

Programmiersprachen II – Wintersemester 2011/2012 Projektarbeit

1 Abgabe

Bitte schicken Sie Ihre Lösung bis spätestens zum **17.2.2012 (Freitag)** an folgende Email-Adresse:
John.Witulski@uni-duesseldorf.de.

Falls Sie Fragen haben, können Sie sich jederzeit bei dieser Email-Adresse melden.

2 Format und Dokumentation

Beachten Sie bitte folgende Aspekte bei der Abgabe. Diese sind **relevant für die Punktevergabe**:

1. Sie sollten zu den implementierten Funktionen einige Tests schreiben. Diese werden Ihnen auch die Implementierung erleichtern. Falls Sie Tests systematisch anwenden möchten, um Ihr Programm zu schreiben, empfehlen wir Ihnen das Thema *Test-driven development*. Eine Erklärung finden Sie z.B. unter http://en.wikipedia.org/wiki/Test-driven_development
2. Nutzen Sie Kommentare im Quellcode, um Ihre Implementierung und die geschriebenen Prädikate zu erläutern.
3. Schreiben Sie eine kurze Erläuterung zu den von Ihnen implementierten Funktionen. Wichtig dabei ist:
 - a) Welche Aufgaben haben Sie bearbeitet?
 - b) Welche Probleme hat Ihre Implementierung?
 - c) Welche Besonderheiten hat Ihre Implementierung?

Es reicht dazu ein Absatz in einer ".txt"-Datei. Sie brauchen keine mehrseitige Dokumentation abgeben!

4. Geben Sie Ihre Implementierung als ".pl"-Dateien ab. Verwenden Sie für jede Codeanalyse bzw. Codeoptimierung eine eigene Prolog-Datei.
5. Geben Sie weiterhin ebenfalls selbst geschriebene (oder erzeugte) Java-Bytecode Programme als ".byc"-Dateien ab.

3 Benotung

Diese Projektarbeit ist keine Zulassungsvoraussetzung für die Klausur. Stattdessen erhalten Sie, bei ausreichender guter Lösung, zusätzliche Punkte für die anstehende Klausur. Die maximal erreichbare Punktezahl beschränkt sich hierbei auf 5 Punkte. Diese fließen in die Klausur als "Bonuspunkte" ein.

4 Aufgaben

Aufgabe 1 (Einführung)

Ihre Aufgabe ist, es den aus der Vorlesung bekannten Abstrakten Java-Bytecode Interpreter zu erweitern. Die Dateien zum Java-Bytecode Interpreter finden Sie online auf der STUPS-Seite. Sie benötigen folgende Dateien:

- **javabc_parser.pl**: Einlesen einer ".byc"-Datei und Ablegen der Fakten für den Interpreter
- **javabc_interpreter.pl**: Interpretieren der abgelegten Fakten
- **abstract_interpreter_loop.pl**: Fixpunktschleife und Test der abstrakten Domäne
- **abstract_environment.pl**: Definition der abstrakten Domäne
- **abstract_constantpropagation.pl**: Eine implementierte Konstantenpropagation
- Ein paar ".byc"-Dateien zum testen

Ausführungsbeispiel:

1. Setzen Sie ein paar Pfade zu den Testdateien an den Anfang der Datei *javabc_parser.pl*.
2. Consultieren Sie die Datei *javabc_interpreter.pl* (Die Datei *javabc_parser.pl* wird automatisch mitconsultiert).
3. Consultieren Sie nun je nachdem, welche Analyse Sie ausführen möchten die Datei *abstract_environment.pl* oder *abstract_constantpropagation.pl* (Die Datei *abstract_interpreter_loop.pl* wird automatisch mitconsultiert).
4. Parsen Sie zunächst eine Datei mit den Prädikaten `parse/0`, `parse1/0`, ... oder `parse/1`.
5. Führen Sie nun eine Interpretation/Analyse aus mit `execute/1` bzw. `aint/0`

Aufgabe 2 (Allgemeine Erweiterungen)

Erweitern Sie zunächst den Parser, Interpreter und abstrakten Interpreter um folgende Operationen:

1. Die Operationen `idiv`, wobei gilt:

```
iconst_4
iconst_2
idiv ; Ergebnis ist: 4/2 = 2
```

Ausserdem muss bei dieser Operation das Ergebnis wieder ein Integer sein. Benutzen Sie hierfür das Prolog-Builtin `floor/1`:

```
?- A is floor(3.1).
A = 3 ?
yes
```

Geben Sie ausserdem bei Division durch 0 einen Fehler aus.

2. Die Operation `ipow`, wobei gilt:

```
iconst_3
iconst_2
ipow ; Ergebnis ist 3^2 = 9
```

Geben Sie hier einen Fehler aus für folgenden Fall: a^x , wobei $a = 0$ und $x < 0$.
Z.B.: $0^{-1} = \frac{1}{0}$

Aufgabe 3 (Spezielle Erweiterungen)

Sie erhalten nachfolgend eine Auswahl an Codeoptimierungen und Codeanalysen: Wählen sie hieraus eine Optimierung/Analyse aus:

- a) Endlosschleifen erkennen
- b) Definite Assignment und Unbenutzte Variablen
- c) Konstantenfaltung ¹
- d) Unreachable code eliminieren ²
- e) Strength reduction
Hinweis: Benötigt eine kleine Anpassung an den Parser und Interpreter

