P. Thiemann, S. Wehr Wintersemester 2006/2007

Compilerbaupraktikum

`http://proglang.informatik.uni-freiburg.de/teaching/compilerpraktikum/2006ws/`

Aufgabenblatt 2

Semantische Analyse

24.11.2006

In dieser Aufgabe soll ein Typchecker für eine kleine Teilmenge von Waitomo bzw. Java implementiert werden. Die Teilmenge trägt den Namen MiniJava.

Laden sie sich zunächst die Datei `mini-java-20061123.tar.gz` von der Homepage des Praktikums herunter und entpacken sie das Archiv im Wurzelverzeichnis ihres Compilercodes. (**Achtung:** Beim Entpacken werden die Dateien `Makefile` und `utils.ml` überschrieben; eigene Änderungen ggf. sichern!)

Aufgabe 1 (Transformation Waitomo nach MiniJava)

Die abstrakte Syntax von MiniJava ist im Module `Mini_java` definiert. Implementieren sie eine Transformation von Waitomo nach MiniJava; d.h. übersetzen sie die Datentypen aus dem Modul `Syntax.Dst` in die entsprechenden Datentypen des Modules `Mini_java`. Falls es in `Mini_java` keinen solchen Datentyp gibt, soll die Übersetzung mit einer geeigneten Ausnahme abgebrochen werden.

Fügen sie dem Treibermodule `Driver` eine neue Phase hinzu, welche ihre neu erstellte Übersetzung aufruft.

Aufgabe 2 (Typchecker für MiniJava)

In dieser Aufgabe soll ein Typchecker für MiniJava implementiert werden. Aufgabe des Typcheckers ist Programme mit Typfehlern zu erkennen und zurückzuweisen. Unter anderem entdeckt der Typchecker folgende Fehler:

- Zugriff auf eine nicht-existente Klasse/Methode/Instanzvariable oder lokale Variable.
- Operanden eines Operators haben den falschen Typ.
  Beispiele: `1 + true`, `"string" && false`, ...
- Methoden werden mit einer falschen Anzahl von Argumenten oder mit Argumenten vom falschen Typ aufgerufen.
- Methoden oder Felder werden außerhalb ihres Sichtbarkeitsbereichs benutzt.
- Eine Zuweisung ist nicht typkompatibel. Beispiel: `int i = new Object();`
- Eine Methode wird in einer Subklasse nicht korrekt überschrieben. Beispiel:

```
class C {
  void m(int x) { }
}
class D extends C {
  void m(bool b) { }
}
```

Das letzte Beispiel zeigt auch, dass MiniJava keine Überladung von Methoden ("overloading") unterstützt. Allerdings ist das Überdecken von Instanzvariablen ("shadowing") und das Überschreiben von Methoden ("overriding") erlaubt. Folgender Code ist also korrekt:

```
class C {
  int f;
  void m(int x) { print("in C"); }
}
class D extends C {
  bool f;
  void m(int x) { print("in D"); }
}
```

Eine genau Definition der Typregeln für Java Programme können sie in der *Java Language Spezification* (`http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html`) finden. Im folgenden werden lediglich Hinweise zum allgemeinen Vorgehen bei der Implementierung des Typcheckers beschrieben.

- Zunächst werden alle vorhandenen Klassen, Felder und Methoden in einer Symboltabelle gespeichert. Für die Symboltabelle steht mit dem Modul `Mini_java_symtab` bereits eine geeignete Datenstruktur mit folgender Signatur zur Verfügung:

  ```
  exception Type_error of string
  open Mini_java

  type t

  val add_class : t -> class_def -> t
  val add_field : t -> Id.clazz -> fdecl -> t
  val add_meth : t -> Id.clazz -> mdef -> t
  val find_class : t -> Id.clazz -> class_def
  val find_superclass : t -> Id.clazz -> class_def option
  val find_field : t -> Id.clazz -> Id.field -> (class_def * fdecl)
  val find_meth : t -> Id.clazz -> Id.meth -> (class_def * mdef)

  type env
  val add_var : env -> Id.var -> ty -> env
  val find_var : env -> Id.var -> ty
  ```

  Der Typ `t` ist der Typ einer Symboltabelle. Die Funktionen `find_field s c f` und `find_meth s c m` suchen das Feld mit Namen `f` bzw. die Methode mit Namen `m` in der Klasse namens `c` bezüglich der Symboltabelle `s`. Falls die Klasse `c` ein solches Feld bzw. eine solche Methode nicht besitzt, wird rekursiv in der Superklasse von `c` nachgeschlagen. Daher enthält der Rückgabewert auch die Klassendefinition in der das Feld bzw. die Methode schließlich gefunden wurde.

  Der Typ `env` speichert Typinformation über lokale Variablen. Beim initialen Aufbau der Symboltabelle sind lokale Variablen zu ignorieren, diese werden erst beim typchecken der einzelnen Methoden und Felder benötigt.

- Als nächstes müssen sie eine Funktion `is_subtype` implementieren, die testet ob zwei Typen zueinander in der Subtyprelation stehen. Die Typsprache von MiniJava unterstützt Klassen, die primitiven Typen `bool`, `int` und `void`, sowie den speziellen Typ `anyref`, welcher nicht in Quellprogrammen auftreten kann und lediglich zum typen des `null`–Literals verwendet wird.

  Die Subtyprelation `<:` ist reflexiv und transitiv; außerdem gilt C_1 `<:` C_2 für zwei Klassen C_1 und C_2 falls im Program C_1 als Subklasse von C_2 definiert ist (C_1 `extends` C_2). Zusätzlich gilt `anyref` `<:` C für jede Klasse C.

- Dann müssen sie den Typ eines Ausdrucks mittels einer Funktion `tyof_expr` berechnen. Die Funktion liefert den Ausdruck in veränderter Form zurück, da für Zugriffe auf Felder und Methode auch der Typ der Klasse, in der das Field oder die Methode schließlich gefunden wurde, mitgeführt werden muss.

- Abschließend müssen sie Statements, Methoden, Feldinitialisierungsausdrücke sowie ganze Klasse auf Typkorrektheit überprüfen. Dabei sind auch Sichtbarkeitsmodifikatoren zu beachten.

Fügen sie abschließend dem Treibermodule `Driver` eine neue Phase hinzu, welche den Typchecker aufruft. Vergessen sie nicht, Tests für den Typchecker zu schreiben.

Deadline: 11.12.2006