Informatik I

Forum: https://forum-db.informatik.uni-tuebingen.de/c/ws1617-info1
Abgabestatus/Feedback: https://handin-db.informatik.uni-tuebingen.de

Übungsblatt 3 (4.11.2016)

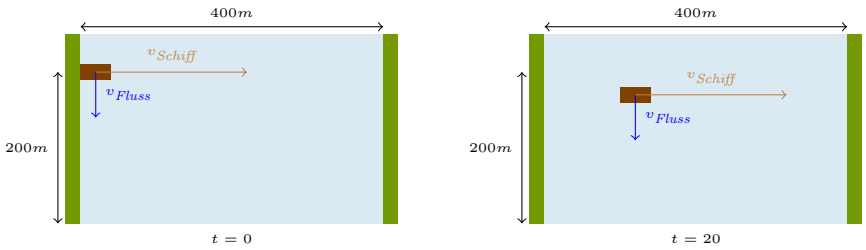
Abgabe: Freitag 11.11.2016, 14:00 Uhr

Sprachebene „Die Macht der Abstraktion — Anfänger"

Haltet euch bei jeder Prozedur, die ihr schreibt, an die Konstruktionsanleitung!

1. [14 Punkte] (*Abgabe:* `Blatt03-A1-river`)

   Vergegenwärtigen wir uns folgendes Szenario:



   Ein Schiff möchte einen $400m$ breiten Fluss überqueren. Leider ist es dabei in seiner Manövrierfähigkeit dahingehend eingeschränkt, dass es nur genau senkrecht zur Flussrichtung steuern und mit einer konstanten Geschwindigkeit von $v_{Schiff} = 5\frac{m}{s}$ fahren kann. Wegen der Strömung des Flusses wird es dadurch mit $v_{Fluss} = 1.5\frac{m}{s}$ entlang des Flusses abgetrieben.

   Der Kapitän möchte nun im Voraus wissen, ob es ihm gelingen kann, noch vor einem Sturz in den Wasserfall — der $200m$ flussabwärts des Ablegestegs liegt — das andere Ufer zu erreichen. Wir wollen ihn nun darin unterstützen, diese Frage zu klären.

   Aufbauend auf den bisher in der Vorlesung erlernten Techniken, stehen uns dazu zwei verschiedene Wege offen: *Berechnung* und *Simulation*. Beide wollen wir im Folgenden angehen.

   (a) Zur *Berechnung* geht ihr wie folgt vor:

      i. Schreibt eine Prozedur `(: plunges? (real real -> boolean))` die, gegeben eine Geschwindigkeit v_{Schiff} des Schiffes und eine Fließgeschwindigkeit v_{Fluss} (jeweils in $\frac{m}{s}$), bestimmt, ob das Schiff den Wasserfall hinunter fallen wird (`#t`) oder zuvor das andere Ufer erreicht (`#f`).

      Wichtig: Wunschdenken! — geht bei der Definition von `plunges?` davon aus, dass bereits eine Prozedur `(: drift (real real real -> real))` existiert, die für eine Geschwindigkeit v_{Schiff}, eine Fließgeschwindigkeit v_{Fluss} sowie eine gefahrene Strecke s (in Meter, senkrecht zur Flussrichtung), errechnet, wie viele Meter das Schiff in Flussrichtung abgetrieben wird.

      Beachtet darüber hinaus auch die **Konstruktionsanleitung für Prozeduren**: (1) *Kurzbeschreibung*, (2) *Signatur*, (3) *Testfälle*, (4) *Definition*.

      ii. Implementiert anschließend die Prozedur `drift`, mit Hilfe der Formel:

$$d = \frac{s}{v_{Schiff}} * v_{Fluss}$$

   (b) Zur *Simulation* geht ihr wie folgt vor:

      Ziel ist es, eine Prozedur `(: crossing (natural -> image))` zu schreiben, die die obige Szene des Schiffes, das den Fluss überquert, zu einem gegeben Zeitpunkt t (in Sekunden) nach dessen Abfahrt vom Ablegesteg, zeichnet.[1] Zur Definition dieser Prozedur benötigen wir:

[1] Die Pfeile in der obigen Darstellung dienen nur der Veranschaulichung und müssen nicht gezeichnet werden.

i. Je ein Literal der Signatur `image`, die die Zeichnung des *Ufers*, des dazwischen liegenden *Flusses* sowie eines *Schiffes* beinhaltet.

