Programmierparadigmen

Wintersemester 2012/13
Torsten Görg, Sandro Degiorgi

1. Übung

30. Oktober / 2. November / 8. November 2012

Organisatorische Hinweise

Die Übungen finden 14-täglich statt. Alle Informationen sind auf der Webseite verfügbar. `http://www.iste.uni-stuttgart.de/ps/lehre/ws1213/programmierparadigmen.html`.

Geben Sie Ihre Lösungen bzw. die Ihres Abgabe-Teams online über eClaus ab (`https://eclaus.informatik.uni-stuttgart.de/`).
Die Abgaben müssen spätestens bis Mo., 29.10.2012, um 13 Uhr vorliegen.

Aufgabe 1.1 . . . . . 9 Punkte

1. Nennen Sie fünf Ausführungsmodelle und geben Sie zu jedem dieser Ausführungsmodelle eine charakteristische Eigenschaft an, die es von allen anderen unterscheidet, und eine konkrete Programmiersprache an, welche das Modell unterstützt.

2. In der Vorlesung wurden gewünschte Eigenschaften von Programmen als "-ilities" eingeführt und dem gegenüber einige Eigenschaften von Programmiersprachen vorgestellt. Geben Sie zu drei "-ilities" korrespondierende Programmiersprachenheigenschaften an und erläutern Sie, inwiefern diese die jeweiligen "-ilities" unterstützen.

Aufgabe 1.2 . . . . . 18 Punkte

Gegeben Sei folgenden Prolog-Wissenbasis:

```
parent( anton, michael, andrea ).     % parent( Child, Father, Mother )
parent( susanne, michael, andrea ).
married( michael, andrea ).           % married( Husband, Wife )
parent( markus, anton, sandra ).
parent( hanna, anton, sandra ).
parent( sandra, sascha, ulrike ).
parent( lukas, claus, susanne ).
married( frank, sabine ).
grand_mother( Child, Grand_Mother ) :- parent( Child, Father, _ ),
                                       parent( Father, _, Grand_Mother ).
grand_mother( Child, Grand_Mother ) :- parent( Child, _, Mother ),
                                       parent( Mother, _, Grand_Mother ).
```

1. `Mary` und `Karl` bekommen eine Tochter und nennen diese `Verena`. Geben Sie die notwendige Ergänzung an, um dies in der Prolog-Wissenbasis auszudrücken.

2. Welche Ergebnisse liefert die Prolog-Anfrage `?- grand_mother( X, andrea ).` ?

3. Wie lauten die Prolog-Anfrage und die dafür notwendigen Prolog-Regeln, um alle Kinder zu ermitteln, deren Eltern nicht verheiratet sind? Welche Ergebnisse liefert diese Anfrage für die gegebene Wissenbasis?

4. Wie lauten die Prolog-Anfrage und die dafür notwendigen Prolog-Regeln, um alle Ehepaare zu ermitteln, die keine Kinder haben? Welche Ergebnisse liefert diese Anfrage für die gegebene Wissenbasis?

5. Programmieren Sie Prolog-Regeln und eine zugehörige Anfrage, um alle (in der Wissensbasis erfassten) Vorfahren einer Person zu ermitteln.

Hinweise: Das Zeichen `_` in einer Regel bedeutet, dass der entsprechende Parameter eines Prädikats an dieser Stelle irrelevant ist. Ein Tutorial mit weiteren ergänzenden Informationen zu Prolog finden Sie z.B. unter `http://www.ifi.uzh.ch/req/courses/logische_programmierung/ws03/documents/Prolog_Tutorial.pdf`.

Aufgabe 1.3 ..... 19 Punkte

Betrachten Sie die folgende Sequenz von Zuweisungs-Operationen. `a`, `b`, `c`, `d`, `e`, `f` und `g` sind lokal definierte `Integer`-Variablen. `x`, `y` und `z` sind formale Parameter vom Typ `Integer`.

```
(01)   a := x + y;          (07)   a := b * 23;
(02)   b := y - 4;          (08)   b := d - a;
(03)   c := a - z;          (09)   c := c / b;
(04)   a := z * z;          (10)   e := 5 * c;
(05)   b := b + a;          (11)   f := c - d;
(06)   d := y * b;          (12)   g := c + a;
```

1. Überführen Sie die gegebene Anweisungssequenz in die äquivalente Single-Assignment-Form. In dieser Form darf einer Variablen nur einmal ein Wert zugewiesen werden, der danach nicht mehr verändert werden kann. Daher müssen alle weiteren Zuweisungen an dieselbe Variable durch Zuweisungen an neue Variablen mit anderen Namen ersetzt werden. Fügen Sie dazu an die ursprünglichen Namen Indexnummern an. Beachten Sie dabei, dass nun auch bei den lesenden Zugriffen auf die Variablen die ggf. geänderten Namen verwendet werden müssen, und zwar so, dass die korrekte Variable gelesen wird.

2. Zeichnen Sie zu der gegebenen Anweisungsfolge einen Abhängigkeitsgraphen (siehe Skript, S. 2-9). Um deutlich zu machen, wo die Eingangsdaten herkommen, fügen Sie auch Knoten für die drei formalen Parameter `x`, `y` und `z` ein, von denen Kanten zu den Anweisungen führen, in denen sie direkt benutzt werden. Die Knoten, die Anweisungen repräsentieren, brauchen nur mit dem Symbol des binären Operators beschriftet zu werden. Beschriften Sie ferner die Kanten mit den zugehörigen lokalen Variablennamen.

