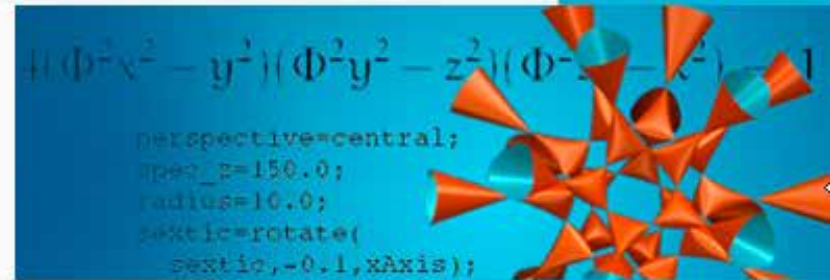


Deklarative Programmierung

Sommersemester 2018

Prof. Christoph Bockisch
(Programmiersprachen und –werkzeuge)
Steffen Dick, Alexander Bille, Johannes Frankenau,
Patrick Frömel, Niclas Schmidt, Jonas Stettin,
Robert Tran, Julian Velten



[Skript 4, 5]

Programme

- Batchprogramm
 - Wird selbstständig ausgeführt
 - Hauptfunktion
 - Eingabe: Funktionsargumente oder Eingabestrom
 - Ausgabe: Funktionsresultat oder Ausgabestrom
- Interaktives Programm
 - Ein-/Ausgabe während Ausführung der Auswertung

Ein-/Ausgabeströme

- Eingabestrom
 - Endlose Zeichenkette
 - Kann Zeichen für Zeichen gelesen werden
- Ausgabestrom
 - Endlose Zeichenkette
 - Kann Zeichen für Zeichen geschrieben werden
- Ausgabestrom einer Funktion kann als Eingabefunktion einer anderen dienen

Ein-/Ausgabeströme

- Beispiel Unix Kommandozeilen-Tools
 - ls – schreibt liste der Dateien im aktuellen Verzeichnis auf Ausgabestrom
 - grep – kopiert Zeilen, die regulärem Ausdruck entsprechen nach Ausgabestrom
 - sort – sortiert die Zeilen eines Eingabestroms

ls -l | grep "05" | sort --reverse

Ein-/Ausgabeströme

• ls -l

| grep "05"

| sort --reverse



DP-01-Folien.pdf
DP-01-Folien.pptx
DP-02-Folien.pdf
DP-02-Folien.pptx
DP-03-Folien.pdf
DP-03-Folien.pptx
DP-04-Folien.pdf
DP-04-Folien.pptx
DP-05-Folien.pdf
DP-05-Folien.pptx

DP-05-Folien.pdf
DP-05-Folien.pptx

DP-05-Folien.pptx
DP-05-Folien.pdf

Batchprogramme

- Programm letter benötigt keine Interaktion
- Beispiel für Batchprogramm
- Möglich solche Programme außerhalb von DrRacket zu starten
- Funktionsargumente müssen von Kommandozeilenargumente übernommen werden

Batchprogramme mit Racket

```
(require racket/base)
```

Verwende Funktionalität
aus Packet Base

```
(define args (current-command-line-arguments))
```

Primitive Konstante

```
(if (= (vector-length args) 3)
```

```
  (display (letter (vector-ref args 0)
```

```
    (vector-ref args 1)
```

```
    (vector-ref args 2)))
```

Schreibe Ergebnis auf
Standard-Ausgabestrom

```
  (error "Bitte genau drei Parameter übergeben"))
```

Details erstmal ignorieren.

```
; String String -> String
```

```
; generates a scam mail for the victim fst last signed as signature-name
```

```
(check-range (string-length (letter "Tillman" "Rendel" "Klaus")) 50 300)
```

```
(define (letter fst lst signature-name) ...
```

Batchprogramme in Racket

- Speichern des Racketprogramms (bspw. letter.rkt)
- Aufruf von Kommandozeile

```
$ racket letter.rkt Tillmann Rendel Klaus  
Dear Mr./Mrs. Rendel,
```

After the last annual calculations of your GNB account activity we have determined that you, Tillmann Rendel, are eligible to receive a tax refund of \$479.30.

