Zwischenbericht Compilerbau Gruppe 7
Listen

Christof Weibel Benjamin Neukom

13. November 2013 Version 0.1

Dieses Dokument beschreibt, wie die Sprache IML um Listen erweitert wird und welche Änderungen dazu am Compiler und der Virtual Machine gemacht werden. Die Spracherweiterung wird mit anderen Sprachen wie Scala oder Haskell verglichen und die Designentscheide werden erläutert.

1 Listen als Spracherweiterung

Listen für IML sind wie folgt definiert

1. Stores vom Typ Liste können mit Brackets definiert werden. Beispiel: *var l* : [*int*] ist ein Store vom Typ Liste von ints.

2. Listen sind geordnete Ansammlungen von Objekten mit dem gleichen Datentyp. Beispiel: $[1, 2, 3]$ ist eine Liste vom Typ *int* mit den Werten 1, 2 und 3. Oder mit Verschachtelung $[[true, false], [true]]$ ist eine Liste vom Typ Liste von bools mit den Werten $[true, false]$ und $[true]$.

3. Der Typ einer Liste ist definiert durch das erste Element (falls das Element eine Liste ist, das erste nicht leere) der Liste. Beispiel: $[[1, 2, 3], [3, 4, 5]]$ ist eine Liste vom Typ Liste von *int*s oder $[[], [true]]$ ist vom Typ Liste von bools.

4. Die leere Liste [] ist vom Typ Liste von *Any* (welcher in IML nicht verwendet werden kann). Der Typ *Any* ist kompatibel mit allen anderen Typen. Beispiel: für die Deklaration des Stores *varl* : [[*int*]] ist $l :=$ [] (Type [*Any*]) so wie auch $l :=$ [[]] (Type [[*Any*]]) eine valide Zuweisung.

5. Listen sind immutable, d.h. die Werte einer Liste können nicht verändert werden. Es können nur mit den operationen :: (concatenation) und *tail* neue Listen konstruiert werden,

2 Operationen auf Listen

2.1 Konkatenation

$$Type :: [Type] \rightarrow [Type]$$

Der :: Operator erstellt eine neue Liste mit dem Operand auf der Linken Seite als Head und dem Operand der rechten Seite als Tail. Beispiel: die Expression $1 :: [2, 3, 4]$ gibt die Liste $[1, 2, 3, 4]$ zurück.

2.2 Head

$$head\ [Type] \rightarrow Type$$

Der *head* Operator gibt das erste Element einer Liste zurück. Beispiel: die Expression head [2,3,4] gibt den Wert 2 zurück. Falls versucht wird der Head einer leeren Liste abzufragen wirft die VM eine Exception.

2.3 Tail

$$tail\ [Type] \rightarrow [Type]$$

Der *tail* Operator gibt eine neue Liste ohne den Head einer bestehenden Liste zurück. Beispiel: Die Expression *tail* $[1, 2, 3, 4]$ gibt die Liste [2,3,4] zurück. Die Tail Operation kann auch eine leere Liste zurückgeben. Beispiel: die Expression *tail* [1] gibt die leere Liste [] zurück.

2.4 Length

$$length\ [Type] \rightarrow Int$$

Der *length* Operator gibt die Länge einer Liste zurück. Beispiel: *length* $[1, 2, 3, 4, 5]$ gibt den Wert 5 zurück.

3 Vergleich mit Haskell und Scala

3.1 Haskell

Die Operatoren *head*, *tail* und *length* verhalten sich identisch zu den Haskell Varianten dieser Operatoren. Einen Unterschied zu Haskell ist die Prezedenz des konkatenations Operator. Bei unserer IML Erweiterung hat der Operator :: die tiefste Priorität und bei Haskell liegt die Priorität zwischen den boolschen, logischen und den arithmetischen Operatoren. Dies hat folgende Konsequenzen:

Beispiel für Haskell:

```
1 > 3 : True : [] // Type Error
(1 > ((3 : True) : [])) // Mit Klammerung um Operator Prezedenz zu zeigen
```

Listing 1: Ungültige Listen Konkatenation in Haskell

Gleiches Beispiel in IML:

```
1 > 3 :: true : [] // Keinen Type Error
((1 > 3) :: (true :: [])) // Mit Klammerung um Operator Prezedenz zu zeigen
```

Listing 2: Gültige Listen Konkatenation in IML

Wir haben uns für diese Prezedenz entschieden, da es unserer Meinung nach natürlicher ist, dass der :: Operator die tiefste Priorität hat.

3.2 Scala

In Scala verhalt sich die Operator Prezedenz für den :: Operator gleich wie bei Haskell, also unterschiedlich zu IML.

4 Änderungen an der Grammatik

Folgende Änderungen wurden an der Grammatik vorgenommen. Die Änderungen wurden mit einem eigens entwickelten Tool (mehr dazu später) getestet und sind *LL*(1) konform.

4.1 Expression

Vorher:

expr ::= *term1* {BOOLOPR term1}
term1 ::= *term2* [RELOPR term2]
term2 ::= *term3* {ADDOPR term3}
term3 ::= *factor* {MULTOPR factor}
factor ::= *literal*
| IDENT [INIT | *exprList*]
| monadicOpr *factor*
| LPAREN *expr* RPAREN;
exprList ::= LPAREN [*expr* {COMMA *expr*}] RPAREN
monadicOpr::= NOT | ADDOPR

Nachher:

expr ::= *term0* {CONCATOPR term0}
term0 ::= *term1* {BOOLOPR term1}
term1 ::= *term2* [RELOPR term2]
term2 ::= *term3* {ADDOPR term3}
term3 ::= *factor* {MULTOPR factor}
factor ::= *literal*
| IDENT [INIT | *exprList*]
| monadicOpr *factor*
| LPAREN *expr* RPAREN;
exprList ::= LPAREN [*expr* {COMMA *expr*}] RPAREN
monadicOpr::= NOT | ADDOPR | HEAD | TAIL | SIZE

4.2 Type

Vorher:

atomType ::= INT | BOOL

Nachher:

type ::= *atomType* | LBRACKET *type* RBRACKET
atomType ::= INT | BOOL

4.3 Literal

Vorher:

literal ::= INTLITERAL | BOOLLITERAL

Nachher:

literal ::= INTLITERAL | BOOLLITERAL | *listLiteral*
listLiteral ::= LBRACKET [*expr* COMMA *expr*] RBRACKET

5 Grammatik auf LL(1) Fehler überprüfen

6 Code Beispiele

7 Quellen

Internet:

Wikipedia	en.wikipedia.org
Haskell Listen	http://andres-loeh.de/haskell/4.pdf
Haskell Language Specification	www.haskell.org/onlinereport/
Scala	www.scala.org

Bücher:

Programmming in Scala	Odersky et. al.
Grundlagen und Techniken des Compilerbaus	Niklaus Wirth