Übungsaufgaben zum Proseminar Semantik

Universität München, CIS, WS 2004

H. Leiß

23. Juli 2004

Lösung zu Aufgabe 4.1, Mehrstellige Quantoren: (freiwilliger Teil)

Die verallgemeinerten Quantoren der Umgangssprache sind in unterschiedlichen Kontexten unterschiedlich zu interpretieren. An den Beispielen

(a) *Alle Barbiere rasieren sich selbst.*

(b) *Alle Nachbarn grüßen einander.*

(c) *Alle Menschen sind Brüder.*

sieht man, daß *alle* in den drei Fällen jeweils etwas anderes bewirkt, je nach der Stelligkeit der beteiligten Nomina und Verben: der Quantor verbindet

(a) *Barbier*$^{(1)}$ und *rasieren*$^{(2)}$, das durch das Reflexivpronomen zu einem einstelligen Verb *sich rasieren*$^{(1)}$ gemacht wird,

(b) *Nachbar*$^{(2)}$ und *grüßen*$^{(2)}$, das durch das Reziprokpronomen zu einem symmetrischen Verb *einander grüßen*$^{(2)}$ gemacht wird,

(c) *Mensch*$^{(1)}$ und das (hier als symmetrisch lesbare) *Bruder*$^{(2)}$,

Fall (a): Das können wir einfach auf die schon behandelten Fälle wie in *Alle Menschen* $^{(1)}$ *arbeiten*$^{(1)}$ zurückführen, indem wir ein transitives Verb mit dem Reflexivpronomen zu einem intransitiven Verb verbinden.

Die Grammatik muß dazu nur um eine Regel wie

⟨*Reflexivierung*⟩≡
```
  iv : ...
     | tv SICH     (fn x => tv(x,x))
```

erweitert werden.[1] Den Rest, abgesehen von der Änderung des Lexers unten, besorgt dann die bisherige Auswertung von Sätzen der Form `qnp iv`.

Fall (b): Zunächst führen wir eine neue syntaktische Kategorie `stv` der *symmetrischen transitiven Verben* ein, mit folgender Regel:

⟨*Bildung symmetrischer Verben*⟩≡
```
  stv : tv EINANDER   (fn (x,y) => tv(x,y) andalso tv(y,x))
```

[1] Das Reflexivpronomen hat dann nur synkategorematische Bedeutung. In der Übungsstunde haben wir Fall (a) etwas anders behandelt.

Solche Verben erfordern stets ein Subjekt im Plural: *alle Nachbarn grüßen einander* ist grammatisch korrekt, *jeder Nachbar grüßt einander* ist inkorrekt. Der Grund für diese syntaktische Besonderheit –die wir in die Beispielgrammatik nicht einbauen wollen– ist semantischer Natur: das Subjekt ist kein Individuum, sondern ein Paar von Individuen (die syntaktische Objektstelle ist schon durch das Reziprokpronomen besetzt). Entsprechend brauchen wir hier besondere quantifizierte Subjekte, die nur mit zweistelligen Nomina gebildet werden können:

⟨*Zweistellige Nomen und Nominalphrasen*⟩≡
```
n2 : NACHBAR    (P.Nachbar)
   | BRUDER     (P.Bruder)

qnp2 : ALLE n2  (P.alle n2)
```

Eine Bedeutung `P.alle` haben wir aber bisher noch nicht angegeben. Im Unterschied zu `jeder` soll `alle` nicht über alle Individuen, sondern über alle *Paare von* Individuen laufen. Dazu fügen wir in das Paket `Quantoren` neben dem bisherigen (hier umbenannten) einstelligen verallgemeinerten Quantor

⟨*Quantoren über Objekte*⟩≡
```
fun forall_1_1 A G = List.all G (List.filter A objects)
```

bei dem `A` und `G` für Tests $A, G : (U \to \mathbb{B})$ auf Objekten stehen, einen entsprechenden zweistelligen verallgemeinerten Quantor ein, bei dem sie für Tests $A, G : (U \times U \to \mathbb{B})$ auf Paaren von Objekten stehen.

⟨*Quantoren über Paare von Objekten*⟩≡
```
val object-pairs = [(a,a),(a,b),...,(a,z),...,(z,a),...,(z,z)];
fun forall_2_2 A G = List.all G (List.filter A object-pairs)
```

Die bei (b) nötige Bedeutung von `alle` muß in der Struktur `Personen` hierauf zurückgeführt werden:

⟨*'alle' als Quantor über Paare*⟩≡
```
val alle = Q.forall_2_2
```

Damit ist für Nominalphrasen der Art `qnp2` eine Bedeutung erklärt, und Beispiele der Form (b) können mit folgender Satzregel behandelt werden:

⟨*Sätze mit paar-quantifiziertem Subjekt und symmetrischem Verb*⟩≡
```
satz : ...
     | qnp2 stv    (qnp2 stv)
```

Fall (c): Wie in (b) ist auch hier das Prädikat *sind Brüder* eine symmetrische zweistellige Relation, so daß wieder ein Subjekt im Plural nötig ist. Aber das Subjekt wird aus einem *einstelligen* Nomen gebildet. Deshalb muß in `Quantoren` eine dritte Bedeutung für die Quantoren eingefügt werden:

⟨*Doppelte Quantifizierung über Objekte*⟩≡
```
fun forall_1_2 A G =
    forall_1_1 A (fn x => forall_1_1 A (fn y => G(x,y)) objects)
```