Please submit the tax refund request (<http://www...>) and allow us 2-6 days in order to process it.

With best regards,
Klaus

Batchprogramme in Racket

- Ausgabe auf Standard-Ausgabestrom
- Kombinierbar mit anderen Kommandozeilen-Tools
- `wc` – Wörter zählen

```
$ racket letter.rkt Tillmann Rendel Klaus | wc -w  
49
```

Ausführbare Programme mit Racket

- Menüeintrag Racket -> Programmdatei eine ausführbare Datei erzeugen
- Erzeugt nativ ausführbare Datei
- Racket muss auf Zielcomputer nicht mehr installiert sein
- Dateien lesen/schreiben: teachpack 2htdp/batch-io hinzufügen

```
(write-file "letter.txt" (letter "Tillman" "Rendel" "Klaus"))  
(read-file "letter.txt")
```

Interaktive Programme

- Universe teachpack
 - Unterstützung für interaktive Programme
 - Zeitsignale
 - Mausevents
 - Tastatureingabe
 - Netzverkehr
 - Graphische Ausgabe
- Mehrere Hauptfunktionen
 - Handler werden als Reaktion auf Ereignisse ausgewertet

Interaktive Programme

Fallbeispiel:
Siehe Life Demo oder Skript

Zustände

- Interaktives Programm hat Zustand
 - Während der Ausführung
 - Welt hat Eigenschaften mit Werten
 - Zustand verändert sich
- Bei interaktiven Programmen
 - Können beeinflussen wie sich Zustand ändert
 - Funktionsaufrufe als Reaktion auf Ereignisse
 - Ereignisse lassen sich steuern
- Programmzustand: “WorldState“

Eventhandler

- Programm definiert Event-Handler
 - Funktion
 - Bei Auftreten des Ereignisses: Aufruf der Funktion
 - Eingabe: aktueller WorldState + Beschreibung des Ereignisses
 - Ausgabe: neuer WorldState
- Ereignisse:
 - **on-mouse-event**: Position des Cursors + Ereignistyp (Bewegung, Click, ...)
 - **on-tick-event**

Handler installieren

- Handler müssen mit Ereignis verknüpft werden
- Funktion big-bang aus Universe
- Ergebnis: letzter WorldState

; install all event handlers; initialize world state to 500

```
(big-bang 500  
  (on-tick on-tick-event 0.1)  
  (on-mouse on-mouse-event)  
  (to-draw render)  
  (stop-when end-of-the-world))
```

Handler installieren

- Handler müssen angeschlossen werden
- Funktion big-bang
- Ergebnis: letzter

Zählt von 500
herunter. Initialer
WorldState

Händlerfunktion für
Zeitsignal

Ein Zählerschritt
nach 0,1 Sekunde

Händlerfunktion
für Maus-Event

Funktion zum
graphischen Darstellen
des WorldState

Funktion zum Bestimmen, ob
Programm beendet ist.

; install all event handlers

(big-bang 500

(on-tick on-tick-event 0.1)

(on-mouse on-mouse-event)

(to-draw render)

(stop-when end-condition)

big-bang

- Handler müssen mit Ereignissen verknüpft werden
- Funktion
- Ergebnis

Sonderform, keine
gewöhnliche Funktion.

; install event handlers; initialize

(big-bang 500

(on-tick on-tick-event 0.1)

(on-mouse on-mouse-event)

(to-draw render)

(stop-when end-of-the-world)))

Kein Funktionsaufruf.
Spezielle Klausel von
big-bang.

big-bang

- Optionale Klauseln
 - Wird für Event-Typ kein Handler installiert, werden diese Events ignoriert
- Nur to-draw Klausel ist erforderlich
- Bindung von Event an Funktion geschieht bei big bang
→ Namen sind egal

```
; install all event handlers;  
; initialize world state to 500
```

```
(big-bang 500
```

```
  (on-tick count-down 0.1)
```

```
  (on-mouse on-mouse-event)
```

