Prof. Dr. Arnd Poetzsch-Heffter
Dr.-Ing. Jan Schäfer

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 6: Übersetzer und sprachverarbeitende Werkzeuge (SS 2011)

Hand Out: 25. Mai 2011
Hand In: 8. Juni 2011

Aufgabe 1 Term-Positionstrukturen transformieren

Basierend auf der Expression-Spezifikation aus der Katja-Einführung sollen Sie nun eine Simplifikation von Ausdrücken implementieren, d.h. Sie sollen eine Methode schreiben, die ein `ExpressionPos`-Objekt nimmt und den zugrunde liegenden Ausdruck mathematisch korrekt vereinfacht, in dem konstante Teilausdrücke ausgewertet und durch Konstanten in der Positionsstruktur ersetzt werden. *Zur Erinnerung:* Die Positionsstruktur ist immutable.

Dennoch gibt es in jedem Pos-Typ eine Methode `replace(Object o)`, die eine neue Positionsstruktur erstellt, in dem die alte kopiert wird. Nur das aktuelle Objekt wird durch `o` ersetzt. `replace` wird ein Objekt von einem Termtyp und nicht von einem Positionstyp übergeben. Der Typ von `o` muss so sein, dass die entstehende Positionsstruktur zur ursprünglichen Spezifikation passt. `replace` liefert dann aus der neuen Positionsstruktur ein Pos-Objekt zurück. Dies entspricht von seiner Position in der Struktur genau dem Objekt auf dem `replace` aufgerufen werden.

Zum Beispiel gilt also:

```
((PlusPos) ExpressionPos(Plus(Literal(1),Literal(2)))).left().replace(Literal(2)).root()
==
ExpressionPos(Plus(Literal(2),Literal(2))).root()
```

Die Methode `root` liefert immer den obersten Parent eines Positionsobjektes zurück, also die Wurzel des Baums.

Zum Durchlaufen einer Positionsstruktur hat jeder Positionstyp die Methode `postOrder()` und `postOrderStart()`. Erstere liefert zu einem Positionsobjekt, das nächste Positionsobjekt in Postorder-Reihenfolge. Dazu darf man aber nicht an der Wurzel beginnen, die soll als letztes besucht werden, sondern muss an dem Knoten, der am weitesten links unten im Baum ist, beginnen. Dieser wird von der Methode `preOrderStart()` zurückgeliefert. Man kann dies also nutzen, um in einer Schleife durch eine Positionsstruktur zu laufen.

Implementieren Sie nun eine Methode `ExpressionPos simplify(ExpressionPos e)`, die `e`, wie oben beschrieben, vereinfacht.

Namensanalyse

Die Namensanalyse ordnet jedem Bezeichner seine Deklarationsstelle zu. Implementieren Sie dazu ein Attribut `resolveName`, welches auf jedem `IdentifierPos` definiert ist und die zugehörige Deklarationsstelle liefert. Für Deklarationsstellen fügen wir der Katja-Spezifikation die Variante `Declaration` hinzu:

```
Declaration = FieldOrVarDecl
            | MethodDecl
            | FormalParameter
            | ClassDecl
```

Attribute in Katja

Attribute in Katja werden durch einfache Methoden in Java implementiert. Wenn Sie zum Beispiel das Attribut `resolveName` implementieren, können Sie folgende statische Methode schreiben:

```
public class NameResolution {
  public static DeclarationPos resolveName(IdentifierPos p) {
    // hier die Berechnung einfuegen
  }
}
```

Da Katja-Positionen immutable sind, kann man eine Mehrfachberechnung vermeiden, indem man sich einmal berechnete Ergebnisse merkt (z.B. in einer `HashMap`). Auf der Webseite der Vorlesung finden sie eine Klasse `KatjaAttribute`, die diesen Cache implementiert. Die Klasse kann folgendermaßen verwendet werden:

```
public class NameResolution {
  private final static KatjaAttribute<DeclarationPos, IdentifierPos>
    resolveNameAttribute = new KatjaAttribute<DeclarationPos, IdentifierPos>() {
       protected DeclarationPos calculate(IdentifierPos p) {
         // hier die Berechnung einfuegen
       }
    };

  public static DeclarationPos resolveName(IdentifierPos id) {
    return resolveNameAttribute.get(id);
  }
}
```

Allgemeines zu den Aufgaben: Im Folgenden wird nur beschrieben was im Erfolgsfall passieren soll. Fehler sollten Sie natürlich ebenfalls behandeln und ausgeben.

Aufgabe 2 Namensanalyse mittels Symboltabellen

In der Vorlesung haben Sie ein Verfahren zur Namensanalyse mittels Symboltabellen für eine sehr kleine Beispielsprache kennengelernt. Verallgemeinern Sie den Algorithmus und implementieren Sie die Namensanalyse für MiniJava mittels Symboltabellen. Es reicht aus, wenn Sie die einfache Variante (Folien 41-44) implementieren. Optional können Sie auch die Variante mittels einer matrix-verlinkten Datenstruktur implementieren.

Speichern Sie die Ergebnisse der Namensanalyse in einer `HashTable` die jedem `IdentifierPos` des ASTs die entsprechende `DeclarationPos` zuweist. Diese `HashTable` verwenden Sie dann, um das eigentlich `resolveName`-Attribut zu implementieren.

Aufgabe 3 Namensanalyse mittels rekursiver Attribute

Wenn ein vollständiger AST zur Verfügung steht, kann man die Namensanalyse auch direkt durch Attribute durchführen. Anstelle in einem ersten Durchgang Symboltabellen aufzubauen, analysiert man direkt den Kontext in dem sich der entsprechende Identifier befindet.

Als Hilfestellung definieren Sie sich zunächst ein Attribut `context`, welches zu einem `Identifier` den nächsten Vater findet, der von der Sorte `IdentifierContext` ist. Wobei `IdentifierContext` wie folgt in Ihrer Katja-Datei definiert werden soll:

```
IdentifierContext = MethodCall
             | FieldOrVariableAccess
             | Declaration
```

Implementieren Sie die Namensanalyse im Anschluss, indem Sie ein Attribute `resolveName` implementieren, dass für einen Bezeichner die entsprechende Deklaration zurückliefert. Verwenden Sie zur Implementierung von `resolveName` das entsprechende Switch-Interface. Dieses gibt es auf Term-Positionen genauso wie Sie es auf Termen kennen gelernt haben.

Unsere Sprache umfasst momentan noch keine Referenztypen, daher können auch nur statische Methoden und Klassenattribute definiert werden. Dies ermöglicht es Ihnen erst, die Namensanalyse unabhängig von der Typanalyse zu definieren.