Arrays und Array-Slicing

Idee

Syntax

Lexikalisch

Zur lexikalischen Syntax haben wir eine ausführliche Diskussion geführt, und dabei verschiedene Varianten ent- und wieder ver-worfen.

Zum Vergleich zunächst die Deklaration einer Variablen, wie sie bereits in IML existiert:
var m:int;
Zunächst wird deklariert, dass es eine Variable (und nicht eine Konstante) ist, dann wird sie benannt und dann der Typ angegeben.
Wir wollten uns möglichst nahe an diese Vorgabe halten, da wir nicht einfach Teile einer anderen Sprache in IML einbauen wollten, sondern
IML erweitern.

Unser erster Versuch, dies zu erreichen, sah so aus:
var a:TYPE[LENGTH][DIMENSION];
TYPE würde hier INT oder BOOL sein, und so bestimmen, was der Typ der Elemente im Array ist. Obwohl diese Schreibweise durchaus elegant
ist, macht sie es unmöglich, mehrdimensionale Arrays zu deklarieren, welche nicht in allen Dimensionen gleich lang sind. Zudem ist diese
Schreibweise noch sehr deutlich von Sprachen wie Java beeinflusst, und führt die eckigen Klammern neu ein.

Als nächste Schreibweise überlegten wir uns, die folgende zu Verwenden:
var b:arr (array_decl) LENGTH;
var b:arr (arr (arr (arr int 8) 5) 3) 10;
var b:arr int 3;
Während dies für ein eindimensionales Array uns sehr intuitiv erscheint, wird es für mehrdimensionale Arrays eher verwirrend, da die
Längenangaben dann in der “verkehrten” Reihenfolge dastehen.
Daher modifizierten wir diese Schreibweise und verlegten die Länge nach vorne:
var c:arr LENGTH (array_decl);
var c:arr 10 (arr 5 (arr 6 (arr 2 int)));
var c:arr 3 int;
Statt einem “Array von ints mit Länge 3” würde man also ein “Array, Länge 3, von ints” deklarieren. Dies schien uns nur wenig
unintuitiver, und bei mehrdimensionalen Arrays deutlich besser. Ganz zufrieden waren wir aber noch nicht. Klammern zur Verschachtelung
werden nämlich ab einigen Levels von Verschachtelung immer schlechter lesbar.

Da alle nicht-Array Elemente eines Arrays vom gleichen Typ sind, und das einzige Verschachtelbare Arrays sind, kamen wir darauf, die
Redundanzen zu entfernen:
var d:arr (4,10,5) int;
var d:arr (3) int;
Die Länge der einzelnen Dimensionen ist das einzige, was sie unterscheidet und mit dieser Deklaration ist dies auch klar.

WIP Array Initialisierung und Zugriff:
// Arrays müssen vollständig initialisiert werden; sugar für init:= 0;
var a: arr 7 int;
var foo: int;
a init := [0, 1, 2, 3, 4, 5, 6];
foo init := a[0];

WIP Array Init und Zugriff bei verschachtelten Arrays:
var a: arr (4,2) bool;
a init:= [[true, true],[true, false],[false, true],[false,false]];
a[0] = [true, true];
a[0][1] = true; \ this gets REAL ugly if you go several levels deep

WIP Array Deklaration mit Array Literalen:
var a: arr 4 int;

a := [0, 1, 5, 6];

WIP Array Slice Notation:
var a: arr 20 int;
var b: arr 4 int;
b := a[1:3]; // the indices are both inclusive, making this a 3

WIP Array Zuweisungen:
a[0] := EXPR;
a[0:2] := ARR_EXPR;

Grammatikalisch

Kontext- und Typeinschränkung

Codeerzeugung (Final Report only)

— EMPTY FOR NOW —

Vergleich mit anderen Sprachen

Matlab: Schrittlänge
Python: Slice notation, aber inklusive
??: ++ als concatenation

Warum so und nicht anders?

Maximale Lesbarkeit ohne Unklarheit und ohne die Typsicherheit zu verletzen. Wir haben uns ueberlegt, Arrays auch mit basic types zu konkatenieren. Dies fuehrt aber zu einer komplexeren Grammatik. Da wir Array Literale haben, kann ein basic type ohne grossen Aufwand manuell in ein Array verpackt werden.

Appendix: Testprogramme

Gewichtung abhängig von Thema

Notes

var a: arr <20> int
var a: arr <20

var a: arr 20 x int

** Array creation

** Array slice notation in Python
ford% python
Python 2.7.8 (default, Jul 25 2014, 14:04:36)
[GCC 4.8.3] on cygwin
Type “help”, “copyright”, “credits” or “license” for more information.

a = [0,1,2,3,4,5,6]
a[0:2][0, 1]
a[0:6][0, 1, 2, 3, 4, 5]
a[0:7][0, 1, 2, 3, 4, 5, 6]

** Array declaration (with

Table exmaple

Left-Aligned	Center Aligned	Right Aligned
col 3 is	some wordy text	$1600
col 2 is	centered	$12
zebra stripes	are neat	$1