Universität Koblenz-Landau
Fachbereich Informatik

Klausur

KI-Programmierung

WS 2007/2008

Jun.-Prof. Dr. B. Beckert

21. Februar 2008

Name, Vorname: _____

Matrikel-Nr.: _____

Studiengang:

- ☐ Informatik (Diplom)
- ☐ Computervisualistik (Diplom)
- ☐ Anderer: _____

A1 (7)	A2 (12)	A3 (10)	A4 (12)	A5 (19)	Σ (60)

Bewertungstabelle bitte frei lassen !!!

Zum Bestehen der Klausur genügen 30 der erreichbaren 60 Punkte.

Wichtiger Hinweis:

Bei Ankreuzaufgaben wird für falsche Kreuze ein Punkt abgezogen!

Dabei werden insgesamt jedoch keinesfalls weniger als 0 Punkte für die jeweilige Teilaufgabe vergeben.

1 Einführung (4 + 3 Punkte)

a) (4 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig

Rationales Verhalten setzt Intelligenz voraus.	richtig ☐ falsch ☒
Rationales Verhalten setzt Lernfähigkeit voraus.	richtig ☐ falsch ☒
Eine Aktion ist rational, wenn sie unter Berücksichtigung der bis dahin gemachten Beobachtungen den zu erwartenden Gewinn (das *performance measure*) maximiert.	richtig ☒ falsch ☐
Suchalgorithmen sind eine der wichtigsten Techniken der KI-Programmierung.	richtig ☒ falsch ☐

b) (3 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

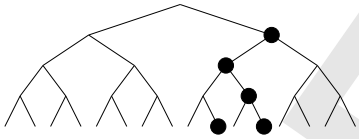
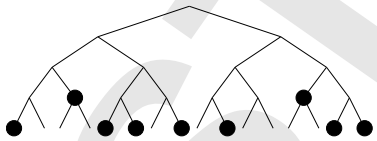
Wenn ein Prologsystem ein Programm ausführt, entspricht dies einer Tiefensuche.	richtig ☒ falsch ☐
Prolog ist eine streng typisierte Sprache.	richtig ☐ falsch ☒
Prolog ist eine funktionale Sprache.	richtig ☐ falsch ☒

2 Suchen (5 + 4 + 3 Punkte)

a) (5 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Der Zeitaufwand einer Suche ist – unabhängig vom verwendeten Suchverfahren – proportional zu der Zahl der Knoten im Suchraum, die besucht werden, bis eine Lösung gefunden ist.	richtig ☒ falsch ☐
Der Zeitaufwand einer Suche ist – unabhängig vom verwendeten Suchverfahren – proportional zu dem für die Suche verwendeten Speicherplatz.	richtig ☐ falsch ☒
Iterative-deepening search ist in praktischen KI-Anwendungen Breiten- und Tiefensuche zumeist überlegen.	richtig ☒ falsch ☐

Für einen Suchraum, wie den hier dargestellten, verwendet man besser (die schwarzen Kreise stellen Ziele dar) . . . 	Tiefensuche ☐ Breitensuche ☒
Für einen Suchraum, wie den hier dargestellten, verwendet man besser (die schwarzen Kreise stellen Ziele dar) . . . 	Tiefensuche ☒ Breitensuche ☐

b) (4 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

Mit α-β-Pruning verdoppelt sich in etwa die Tiefe des Suchraumes, der in bestimmter Zeit durchsucht werden kann.	richtig ☒ falsch ☐
Min-Max-Suche findet mit und ohne α-β-Pruning – bei gleichbleibender Suchtiefe – die gleiche Antwort (d. h. den gleichen optimalen Spielzug).	richtig ☒ falsch ☐
Wie lange es dauert, mit A^*-Suche eine Lösung zu finden, hängt in hohem Maße von der verwendeten Heuristikfunktion ab.	richtig ☒ falsch ☐
Ob die mit A^*-Suche gefundene Lösung optimal ist, hängt von der verwendeten (zulässigen) Heuristikfunktion ab.	richtig ☐ falsch ☒

c) (3 Punkte)

Bei A^*-Suche wird jeweils derjenige Knoten K als nächster expandiert, für den gilt (geben Sie eine präzise Definition):

Es wird derjenige Knoten K als nächster expandiert,

für den die Summe aus den tatsächlichen Pfadkosten vom Startknoten zu K

und den heuristisch geschätzten Kosten von K zum nächstgelegenen Ziel

minimal ist.