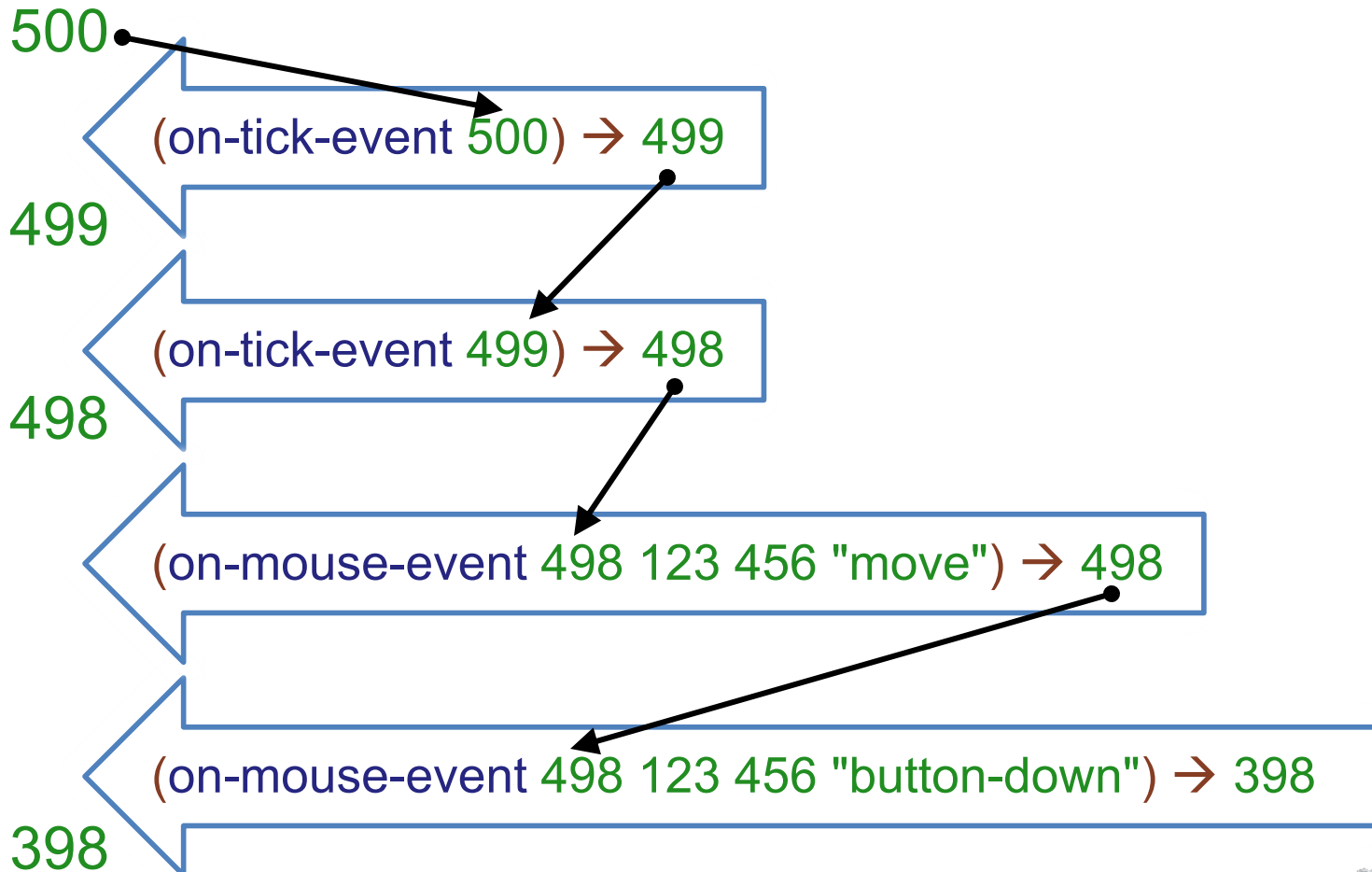
```
  (to-draw render)
```

```
  (stop-when end-of-the-world))
```

```
(define (count-down world)  
  (- world 1))
```

Sequenz von Event-Handlern

WorldState:



