalyse: **Theorem 2.16**

Arrangement mit  $O(nm)$  Kanten!!

$$\begin{aligned} T(m, n) &\leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)} + C \times mn \log mn \text{ (Merge)} \\ &\leq 2\left(2T\left(m, \frac{n}{4}\right) + C\frac{mn}{2} \log \frac{mn}{2}\right) + C \times mn \log mn \\ &\quad \vdots \quad \vdots \\ &\leq nT(m, 1) + Cmn \sum_{i=0}^{\log n} \log \frac{mn}{2^i} \end{aligned}$$

## Laufzeitanalyse: Theorem 2.16

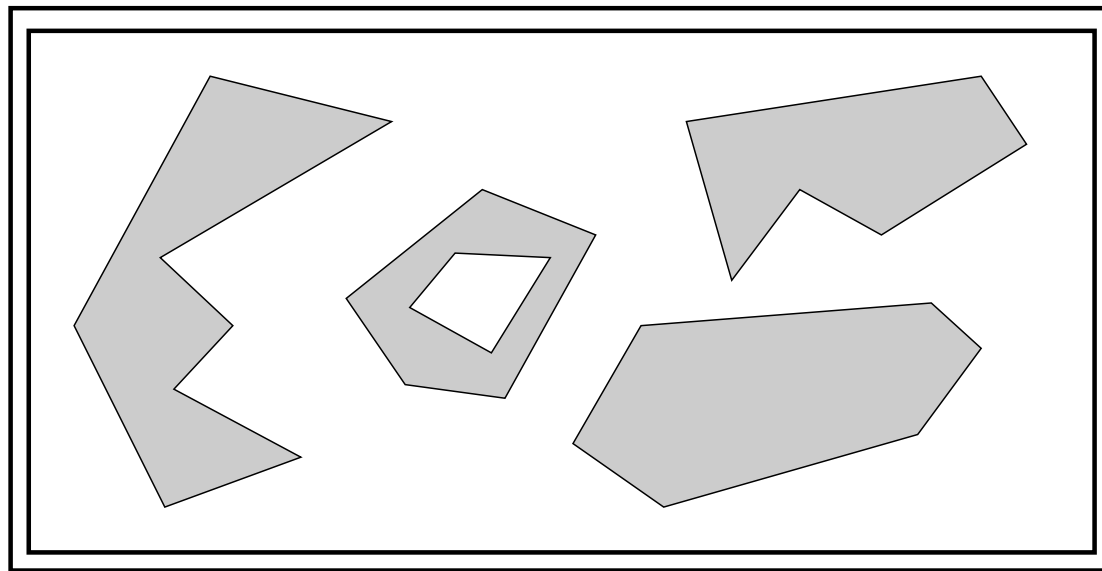
Arrangement mit  $O(nm)$  Kanten!!

$$\begin{aligned} T(m, n) &\leq 2T\left(m, \frac{n}{2}\right) \text{ (Rek.)} + C \times mn \log mn \text{ (Merge)} \\ &\leq 2\left(2T\left(m, \frac{n}{4}\right) + C\frac{mn}{2} \log \frac{mn}{2}\right) + C \times mn \log mn \\ &\quad \vdots \quad \vdots \\ &\leq nT(m, 1) + Cmn \sum_{i=0}^{\log n} \log \frac{mn}{2^i} \\ &\in O(mn \log^2 mn) \end{aligned}$$

$C_{\text{frei}}$  verwenden! Roadmap!!

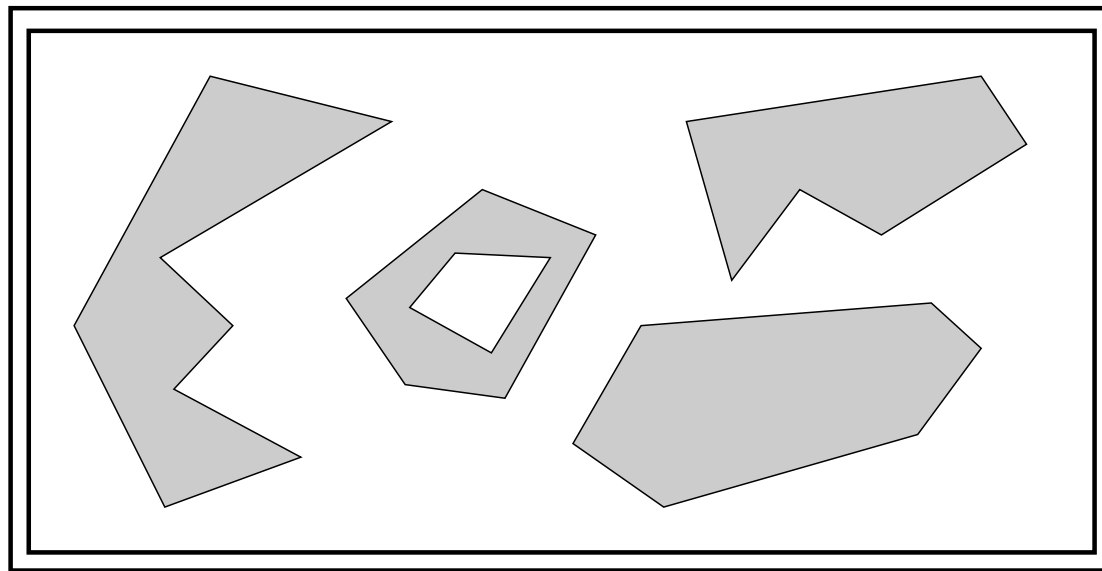
# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)



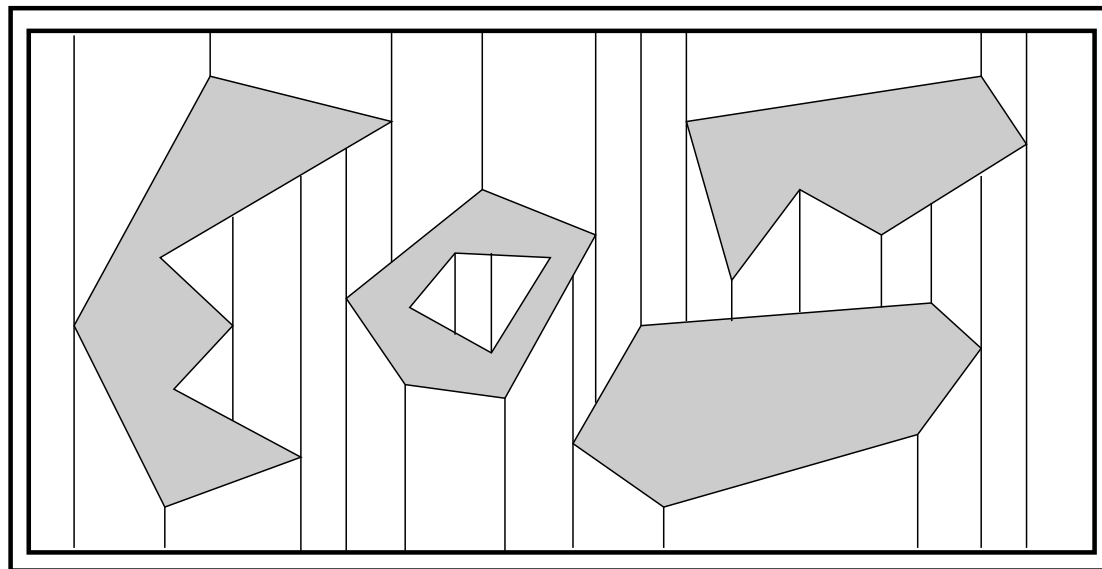
# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)



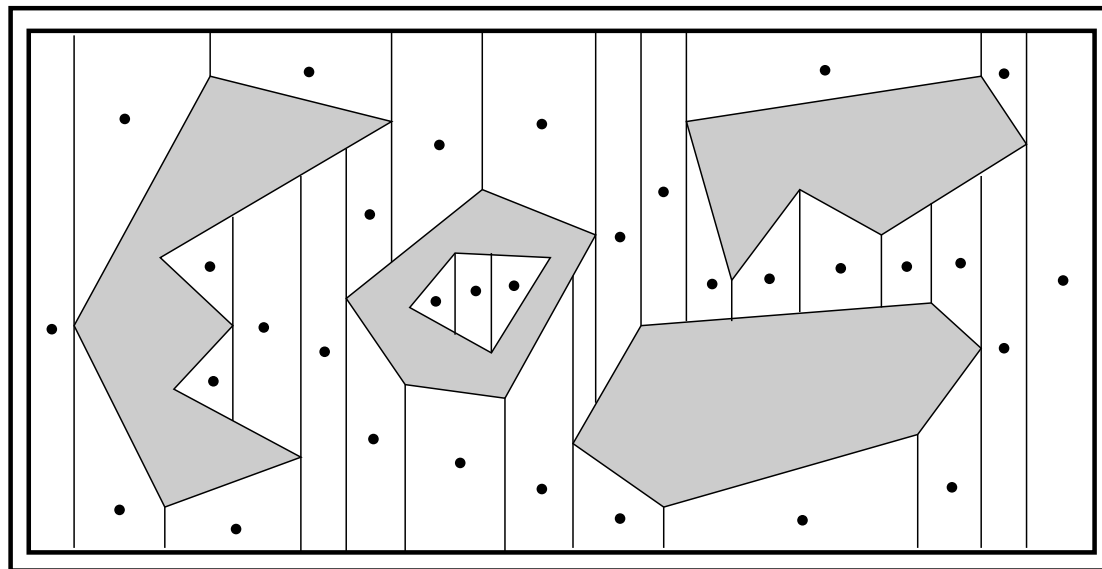
# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)



# $C_{\text{frei}}$ verwenden! Roadmap!!

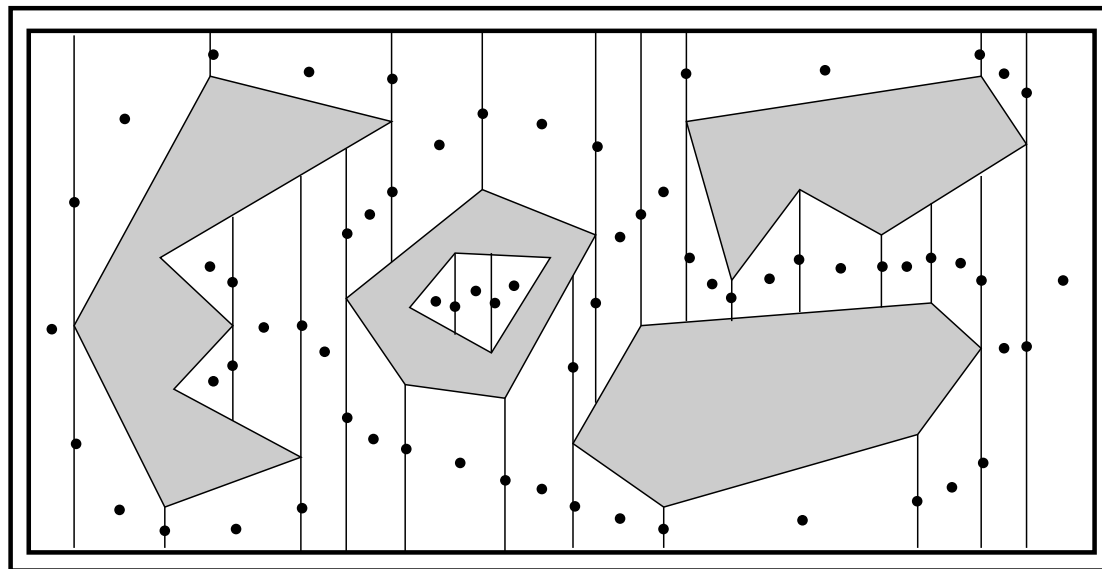
- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)





