Künstliche Intelligenz

Übungsblatt #3
STRIPS
Version 1.1

Prof. Dr. J. Fürnkranz, Dr. G. Grieser

Aufgabe 3.1

Betrachten Sie folgende Situation: In einem Labor versuchen Wissenschaftler das Verhalten von Affen zu studieren. Die Affen sind hungrig und sollen versuchen, an die an der Decke hängenden Bananen heranzukommen. Dies gelingt ihnen jedoch nur, wenn sie sich auf eine Kiste direkt unter eine Banane stellen. Die Kisten sind so leicht, daß die Affen sie herumschieben können. Außerdem liegt noch Spielzeug herum, das die Affen durch die Gegend schmeißen können.

Wir vereinfachen die Aufgabe wie folgt: es gibt nur eine endliche Anzahl fest vorgegebener Positionen im Käfig. An jeder Position können sich beliebig viele Affen, Kisten und Spielzeuge befinden. Auf einer Kiste können ebenfalls beliebig viele Affen stehen. Kisten können nicht übereinandergestapelt werden. Wenn gleichzeitig mehrere Affen nach einer Banane greifen dann bekommen alle ein Stückchen - und jeder ist hinterher satt. Spielzeug landet immer auf dem Boden.

Ihre Aufgabe ist nun, den Affen zu Bananen zu verhelfen, indem Sie Pläne entwickeln, wie die Affen an die Bananen herankommen.

a) Überlegen Sie zunächst, welche Literale Sie zur Beschreibung dieser Welt benötigen.

b) Modellieren Sie mittels dieser Literale die folgende Situation: Der Käfig hat 3 Positionen A, B und C. Im Käfig sind:

   - Ein Affe an Position A.
   - Eine Banane an Position B.
   - Eine Kiste an Position C.
   - In der Position A, B, C liegt ein Spielzeug.

Lösungsvorschlag:

```
at_a(a,a)       % Affe in Position A
at_b(b,b)       % Banane in Position B
at_k(k,c)       % Kiste in Position C
at_s(s_1,a)     % Spielzeug in Position A
at_s(s_2,b)     % Spielzeug in Position B
at_s(s_3,c)     % Spielzeug in Position C
hungrig(a)      % Der Affe ist hungrig
auf_boden(a)    % Der Affe ist auf dem Boden
```

c) Als nächstes modellieren Sie die folgenden Aktionen, die von den Affen ausgeführt werden können:

`go` : Ein Affe bewegt sich zu der angebenen Position

Lösungsvorschlag:

action:	go(A,P)	
	preconditions:	at_a(A,Q), auf_boden(A)
	add:	at_a(A,P)
	delete:	at_a(A,Q)

`push` : Ein Affe kann eine Kiste zu der angebenen Position schieben, wenn sich Affe und Kiste an der gleichen Stelle befinden.

Lösungsvorschlag:

action:	push(A,K,P)	
	preconditions:	at_a(A,Q), at_k(K,Q), auf_boden(A)
	add:	at_a(A,P), at_k(K,P)
	delete:	at_a(A,Q), at_k(K,Q)

`throw` : Ein Affe kann ein Spielzeug herumwerfen, das sich auf seiner Position befindet.

Lösungsvorschlag:

action:	throw(A,S,P)	
	preconditions:	at_a(A,Q), at_s(S,Q), auf_boden(A)
	add:	at_s(S,P)
	delete:	at_s(S,Q)

`up` : Ein Affe kann auf eine Kiste klettern, wenn er auf dem Fußboden auf gleicher Position wie die Kiste ist.

Lösungsvorschlag:

action:	up(A)	
	preconditions:	at_a(A,P), at_k(K,P), auf_boden(A)
	add:	auf_kiste(A)
	delete:	auf_boden(A)

`down` : Der Affe klettert von der Kiste herunter.

Lösungsvorschlag:

action:	down(A)	
	preconditions:	auf_kiste(A)
	add:	auf_boden(A)
	delete:	auf_kiste(A)

`eat` : Der Affe ißt die Banane, wenn er auf einer Kiste direkt unter ihr steht. Danach ist der Affe satt und die Banane verschwunden.

Lösungsvorschlag:

action:	eat(A,B)	
	preconditions:	at_a(A,P), at_b(B,P), auf_kiste(A)
	add:	satt(A)
	delete:	hungrig(A), at_b(B,P)

d) Überlegen Sie sich mittels einer Ihrer Lieblingsmethoden[1] einen Plan, wie der Affe in der unter b) beschriebenen Situation satt wird und schreiben sie ihn formal auf.