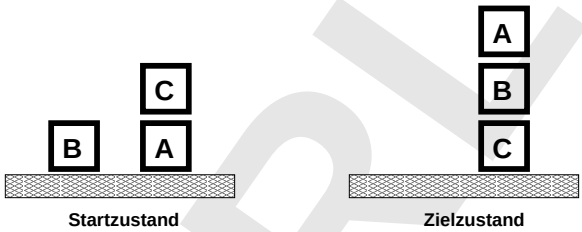
3 Planen (10 Punkte)

Die folgenden beiden STRIPS-Operatoren seien gegeben
(dies sind Standard-Operatoren der *Blocks world*).

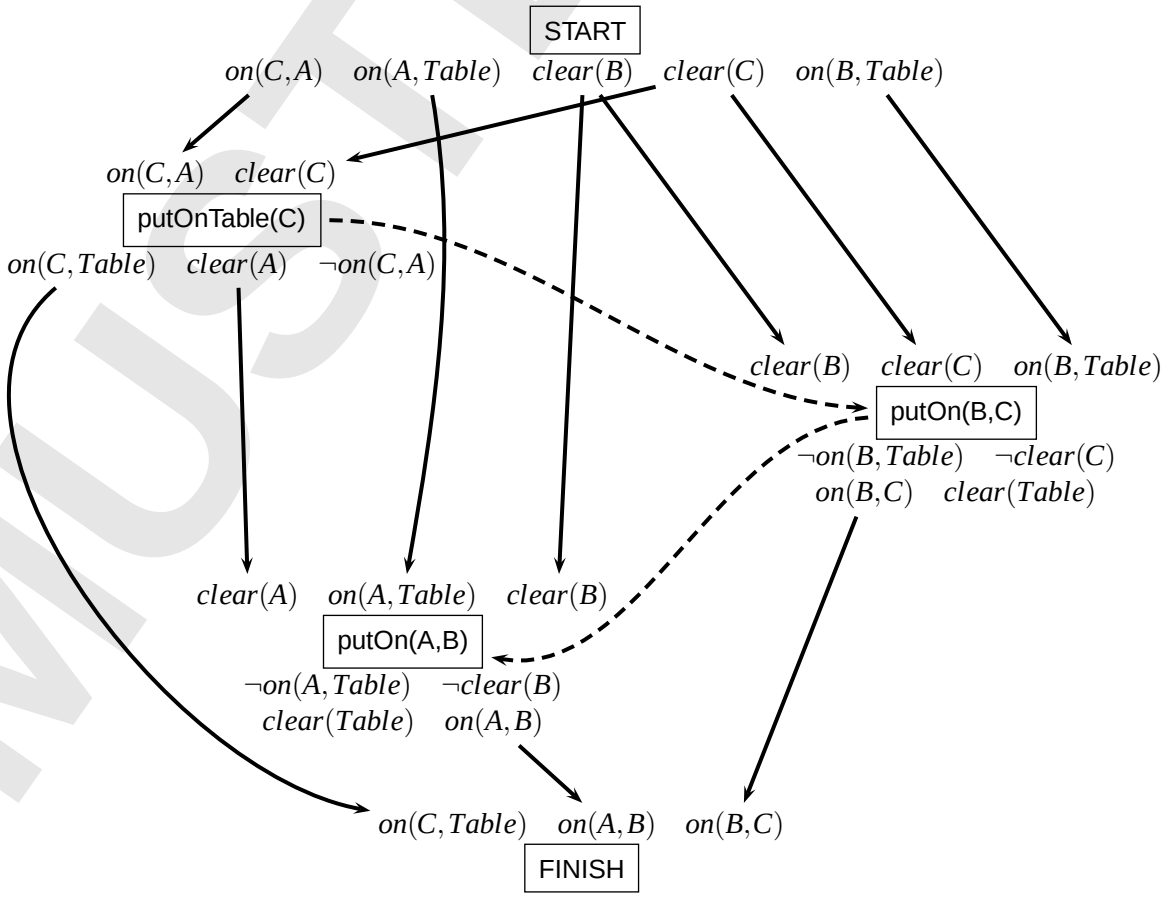
Action:	$putOn(x,y)$
Precond:	$clear(x) \wedge on(x,z) \wedge clear(y)$
Effect:	$on(x,y) \wedge clear(z) \wedge \neg on(x,z) \wedge \neg clear(y)$