   Hinweis: Ihr müsst das Teachpack `image2.ss` in DrRacket laden, um mit Bildern arbeiten zu können (*Sprache* → *Teachpack hinzufügen...*).

ii. Eine Prozedur `(: boat-at (natural -> image))`, die das Schiff an diejenige Position verschiebt an, der es sich zu einem gegeben Zeitpunkt t (in Sekunden) nach der Abfahrt befindet. Nutzt dabei die von euch in Teilaufgabe 1(a)ii erstellte Prozedur `drift`, um die aktuelle Position (x,y) des Schiffes zu ermitteln.[2] **Versucht darüber hinaus, in sich abgeschlossene Teilprobleme in ihren eigenen Funktionen zu lösen!**

   Hinweis: Zum Zeichnen des Schiffes an der Position (x,y), sollt ihr — wie in der Vorlesung — die eingebaute Prozedur `(: put-pinhole (real real image -> image))` verwenden.

Fügt in `crossing` die Bilder von Fluss, Ufer und Schiff zum Zeitpunkt t mit Hilfe der eingebauten Prozedur `overlay/pinhole` zusammen und blendet das *pinhole* mit `clear-pinhole` aus – entsprechend dem Vorgehen in der Vorlesung.

Bemerkung: Wenn ihr nun zusätzlich das Teachpack `universe.ss` ladet, könnt ihr anschließend mit dem Befehl `(animate crossing)` die Simulation durchführen.

BONUS [2 Punkte]: Um die einzelnen Objekte zu zeichnen, könnt ihr entweder die eingebaute Prozedur `rectangle` nutzen, oder auf andere treffende Grafiken zurückgreifen (erinnert euch an *Einfügen* → *Bild einfügen...*). Für besonders schöne Abgaben könnt ihr bis zu zwei Bonuspunkte erhalten!

2. [6 Punkte] (*Abgabe:* `Blatt03-A2-stepper`)

Beobachtet die Auswertung der folgenden Ausdrücke im Stepper von DrRacket und gebt die Reduktionsregeln an, die der Stepper ausführt!

Um Schreibarbeit zu sparen, ist es sinnvoll, die Regeln des Substitutionsmodells wie folgt abzukürzen:

eval_{id}	Auswertung eines gebundenen Identifiers
$\text{apply}_{\text{prim}(f)}$	Applikation einer eingebauten Funktion f
apply_{λ}	Applikation einer `lambda`-Abstraktion

Für den Ausdruck `(+ (* 1 2) 3)` zeigt der Stepper zum Beispiel `(+ 2 3)` als erstes Zwischenergebnis an. In diesem Schritt kommt die Regel $\text{apply}_{\text{prim}(*)}$ (für die Applikation der eingebauten Prozedur `*`) zur Anwendung.

Benennt für die folgenden Ausdrücke für jeden Schritt des Steppers die Reduktionsregel, die der Stepper angewendet hat. Haltet dabei die einzelnen Zwischenergebnisse fest.

Bemerkung: In der Vorlesung wurde darauf hingewiesen, dass der Stepper bei der Auswertung eingebauter Funktionen den eval_{id}- und den darauffolgenden $\text{apply}_{\text{prim}(f)}$-Schritt zusammenfasst. Beachtet, dass ihr diesen expliziten eval_{id}-Schritt dennoch angeben sollt.

```
((lambda (a) a)
 (+ ((lambda (a) (+ a 2)) 3) 2))

(define pi 3.141)
(* 2 pi)

((lambda (pi) (* 2 pi)) pi)

(define quadrat
  (lambda (n)
    (* n n)))
(quadrat (+ 4 2))

((lambda (x) x) (lambda (x) x))
```

[2] Damit sich das Schiff nicht ab einem bestimmten Zeitpunkt an einer Position x jenseits des Ufers befindet, oder nach Ankunft am Ufer flussabwärts (mit wachsender Position y) weiter bewegt, könnt ihr mit Hilfe der eingebauten Prozedur `(: min (real real -> real))` (gibt die kleinere zweier Zahlen zurück), die x und y Positionen des Schiffes beschränken. **Alternativ dürft ihr diesen Umstand aber auch einfach außer Acht lassen!**