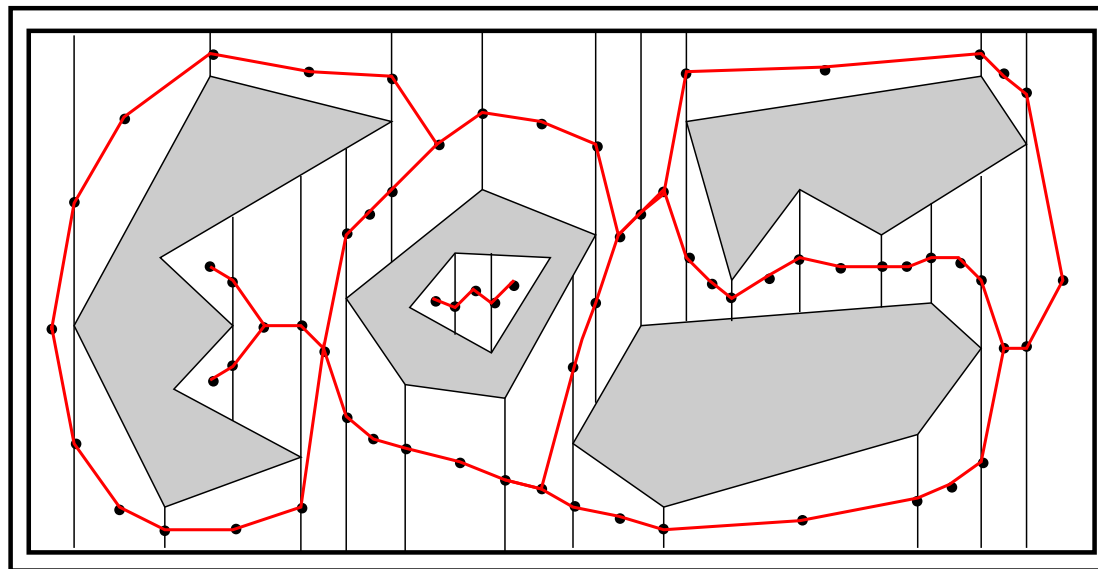
# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)



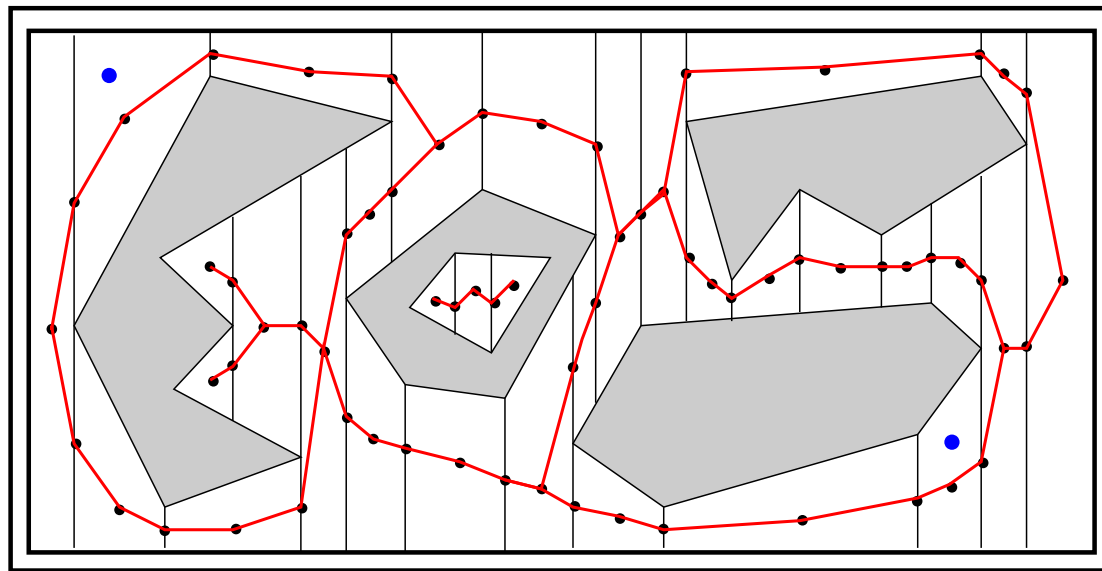
# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)
- Zusammenhangsgraph, Roadmap



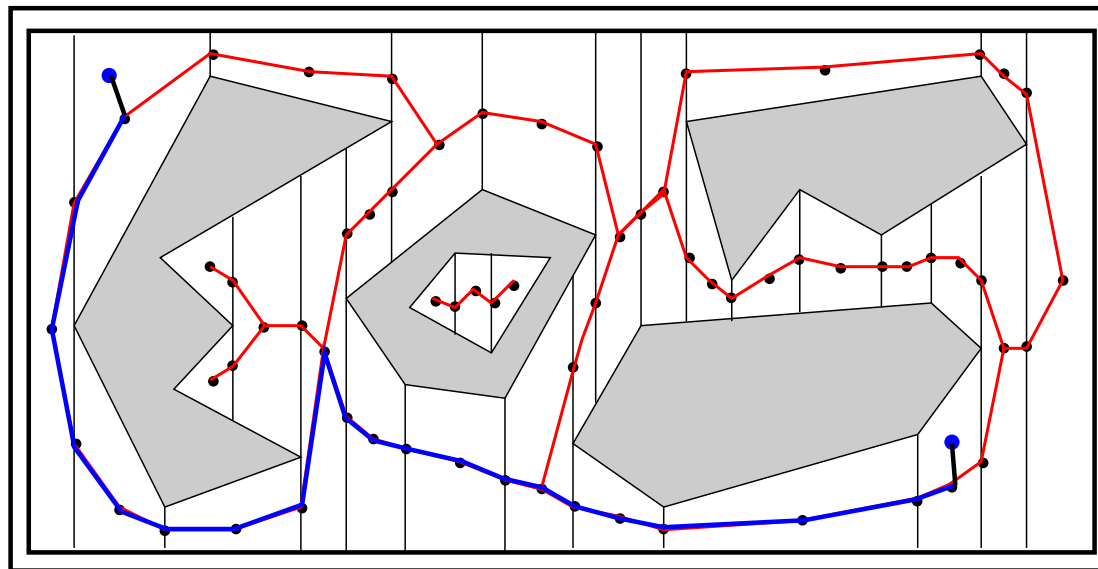
# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)
- Zusammenhangsgraph, Roadmap
- Lokalisation  $s$  und  $t$  und verbinden (BFS)



# $C_{\text{frei}}$ verwenden! Roadmap!!

- Zerlegung in Trapezen (Seidel)
- Benachbarte Trapeze verbinden (Schwerpunkt/Mittelpunkt)
- Zusammenhangsgraph, Roadmap
- Lokalisation  $s$  und  $t$  und verbinden (BFS)



# Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

# Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

Vorbereitung:

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur,

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur, (randomisiert!!)  $O(mn \log(mn))$



## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point–Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

- Lokalisierere  $s$  und  $t$  Trapeze,

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point–Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

- Lokalisierere  $s$  und  $t$  Trapeze,  $O(\log(mn))$

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

- Lokalisierere  $s$  und  $t$  Trapeze,  $O(\log(mn))$
- Finde Weg der Trapeze mit BFS,

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

- Lokalisierere  $s$  und  $t$  Trapeze,  $O(\log(mn))$
- Finde Weg der Trapeze mit BFS,  $O(mn)$

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point–Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

- Lokalisierere  $s$  und  $t$  Trapeze,  $O(\log(mn))$
- Finde Weg der Trapeze mit BFS,  $O(mn)$

Roadmap!! Einfach aber effizient!!

## Algorithmus 2.3: $C_{\text{frei}}$ gegeben!

### Vorbereitung:

- Trapezzerlegung  $C_{\text{frei}}$ /Point-Location Struktur, (randomisiert!!)  $O(mn \log(mn))$
- Roadmap-Konstruktion: Verbindung adjazenter Trapeze,  $O(mn)$

### Query:

- Lokalisierere  $s$  und  $t$  Trapeze,  $O(\log(mn))$
- Finde Weg der Trapeze mit BFS,  $O(mn)$

Roadmap!! Einfach aber effizient!! Probabilistisch!



# Ergebnis: **Theorem 2.17**

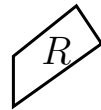
## Ergebnis: Theorem 2.17

Translationsbewegungen eines konvexen, polygonalen Roboters mit  $m$  Ecken in einer Umgebung mit polygonalen Hindernissen mit insgesamt  $n$  Ecken können nach  $O(mn \log^2(mn))$  (randomisierter) Vorbereitungszeit in Zeit  $O(mn)$  geplant werden.

# Kreisförmiger! Voronoi Diagramme 2.1.1

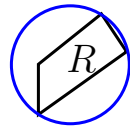
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter



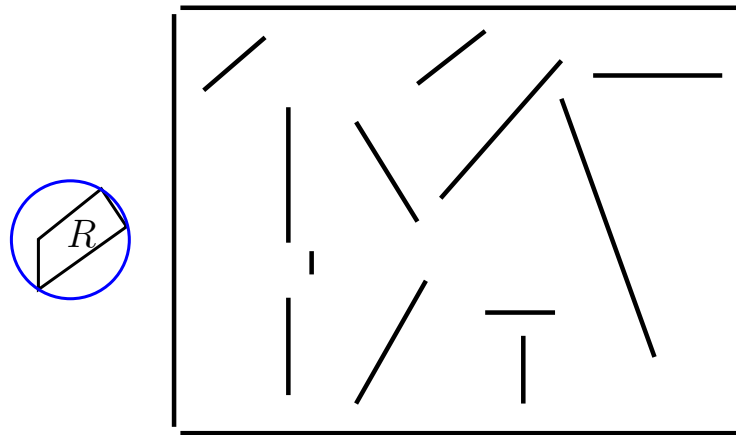
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse



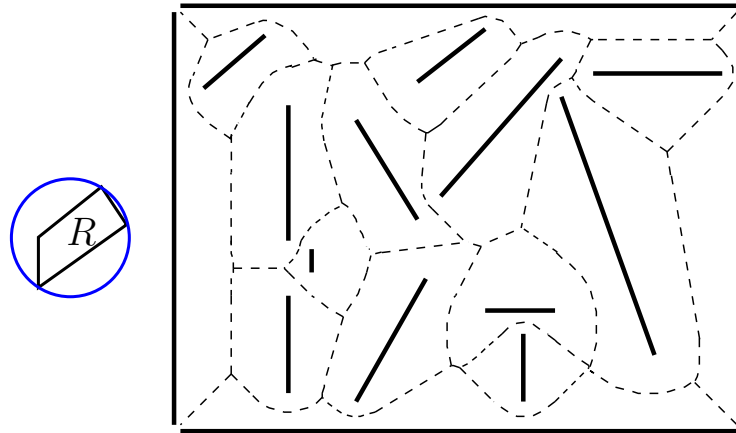
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse
- Weg auf Bisektoren:



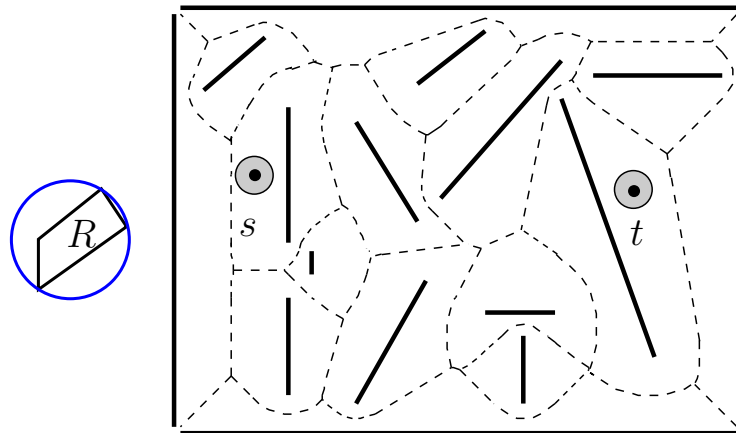
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse
- Weg auf Bisektoren:



# Kreisförmiger! Voronoi Diagramme 2.1.1

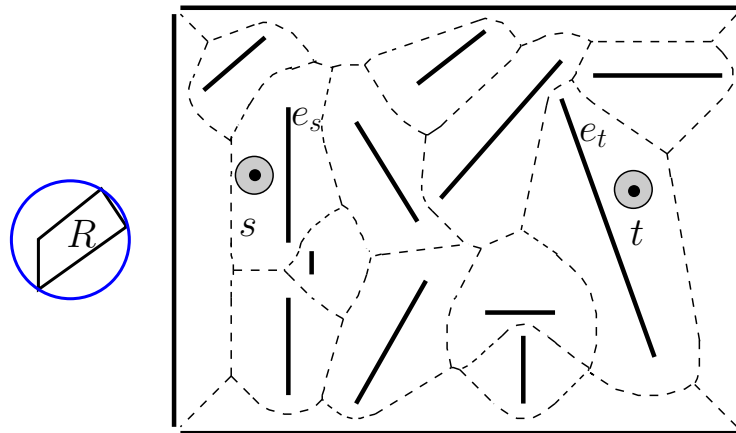
- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse
- Weg auf Bisektoren:





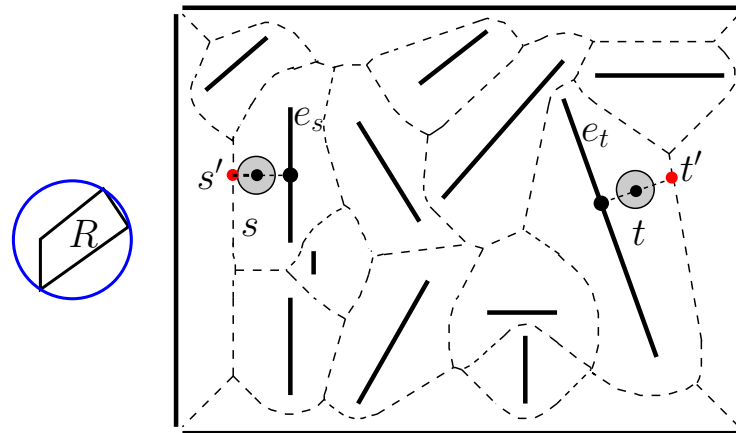
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse
- Weg auf Bisektoren:



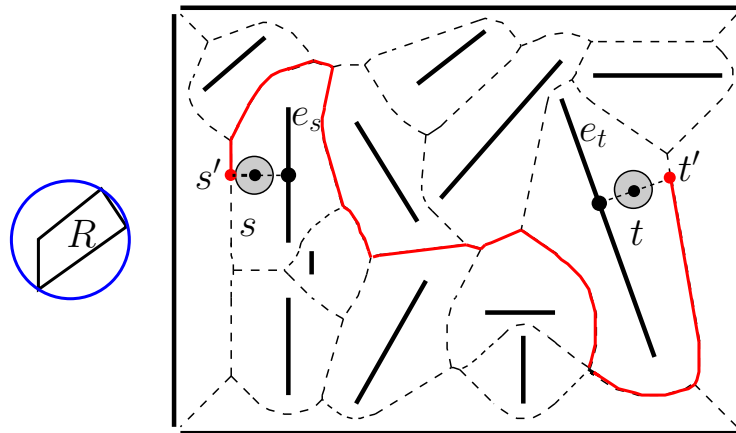
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse
- Weg auf Bisektoren:



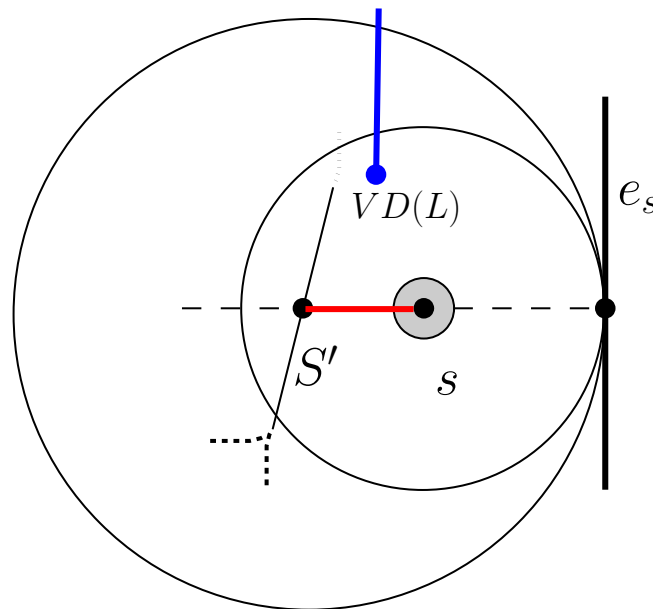
# Kreisförmiger! Voronoi Diagramme 2.1.1

- Verwende kleinsten Kreis um Roboter
- Voronoi Diagramm der Segmente der Hindernisse
- Weg auf Bisektoren:



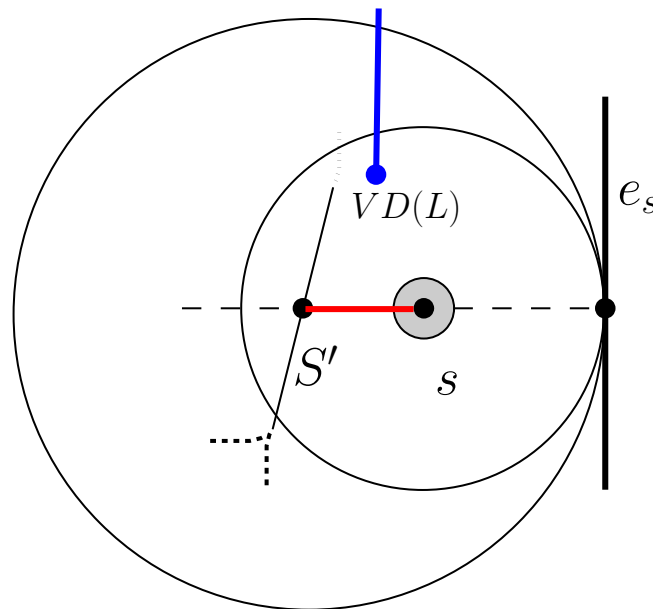


# Start $s'$ kann stets angelaufen werden



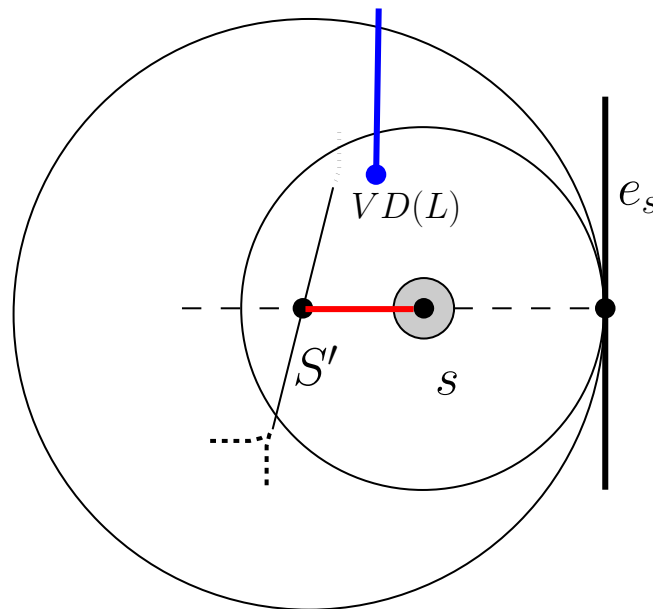
# Start $s'$ kann stets angelaufen werden

- $s$  in Region von  $e_s$ ,



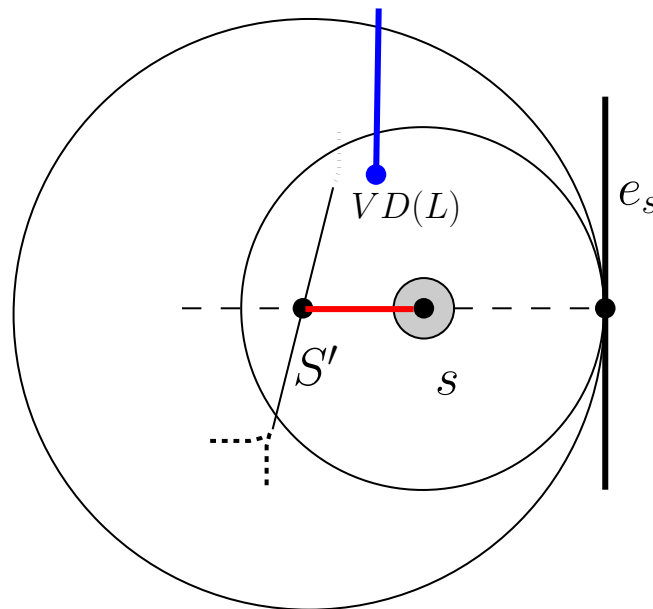
# Start $s'$ kann stets angelaufen werden

- $s$  in Region von  $e_s$ , Kreis frei



# Start $s'$ kann stets angelaufen werden

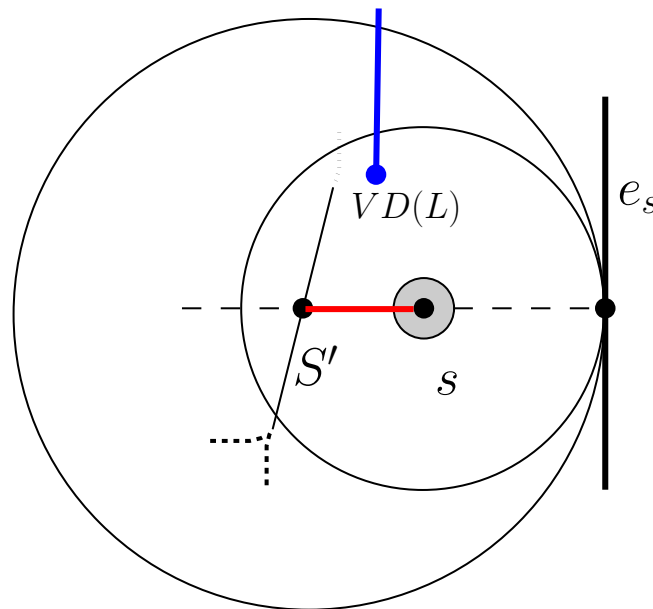
- $s$  in Region von  $e_s$ , Kreis frei
- Kürzester Weg zu  $e_s$ , Strahl Richtung Bisector





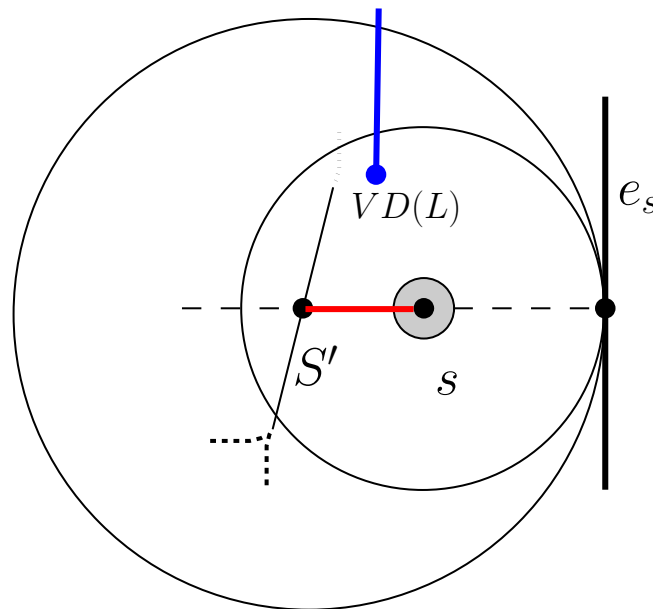
# Start $s'$ kann stets angelaufen werden

- $s$  in Region von  $e_s$ , Kreis frei
- Kürzester Weg zu  $e_s$ , Strahl Richtung Bisector
- Trifft Bisector bei  $s'$ ,



# Start $s'$ kann stets angelaufen werden

- $s$  in Region von  $e_s$ , Kreis frei
- Kürzester Weg zu  $e_s$ , Strahl Richtung Bisector
- Trifft Bisector bei  $s'$ , Weg ist frei!!



