Übungsblatt 11

Abgabe der Lösungen: Do, 25.07., 12:00 im Briefkasten am blauen Hochhaus

PRÄSENZAUFGABEN

Übung 1 Ein koinduktives Animationsstudio

Eine *Animation* ist eine unendliche Sequenz von *Sprites*. Insbesondere ist eine *Animationsschleife* eine Animation, in der sich eine *endliche* Anzahl von Sprites unendlich oft wiederholt; beispielsweise ließe sich ein nach rechts laufender Mensch mittels einer Animationsschleife darstellen. Eine Animation entspricht dann auf natürliche Weise einem *koinduktiven Typ*, und wir können mittels *Korekursion* eine Funktion $loop$: **List Sprite** $\to$ **Animation** definieren, die aus einer (als abstrakter Typ betrachteten) Liste von Sprites eine Animationsschleife erzeugt:

codata Animation where
$sprite$: **Animation** $\to$ **Sprite**
$advance$: **Animation** $\to$ **Animation**

$$\begin{aligned} sprite\ \ (loop\ \ (Cons\ s\ \ ss)) &= s \\ advance\ (loop\ \ (Cons\ s\ \ ss)) &= loop\ (snoc\ \ ss\ \ s) \end{aligned}$$

$$\begin{aligned} sprite\ \ (loop\ \ Nil) &= blankSprite \\ advance\ (loop\ \ Nil) &= loop\ Nil \end{aligned}$$

1. Es sei *walk_right* die aus sechs Sprites bestehende Liste [*s1*,*s2*,*s3*,*s4*,*s5*,*s6*], die die folgende Sequenz bezeichnen soll:

Was ist das Ergebnis der Auswertung des folgenden Terms?

$$sprite\ \ (advance\ (advance\ (advance\ (loop\ walk\_right))))$$

2. Wir möchten nun eine gegebene Animation a um einen Frame *verzögern*. Das heißt, $delay\ \ a$ soll sich genau so wie a verhalten, allerdings soll der erste Frame von a zu Beginn von $delay\ \ a$ zweimal angezeigt werden. Definieren Sie eine korekursive Funktion

$$delay\ :\ \textbf{Animation} \to \textbf{Animation},$$

die die folgende Eigenschaft für alle $n \in \mathbb{N}$ erfüllt:

$$sprite\ \ (advance^n\ \ (delay\ \ a)) = \begin{cases} sprite\ \ a & \text{falls } n = 0 \\ sprite\ \ (advance^{n-1}\ \ a) & \text{falls } n > 0 \end{cases}$$

3. Das Verzögern von Animationen ist nützlich, um die Geschwindigkeit einer Animation zu beeinflussen. Nehmen wir beispielsweise an, dass die Anzahl von Frames pro Sekunde fest vorgegeben ist, so können wir die Animationsgeschwindigkeit halbieren, indem wir *jeden* Frame (anstelle – wie oben – nur den ersten Frame) einmal verzögern. Definieren Sie eine korekursive Funktion *halfspeed* : **Animation** $\to$ **Animation**, die jeden Frame verzögert.

4. Analog ist es möglich, die Geschwindigkeit einer Animation zu verdoppeln, indem jeder zweite Frame übersprungen wird. Offensichtlich gibt es zwei Möglichkeiten, dies zu erreichen: Durch das Überspringen entweder aller Frames in *geraden* oder aber aller Frames in *ungeraden* Positionen. Definieren Sie beide Varianten als korekursive Funktionen, d.h. definieren Sie die zwei Funktionen *doublespeed_e*, *doublespeed_o* : **Animation** $\to$ **Animation** korekursiv. *Hinweis:* Möglicherweise werden Sie eine Hilfsfunktion definieren müssen.

Übung 2 Koinduktion

Um sicherzustellen, dass die Funktionen aus der vorherigen Übung korrekt sind, möchten wir nun beweisen, dass sie einige sinnvolle Eigenschaften erfüllen.

1. Es seien a und b zwei Terme des Typs **Animation**. Um mittels Koinduktion zu zeigen, dass $a = b$ gilt, müssen wir eine *Bisimulation* R (für den Typ **Animation**) finden, so dass $a\ R\ b$. Definieren Sie den Begriff einer Bisimulation für den Typ **Animation**.

2. Beweisen Sie die folgenden Eigenschaften mittels Koinduktion (falls nötig). Falls ein Beweisversuch fehlschlägt, so kann das daran liegen, dass die entsprechende Funktion nicht korrekt definiert ist!

   (a) $\forall a.\ \mathit{doublespeed\_e}\ (\mathit{halfspeed}\ a) = a$

   (b) $\forall a.\ \mathit{doublespeed\_e}\ a = \mathit{doublespeed\_o}\ (\mathit{delay}\ a)$

   Hinweis: Falls Sie Ihren Beweis nicht vollenden können, Ihre Funktion aber für korrekt halten, so ist es vielleicht notwendig, dass sie den Kandidaten für Ihre Bisimulation *vergrößern*.