Sequenz von Event-Handlern

- Reihenfolge der Events bestimmt Reihenfolge der Aufrufe von Handlerfunktionen
- In funktionalen Programmiersprachen: Sequenz durch Schachtelung

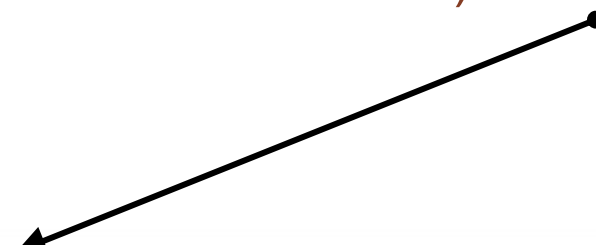
Sequenz von Event-Handlern

(on-tick-event 500) → 499

(on-tick-event 499) → 498

(on-mouse-event 498 123 456 "move") → 498

(on-mouse-event 498 123 456 "button-down") → 398

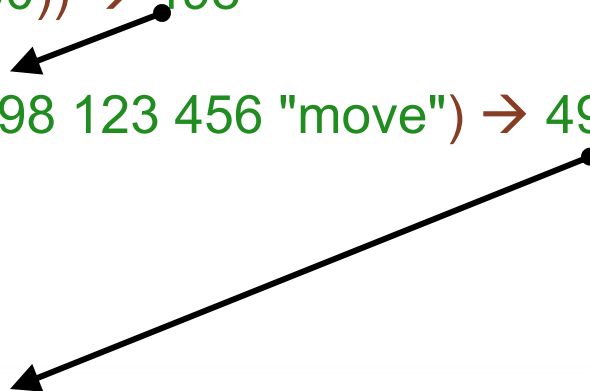


Sequenz von Event-Handlern

(on-tick-event
 (on-tick-event 500)) → 498


(on-mouse-event 498 123 456 "move") → 498

(on-mouse-event 498 123 456 "button-down") → 398



Sequenz von Event-Handlern

```
(on-mouse-event  
  (on-tick-event  
    (on-tick-event 500))  
    123 456 "move") → 498  
  (on-mouse-event 498 123 456 "button-down") → 398
```

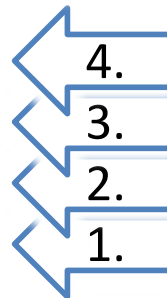


Sequenz von Event-Handlern

So können in Tests Event-Sequenzen simuliert werden.

```
(check-expect
  (on-mouse-event
    (on-mouse-event
      (on-tick-event
        (on-tick-event 500))
        123 456 "move")
      123 456 "button-down")
    398)
```

```
(on-mouse-event
  (on-mouse-event
    (on-tick-event
      (on-tick-event 500))
      123 456 "move")
    123 456 "button-down") → 398
```



Datentypen

- Menge von Werten mit gemeinsamer Bedeutung

- Bisher:

- Zahlen
- Strings
- Bilder
- Wahrheitswerte
- Selbstdefinierte Datentypen

unbegrenzt

begrenzt

Interpretation
möglicherweise
begrenzt

Enumerationstypen

- Bisher:
 - Selbstdefinierte Datentypen als Interpretation von primitiven Typen
 - Nur bestimmte Werte machen Sinn, dann:
 - “Aufzählungstyp” oder “Enumerationstyp”

; A TrafficLight shows one of three colors:

; – "red"

; – "green"

; – "yellow"

; interp. each element of TrafficLight represents which colored

; bulb is currently turned on

Enumerationstypen

- Geeignet für Fallunterscheidungen

; TrafficLight -> TrafficLight

; given state s, determine the next state of the traffic light

(check-expect (traffic-light-next "red") "green")

```
(define (traffic-light-next s)
  (cond
    [(string=? "red" s) "green"]
    [(string=? "green" s) "yellow"]
    [(string=? "yellow" s) "red"])))
```

Was passiert, wenn ein anderer String als "red", "gren", "yellow" übergeben wird?