3. Anweisungen, zwischen denen keine Datenabhängigkeiten bestehen, können in der Reihenfolge ihrer Ausführung vertauscht werden, ohne die Semantik der Anweisungsfolge damit zu verändern. Damit können die Anweisungen der Anweisungsfolge in verschiedenen Reihenfolgen ausgeführt werden. Geben Sie fünf Varianten der obigen Anweisungsfolge in unterschiedlichen Ausführungsreihenfolgen an, die die Semantik nicht verändern. Es sollen dabei keine Optimierungen durchgeführt, sondern alle Anweisungen in ihrer ursprünglichen Form belassen werden.

4. Um die Berechnung zu beschleunigen, soll diese nun parallelisiert werden. Gehen Sie dazu davon aus, dass beliebig viele, parallel arbeitende Recheneinheiten zur Verfügung stehen, die nach dem Datenfluss-Modell arbeiten. Verteilen Sie die Bearbeitung der Anweisungen möglichst optimal auf mehrere Recheneinheiten, so weit dies bei den bestehenden Datenabhängigkeiten möglich ist. Jede Anweisung benötigt zu ihrer Ausführung eine Zeiteinheit. Idealisierter Weise wird für den Datentransfer zwischen den Recheneinheiten keine Zeit benötigt. Wieviele Zeiteinheiten werden mindestens benötigt? Geben Sie als Beleg für Ihre Aussage eine zugehörige Aufteilung der Anweisungen auf Recheneinheiten an.

5. Die parallele Ausführung sei nun auf nur zwei Recheneinheiten beschränkt. Wieviel Zeiteinheiten werden unter dieser Restriktion für die Berechnung mindestens benötigt? Geben Sie eine zugehörige Aufteilung der Anweisungen auf die beiden Recheneinheiten an.

Aufgabe 1.4 ..... 24 Punkte

Gegeben sei folgende Ada-Prozedur:

```
type Int_Array is array (Integer range 1..5) of Integer;

procedure Sort( Data : in out Int_Array ) is
   J, K : Integer;
begin
   for I in reverse Data'First .. Data'Last-1 loop
      J := I;
      K := Data( I );
      while J < Data'Last and then Data( J+1 ) < K loop
         Data( J ) := Data( J+1 );
         J := J + 1;
      end loop;
      Data( J ) := K;
   end loop;
end Sort;
```

Im Gegensatz zum imperativen Von-Neumann-Modell sind im funktionalen Model keine Wertzuweisungen möglich. Variablen dürfen nur mit einem initialen Wert belegt und danach nicht mehr verändert werden. Neue Werte können nur als aktuelle Parameter

beim Aufruf einer Funktion oder durch das Initialisieren neuer, lokaler Variablen entstehen. Beispielsweise ist es in der Prozedur `Sort` demnach nicht möglich, dem bereits initialisierten Array `Data` wahlfrei neue Werte zuzuweisen.

1. Implementieren Sie zunächst unter Berücksichtigung der Einschränkungen des funktionalen Programmiermodells eine Funktion

   ```
   Replace( Data : Int_Array;
            Replace_Position : Integer;
            Replace_Value : Integer ) return Int_Array;
   ```

   welche ein Array liefert, welches an der Stelle `Replace_Position` den Wert `Replace_Value` hat und ansonsten mit `Data` übereinstimmt. Hinweis: Verwenden Sie Array-Slices und Array-Concatenation.

2. Überführen Sie nun die Prozedur `Sort` in eine Funktion entsprechend des funktionalen Paradigmas, indem Sie das gegebene Programm so umformen, dass es keine Zuweisungsoperationen mehr enthält. Wo im Original-Code eine Zuweisung erfolgt, muss stattdessen eine Funktion aufgerufen oder eine neue Variable eingeführt werden. Insbesondere müssen alle Schleifen durch Rekursionen ersetzt werden.

3. Implementieren Sie eine Variante des obigen Original-Codes, in der die Array-Datenstruktur `Int_Array` durch diese einfach verkettete, lineare Liste ersetzt ist:

   ```
   type Int_List is access Int_List_Element;
   type Int_List_Element is record
                               Element : Integer;
                               Next : Int_List;
                            end record;
   ```

4. Überführen Sie nun auch die Variante mit linearen Listen aus dem vorigen Aufgabenteil in eine Funktion, die dem funktionalen Paradigma genügt. Bauen Sie bei Änderungen an der Liste aber nicht jeweils eine komplett neue Liste auf, sondern führen nur die minimal notwendigen Modifikationen durch.

Aufgabe 1.5 . . . . . 10 Punkte

1. Geben Sie ein Beispiel für ein Ada-Programmfragment an, bei dessen Ausführung ein Speicherloch (Abfall) entsteht. Kann ein ähnliches Problem auch in Java auftreten? Wenn ja, wie? Wenn nein, warum nicht?

2. Geben Sie zwei unterschiedliche Ada-Beispiel-Programmfragmente an, die auf verschiedene Weisen zu einer Dangling Reference führen. Können diese Situationen auch in Java auftreten? Wenn ja, wie? Wenn nein, warum nicht?