Lösungsvorschlag:

Die Formulierung des Ziels lautet: satt(a)

Der kürzeste Plan hierfür ist:
go(a,c)
push(a,k,b)
up(a,k)
eat(a)

Arbeiten Sie diesen Plan Schritt für Schritt ab, d.h. geben Sie jeweils die geltenden Fakten vor bzw. nach Abarbeiten einer Aktion an.

Lösungsvorschlag:

Fakten am Anfang: at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), hungrig(a), auf_boden(a)

Aktion: go(a,c)

neue Fakten: at_a(a,c), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), hungrig(a), auf_boden(a)

Aktion: push(a,k,b)

neue Fakten: at_a(a,b), at_b(b,b), at_k(k,b), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), hungrig(a), auf_boden(a)

Aktion: up(a,k)

neue Fakten: at_a(a,b), at_b(b,b), at_k(k,b), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), hungrig(a), auf_kiste(a)

Aktion: eat(a)

neue Fakten: at_a(a,b), at_k(k,b), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), satt(a), auf_kiste(a)

[1] Da wären z.B. die Methode des vollständigen Draufschauens und die Methode des unbekümmerten Probierens. Sehr beliebt ist auch die Methode des Nachbarfragens.

e) Suchen Sie mittels Vorwärtsplanung (*Progression*) einen Plan.

Lösungsvorschlag:

In der Anfangssituation $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a) können folgende Aktionen angewendet werden, die jeweils die angeführten Faktenmengen nach sich ziehen:

1) go(a,a):
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
2) go(a,b):
   $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
3) go(a,c):
   $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
4) $\texttt{throw(s}_1\texttt{,a)}$:
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
5) $\texttt{throw(s}_1\texttt{,b)}$:
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
6) $\texttt{throw(s}_1\texttt{,b)}$:
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,c)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)

Die Knoten 1 und 4 werden nicht weiter verfolgt, da sie bereits expandiert worden sind (d.h. auf der *Closed-List* stehen).

Knoten 2 kann wie folgt expandiert werden:

2.1) go(a,a):
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
2.2) go(a,b):
   $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
2.3) go(a,c):
   $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)
2.4) $\texttt{throw(s}_2\texttt{,a)}$:
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)

2.5) throw(s_2,b):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

2.6) throw(s_2,b):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,c), at_s(s_3,c),
hungrig(a), auf_boden(a)

Auch hier werden die Knoten 2.1, 2.2, 2.3 und 2.5 sofort geschlossen.

Knoten 3 kann wie folgt expandiert werden:

3.1) go(a,a):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.2) go(a,b):
at_a(a,b), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.3) go(a,c):
at_a(a,c), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.4) push(a,k,a):
at_a(a,a), at_b(b,b), at_k(k,a), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.5) push(a,k,b):
at_a(a,b), at_b(b,b), at_k(k,b), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.6) push(a,k,c):
at_a(a,c), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.7) throw(s_3,a):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,a),
hungrig(a), auf_boden(a)

3.8) throw(s_3,b):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,b),
hungrig(a), auf_boden(a)

3.9) throw(s_3,b):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_boden(a)

3.10) up(a):
at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c),
hungrig(a), auf_kiste(a)

Auch hier werden die Knoten 3.1, 3.2, 3.3, 3.6 und 3.9 sofort geschlossen.

---

usw. Wir verzichten darauf, den Baum vollständig anzugeben, da das Prinzip klargeworden sein sollte.

Wir expandieren als nächstes beispielhaft den Knoten 3.5. (Ausgangsfakten $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)).

3.5.1) go(a,a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)

3.5.2) go(a,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a) → closed

3.5.3) go(a,c):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)

3.5.4) push(a,k,a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,a)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a) → closed

3.5.5) push(a,k,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a) → closed

3.5.6) push(a,k,c):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a) → closed

3.5.7) throw($\texttt{s}_2$,a):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)

3.5.8) throw($\texttt{s}_2$,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a) → closed

3.5.9) throw($\texttt{s}_2$,c):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,c)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_boden(a)

3.5.10) up:
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_s\texttt{(s}_1\texttt{,a)}$, $\texttt{at}_s\texttt{(s}_2\texttt{,b)}$, $\texttt{at}_s\texttt{(s}_3\texttt{,c)}$, hungrig(a), auf_kiste(a) → closed

---

Expansion von 3.5.10 bspw. ergibt

3.5.10.1) down:
at_a(a,b), at_b(b,b), at_k(k,b), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), hungrig(a), auf_boden(a) → closed

3.5.10.2) eat:
at_a(a,b), at_k(k,b), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), satt(a), auf_kiste(a)

Zustand 3.5.10.2 ist ein Endzustand, da lle Zielfakten (hier: satt(a)) enthalten sind.