# Algorithmus 2.1: Kreisförmiger Roboter

# Algorithmus 2.1: Kreisförmiger Roboter

Vorbereitung:

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi-Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$



# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

**Query** für zwei Punkte  $s, t$ :

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:  $O(\log n)$

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:  $O(\log n)$
- $s$  gehört zu  $e_s$ ,  $t$  gehört zu  $e_t$

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:  $O(\log n)$
- $s$  gehört zu  $e_s$ ,  $t$  gehört zu  $e_t$
- Bestimme mit Strahlen Punkt  $t'$  und  $s'$  auf  $VD(L)$ :

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:  $O(\log n)$
- $s$  gehört zu  $e_s$ ,  $t$  gehört zu  $e_t$
- Bestimme mit Strahlen Punkt  $t'$  und  $s'$  auf  $VD(L)$ :  $O(1)$

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:  $O(\log n)$
- $s$  gehört zu  $e_s$ ,  $t$  gehört zu  $e_t$
- Bestimme mit Strahlen Punkt  $t'$  und  $s'$  auf  $VD(L)$ :  $O(1)$
- Berechne (BFS) in  $VD(L)$  Pfad von  $s'$  nach  $t'$  mit zul. Mindestabstand oder berichte, dass es keinen gibt:

# Algorithmus 2.1: Kreisförmiger Roboter

## Vorbereitung:

- Konstruiere das Voronoi–Diagramm  $VD(L)$ , Kanten der Hindernisse als Liniensegmente:  $O(n \log n)$
- Point–Location Trapezzerlegung:  $O(n \log n)$

## Query für zwei Punkte $s, t$ :

- Lokalisierere  $s, t$  im VD:  $O(\log n)$
- $s$  gehört zu  $e_s$ ,  $t$  gehört zu  $e_t$
- Bestimme mit Strahlen Punkt  $t'$  und  $s'$  auf  $VD(L)$ :  $O(1)$
- Berechne (BFS) in  $VD(L)$  Pfad von  $s'$  nach  $t'$  mit zul. Mindestabstand oder berichte, dass es keinen gibt:  $O(n)$



# Kreisförmig, konvex! Beliebiger Roboter!

# Kreisförmig, konvex! Beliebiger Roboter!

- Bisher: Divide and Conquer Konfigurationsraum

# Kreisförmig, konvex! Beliebiger Roboter!

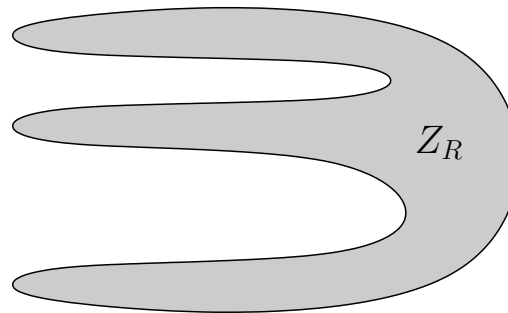
- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$

# Kreisförmig, konvex! Beliebiger Roboter!

- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$
- Merge komplett:  $\Theta((nm)^2)$

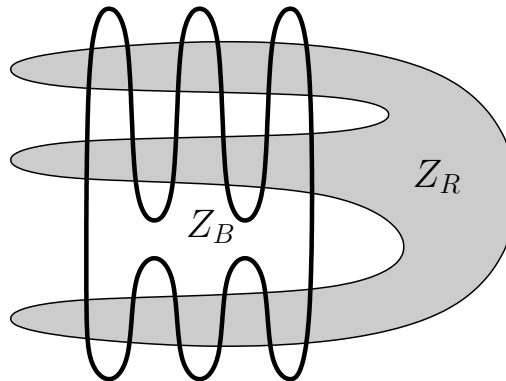
# Kreisförmig, konvex! Beliebiger Roboter!

- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$
- Merge komplett:  $\Theta((nm)^2)$
- Beobachtung: Zelle  $Z_s$  nicht so komplex??



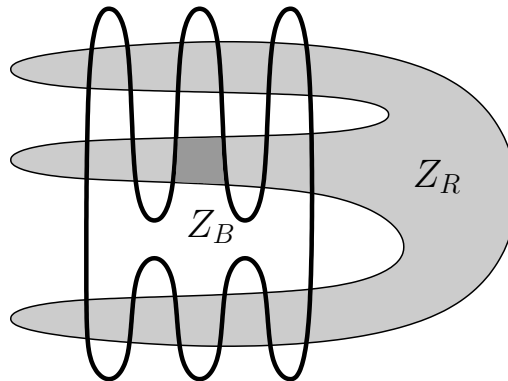
# Kreisförmig, konvex! Beliebiger Roboter!

- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$
- Merge komplett:  $\Theta((nm)^2)$
- Beobachtung: Zelle  $Z_s$  nicht so komplex??



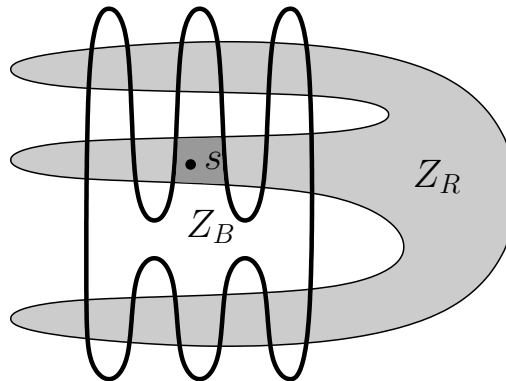
# Kreisförmig, konvex! Beliebiger Roboter!

- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$
- Merge komplett:  $\Theta((nm)^2)$
- Beobachtung: Zelle  $Z_s$  nicht so komplex??



# Kreisförmig, konvex! Beliebiger Roboter!

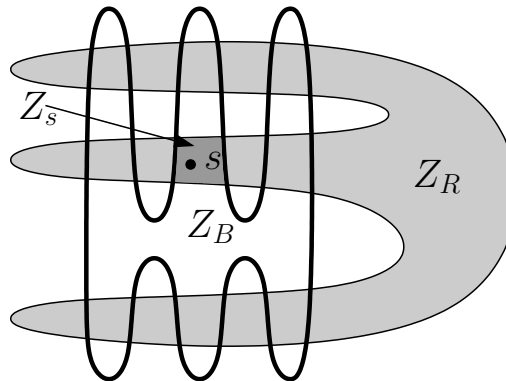
- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$
- Merge komplett:  $\Theta((nm)^2)$
- Beobachtung: Zelle  $Z_s$  nicht so komplex??





# Kreisförmig, konvex! Beliebiger Roboter!

- Bisher: Divide and Conquer Konfigurationsraum
- Nicht konvex: Komplexität  $\Theta((nm)^2)$
- Merge komplett:  $\Theta((nm)^2)$
- Beobachtung: Zelle  $Z_s$  nicht so komplex??



# Begrenzung einer Zelle!!

# Begrenzung einer Zelle!!

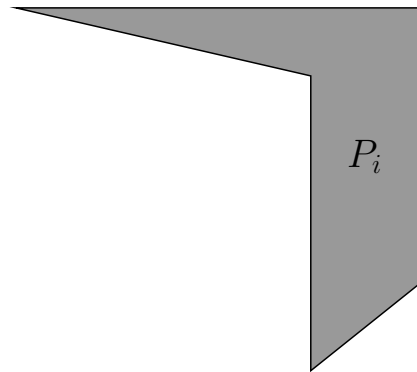
- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ .

# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken

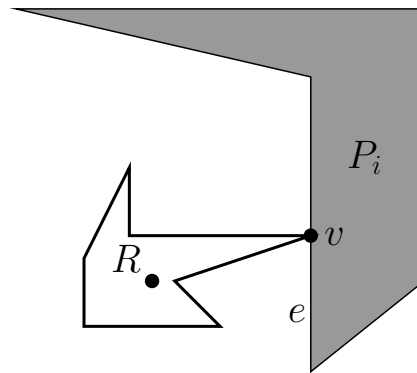
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis



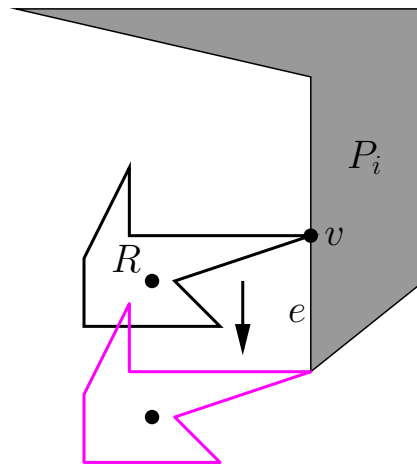
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis



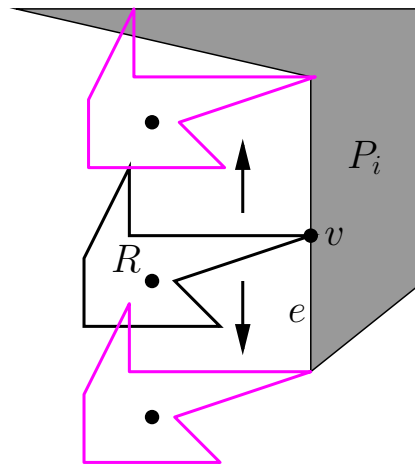
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis



# Begrenzung einer Zelle!!

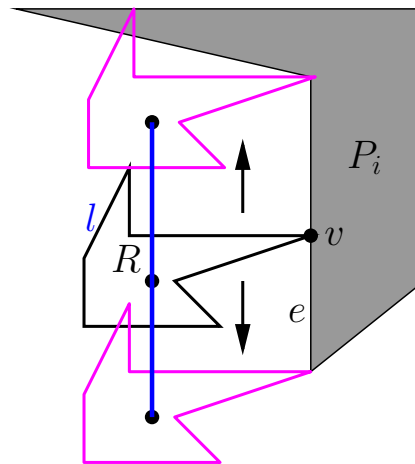
- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis





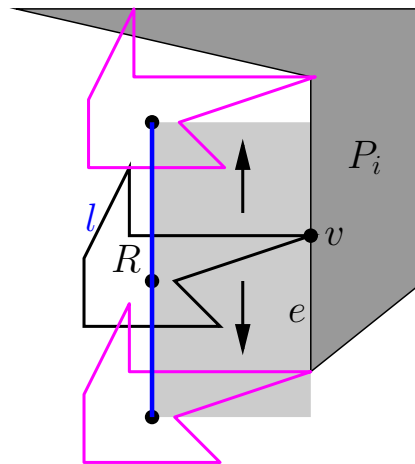
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis



