GWV – Grundlagen der Wissensverarbeitung
Tutorial 2 : State Spaces

Enrico Milutzki (6671784), Love Kumar (7195374),
Nikolai Poets (6911432), Jan Willruth (6768273)

Exercise 2.3 : (State Space Construction 2)

In einem State muss definiert sein:

- Das gesamte Brett
- Liste aller möglichen Übergänge auf dem Brett (als Liste von Tupeln)
- Position Mister X
- Positionen der Detektive
- Verbleibende Schritte A
- Verbleibende Schritte B

Start State:

<Feld-2d-array, List<Tupel<Startfeld, Transportmittel, Endfeld>>, Startposition Mr. X, Startposition Detektiv 1, Startposition Detektiv 2, ..., Startposition Detektiv n, A, B>

End State:

<Feld-2d-array, List<Tupel<Startfeld, Transportmittel, Endfeld>>, Endposition Mr. X, Endposition Detektiv 1, Endposition Detektiv 2, ..., Endposition Detektiv n, 0, 0>

Übergänge:

Start State
→ (1. Bewegung von Mr. X) →
<Feld-2d-array, List<Tupel<Startfeld, Transportmittel, Endfeld>>, 2. Position Mr. X, Startposition Detektiv 1, Startposition Detektiv 2, ..., Startposition Detektiv n, A-1, B>
→ (m Bewegungen von Mr. X) →

<Feld-2d-array, List<Tupel<Startfeld, Transportmittel, Endfeld>>, ~~End~~position Mr. X, Startposition Detektiv 1, Startposition Detektiv 2, ..., Startposition Detektiv n, A-m, B> **solange 0 ≤ A**
→ (1. Bewegung der Detektive) →
<Feld-2d-array, List<Tupel<Startfeld, Transportmittel, Endfeld>>, Endposition Mr. X, 2. Position Detektiv 1, 2. Position Detektiv 2, ..., 2. Position Detektiv n, A-m (0), B - 1>
→ (n Bewegungen der Detektive) **solange 0 ≤ B**→ Endtuple

Transition:
Mister x: <Startposition, Transportmittel, Endposition>
Detektive 1 bis n: <Startposition, Transportmittel, Endposition>

Lösungsweg:
Als Mr. X simuliert man jeweils alle möglichen Schritte von sich selbst und von den Detektiven hintereinander. Dabei markiert man alle möglichen Felder (Mr. X = x, Detektive = d), wobei die Markierung der Detektive die von Mr. X überschreibt. Anschließend bewegt Mr. X sich zu einer möglichen Position, die sich nicht in den mit d markierten Bereichen befindet und mit x markiert ist. Falls kein mit x markiertes Feld übrig bleibt, gibt es keine Lösung.

Exercise 2.4 : (State Space Construction 3)

Furniture Placement:

State definition:

- Der Raum als 2D-Array
- Liste der Gegenstände als Tupel mit Arrayposition[x, y] (-1 wenn nicht platziert, Länge, Breite)
- Radius der Tür wird mit x markiert
- Tisch wird mit t markiert, sodass man die Stühle am Tisch platzieren kann; Tisch muss zuerst platziert werden

<Feld-2D-Array, List<Tupel<Position (-1 wenn nicht platziert, sonst Array Index), Länge x, Breite y>>>

Start state:

<Feld-2D-Array, List<Tupel<Gegenstand n, -1, x, y>>>

End state:

Wenn alle Gegenstände im Raum platziert sind, die Tür frei ist und die Gegenstände sich nicht gegenseitig behindern.

Lösungsweg:
Zunächst wird der Tisch mit mindestens einem freien Platz in alle Richtungen (außer diagonal) platziert, anschließend werden alle Stühle um den Tisch platziert. Danach können alle restlichen Objekte hintereinander platziert werden, wobei jeweils das Größte als nächstes platziert wird.

Construction Site Planning:

Hausbau in einzelne Einheiten unterteilen und der Reihenfolge (Anfang bis Ende) nach ordnen und Arbeiter anhand dessen einteilen.

State Definition

- Liste aus Tupeln mit allen nötigen Aktionen <Aktion, Startpunkt, Dauer, Handwerker, Vorbedingungen>
- Zeitpunkt
- Liste der bearbeiteten Aktionen <Aktion, Startpunkt, Dauer, Handwerker, Vorbedingung>

Lösungsweg:
-Breitensuche:
1.) Finde alle Aktionen, für welche die Vorbedingungen erfüllt sind.
2.) Dann Lege Sie in eine Liste mit Startpunkt t_0.
3.) Dann davon ausgehend, dass die Aktionen erfüllt sind. Finde alle Aktionen, für welche die Vorbedingungen erfüllt sind.
4.) Dann lege sie in die Liste mit Startpunkt als: Startpunkt der letzten Iteration + deren längste Dauer.
5.) wiederhole 3.) bis die Liste der nötigen Aktionen leer ist.

→ Dann führe die Liste der Aktionen aus. Starte die Aktionen, die den Zeitpunkt t als Startpunkt haben.

Elevator:

State Definition:

- EtagenArray mit Boolean-Werten: true= knopf gedrückt
- Etagenziel(e) als Liste von Integers
- Aktuelle Etage
- Fahrtrichtung 0 oder 1 (1= aufwärts)

<EtagenArray, ZielList, Aktuelle Etage>

Start-State:
<EtagenArray, null, 0>

End-State:
Dort, wo er kaputt geht.

Lösungsweg:
Wenn der Aufzug leer ist und ein oder mehrere Knöpfe gedrückt sind, fährt der Aufzug dorthin wo der Knopf von außen als erstes betätigt wurde. Wenn der Aufzug nicht leer ist, fährt er in die Richtung, in die er sich vorher bewegt hat solange ein sich im Aufzug befindlicher Fahrgast in eine Etage auf dem Weg möchte oder ein Fahrgast von außen gedrückt hat. Falls er sich vorher nicht bewegt hat, fährt er in Richtung der vom Fahrgast gedrückten Etage. Ansonsten fährt der Aufzug in die Richtung des zuerst gedrückten Knopfs einer Etage. Falls kein Knopf gedrückt ist, bleibt der Fahrstuhl stehen.