(traffic-light-next "blue")

Fehler: *cond: all question results were false*

Enumerationstypen

- Weiteres Beispiel: MouseEvent

; A MouseEvent is one of these strings:

; – "button-down"

; – "button-up"

; – "drag"

; – "move"

; – "enter"

; – "leave"

Entwurfsrezept mit Enumerationstypen

- Entwurfsrezept Schritt 3: Tests
- Funktion mit Enumerationstyp für Argument
- Einen Test für jeden möglichen Wert des Arguments

(check-expect (traffic-light-next "red") "green")

(check-expect (traffic-light-next "green") "yellow")

(check-expect (traffic-light-next "yellow") "red")

Entwurfsrezept mit Enumerationstypen

- Entwurfsrezept Schritt 4: Schablone
- Bei Parametertyp mit Enumerationstyp:
Fallunterscheidung mit möglichen Werten

```
(define (traffic-light-next s)
  (cond
    [(string=? "red" s) ...]
    [(string=? "green" s) ...]
    [(string=? "yellow" s) ...])))
```

Wenn Argument mit Enumerationstyp nur in Hilfsfunktion verwendet wird, taucht hier keine Fallunterscheidung auf

Intervalltypen

- Aufzählung aller möglicher Werte
 - Unmöglich bei unendlich vielen Werte
 - Unsinnig bei großer Anzahl von Werten
 - Dann auch schlecht lesbar
- Angabe von Wertebereichen bzw. “Intervallen“

Intervalltypen

- Beispiel: Simulation eines landenden Ufos
 - Mittels Funktion big-bang
 - WorldState entspricht der Höhe des Ufos (von oben)
- Erweiterung: Statuszeile
 - “decending“ bei Höhe oberhalb $\frac{1}{3}$ des Bildes
 - “closing“ darunter
 - “landed“ beim Aufsetzen

Intervalltypen

Fallbeispiel:
Siehe Life Demo oder Skript

Intervalltypen

- Zahlen und Strings sind vergleichbar:

```
> ( string<? "a" "b")
```

```
#true
```

```
> ( > 9 2)
```

```
#true
```

```
> (< 2 2.1)
```

```
#true
```

```
> (string>=? "ab" "abc")
```

```
#false
```

Intervalle

- Definieren von Bereichen/Intervallen durch größer/kleiner Vergleiche
- Eine oder zwei Grenzen
- Abgeschlossene Grenze
 - Grenzwert ist enthalten
 - \geq oder \leq
- Offene Grenze
 - Grenzwert ist nicht enthalten
 - $>$ oder $<$

Intervalle

- Definition von Konstanten für Intervallgrenzen

; constants:

```
(define WIDTH 300)
```

```
(define HEIGHT 100)
```

...

```
(define BOTTOM (- HEIGHT (/ (image-height UFO) 2)))
```

```
(define CLOSE (* 2 (/ HEIGHT 3)))
```

; A WorldState is a number. It falls into one of three intervals:

; – between 0 and CLOSE

; – between CLOSE and BOTTOM

; – at BOTTOM

; interp. height of UFO (from top)

Definition von
Intervallgrenzen

Verwendung von
Intervallgrenzen

Intervalltypen

- Intervalle nicht für alle Funktionen relevant
 - render muss nicht angepasst werden
- Im Beispiel ist Rendern der Statuszeile abhängig vom Intervall
- Funktionen, die von Intervallen abhängig sind haben in der Regel eine Fallunterscheidung

Entwurfsrezept mit Intervalltypen

- Entwurfsrezept Schritt 3: Tests
- Bei Parameter mit Intervalltyp
 - Mindestens ein Test pro Intervall
 - Insbesondere Intervallgrenzen testen

Entwurfsrezept mit Intervalltypen

- Entwurfsrezept Schritt 4: Schablone
- Bei Parameter mit Intervalltyp:
Fallunterscheidung mit möglichen Werten

; WorldState -> Image

; add a status line to the scene create by render

(define (render/status y)

(cond

[(\leq 0 y CLOSE) ...]

[($<$ CLOSE y BOTTOM) ...]

[($=$ y BOTTOM) ...]))