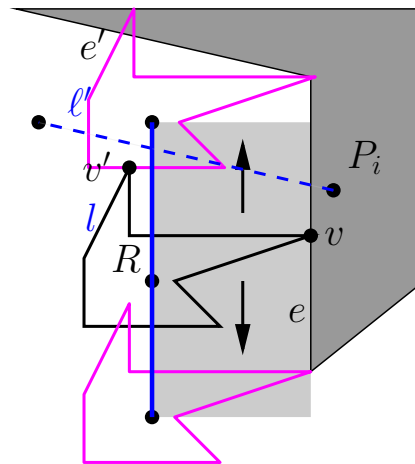
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis



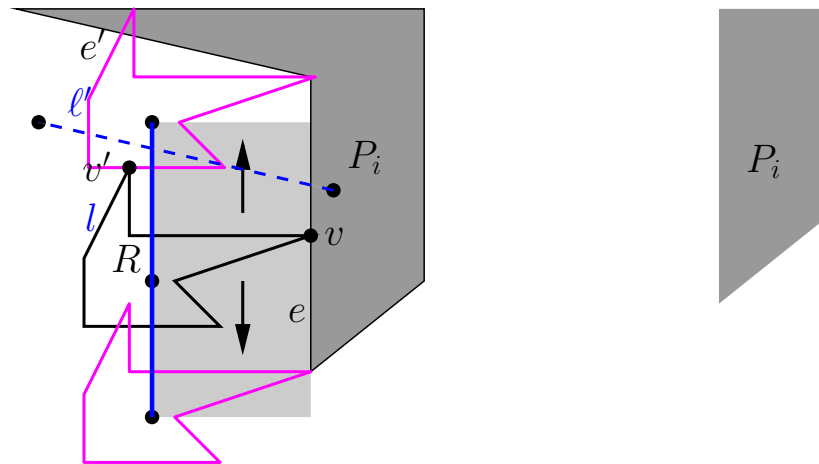
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis



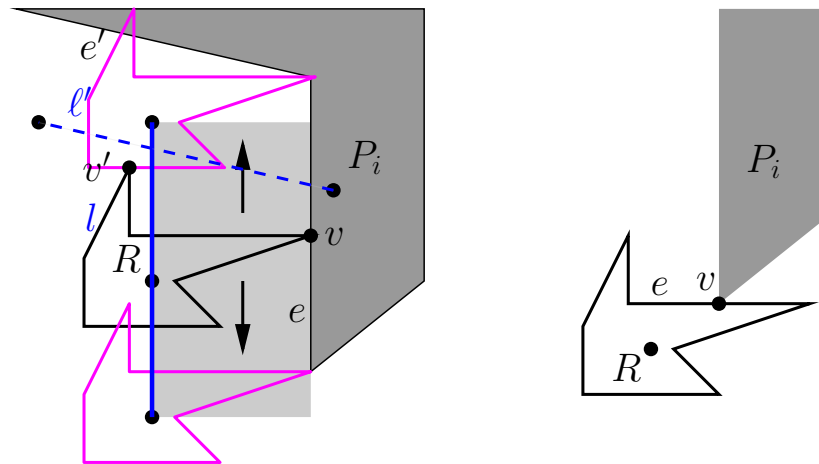
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis
- Ecke Hindernis, Kante Roboter



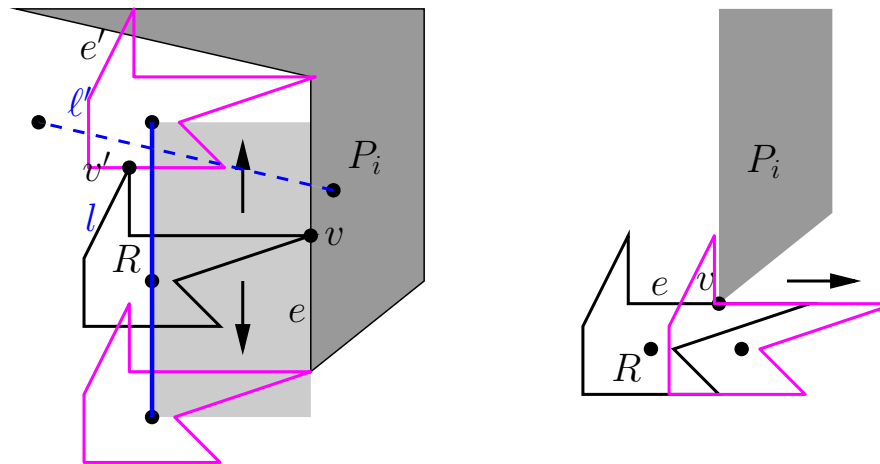
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis
- Ecke Hindernis, Kante Roboter



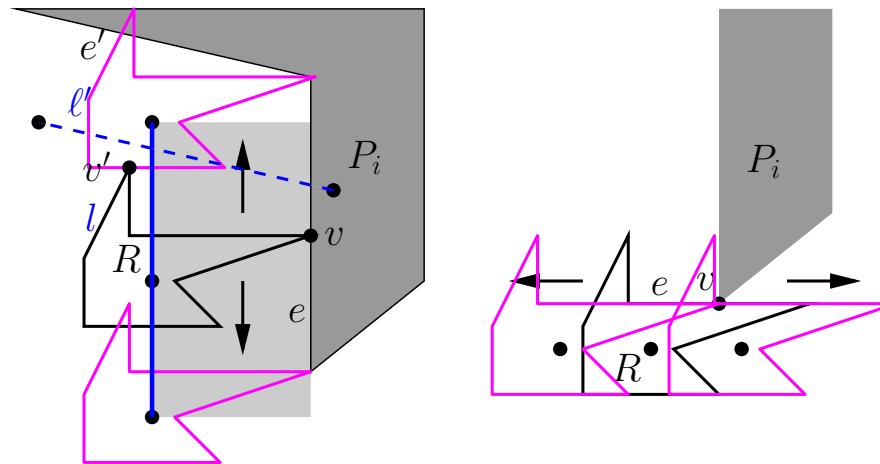
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis
- Ecke Hindernis, Kante Roboter



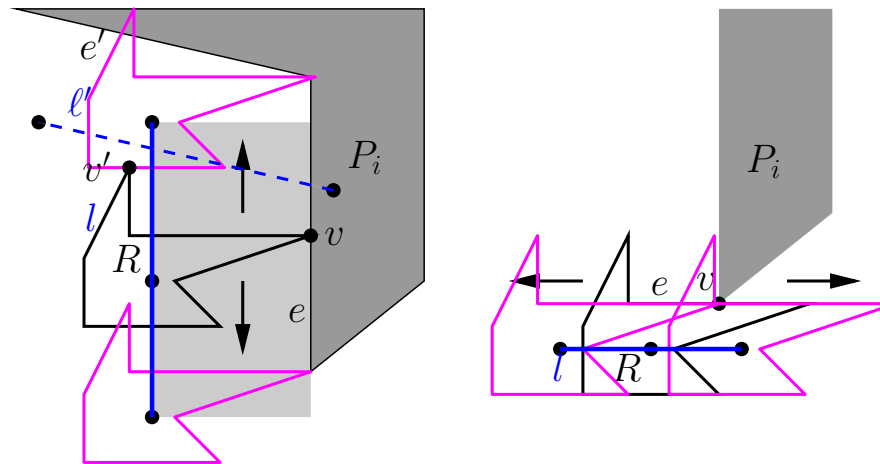
# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis
- Ecke Hindernis, Kante Roboter



# Begrenzung einer Zelle!!

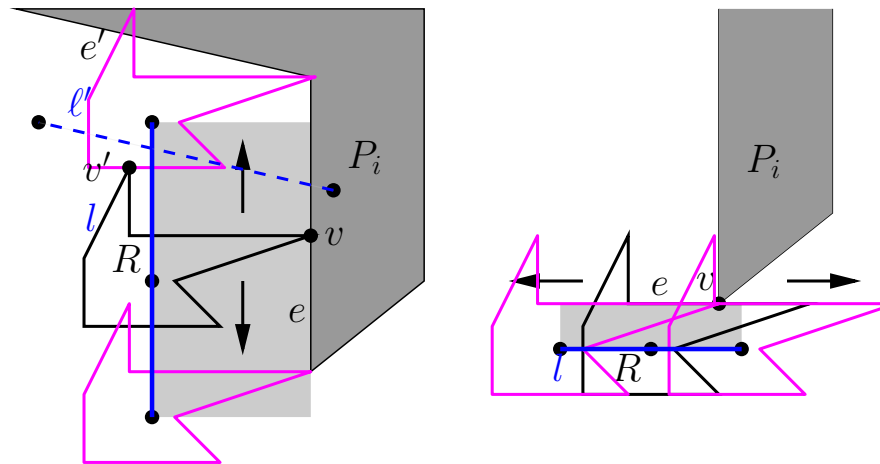
- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis
- Ecke Hindernis, Kante Roboter



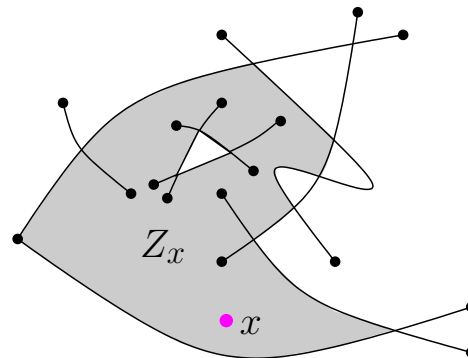


# Begrenzung einer Zelle!!

- Nicht-konvexer Roboter  $R$  mit  $|R| = m$ . Polygonale Szene  $n$  Ecken
- Ecke Roboter, Kante Hindernis
- Ecke Hindernis, Kante Roboter
- $\Theta(mn)$  viele Kanten

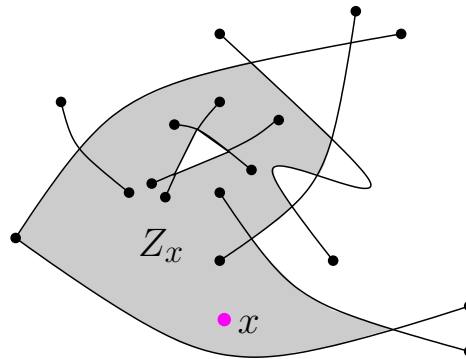


# Allgemeiner!!



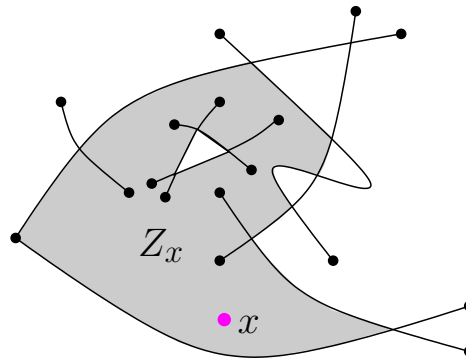
# Allgemeiner!!

- Menge von Segmenten



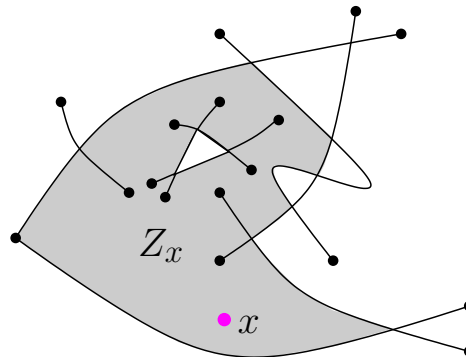
# Allgemeiner!!

- Menge von Segmenten
- Je zwei schneiden sich  $s$  mal



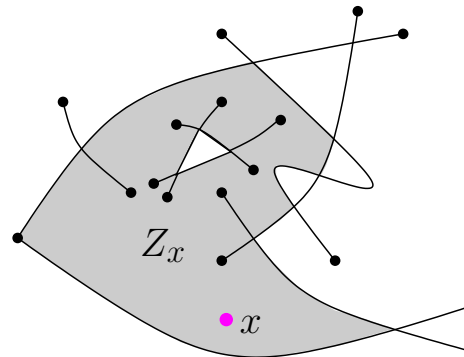
# Allgemeiner!!

