Prof. Dr. Günther Görz
Lehrstuhl Informatik 8 (Künstliche Intelligenz)
der Universität Erlangen-Nürnberg

27. September 2010

Bachelorprüfung
Grundlagen der Logik und Logikprogrammierung
— September 2010

Angaben zur Person (Bitte in Druckschrift ausfüllen!):

Name, Vorname: .....

Geburtsdatum: .....

Matrikelnummer: .....

Studienfach (Bitte ankreuzen):

☐ Informatik ☐ Wirtschafts-informatik ☐ Linguistische Informatik ☐ Computational Engineering ☐ Mathematik ☐ Techno-mathematik

☐ Sonstiges (Bitte angeben): .....

Nicht von der Kandidatin bzw. vom Kandidaten auszufüllen !!!

Punktzahlen:

Aufgabe	1	2	3	4	5	Σ
Max. Punkte	25	20	10	20	15	90
Erreichte Punkte						

Note: ☐ **Bonuspunkte:** ☐ **Endnote:** ☐

Organisatorische Hinweise

Die folgenden Hinweise bitte aufmerksam lesen und die Kenntnisnahme durch Unterschrift bestätigen!

- Bitte legen Sie Ihren Studentenausweis und einen Lichtbildausweis zur Personenkontrolle bereit!
- Die angegebene Punkteverteilung gilt unter Vorbehalt.
- Hilfsmittel (außer Schreibmaterial) sind **nicht** zugelassen.
- Fragen zu den Prüfungsaufgaben werden grundsätzlich **nicht** beantwortet!
- Wenn Sie die Prüfung aus gesundheitlichen Gründen abbrechen müssen, so muss Ihre Prüfungsunfähigkeit durch eine Untersuchung in der Universitätsklinik nachgewiesen werden. Melden Sie sich in jedem Fall bei der Aufsicht und lassen Sie sich das entsprechende Formular aushändigen.
- Die Bearbeitungszeit beträgt 90 Minuten.
- **Überprüfen Sie die Prüfungsaufgaben auf Vollständigkeit (11 Seiten inklusive Deckblatt) und einwandfreies Druckbild!** Vergessen Sie nicht, auf dem Deckblatt die Angaben zur Person einzutragen!
- Verwenden Sie für die Darstellung der Wahrheitswerte `t` bzw. $\top$ und `f` bzw. $\bot$.

  Ausnahmen werden explizit in der jeweiligen Aufgabenstellung gestattet!

Erklärung

Durch meine Unterschrift bestätige ich den Empfang der vollständigen Klausurunterlagen und die Kenntnisnahme der obigen Informationen.

Erlangen, 27. September 2010 .....
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis unter Angabe der Matrikelnummer anonymisiert veröffentlicht wird:

ja: ☐ nein: ☐

Erlangen, 27. September 2010 .....
(Unterschrift)

Aufgabe 1 (25 Punkte)

Beantworten Sie jeweils in wenigen vollständigen Sätzen, gegebenenfalls unter Zuhilfenahme von Formeln oder Beispielen, die nachfolgenden Fragen.

a) Wann heißt eine Aussage *klassisch erfüllbar*?

b) Wann heißt eine Aussage *klassisch allgemeingültig*?

c) Wann heißt eine Aussage *kontradiktorisch*?

d) Welche syntaktische Eigenschaft zeichnet Horn-Formeln aus?

e) Welche Form besitzen Aussagen in KNF?

f) Welche Form besitzen Aussagen in DNF?

g) Wann heißt eine Regel in einem Kalkül zulässig?

h) Nennen Sie zwei Resolutionsvarianten und erklären Sie den Unterschied zwischen den beiden!

Aufgabe 2 (20 Punkte)

Logik

a) Wodurch unterscheiden sich klassische Logik und effektive Logik?

b) Zeigen Sie die Gültigkeit von *duplex negatio affirmat* $\neg\neg A \rightarrow A$ mithilfe eines klassischen und eines effektiven Dialogs.

Hinweis: Verwenden Sie im effektiven Fall *tertium non datur* als Hypothese.

c) Ist die Aussage $\neg\Big(\big((A \wedge B) \vee \neg C\big) \longrightarrow \big((A \wedge \neg C) \vee B\big)\Big)$ kontradiktorisch? Begründen Sie! Geben Sie, wenn möglich, ein Modell für die Aussage an!

d) Aussagenlogische Resolution: Zeigen Sie mit der Resolutionsmethode, dass

$$(\neg A \wedge \neg B \wedge \neg C) \vee (\neg C \wedge B) \vee (\neg C \wedge A) \vee C$$

eine Tautologie ist.