Fallunterscheidung nicht
notwendigerweise auf
höchster Ebene

Entwurfsrezept mit Intervalltypen

- In allen drei Fällen
 - Statuszeile rendern
 - Einziger Unterschied: Text
- DRY-Prinzip: Fallunterscheidung für das Text-Argument

; WorldState -> Image

; add a status line to the scene create by render

```
(define (render/status y)
  (above
    (text
      (cond
        [(<= 0 y CLOSE) "descending"]
        [(< CLOSE y BOTTOM) "closing in"]
        [(= y BOTTOM) "landed"])
      12 "black")
    (render y)))
```

Verwenden in der
to-draw Klausel
von big-bang.

Summentypen

- Bisher:
 - Entweder Primitiver Typ
 - Oder Enumerationstyp
 - Oder Intervalltyp
- Summentypen
 - Kombination von Typen
 - Wert eines Summentyps gehört genau einer der Alternativen an

Summentypen

- Beispiel: Funktion `string->number`

- Resultat entweder Zahl oder `false`
- Datentyp für Resultat

; A NorF is one of:

; – a Number

; – false

; String -> NorF

; converts the given string into a number;

; produces false if impossible

(define (`string->number` str) ...)

Summentypen

- Vereinigung von Typen, daher auch
 - “Vereinigungstyp”
 - “Itemization”
- Aus der BSL Dokumentation:

```
(string->number s) → (union number #false)      procedure  
s : string
```

Converts a string into a number, produce false if impossible.

```
> (string->number "-2.03")  
-2.03  
> (string->number "1-2i")  
1-2i
```

Rezept: Entwurf mit Summentypen

- Beispiel

The tax on an item is either an absolute tax of 5 or 10 currency units, or a linear tax. Design a function that computes the price of an item after applying its tax.

- 1. Informationsrepräsentation

- Problemstellung unterteilt mögliche Werte in Klassen
- Explizite Angabe der Klassen

; A Tax is one of

; - "absolute5"

; - "absolute10"

; - a Number representing a linear tax rate in percent

Rezept: Entwurf mit Summentypen

- 2. Signatur
 - Verwendung des Summentypen in Signatur

; Number Tax -> Number

; computes price of an item after applying tax

(define (total-price itemprice tax) 0)

Rezept: Entwurf mit Summentypen

- 3. Tests
 - Pro Alternative mindestens ein Test
 - Bei Alternative Enumerationstyp: ein Test pro Wert
 - Bei Alternative Intervalltyp: Test für Grenzen

(check-expect (total-price 10 "absolute5") 15)

(check-expect (total-price 10 "absolute10") 20)

(check-expect (total-price 10 25) 12.5)

Rezept: Entwurf mit Summentypen

- 4. Aufteilen der Hauptfunktion (Schablone)

- Fallunterscheidung der Alternativen
- Bedingung kann auf Typ testen
Funktionen wie `number?` oder `string?`

Allgemein: Struktur der Funktion folgt aus Struktur der Daten.

```
(define (total-price itemprice tax)
  (cond [(number? tax) ...]
        [(string=? tax "absolute5") ...]
        [(string=? tax "absolute10") ...]))
```

Ist die Reihenfolge der Fälle bedeutsam?

Ja: nur wenn der Wert keine Zahl ist, dürfen wir Stringvergleich durchführen.

Rezept: Entwurf mit Summentypen

- 5. Funktionsbody implementieren
- Fälle der Schablone einzeln implementieren
- Fokus auf einzelne Alternative konzentrieren
- Ggf. Vereinfachung des cond Ausdrucks in der Nachbearbeitung

```
(define (total-price itemprice tax)
  (cond [(number? tax) (* itemprice (+ 1 (/ tax 100)))]
        [(string=? tax "absolute5") (+ itemprice 5)]
        [(string=? tax "absolute10") (+ itemprice 10)]))
```


Unterscheidbarkeit der Fälle

- ; A Tax is one of
- ; - a Number representing an absolute tax in currency units
- ; - a Number representing a linear tax rate in percent

Können wir diesen Summentyp nach dem gegebenen Rezept behandeln?

Nein: Können für einen Wert nicht entscheiden, welcher Alternative er angehört.

Alternativen müssen disjunkt sein. Ansonsten muss Zugehörigkeit durch "Tag" markiert werden.