Der gefundene Plan entspricht dem Pfad zum Knoten 3.5.10.2.

f) Suchen Sie mittels Rückwärtsplanung (*Regression*) einen Plan.

Lösungsvorschlag:

Diesmal beginnen wir rückwärts, d.h. unsere Ausgangssituation ist die Zielliste `satt(a)`.

Wir müssen diesmal nur *relevante* Aktionen betrachten, d.h. Aktionen, die mind. ein Add-Element in der Zielliste wahr machen.

Hier ist nun lediglich die Aktion `eat(a,b)` relevant, da die Addliste `satt(a)` lautet. Die Vorbedingungen sind $\texttt{at}_a\texttt{(a,P)}$, $\texttt{at}_b\texttt{(b,P)}$, `auf_kiste(a)`. Da die Variable `P` die Werte `a,b,c` annehmen kann, ergeben sich für ein und dieselbe Aktion mehrere Nachfolgezustände:

1) `eat(a,b)`:
   $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,a)}$, `auf_kiste(a)`
2) `eat(a,b)`:
   $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, `auf_kiste(a)`
3) `eat(a,b)`:
   $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,c)}$, `auf_kiste(a)`

Anmerkung 1: Wir wissen, daß die Zustände 1 und 3 sinnlos sind, da die Banane niemals in die Positionen `a` oder `c` gelangen kann. Dieses Wissen läßt sich auch beim Planen gewinnen, indem bspw. Erreichbarkeitsgraphen aufgebaut werden. Reale Planungssysteme führen solche Optimierungen auch durch (und müssen dies sogar, um halbwegs effizient zu sein), in unserem Basisansatz jedoch müssen alle 3 Zustände betrachtet werden.

Anmerkung 2: Für diejenigen, die wissen, was damit gemeint ist, die anderen ignorieren dies bitte: Es ist eigentlich nicht nötig, die Variable P bereits hier zu instantiieren. Effizienter ist, nur einen Nachfolgeknoten mit variabler Position zu erzeugen. Dies erfordert jedoch in späteren Schritten, verschiedene Variablen miteinander zu unifizieren.

Betrachten wir nun den Zustand 1: $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,a)}$, auf_kiste(a). Die Aktionen go(a,a) und push(a,k,a) machen als als Nachbedingung das Literal $\texttt{at}_a\texttt{(a,a)}$ wahr. Die Aktion up(a) liefert auf_kiste(a).

Betrachten wir nun die Aktion go(a,a). Als Ausgangssituation haben wir $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,a)}$, auf_kiste(a). Wenn wir eine Aktion rückwärts anwenden wollen, so heißt das, daß wir die Nachbedingungen (genauer: die Elemente der Add-Liste) in unserem Ziel durch die Vorbedingungen ersetzen. Dies führt nun zu der (aber nur aus unserer Meta-Sicht) paradoxen Situation, daß wir gleichzeit sowohl das Literal auf_boden(a) (steht in der Vorbedingung der Aktion und wird deshalb hinzugefügt) als auch auf_kiste(a) (war bereits in der Zielliste) in unserer Zielliste haben! Ohne Metawissen, daß beide Literale nicht gleichzeitig wahr werden können, wird der Planer auch diese Wege versuchen.

Somit sind folgende Aktionen möglich:

1.1) go(a,a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,a)}$, auf_kiste(a), auf_boden(a)

1.2) go(a,a):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,a)}$, auf_kiste(a), auf_boden(a)

1.3) go(a,a):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,a)}$, auf_kiste(a), auf_boden(a)

1.4) push(a,k,a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,a)}$, $\texttt{at}_k\texttt{(k,a)}$, auf_kiste(a), auf_boden(a)

1.5) push(a,k,a):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,a)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_kiste(a), auf_boden(a)

1.6) push(a,k,a):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,a)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_kiste(a), auf_boden(a)

1.7) up(a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,a)}$, $\texttt{at}_k\texttt{(k,a)}$, auf_boden(a)

1.8) up(a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,a)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a)

1.9) up(a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,a)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a)

Anmerkung 3: In 1.8 und auch in 1.9 haben wir eine analoge Situation zu Anmerkung 1, nur diesmal bzgl. des Prädikats $\texttt{at}_a$.