- Menge von Segmenten
- Je zwei schneiden sich  $s$  mal
- Spezieller Punkt  $x$



# Allgemeiner!!

- Menge von Segmenten
- Je zwei schneiden sich  $s$  mal
- Spezieller Punkt  $x$
- Komplexität der Zelle  $Z_x$



# Fahrplan!!

# Fahrplan!!

- Divide and Conquer!!



# Fahrplan!!

- Divide and Conquer!!
- Teile Segmente in zwei gleichgroße Mengen  $Z_1, Z_2$

# Fahrplan!!

- Divide and Conquer!!
- Teile Segmente in zwei gleichgroße Mengen  $Z_1, Z_2$
- Berechne  $Z_{1x}$  und  $Z_{2x}$

# Fahrplan!!

- Divide and Conquer!!
- Teile Segmente in zwei gleichgroße Mengen  $Z_1, Z_2$
- Berechne  $Z_{1x}$  und  $Z_{2x}$
- Merge zu  $\{Z_1 \cup Z_2\}_x$

# Fahrplan!!

- Divide and Conquer!!
- Teile Segmente in zwei gleichgroße Mengen  $Z_1, Z_2$
- Berechne  $Z_{1x}$  und  $Z_{2x}$
- Merge zu  $\{Z_1 \cup Z_2\}_x$
- Spezieller Merge wegen Schnitt mit  $x$

# Fahrplan!!

- Divide and Conquer!!
- Teile Segmente in zwei gleichgroße Mengen  $Z_1, Z_2$
- Berechne  $Z_{1x}$  und  $Z_{2x}$
- Merge zu  $\{Z_1 \cup Z_2\}_x$
- Spezieller Merge wegen Schnitt mit  $x$
- **RED BLUE** Merge

# Fahrplan!!

- Divide and Conquer!!
- Teile Segmente in zwei gleichgroße Mengen  $Z_1, Z_2$
- Berechne  $Z_{1x}$  und  $Z_{2x}$
- Merge zu  $\{Z_1 \cup Z_2\}_x$
- Spezieller Merge wegen Schnitt mit  $x$
- RED BLUE Merge
- Merge: Komplexität des Ergebnisses

# Exkurs: Davenport-Schinzel-Sequenzen

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.



# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport-Schinzel-Sequenz der Ordnung  $s$

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport–Schinzel–Sequenz der Ordnung  $s$

- Wort  $w$  über  $\Sigma$

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport–Schinzel–Sequenz der Ordnung  $s$

- Wort  $w$  über  $\Sigma$
- in  $w$  keine benachbarten Buchstaben gleich

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport–Schinzel–Sequenz der Ordnung  $s$

- Wort  $w$  über  $\Sigma$
- in  $w$  keine benachbarten Buchstaben gleich
- keine zwei verschiedenen Buchstaben wechseln mehr als  $s$  mal

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport–Schinzel–Sequenz der Ordnung  $s$

- Wort  $w$  über  $\Sigma$
- in  $w$  keine benachbarten Buchstaben gleich
- keine zwei verschiedenen Buchstaben wechseln mehr als  $s$  mal

Bsp.: ABRAKADABRA;

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport–Schinzel–Sequenz der Ordnung  $s$

- Wort  $w$  über  $\Sigma$
- in  $w$  keine benachbarten Buchstaben gleich
- keine zwei verschiedenen Buchstaben wechseln mehr als  $s$  mal

Bsp.: ABRAKADABRA; max. 4 Wechsel (A und B)

# Exkurs: Davenport-Schinzel-Sequenzen

Alphabet  $\Sigma = \{A, B, C, \dots\}$  über  $n$  Buchstaben.

Davenport–Schinzel–Sequenz der Ordnung  $s$

- Wort  $w$  über  $\Sigma$
- in  $w$  keine benachbarten Buchstaben gleich
- keine zwei verschiedenen Buchstaben wechseln mehr als  $s$  mal

Bsp.: ABRAKADABRA; max. 4 Wechsel (A und B)

$\lambda_s(n)$  maximale Länge eines solchen Wortes

# Davenport-Schinzel-Sequenzen: **Th. A9**



# Davenport-Schinzel-Sequenzen: **Th. A9**

Fast linear!!

# Davenport-Schinzel-Sequenzen: Th. A9

Fast linear!! (Ohne Beweis!)

$$\lambda_1(n) = n$$

$$\lambda_2(n) = 2n - 1$$

$$\lambda_3(n) \in \Theta(n \alpha(n))$$

$$\lambda_4(n) \in \Theta(n \cdot 2^{\alpha(n)})$$

$$\lambda_s(n) \in O(n \log^*(n)) \in O(n^2)$$

# Davenport-Schinzel-Sequenzen: Th. A9

Fast linear!! (Ohne Beweis!)

$$\lambda_1(n) = n$$

$$\lambda_2(n) = 2n - 1$$

$$\lambda_3(n) \in \Theta(n \alpha(n))$$

$$\lambda_4(n) \in \Theta(n \cdot 2^{\alpha(n)})$$

$$\lambda_s(n) \in O(n \log^*(n)) \in O(n^2)$$

$\alpha(n)$  Inverse Ackermann Fkt.

# Davenport-Schinzel-Sequenzen: Th. A9

Fast linear!! (Ohne Beweis!)

$$\lambda_1(n) = n$$

$$\lambda_2(n) = 2n - 1$$

$$\lambda_3(n) \in \Theta(n \alpha(n))$$

$$\lambda_4(n) \in \Theta(n \cdot 2^{\alpha(n)})$$

$$\lambda_s(n) \in O(n \log^*(n)) \in O(n^2)$$

$\alpha(n)$  Inverse Ackermann Fkt.

$\log^*(n)$

# Untere Kontur: Def. A10

## Untere Kontur: **Def. A10**

$f_1, f_2, \dots, f_n$  reellwertige Funktionen, entweder

## Untere Kontur: **Def. A10**

$f_1, f_2, \dots, f_n$  reellwertige Funktionen, entweder

- über einem gemeinsamen Intervall  $I$  oder

## Untere Kontur: Def. A10

$f_1, f_2, \dots, f_n$  reellwertige Funktionen, entweder

- über einem gemeinsamen Intervall  $I$  oder
- über je einem Intervall  $I_j \subseteq I, 1 \leq j \leq n$ .



## Untere Kontur: Def. A10

$f_1, f_2, \dots, f_n$  reellwertige Funktionen, entweder

- über einem gemeinsamen Intervall  $I$  oder
- über je einem Intervall  $I_j \subseteq I, 1 \leq j \leq n$ .

Je zwei Funktionen max.  $s$  gemeinsame Schnittpunkte

## Untere Kontur: Def. A10

$f_1, f_2, \dots, f_n$  reellwertige Funktionen, entweder

- über einem gemeinsamen Intervall  $I$  oder
- über je einem Intervall  $I_j \subseteq I, 1 \leq j \leq n$ .

Je zwei Funktionen max.  $s$  gemeinsame Schnittpunkte

$$L(x) := \min \{ f_i(x) \mid 1 \leq i \leq n \}$$

## Untere Kontur: Def. A10

$f_1, f_2, \dots, f_n$  reellwertige Funktionen, entweder

- über einem gemeinsamen Intervall  $I$  oder
- über je einem Intervall  $I_j \subseteq I, 1 \leq j \leq n$ .

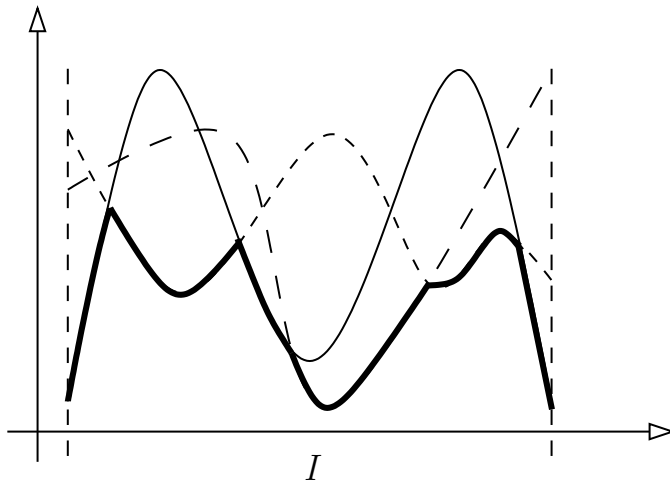
Je zwei Funktionen max.  $s$  gemeinsame Schnittpunkte

$$L(x) := \min \{ f_i(x) \mid 1 \leq i \leq n \}$$

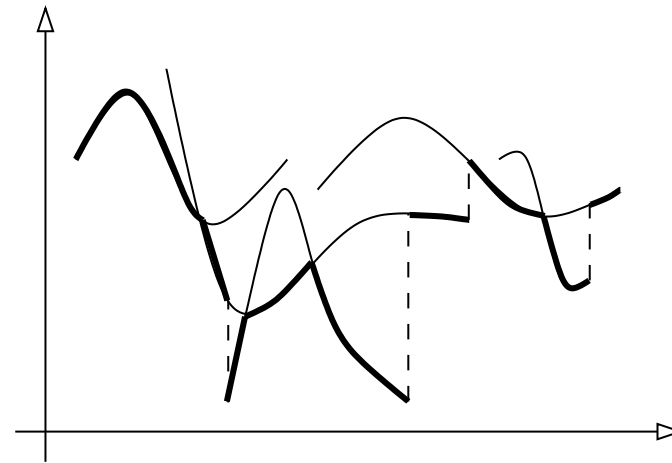
Lower envelope der Funktionsgraphen

# Untere Kontur: **Th. A12**

## Untere Kontur: Th. A12



(1)



(2)

$L(x)$  besteht aus maximal

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

vielen Teilstücken

# Beweis: Th. A12

## Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

1.:



## Beweis: Th. A12

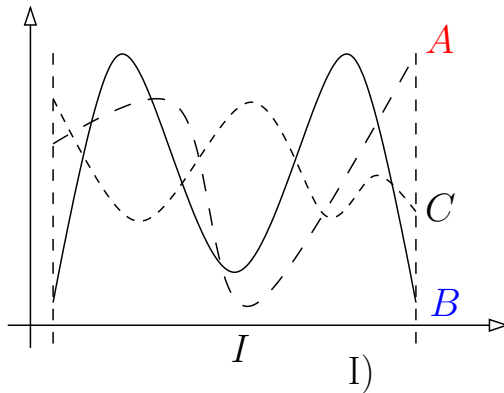
1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen

# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

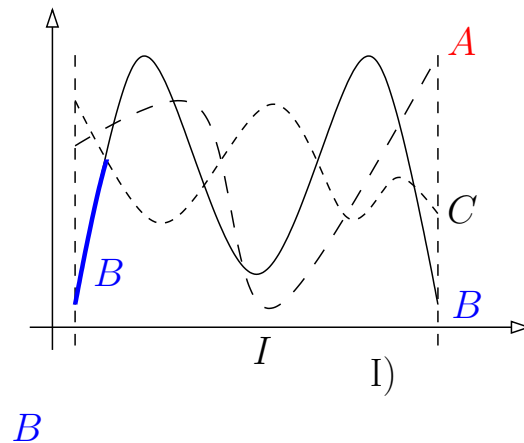
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