Action:	$putOnTable(x)$
Precond:	$clear(x) \wedge on(x,z)$
Effect:	$on(x,Table) \wedge clear(z) \wedge \neg on(x,z)$

Gegeben seien folgende Start- und Zielzustände:



Verwenden Sie den POP-Algorithmus, um den unten dargestellten partiellen Plan zu vervollständigen.
Hinweis:
Zeichnen Sie neben den notwendigen zusätzlichen Aktionen auch deren Vorbedingungen (*preconditions*) und Effekte sowie die kausalen Links und die zur Beseitigung von Clobberings notwendigen Ordnungsconstraints (*ordering constraints*) ein.



4 Prolog I (5 + 4 + 3 Punkte)

a) (5 Punkte)

Entscheiden Sie durch Ankreuzen, ob die folgenden Aussagen richtig oder falsch sind.

In Prolog kann man selbstmodifizierende Programme schreiben.	richtig ☒ falsch ☐
NP-vollständige Probleme (wie bspw. Knapsack) können mit Prolog nicht gelöst werden.	richtig ☐ falsch ☒
Wenn man in Prolog zweimal die gleiche Anfrage stellt, erhält man immer zweimal die gleiche Antwort (d.h., die Berechnung der ersten Antwort kann keinen Seiteneffekt auf die Berechnung der zweiten Antwort haben).	richtig ☐ falsch ☒
Die Standard-Implementierung von `append` ist flexibel.	richtig ☒ falsch ☐
`not(X)` terminiert genau dann, wenn `X` terminiert.	richtig ☒ falsch ☐

b) (2 + 2 Punkte)

i. Welche Antwort liefert Prolog auf folgende Anfrage?

```
?- append(X,Y,[])
```

Lösung:

```
X = [],
Y = []
Yes
```

ii. Welche Antwort liefert Prolog auf folgende Anfrage?

```
?- .(X,Y) = [1,2,3]
```

Lösung:

```
X = 1,
Y = [2,3]
Yes
```

c) (3 Punkte)

Betrachten Sie das folgende, aus drei Klauseln bestehende Prolog-Programm:

```
q(a).
q(b) :- q(b).
p(X) :- not q(X).
```

Welche Antworten liefert Prolog mit diesem Programm auf folgende Anfragen?

Hinweis: Die möglichen Antworten sind `yes`, `no` und „terminiert nicht“.

i. `?- p(a)` liefert: no

ii. `?- p(b)` liefert: terminiert nicht

iii. `?- p(c)` liefert: yes

5 Prolog II (9 + 7 + 3 Punkte)

a) (6 + 3 Punkte)

Betrachten Sie das folgende – *fehlerhafte* – Prolog-Programm, dass die Funktion `length` implementieren, die die Länge einer Liste berechnet:

```
length([],1).
length([H|T],N) :- N1 is N+1, length(T,N1).
```

Dieses Programm enthält drei Fehler.

i. Geben Sie die drei Fehler an:

1. Die Länge der leeren Liste nicht 1 sondern 0.

2. Die Berechnung von `N` muss nach dem rekursiven Aufruf stehen.

3. In der Berechnung mit `is` sind `N` und `N1` vertauscht.

ii. Geben Sie eine korrekte Implementierung von `length` an.

Lösung:

```
length([],0).
length([H|T],N) :- length(T,N1), N is N1+1.
```

b) (7 Punkte)

Geben Sie ein Prädikat

```
list_add(List1,List2,Output)
```

and, dass zwei (gleichlange) Listen von Zahlen elementweise addiert.

Beispiel: Die Anfrage

```
?- list_add( [1,2,3,4], [10,20,30,40], Output)
```

liefert

```
Output = [11,22,33,44]
```

Lösung:

```
list_add([],[],[]).

list_add( [H1|T1], [H2|T2], [Hout|Tout] ) :-
    Hout is H1 + H2,
    list_add(T1,T2,Tout).
```

c) (3 Punkte)

Geben Sie Prolog-Programm an, das auf die Anfrage

```
?- p(t)
```

für keinen Term *t* terminiert.

Hinweis: Eine Klausel genügt.

Lösung:

```
p(X) :- p(X).
```