Als nächstes expandieren wir Zustand 2: $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, auf_kiste(a)

2.1) go(a,a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, auf_kiste(a), auf_boden(a)

2.2) go(a,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, auf_kiste(a), auf_boden(a)

2.3) go(a,b):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, auf_kiste(a), auf_boden(a)

2.4) push(a,k,b):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,a)}$, auf_kiste(a), auf_boden(a)

2.5) push(a,k,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_kiste(a), auf_boden(a)

2.6) push(a,k,b):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_kiste(a), auf_boden(a)

2.7) up(a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,a)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a)

2.8) up(a):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a)

2.9) up(a):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a)

Anmerkung 4: Die Zielliste nach 2.5 ist eine Obermenge von 2.8, deshalb braucht eigentlich nur 2.5 betrachtet werden. Dies wird jedoch von den meisten Suchalgorithmen nicht bemerkt werden.

Und schließlich Zustand 3: $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,c)}$, auf_kiste(a)

3.1) go(a,c):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,c)}$, auf_kiste(a), auf_boden(a)

3.2) go(a,c):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,c)}$, auf_kiste(a), auf_boden(a)

3.3) go(a,c):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,c)}$, auf_kiste(a), auf_boden(a)

3.4) push(a,k,c):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,c)}$, $\texttt{at}_k\texttt{(k,a)}$, auf_kiste(a), auf_boden(a)

3.5) push(a,k,c):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,c)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_kiste(a), auf_boden(a)

3.6) push(a,k,c):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,c)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_kiste(a), auf_boden(a)

3.7) up(a):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,c)}$, $\texttt{at}_k\texttt{(k,a)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a)

3.8) up(a):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,c)}$, $\texttt{at}_k\texttt{(k,b)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a)

3.9) up(a):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,c)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a)

Auch hier geben wir wieder nur einen Ausschnitt des Baumes an...

Betrachten wir bspw. die Expansion von Knoten 2.7: $\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a)

2.7.1) go(a,b):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a)

2.7.2) go(a,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a) → closed

2.7.3) go(a,b):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a)

2.7.4) push(a,k,b):
$\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,a)}$, auf_boden(a)

2.7.5) push(a,k,b):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_boden(a) → closed

2.7.6) push(a,k,b):
$\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a)

2.7.7) down(a):
$\texttt{at}_a\texttt{(a,b)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,b)}$, auf_kiste(a)

Anmerkung 5: Bei Anwendung von push(a,k,b) bspw. auf Knoten 2.7.1 bspw. ergibt sich eine analoge Situation zu Anmerkung 1, diesmal jedoch aus einem anderen Grund:

Betrachten wir die Als Ergebnis von push(a,k,b) werden bspw. die Literale $\texttt{at}_a\texttt{(a,b)}$ und $\texttt{at}_k\texttt{(a,b)}$ aus der Zielliste gelöscht ($\texttt{at}_a\texttt{(a,b)}$ ist jedoch gar nicht enthalten) und die Literale $\texttt{at}_a\texttt{(a,c)}$ und $\texttt{at}_k\texttt{(a,c)}$ hinzugefügt (wenn wir von c kommen). Somit ergibt sich als Zielliste $\texttt{at}_a\texttt{(a,a)}$, $\texttt{at}_a\texttt{(a,c)}$, $\texttt{at}_b\texttt{(b,b)}$, $\texttt{at}_k\texttt{(k,c)}$, auf_boden(a).

Und abschließend sehen wir uns die Nachfolger von 2.7.6 an (at_a(a,c), at_b(b,b), at_k(k,c), auf_boden(a)):

2.7.6.1) go(a,c):
at_a(a,a), at_b(b,b), at_k(k,c), auf_boden(a)

2.7.6.2) go(a,c):
at_a(a,b), at_b(b,b), at_k(k,c), auf_boden(a)

2.7.6.3) go(a,c):
at_a(a,c), at_b(b,b), at_k(k,c), auf_boden(a) → closed

2.7.6.4) push(a,k,c):
at_a(a,a), at_b(b,b), at_k(k,a), auf_boden(a)

2.7.6.5) push(a,k,c):
at_a(a,b), at_b(b,b), at_k(k,b), auf_boden(a)