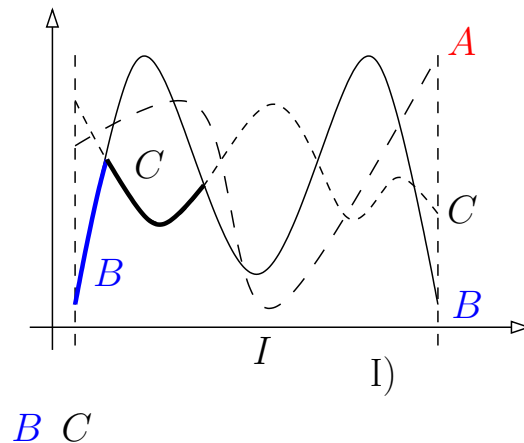
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

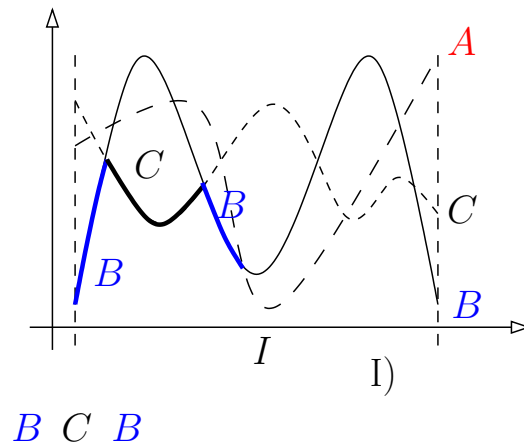
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

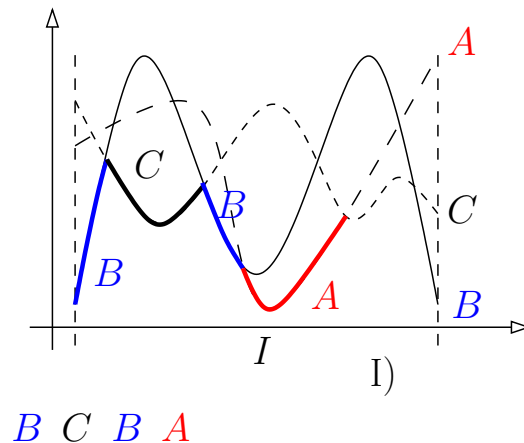
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

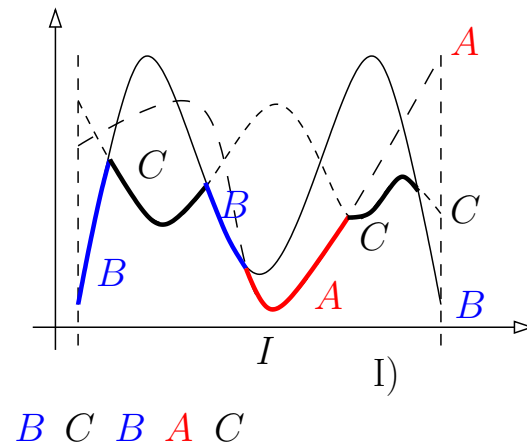
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

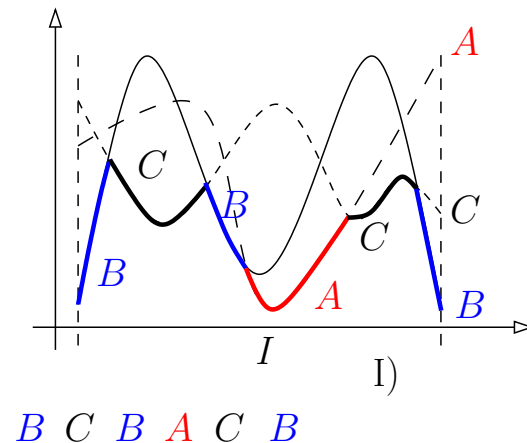
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen

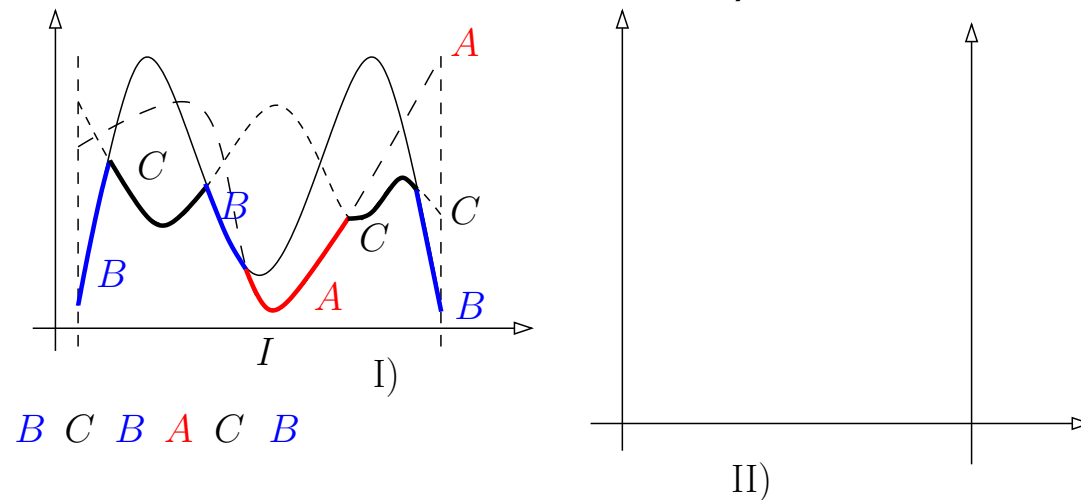




# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

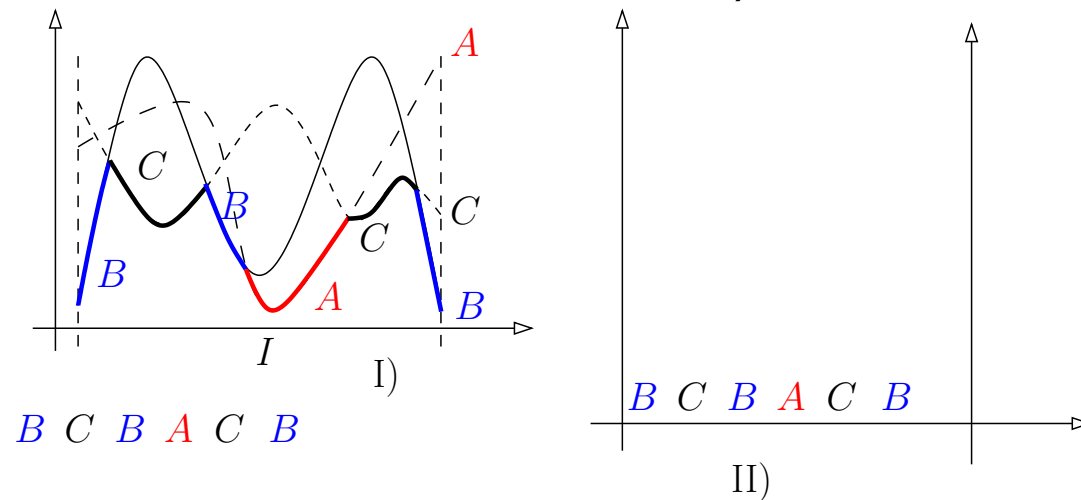
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

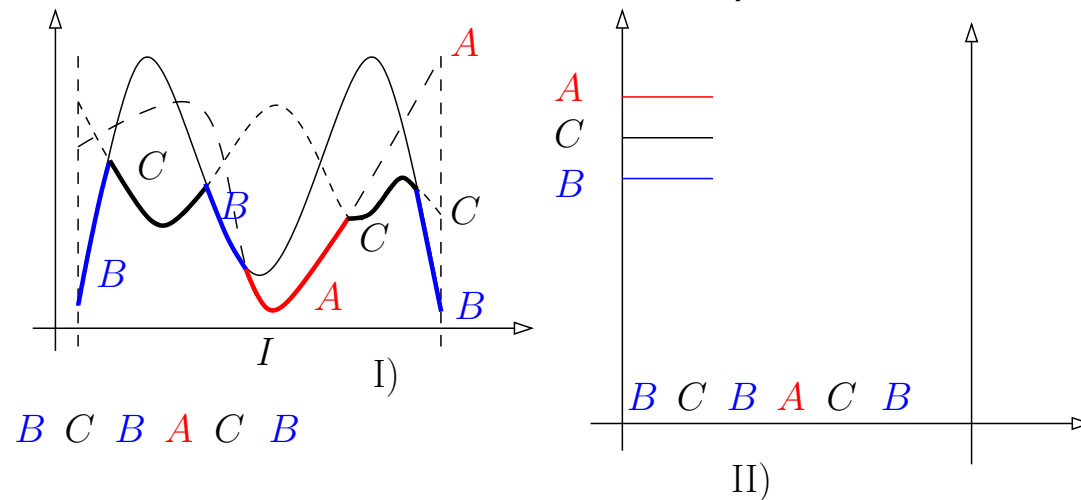
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

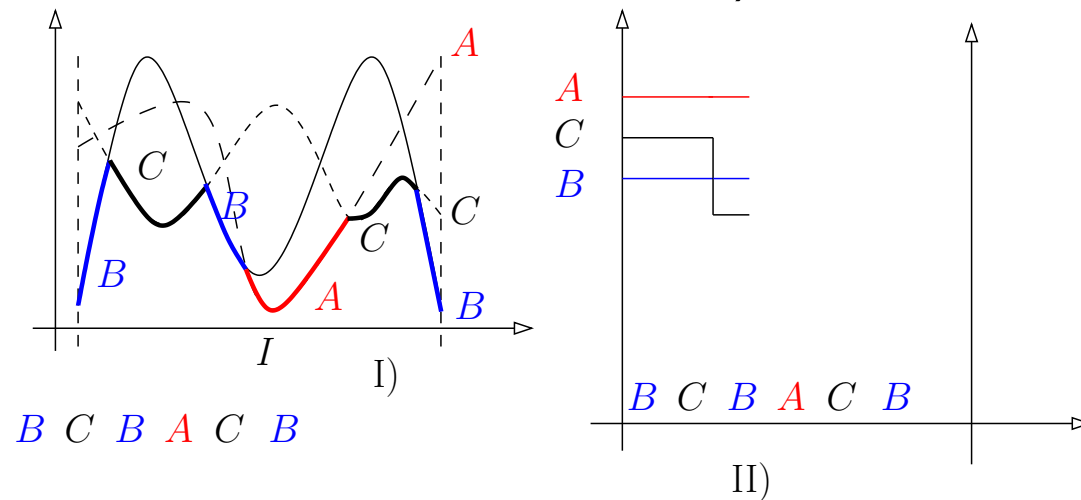
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

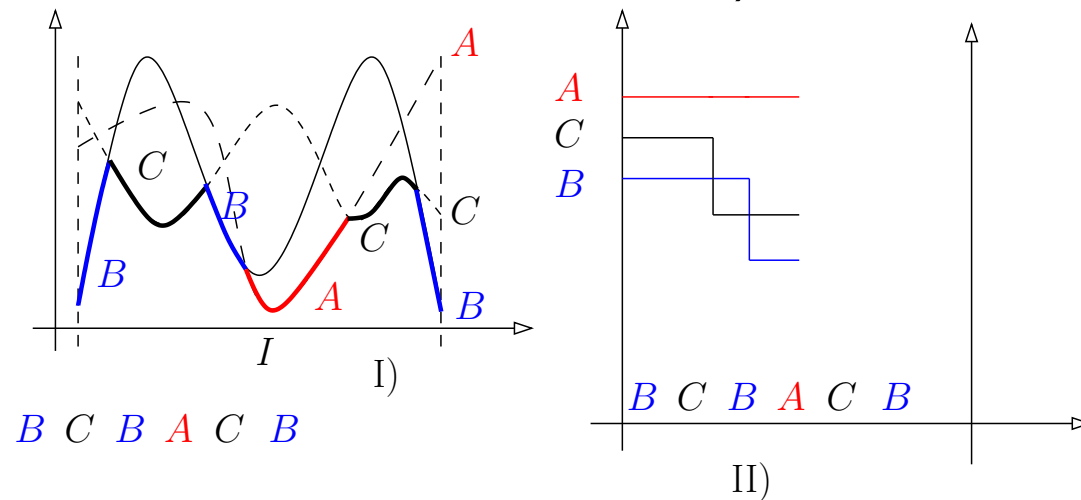
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

