Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Clemens Ballarin

Vorlesung Semantik
Sommersemester 2004
Übungsblatt 12
Besprechung am 14. Juli 2004

Aufgabe 1 (H) (*Typisierung von JinjaVM-Programmen*)

Gegeben ist ein JinjaVM-Programm mit den Klassen `A` und `B`, wobei `B` eine Teilklasse von `A` ist. In der Klasse `A` sind ausserdem das Feld `F` vom Typ `Class A` und die Methoden `M1` und `M2` definiert. Abweichend von der Vorlesung werden die Sprungziele symbolisch mit Lables angegeben, nicht als Sprungweite.

a) `M1: [Integer, Class B] -> Class A`

```
[    Load 2,
 L1: Load 1, Push (Intg 0), CmpEq,
     IfFalse L2, Getfield F A, Goto L1,
 L2: Return ]
```

b) `M2: [Integer, Class B] -> Class A`

```
[    Load 2, Load 0,
 L1: Load 1, Push (Intg 0), CmpEq,
     IfFalse L2, Putfield F A, Load 0, Goto L1,
 L2: Return ]
```

Bestimmen Sie jeweils den Methodentyp.

Aufgabe 2 (Ü) (*Typsicherheit*)

In dieser Aufgabe wird eine stark reduzierte Variante der JinjaVM betrachtet, in der nur die Typen `Boolean` und `Integer` vorkommen. Dies hat unter anderem zur Folge, dass es keine Methodenaufrufe und keine Ausnahmebehandlung gibt. Es gibt lediglich eine globale Methode, die ausgeführt werden kann.

a) Geben Sie das Maschinenmodell für die reduzierte JinjaVM an.

b) Geben Sie an, wann ein Programm P wohlgeformt bezüglich der Typisierung Φ ist, und definieren Sie Konformanz ($P, \Phi \vdash \sigma\surd$) eines Zustands σ bezüglich P und Φ . Welchen Typ hat Φ für die reduzierte JinjaVM?

c) Zeigen Sie jetzt die Typsicherheit. Das heisst, unter geeigneten Annahmen gilt bei einem Aufruf von *exec-instr*, dass *check-instr* mit den gleichen Argumenten zu true auswertet.

Aufgabe 3 (Ü) (*Konstantenpropagation*)

In der Vorlesung wird Kildalls Algorithmus zur Datenflussanalyse eingesetzt, um zu berechnen ob Methoden typkorrekt sind. Eine andere Anwendungsmöglichkeit besteht darin herauszufinden, welche Werte während der Programmausführung konstant sind. Dies kann zur Optimierung, z.B. in einem Compiler benutzt werden. Im Folgenden werden jedoch JinjaVM-Programme betrachtet. Modifizieren Sie den Rahmen zur Bytecode-Verifikation so, dass er zur Konstantenpropagation verwendet werden kann.

a) Geben Sie einen geeigneten Halbverband an. In diesem muss sich ausdrücken lassen, ob ein Register oder Stackelement an einer bestimmten Stelle im Programm initialisiert ist, einen bestimmten konstanten Wert hat, oder ob es verschiedene Werte enthalten kann.

b) Passen Sie die Funktionen für Anwendbarkeit und Effekt an. Geben Sie den Ausdruck an, mit dem die Datenflussanalyse jetzt gestartet werden muss.

c) Wenden Sie Ihren Algorithmus zur Konstantenpropagation auf die Methode

```
M: [Integer] -> Integer

[    Push (Intg 0), Push (Intg 0), CmpEq,
     IfFalse L1, Push (Intg 0), Goto L2,
 L1: Load 1,
 L2: Return]
```

an. Lässt sich das Ergebnis noch weiter verbessern? Wenn ja, wie?