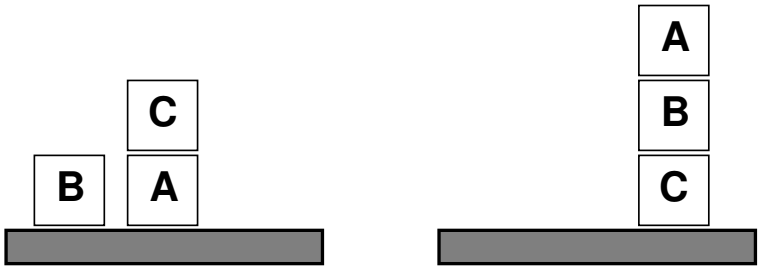
2.7.6.6) push(a,k,c):
at_a(a,c), at_b(b,b), at_k(k,c), auf_boden(a) → closed

2.7.6.7) down(a):
at_a(a,c), at_b(b,b), at_k(k,c), auf_kiste(a)

Die Zielliste nach Zustand 2.7.6.1 (at_a(a,a), at_b(b,b), at_k(k,c), auf_boden(a)) ist Teilmenge unserer Anfangsfakten at_a(a,a), at_b(b,b), at_k(k,c), at_s(s_1,a), at_s(s_2,b), at_s(s_3,c), hungrig(a), auf_boden(a), deshalb ist dieser Zustand ein Endzustand und wir haben einen Plan gefunden.

Aufgabe 3.2

Die folgende Abbildung zeigt ein Blocksworld-Problem, das als *Sussmann-Anomalie* bekannt ist. (Siehe Aufgabe 11.11 in Russel/Norvig).



Start State Goal State

a) Geben Sie eine formale Beschreibung dieser Situation an.

Lösungsvorschlag:

`on(b,table), on(c,a), on(a,table), clear(b), clear(c)`

b) Geben Sie eine formale Beschreibung der Zielsituation an.

Lösungsvorschlag:

`on(a,b), on(b,c)`

c) Geben Sie einen Plan an.

Lösungsvorschlag:

Hierzu benötigen wir zunächst einmal Aktionen. Wir benutzen die in Russel/Norvig angegebenen (Fig. 11.4):

Bewege eine Block B, der auf X liegt, auf ein freies Objekt Y:

```
 action:  move(B,X,Y
          preconditions:  on(B,X), clear(B), clear(Y), block(B),
                          B≠X, B≠Y, X≠Y,
          add:            on(B,Y), clear(X)
          delete:         on(B,X), clear(Y)
```

Bewege eine Block B, der auf X liegt, auf den Tisch:

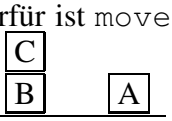
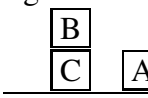
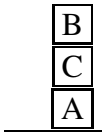
```
 action:  movetotable(B,X
          preconditions:   on(B,X), clear(B), block(B), B≠X
          add:             on(B,table), clear(X)
          delete:          on(B,X)
```

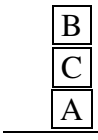
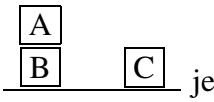
Ein Plan wäre dann bspw.:

`movetotable(c,a), move(b,table,c), move(a,table,b)`

d) Diskutieren Sie, warum *Noninterleaved Planners* hier keine Lösung finden können. *Noninterleaved Planners* bearbeiten eine Konjunktion von Zielen G_1, G_2, indem sie zunächst einen Plan für das Teilziel G_1 und danach einen Plan für G_2 (oder andersherum) suchen. Welche Pläne würden beispielsweise generiert?

Lösungsvorschlag:

Betrachten wir als erstes einen Planer, der zunächst das Teilziel `on(a,b)` bearbeitet. Der kürzeste hierfür ist `movetotable(c,a), move(a,table,b)`. Als Zwischenstand ergibt sich  Nun wird das Ziel `on(b,c)` bearbeitet und es ergibt sich der Plan `movetotable(c,b), move(b,table,c)` mit dem Endstand  . Ein alternativer Plan wäre `move(c,b,a), move(b,table,c)` mit dem Ergebnis  .

Beginnt ein Planer mit dem Teilziel `on(b,c)`, so ist der kürzeste Plan `move(b,table,c)` mit dem Ergebnis  . Das Teilziel `on(a,b)` erhält man mittels `movetotable(b,c), movetotable(c,a), move(a,table,b)`, das Ergebnis  jedoch ist auch nicht wie gewünscht.

In jedem Fall wird also mit der Bearbeitung des zweiten Teilziels das erste erreichte Teilziel wieder zerstört.