Aufgabe 3 (10 Punkte)

Sind die folgenden Ausdrücke unifizierbar? Falls ja, geben Sie jeweils einen Unifikator an (Variablen beginnen mit Großbuchstaben, Konstanten mit Kleinbuchstaben)!

a) $f(a(B)) = f(a(b))$

b) $f(a(b)) = f(b(A))$

c) $a(b(C)) = a(C)$

d) $X = [a, b, c]$

e) $[X, b, Z] = [a, Y, c]$

f) $[[a, b], c] = [X, Y]$

g) $[a(b), c(X)] = [Z, c(d)]$

h) $[X|Y] = [a, b, 1, 2]$

i) $[a, b] = [X, Y|Z]$

j) $[X, Y, Z|A] = [[a, b, c], d]$

Aufgabe 4 (20 Punkte)

Schreiben Sie ohne Verwendung der Prolog-Prädikate `reverse/2`, `delete/3` und `append/3` (aber gegebenenfalls unter Verwendung der in den jeweiligen Teilaufgaben zu entwickelnden Programme)

a) ein Prolog-Programm, `rev/2`, welches die Reihenfolge der Elemente einer Liste umkehrt. Beispiel eines Aufrufs:

```
?- rev([1,2,3,4,5], X).
X = [5, 4, 3, 2, 1].
```

b) ein Prolog-Programm, `tausch/2`,welches das erste und letzte Element einer Liste vertauscht. Beispiel eines Aufrufs:

```
?- tausch([1,2,3,4,5], X).
X = [5, 2, 3, 4, 1].
```

c) ein Prolog-Programm, `entf/3`, welches alle Vorkommen eines bestimmten Elementes aus einer Liste entfernt. Beispiel eines Aufrufs:

```
?- entf([a,b,a,c,b,a], a, X).
X = [b, c, b] .
```

d) ein Prolog-Programm, `doppel/2`, welches eine Liste verdoppelt. Beispiel eines Aufrufs:

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

Aufgabe 5 (15 Punkte)

Ein binärer Baum ist ein Graph, in dem jeder Knoten maximal drei Kanten hat: Alle Knoten haben genau eine eingehende Kante und maximal zwei ausgehende Kanten. Eine Ausnahme bildet der Wurzelknoten mit keiner eingehenden Kante.

Beim binären Suchbaum sind die Knoten so geordnet, dass die Werte aller Knoten des linken Teilbaums kleiner sind als der Wert des Wurzelknotens. Dessen Wert ist wiederum kleiner als die Werte aller Knoten des rechten Teilbaums, und diese Eigenschaft gilt rekursiv für alle Knoten des Baumes.

Folgende Prädikate sind vorgegeben:

```
% leerer Baum
tree_empty(nil).

% Konstruktor
tree_create(V,tree(nil,nil,V)).

% Selektoren
tree_getv(tree(_,_,V),V).
tree_getlb(tree(L,_,_),L).
tree_getrb(tree(_,R,_),R).

% Modifikatoren
tree_setv(tree(L,R,_),VN,tree(L,R,VN)).
tree_setlb(tree(_,R,V),LN,tree(LN,R,V)).
tree_setrb(tree(L,_,V),RN,tree(L,RN,V)).

% Typueberpruefungspradikat
tree_is(tree(_,_,_)).
```

a) Schreiben Sie das Prologprogramm `treeheight(T,H)`, mit welchem man die Höhe eines Baumes berechnen kann. Hinweis: Die Höhe eines Baumes *h(T) = 1 + max(h(lb(T)), h(rb(T)))*, wobei *lb(T)* den linken Teilbaum *rb(T)* den rechten Teilbaum des Baumes T bezeichnet.

b) Man kann einen Baum auf verschiedene Weisen durchlaufen. Implementieren Sie die *preorder*-Traversierung (aktueller Knoten, linker Teilbaum, rechter Teilbaum) wobei alle Werte der Knoten im Baum in der gewünschten Reihenfolge mittels `write/1` ausgegeben werden.