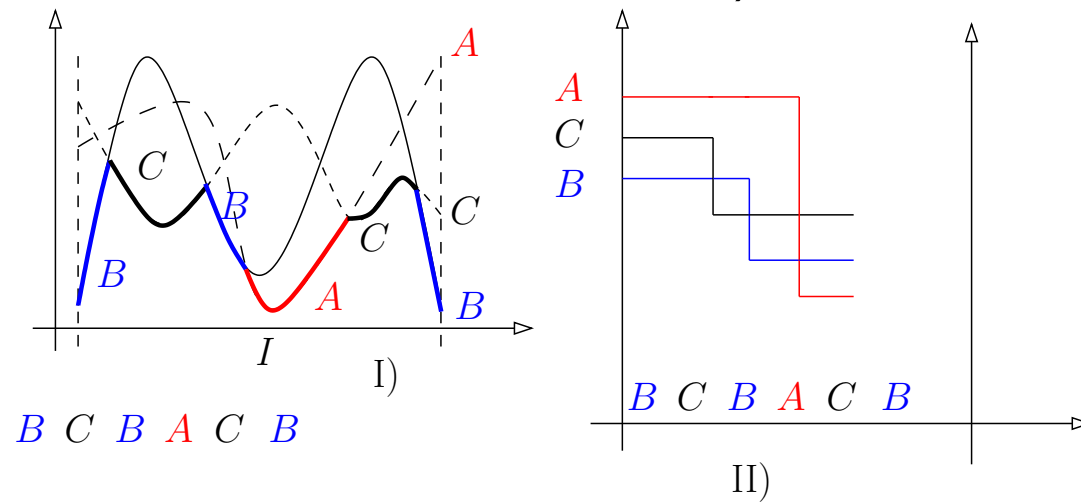
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

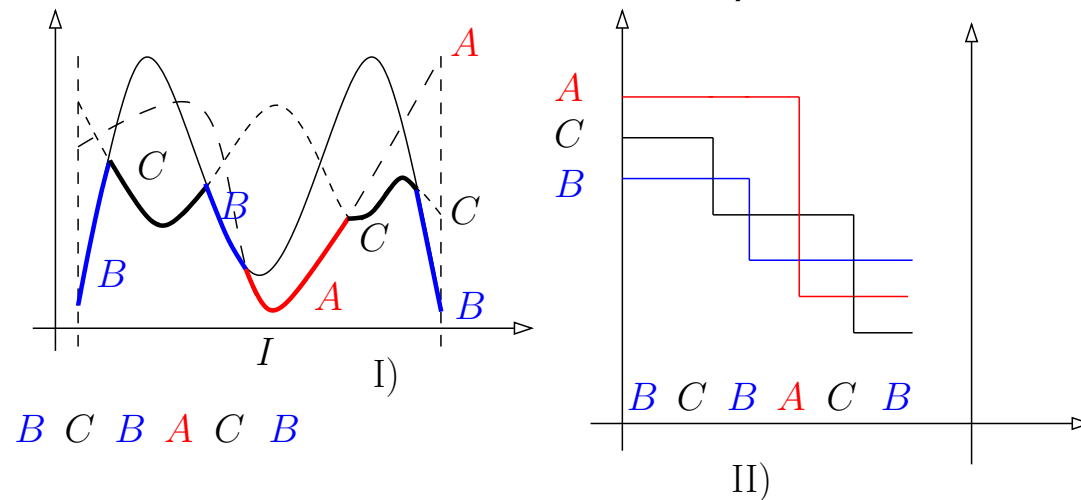
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

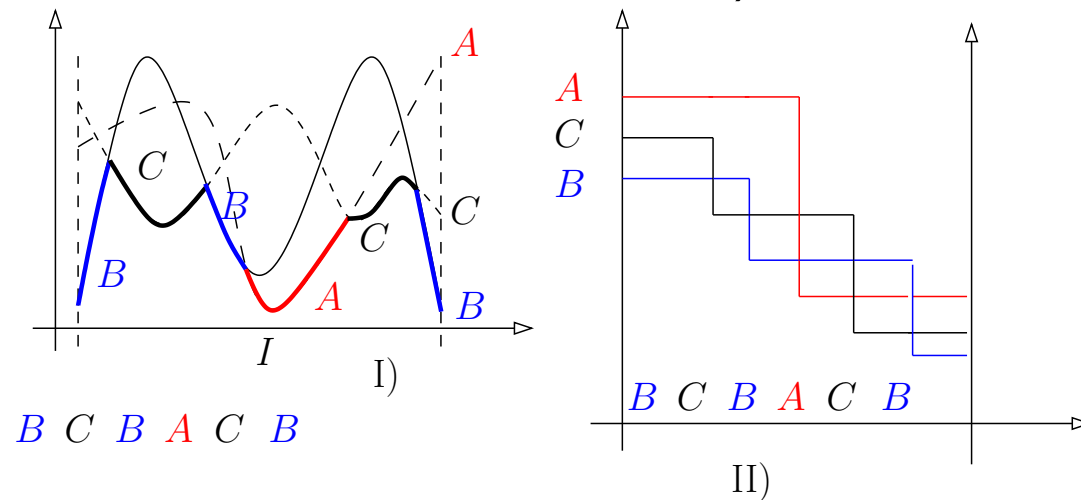
1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen

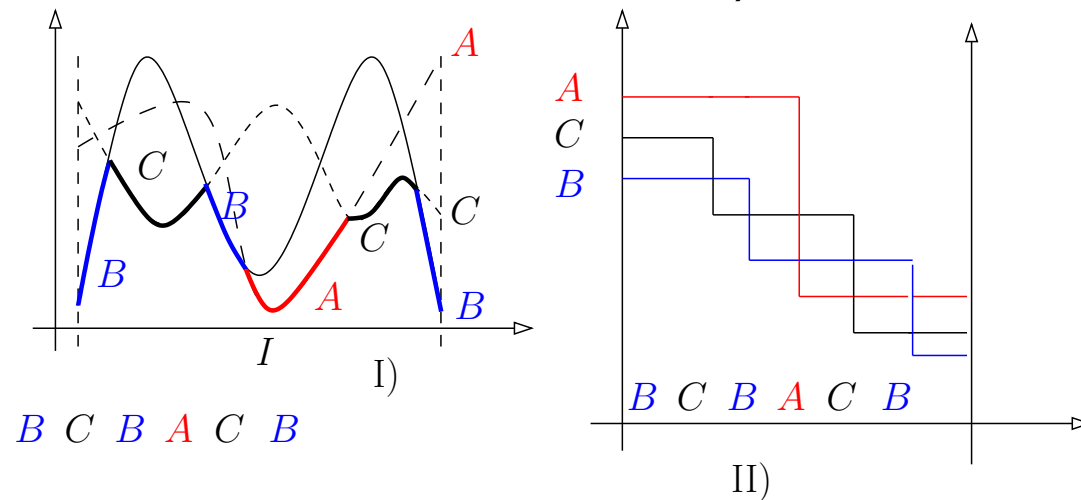




# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

1.: I) Übersetzen in Buchst.-folge oder II) Buchst.-folge übersetzen



# Beweis: Th. A12

## Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

## Beweis: Th. A12

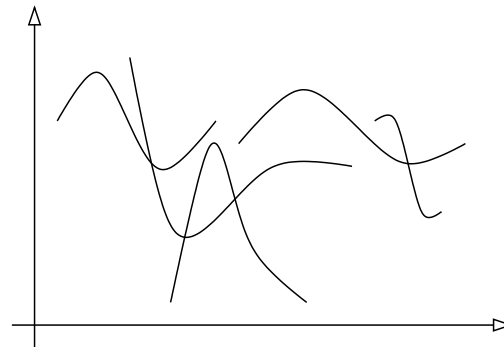
1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

2. Verlängern auf gesamtes Intervall, dann 1. verwenden

## Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

2. Verlängern auf gesamtes Intervall, dann 1. verwenden

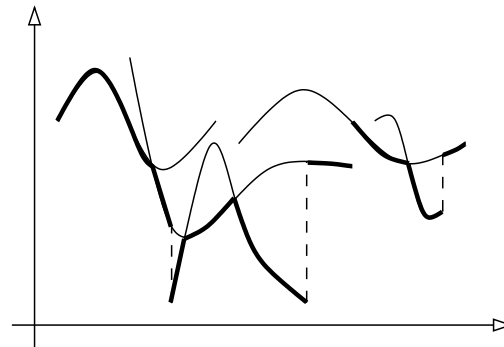


(2)

## Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

2. Verlängern auf gesamtes Intervall, dann 1. verwenden

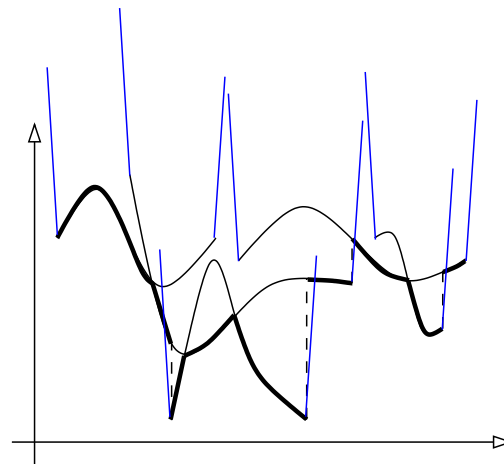


(2)

# Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

2. Verlängern auf gesamtes Intervall, dann 1. verwenden



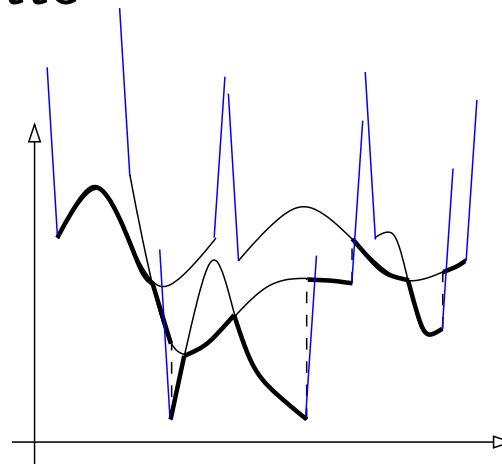
(2)

## Beweis: Th. A12

1.  $\lambda_s(n)$
2.  $\lambda_{s+2}(n)$

2. Verlängern auf gesamtes Intervall, dann 1. verwenden

Max zwei zusätzliche Schnitte



(2)



# Zellen: **Th. 2.18**

## Zellen: Th. 2.18

$A$  Arrangement von  $n$  Kurvenstücken von denen sich zwei nur  $s$  mal schneiden. Jede Zelle von  $A$  hat Komplexität  $O(\lambda_{s+2}(n))$ .

## Zellen: Th. 2.18

$A$  Arrangement von  $n$  Kurvenstücken von denen sich zwei nur  $s$  mal schneiden. Jede Zelle von  $A$  hat Komplexität  $O(\lambda_{s+2}(n))$ .

Weniger als  $\Omega(n^2)$ !!

## Zellen: Th. 2.18

$A$  Arrangement von  $n$  Kurvenstücken von denen sich zwei nur  $s$  mal schneiden. Jede Zelle von  $A$  hat Komplexität  $O(\lambda_{s+2}(n))$ .

Weniger als  $\Omega(n^2)$ !! Beweis!!