HAUSAUFGABEN

Übung 3 Ein koinduktives Animationsstudio II (10 Punkte)

1. Eine Methode zur Erzeugung komplexerer Animationen ist das Voranstellen einer (endlichen) Startsequenz vor eine gegebene Animation. Definieren Sie mittels Korekursion eine Funktion *prepend* : **List Sprite** $\to$ **Animation** $\to$ **Animation**, die solch eine einmalig abzuspielende Startsequenz vor einer Animation einfügt.

2. Animationsschleifen werden üblicherweise verwendet, um eine kontinuierliche Bewegung darzustellen, wie beispielsweise eine Person, die wartet, nach links oder rechts geht, nach links oder rechts rennt, klettert etc. Beim *Übergang* von einer Schleife *a1* zu einer anderen Schleife *a2* (beispielsweise beim Übergang vom Rennen zum Gehen) sollte darauf geachtet werden, dass mit dem Übergang so lange gewartet wird, bis das "aktuelle" Sprite von *a1 kompatibel* mit dem ersten Sprite von *a2* ist (ansonsten wird der Übergang zu abrupt

erfolgen). Wir nehmen an, dass eine Funktion *compatible* : **Sprite** → **Sprite** → **Bool** gegeben ist. Definieren Sie eine Funktion *transition* : **Animation** → **Animation** → **Animation**, so dass *transition a1 a2* eine Animation ist, die so lange *a1* abspielt, bis ein mit dem ersten Sprite von *a2* kompatibles Sprite erreicht wird, und sodann zu *a2* übergeht.

3. Beweisen Sie die folgenden Eigenschaften mittels Koinduktion (falls nötig). Falls ein Beweisversuch fehlschlägt, so kann das daran liegen, dass die entsprechende Funktion nicht korrekt definiert ist!

   (a) $\forall s\ t\ ts.\ compatible\ s\ t = False \Rightarrow$
   $transition\ (loop\ [s])\ (loop\ (Cons\ t\ ts)) = loop\ [s]$

   (b) $\forall s\ ss\ t\ ts.\ compatible\ s\ t = True \Rightarrow$
   $transition\ (loop\ (Cons\ s\ ss))\ (loop\ (Cons\ t\ ts)) = prepend\ [s]\ (loop\ (snoc\ ts\ t))$

Übung 4 Streams und unendliche Bäume (10 Punkte)

Aus der Vorlesung kennen Sie bereits den Kodatentyp der Streams. Als weiteres Beispiel betrachten wir den Kodatentyp unendlicher binärer Bäume:

codata ITree a **where**
inner : **ITree** $a \to a$
left : **ITree** $a \to$ **ITree** a
right : **ITree** $a \to$ **ITree** a

Der Destruktor *inner* liest das Label (vom Typ a) der Wurzel des übergebenen Baumes aus, die Destruktoren *left* und *right* liefern den linken bzw. rechten Teilbaum.

1. Welcher Mengenoperator gehört zu dieser Kodatentypdeklaration?

2. Sei beispielsweise **Nat** der Datentyp natürlicher Zahlen mit den üblichen Rechenoperationen. Definieren Sie korekursiv eine Funktion

   itadd : **ITree Nat** → **ITree Nat** → **ITree Nat**,

   die zwei unendliche Bäume von natürlichen Zahlen knotenweise addiert.

3. Für die folgende Aufgabe benötigen wir Hilfsfunktionen über Streams: definieren Sie Funktionen

   flip : **Stream Bool** → **Stream Bool**
   choose: **Stream Bool** → **ITree** a → **Stream** a.
   mirror: **ITree** a → **ITree** a.

   Dabei soll *flip* die Bits in einem Bitstream (**Stream Bool**) invertieren und *choose* soll in Abhängigkeit der booleschen Werte im Eingabestream einen unendlichen Pfad im Eingabebaum auswählen (ist das nächste Element im Stream *True*, so wird nach links abgestiegen, ist es *False*, dann nach rechts) und diesen als Stream speichern; *mirror* soll einen unendlichen Baum analog zu Blatt 9, Übung 4 spiegeln.

4. Definieren Sie den Begriff einer Bisimulation für den Typ **ITree** a.

5. Beweisen Sie die folgenden Eigenschaften mittels Koinduktion (über Streams oder unendlichen Bäumen):

   (a) $\forall t1\ t2.\ itadd\ t1\ t2 = itadd\ t2\ t1$

(b) $\forall s\ t.\ \mathit{choose}\ (\mathit{flip}\ \ s)\ (\mathit{mirror}\ t) = \mathit{choose}\ s\ t$

Hinweis: Verwenden Sie, dass für alle Funktionen f gilt:

$$f(\textbf{if}\ a\ \textbf{then}\ b\ \textbf{else}\ c) = \textbf{if}\ a\ \textbf{then}\ f(b)\ \textbf{else}\ f(c).$$