Minitest WS 2001/2002 • Musterlösung

Programmiertechniken in der Computerlinguistik I · 7. Februar 2002 · Dauer 45 Min

1. Anfragen und Unifikation (30 Punkte)

Kann der PROLOG-Interpreter die beiden nachfolgenden Anfragen beweisen? Wenn ja, mit welchen Variablenbindungen? *(je 5 Punkte)*

a) `?- d(A,b,C) = d(b(B),C,d).`	❑ Ja	✔ Nein	Bindung(en):
b `?- d(a(a,b)) = d(A,B).`	❑ Ja	✔ Nein	Bindung(en):
c) `?- X is 1 + 2.`	✔ Ja	❑ Nein	Bindung(en): `{X/3}`
d) `?- [a,b,c] = [A,B].`	❑ Ja	✔ Nein	Bindung(en):
e) `?- \+ (fail, X = x, true).`	✔ Ja	❑ Nein	Bindung(en):
f) `?- [[a],b] = [A|B].`	✔ Ja	❑ Nein	Bindung(en): `{A/[a],B/[b]}`

2. Unifikation (20 Punkte)

Unifikation ist in der Computerlinguistik ein wichtiges Konzept. Unter welchen Bedingungen sind 2 beliebige Terme *T* und *U* unifizierbar? (*20 Punkte*)

Für 20 Punkte mussten folgende Dinge erwähnt werden:

- Falls U und T identische atomare Terme sind.
- Falls U oder T eine Variable ist.
- Falls U und T komplexe Terme mit gleichem Hauptfunktor und Stelligkeit desselben sind und zudem die Argumente paarweise unifizierbar sind.

3. Flektieren (aka. Listenverarbeitung) (40 Punkte)

a) Definiere das Prolog-Prädikat `regind3sg/2`, das aus der Infinitivform eines regelmässigen Verbs mit der Endung *-en* die Form für die 3. Person Indikativ Präsens bildet (*30 Punkte*):

```
?- regind3sg([g,e,h,e,n],X).
X = [g,e,h,t] ? ;
no
```

```
regind3sg([e,n], [t]).          % Abbruchbedingung
regind3sg([C|Cs], [C|Ds]) :-    % Rekursionsschritt
       regind3sg(Cs, Ds).
```

b) Definiere das Prädikat `ind3sg/2`, das wie `regind3sg/2` flektiert, aber zusätzlich für "haben" und "sein" nur die korrekten Formen "hat" bzw. "ist" berechnet (*10 Punkte*):

```
?- ind3sg([h,a,b,e,n],X).
X = [h,a,t] ? ;
no

?- ind3sg([g,e,h,e,n],X).
X = [g,e,h,t] ? ;
no
```

```
ind3sg([h,a,b,e,n], [h,a,t]) :-
       !.                       % Red Cut
ind3sg([s,e,i,n], [i,s,t]) :-
       !.                       % Green Cut
ind3sg(Inf, Fin) :-
       regind3sg(Inf, Fin).
```

4. Parsen mit DCG (30 Punkte)

Gegeben sei folgender Ausschnitt aus einer Sprachbeschreibung:

Grammatikregeln		***Lexikonregeln***	
VP → *V*	*NP* → *Det Adj N*	*V* → knurrt	*N* → Pudel | Dackel
Adj → *ε*	*NP* → *NP Conj NP*	*Adj* → wilde	*Det* → der
		Conj → und	

a) Schreibe obige Regeln und die dazugehörigen Lexikonregeln als DCG. Füge dabei ein zusätzliches Argument ein, das am Schluss den Parsebaum als Prolog-Term enthält. (*25 Punkte*)

```
?- phrase(vp(B), [knurrt]).
B = vp(v(knurrt))
```

```
% Syntax

vp(vp(V)) --> v(V).

adj(adj('')) --> [].

np(np(Det,Adj,N)) --> det(Det), adj(Adj), n(N).

np(np(NP1,Conj,NP2)) --> np(NP1), conj(Conj), np(NP2).

% Lexikon

v(v(knurrt)) --> [knurrt].

adj(adj(wilde)) --> [wilde].

conj(conj(und)) --> [und].

n(n('Pudel')) --> ['Pudel'].

n(n('Dackel')) --> ['Dackel'].

det(det(der)) --> [der].
```

b) Was ist an obigen Grammatikregeln für den Standard-DCG-Parser von Prolog problematisch und wieso? (nur Stichworte!) (*5 Punkte*)

```
Die linksrekursive Regel

   np(np(NP1,Conj,NP2)) --> np(NP1), conj(Conj), np(NP2).

kann zur Nicht-Terminierung des Beweisvorgangs führen.
```