Erläuterung zu den Aufgabenteilen (Beispiele in Java):

- **Endlosschleifen erkennen:**

Man kann gewisse offensichtliche Endlosschleifen erkennen, z.B.:

```
for(int i = 1; i > 0; i = i + 1)
    ...
```

In einer möglichen Abstraktion wäre dies:

```
for(int i = pos; i > 0; i = i + pos)
    ...
```

Da pos und $(pos +_a pos)$ immer größer als 0 sind, ist der Ausdruck $i > 0$ immer wahr. Weitere Beispiele:

```
while(true)
    ...

int b = 7;
int a = 1;
while(a > 0) {
    a = b ** 2;
    b = (-2) * a;
}
```

Für die volle Punktzahl sollten Sie mindestens diese 3 Beispiele erkennen können.

Anforderung: Geben Sie einen Hinweis aus, falls Sie eine Endlosschleife entdeckt haben, in der der Programmzähler der zugehörigen Abfrage angegeben wird.

- **Definite assignment und unbenutzte Variablen**

Definite assignment bedeutet, dass jede Variable vor einer Benutzung einen Wert erhalten muss.

Unbenutzte Variablen sind Variablen, die zwar einen Wert erhalten, aber nicht benutzt werden und daher unnötig sind.

Nutzen Sie hierfür **def** und **use**, wie aus der Vorlesung bekannt.

Hierbei ist es egal, ob die Verwendung der Variable in "totem" Code stattfindet oder nicht.

Anforderung: Geben Sie den Namen jeder unbenutzten Variable und die Namen nicht initialisierter Variablen sowie den Ort ihrer Verwendung aus.

¹http://en.wikipedia.org/wiki/Constant_folding

²http://en.wikipedia.org/wiki/Unreachable_code

- **Konstantenfaltung**

Konstantenfaltung ist die Vorauswertung von statischen Ausdrücken. Betrachten Sie hierbei nur Integer-Zuweisungen.

```
int a = 7 * (3 ** 2);
int b = a / (-3);
int x = y * (2**3);
```

Dies lässt sich falten zu:

```
int a = 63;
int b = -21;
int x = y * 8;
```

Anforderung: Falten Sie (Integer-)Ausdrücke, die aus Konstanten bestehen. Falten Sie solange, bis keine weitere Faltung mehr möglich ist. Nach der Faltung soll das neu berechnete Programm als Fakten `instr/3` das vorherige Programm ersetzen.

- **Unreachable code eliminieren**

Unreachable code sind Programmabschnitte oder Anweisungen, die nie erreicht werden können, da kein Pfad des Kontrollflusses zu ihnen führt.

```
if(false) {
    ...          // unreachable code
}

int a = 2 + 1;
if(a == -1) {
    ...          // unreachable code
}

return;
int a = 2000;  // unreachable code
```

Suchen Sie sich selbst aus, ob Sie mit einer abstrakten Domäne oder einer konkreten Domäne arbeiten möchten. Geben Sie Ihre Wahl in der Dokumentation an.

Anforderung: Die oben stehenden Beispiele sollte als unreachable code erkannt werden. Nach der Berechnung soll das Programm ohne den unreachable code als Fakten `instr/3` das vorherige Programm ersetzen.

- **Strength reduction**

Erweitern Sie zunächst den Parser und Interpreter um die *Shift*-Funktion. Definition der Operatoren:

- **ishiftl x** Nimmt einen Wert vom Stack, shiftet ihn um x nach links (*ishiftl 1* entspricht der Operation $a * 2$ auf a). Anschliessend wird das Ergebnis auf den Stack gelegt.
- **ishiftr x** Nimmt einen Wert vom Stack, shiftet ihn um x nach rechts (*ishiftr 1* entspricht der Operation $a/2$ auf a). Anschliessend wird das Ergebnis auf den Stack gelegt.

Benutzen Sie für den Interpreter die in Prolog eingebauten Operatoren `<<` bzw. `>>` unter `is/2`:

```
?- A is 2 << 1.
A = 4 ?
yes
?- A is 8 >> 2.
A = 2 ?
yes
```

Unter Strength reduction versteht man eine Reduktion von Arithmetischen Ausdrücken. Der Vorteil soll sein, dass der neue Ausdruck Ressourcen sparen soll. Man versucht z.B. einen Ausdruck wie $x * 2$ auf $x \ll 1$ zu reduzieren, da dieser schneller berechnet werden kann (Randbemerkung: auf modernen Desktop-Prozessoren ist Strength Reduction nicht mehr zwingenderweise notwendig, da diese Multiplikation mit kleine Konstanten ähnlich schnell ausführen wie Addition).

Weitere Beispiele:

| Ausdruck | Reduzierter Ausdruck |
|--------------|----------------------|
| $y = x/8$ | $y = x \gg 3$ |
| $y = x * 64$ | $y = x \ll 6$ |
| $y = x * 2$ | $y = x + x$ |
| $y = x * 15$ | $y = (x \ll 4) - x$ |
| $y = (-x)^2$ | $y = x * x$ |

Anforderungen: Die in der Tabelle stehenden Beispiele (ausser $y = x * 15$) sollten wie angegeben funktionieren. Nach der Berechnung soll das Strength-reduzierte Programm als Fakten `instr/3` das vorherige Programm ersetzen.

Viel Erfolg!