PR ÜBERSETZERBAU 621.401 – 621.402 **SS 2007**

Institut für Informationstechnologie (ITEC) **Taschwer**

Übungsblatt 06

Termin: 2007/05/08

Ü 6.1 Typprüfung, Typausdrücke (Gewicht = 30%)

Die folgende Grammatik definiert eine einfache Sprache für ein Programm (P), in dem eine Sequenz von Deklarationen (D) von einem einzigen Ausdruck (E) gefolgt wird. Neben den Basisdatentypen für Zeichenketten (*char*) und ganze Zahlen (*integer*) wird auch ein Datentyp *list*(T) unterstützt, der eine Liste von Elementen des Typs T darstellt.

P → D ; E
D → D ; D | **id** : T
T → **list of** T | **char** | **integer**
E → (L) | **literal** | **num** | **id**
L → E , L | E

a) Schreiben Sie ein Übersetzungsschema, das den Datentyp eines deklarierten Bezeichners (**id**) in der Symboltabelle speichert und den Datentyp eines Ausdrucks (E) in Form eines Typausdrucks bestimmt (vgl. VO-Folie 6.3). Falls die Elemente einer Liste nicht alle denselben Typ haben, soll die Liste den speziellen Typ *type_error* erhalten, um einen Typfehler zu signalisieren.

b) Geben Sie den von Ihrem Übersetzungsschema erzeugten Typausdruck für die Ausdrücke in folgenden Programmen an:

```
list1: list of integer; list2: list of list of integer;
(((1,2),list1),list2)
```

```
list1: list of integer;
((1),((2),list1))
```

Ü 6.2 Typäquivalenz (Gewicht = 30%)

Die Programmiersprache C benutzt strukturelle Äquivalenz für alle Datentypen außer für Records. Betrachten wir folgende Deklarationen in C:

```
struct cell {  /* benannte Record-Deklaration */
    int info;
    struct cell *next;  /* Zeiger auf Record desselben Typs */
};
typedef struct cell celltype;  /* celltype ist Typname vom Typ struct cell */
typedef celltype *cellpointer; /* cellpointer: Typ pointer(celltype) */
```

Der Typname `struct cell` steht für folgenden Typausdruck (vgl. VO-Folie 6.2):

record((`info` × *integer*) × (`next` × *pointer*(`struct cell`)))

Zeiger auf benannte Recordtypen bieten die einzige Möglichkeit in C, rekursive Datenstrukturen zu definieren. Daher benutzt C die Namensäquivalenz für benannte Recordtypen, deren Name mit *struct ...* beginnt. Ein unbenannter Recordtyp (`struct {...}`)

ist zu keinem anderen Recordtyp äquivalent. Ein Arraytyp ist in C äquivalent zum Typ eines Zeigers auf ein Arrayelement, als Indextyp wir implizit *integer* verwendet.

a) Welche der folgenden Typausdrücke sind laut C-Semantik äquivalent?
   (1) `cellpointer`
   (2) *pointer*(`celltype`)
   (3) *pointer*(`struct cell`)
   (4) *pointer*(*record*((`info` × *integer*) × (`next` × *pointer*(`struct cell`))))
   (5) *array*(*integer*, `struct cell`)

b) Modifizieren Sie den Algorithmus auf VO-Folie 6.5 für die Prüfung der Äquivalenz von C-Datentypen (ohne `enum`).

Ü 6.3 Überladen von Operatoren (Gewicht = 40%)

Angenommen, der Multiplikationsoperator `*` einer Programmiersprache sei überladen worden, sodass er für ganze und komplexe Zahlen definiert ist:

(A) *integer* × *integer* → *integer*
(B) *integer* × *integer* → *complex*
(C) *complex* × *complex* → *complex*

Die Semantik der Sprache verlangt, dass die Interpretation (A, B oder C) des Operators `*` eindeutig aus dem Kontext des Teilausdrucks hervorgeht. Beispielsweise muss der Teilausdruck `3*5` in `2*(3*5)` den Ergebnistyp *integer* haben, in `(3*5)*z` aber *complex*, falls `z` als komplexe Variable deklariert worden ist. Der Einfachheit halber betrachten wir folgende Grammatik für vollständige Ausdrücke (E) der Programmiersprache:

E → T
T → **id** | **num** | (T) | T * T

a) Erstellen Sie eine syntaxgesteuerte Definition, welche die Menge der möglichen Ergebnistypen eines vollständigen Ausdrucks als Attribut E.*types* bestimmt. Das Terminal **num** hat den Typ *integer*, der Typ eines Bezeichners kann durch den Funktionsaufruf *lookup*(**id**.*entry*) ermittelt werden und ist entweder *integer* oder *complex*.

b) Um die Typprüfung für vollständige Ausdrücke umzusetzen, traversiert der durch a) definierte Übersetzer den Syntaxbaum ein weiteres Mal und bestimmt für alle Knoten ein vererbtes Attribut *unique*, das den eindeutigen Ergebnistyp des vollständigen Ausdrucks (E) darstellt. E.*unique* wird mit dem in E.*types* enthaltenen Typ initialisiert, sofern E.*types* nur 1 Element enthält; falls E.*types* 2 Elemente enthält, wird E.*unique* mit *integer* initialisiert; sonst erhält E.*unique* den Wert *type_error*. Bei der Anwendung der Multiplikation wird aus dem vererbten Ergebnistyp T.*unique* und den Typmengen der Operanden ein eindeutiger zulässiger Wert für das *unique* Attribut der Operanden bestimmt, wobei analoge Regeln wie für E.*unique* gelten. Erweitern Sie die syntaxgesteuerte Definition aus a) entsprechend.

c) Geben Sie für folgende Ausdrücke den mit den Attributen *types* und *unique* annotierten Syntaxbaum an. Der Bezeichner `z` hat den Typ *complex*. Welche Interpretation (A, B oder C) des Operators `*` wird jeweils verwendet, falls kein Typfehler auftritt?
   (i) `1*2*3` (ii) `1*(z*2)` (iii) `(1*2)*z`