Dieser Quantor drückt aus, daß *je zwei* Objekte mit der Eigenschaft $A^{(1)}$ in der Relation $G^{(2)}$ zueinander stehen. In der Struktur `Personen` definieren wir also:

⟨*'je-zwei' als Quantor*⟩≡
```
val je_zwei = Q.forall_1_2
```

Dann können Beispiele der Form (c) mit folgenden Regeln behandelt werden:

⟨*Sätze mit doppelt quantifiziertem Subjekt und symm.transitiven Verb*⟩≡
```
qnp_1_2 : ALLE n     (P.je_zwei n)

satz : ...
     | qnp_1_2 stv   (qnp_1_2 stv)
```

Man sieht also, daß ein einfach erscheinender Quantor wie *alle* auf drei sehr unterschiedliche Weisen interpretiert wereden muß, je nach dem Kontext! Entsprechend hat auch *einige* die drei Bedeutungen von *mindestens ein Objekt hat eine Eigenschaft* $G^{(1)}$, *mindestens ein Paar von Objekten steht in einer Beziehung* $G^{(2)}$, und *mindestens ein Objekt steht zu mindestens einem Objekt in einer symmetrischen Beziehung*.

Der Tokenizer muß natürlich passend um

⟨*tokenizer*⟩≡
```
"sich"     => Tokens.SICH(!line,!line));
"einander" => Tokens.EINANDER(!line,!line));
"Nachbarn" => Tokens.NACHBAR(!line,!line));
"Brüder"   => Tokens.BRUDER(!line,!line));
"alle"     => Tokens.ALLE(!line,!line));
```

erweitert werden.

Wenn man die Dateien des Pakets `TNL` entsprechend ändert, sollte es gehen. (Ich hatte aber keine Zeit, es zu testen.)

Lösung zu Aufgabe 4.2:

Weil niemand die Aufgabe richtig verstanden hatte, gebe ich hier nur einen ein Hinweis, damit man es nochmal versuchen kann: in `TNL/zeit.sml` haben wir schon Bedeutungen für `seit` und `bis` definiert, und zwar:[2]

⟨*aus zeit.sml*⟩≡

```
val bis = Until
val seit = Since
```

Die Grammatik hat das zur Interpretation zusammengesetzter *Sätze* benutzt:

⟨*aus tnl.grm*⟩≡

```
satz : ...
    | satz BIS satz (Z.bis(satz2,satz1))
    | satz SEIT satz (Z.seit(satz2,satz1))
```

Hierin werden durch `Z.bis` und `Z.seit` zwei *Satzbedeutungen* $f, g : \mathbb{Z} \to \mathbb{B}$ kombiniiert. Die indexikalischen Zeitadverbien werden aber durch

⟨*aus tnl.grm*⟩+≡

```
%nonterm
      ...
    | advti of D.ADVti
```

und die Definition

⟨*aus tnl.sml*⟩≡

```
type ADVti = zeit -> zeit
```

aus der Interpretation `D` als Funktionen in $\mathbb{Z} \to \mathbb{Z}$ definiert.

Die Aufgabe war, Sätze der Form

⟨*erweiterung von tnl.grm*⟩≡

```
satz ...
    | satz BIS advti    (Auswertungsregel-1 ?)
    | satz SEIT advti   (Auswertungsregel-2 ?)
```

zuzulassen und passende Auswertungsregeln anzugeben, die die schon definierten `Z.bis` und `Z.seit` benutzen. Manche haben

⟨*fehlerhaft*⟩≡

```
satz ...
    | satz BIS advti    (Z.bis(advti,satz))
    | satz SEIT advti   (Z.seit(advti,satz))
```

vorgeschlagen, aber das geht nicht, da im ersten Argument eine Funktion $f : \mathbb{Z} \to \mathbb{B}$ nötig ist, aber `advti` $: \mathbb{Z} \to \mathbb{Z}$ ist. Der Tip in der Aufgabe war, man solle die „Zeitkonstante“ `jetzt` (mit dem festen Wert 10) benutzen.

Man muß für das erste Argument eine Funktion $f : \mathbb{Z} \to \mathbb{B}$ angeben, sodaß $f(t)$ genau dann 1 ist, wenn t der durch das indexikalische Zeitadverb `advti` im Satz (relativ zu `jetzt`) benannte Zeitpunkt ist. Wenn das Adverb z.B. `morgen` lautet, muß $f(t) = 1 \iff t = 10 + 1$ gelten. Wie definiert man f also allgemein aus `advti` und `jetzt`?

Mehr kann ich nicht sagen, ohne die vollständige Lösung hinzuschreiben.

[2] Es wäre geschickter gewesen, wenn ich `bis` durch `fun bis(f,g) = Until(g,f)` definiert hätte: dann bräuchte in der Auswertungsregel `satz BIS satz (Z.bis(satz2,satz1))` die Vertauschung der Argumente nicht gemacht weren. (Analog bei `seit`.)

Der Grund für die Vertauschung ist, daß die Definition von `Until` von der Präfixschreibweise `Until(f,g)` ausgeht, was dem umgangssprachlichen `bis f (gilt), (gilt) g` entspricht (Adverbial `bis f` im Vorfeld). Die Grammatik sieht aber die Infixschreibweise `satz BIS satz` vor, und die entspricht `g (gilt), bis f (gilt))`, (Adverbial `bis f` im Nachfeld). Die Auswertungsregel `Z.bis(satz2,satz1)` bringt sozusagen das Adverbial aus dem Nachfeld in die von `Until = Z.bis` gewünschte Stellung im Vorfeld. Kurz: `g, bis f` bedeutet `bis f, g` bzw. `Until(f,g)`