

# Praktikum Teil 3 - Aufgabe 1 (1)

- ≡ Vervollständigen Sie den in der Vorlesung entwickelten allgemeinen Suchalgorithmus um die anwendungsspezifischen Bestandteile für das nachfolgend beschriebene Planungsproblem.
- ≡ Hinweis:
  - ≡ Es ist günstiger, zur Entwicklung des Programms zunächst den Block 4 zu entfernen, weil sich der Programmablauf für die leichtere Aufgabenstellung mit nur 3 Blöcken leichter nachvollziehen lässt.

# Praktikum Teil 3 - Aufgabe 1 (2)

☰ Startzustand:

```
[block(block1) ,  
  block(block2) ,  
  block(block3) ,  
  block(block4) ,      %mit Block4  
  on(table,block2) ,  
  on(table,block3) ,  
  on(block2,block1) ,  
  on(table,block4) , %mit Block4  
  clear(block1) ,  
  clear(block3) ,  
  clear(block4) ,      %mit Block4  
  handempty]
```

# Praktikum Teil 3 - Aufgabe 1 (3)

≡ Zielzustand:

```
[block(block1) ,  
  block(block2) ,  
  block(block3) ,  
  block(block4) ,      %mit Block4  
  on(block4,block2) ,  %mit Block4  
  on(table,block3) ,  
  on(table,block1) ,  
  on(block1,block4) ,  %mit Block4  
  %on(block1,block2) , %ohne Block4  
  clear(block3) ,  
  clear(block2) ,  
  handempty]
```

# Praktikum Teil 3 - Aufgabe 2

- Realisieren Sie nun eine informierte Suche. Führen Sie hierzu eine Bewertungsfunktion für Pfade bzw. Zustände ein. (Hierzu muss "eval\_path" definiert werden. Da nur das erste Element des Pfades zur Bewertung herangezogen werden muss, bietet es sich an, ein weiteres Prädikat "eval\_state" zu definieren. Schauen Sie sich in diesem Zusammenhang auch den Code für die informierte Suche an.)
- Formulieren Sie verschiedene Heuristiken und testen Sie diese auch mit komplexeren Aufgabenstellungen (Komplexere Start- und Zielzustände, also viele Blöcke. In der Zielbeschreibung sind nicht alle Blöcke enthalten, weil es für einige egal ist, wo sie stehen). Es soll eine komplexe Heuristik dabei sein, die nicht einfach nur die Länge einer Liste zählt.
- Realisieren Sie A\* (und begründen Sie dabei, ob Ihre Heuristik zulässig ist), gierige Bestensuche, optimistisches Bergsteigen und Bergsteigen mit Backtracking.**
- Implementieren Sie auch die iterative Tiefensuche. Dazu sind etwas größere Anpassungen nötig.**
- Vergleichen Sie die verschiedenen Suchverfahren mittels des "time"-Prädikats, das Ihnen auch die Anzahl der benötigten Inferenzen liefert. (Aufruf: `time(<Anfrage>)` . Für eine bessere Vergleichbarkeit sollten die write-Anweisungen, die das Tracing betreffen, herausgenommen werden.

# Praktikum Teil 3 - Hinweise

- ≡ Ein Knoten des Suchbaums kann in folgender Weise beschrieben werden:

## (Action, State, Value)

- ≡ **Action**: Die Aktion, die vom Vorgänger-Zustand zum beschriebenen Zustand geführt hat.
  - ≡ **State**: Die Beschreibung eines Zustands (siehe Start-Zustand oder Ziel-Zustand)
  - ≡ **Value**: Der Wert der Kostenfunktion zur Umsetzung einer informierten Suche
- ≡ Alle Module des Programms werden in "**Suche.pl**" aufgerufen. Sie müssen also vor dem Start **Suche.pl** laden.

# Praktikum Teil 3 – Alternative Aufgabe

- Alternativ können Sie auch einen allgemeinen Suchalgorithmus in einer Programmiersprache Ihrer Wahl erstellen. Orientieren Sie sich dabei am Pseudocode aus den Vorlesungsfolien. Die Anforderungen bezüglich der Blöcke, Heuristiken und untersuchten Suchverfahren gelten in gleicher Weise wie für die Umsetzung in Prolog.